```
################################################################################
#Name: Daniel Lewis
#Description: Homework Assignment 2
#Date: 01/31/019 (JST)
#input: CSV file from the U.S. Census
#output:
#update history:
################################################################################
######################HW3 Cleaning/munging      Dataframes#######################
################################################################################
#
#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 (String) {
  csvSource <- String
  tempFrame <- data.frame((read.csv(csvSource)))
  return (tempFrame)
}

dfStates <- readStates ("http://www2.census.gov/programs-surveys/popest/tables/2010-2011/state/totals/nst-est2011-
01.csv")

#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 thefollowing names (stateName,  base2010, base2011,Jul2010, Jul2011).
#Clean up the rows
cleanStates<-function(dfStates) {
  dfStates<-dfStates[-1:-8,]
  dfStates<-dfStates[-52:-58,]

  #Clean up the columns
  dfStates<-dfStates [,1:5]

  #Rename the Columns
  colnames(dfStates)[1]<-"StateName"
  colnames(dfStates)[2]<-"base2010"
  colnames(dfStates)[3]<-"base2011"
  colnames(dfStates)[4]<-"Jul2010"
  colnames(dfStates)[5]<-"Jul2011"
  rownames(dfStates)<-NULL

  #clean up fields
  dfStates$StateName<-gsub("\\.", "", dfStates$StateName)

  #5. Make sure the last four columns are numbers (i.e. not strings).
  conversion.integer <- function(submission) {
    submission<-gsub (",","",submission)
    submission<-gsub (" ","",submission)
    return(as.numeric(submission))
  }
  dfStates$base2010<-conversion.integer(dfStates$base2010)
  dfStates$base2011<-conversion.integer(dfStates$base2011)
  dfStates$Jul2010<-conversion.integer(dfStates$Jul2010)
  dfStates$Jul2011<-conversion.integer(dfStates$Jul2011)
  return(dfStates)
}

dfStates<-cleanStates(dfStates)

#Step 3: Store and Explore the dataset
#6. Store  the dataset into a dataframe, called dfStates.

#accomplished above printed below
print(dfStates)

#7. Test your dataframe by calculating the mean for the July2011 data, by doing:mean(dfStates$Jul2011) you should get
an answer of  6,109,645
testMean<-mean(dfStates$Jul2011)
print(testMean)

#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?
jully2011.max<-which.max(dfStates$Jul2011)
dfStates$StateName[jully2011.max]
"with a population of"
dfStates$Jul2011[jully2011.max]

#9. Sort the data, in increasing order, based on the July2011 data.
#interpreted this to mean sorted by ascending order aka smallest first
```

```
dfStates.ascending<-dfStates[order(dfStates$Jul2011),]
print(dfStates.ascending)

#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'.
#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?

Distribution<-function(vectorInput, IntegerInput){
  inputVector<-c(vectorInput)
  indexValues<-inputVector<IntegerInput
  distroStorage<-inputVector[indexValues]
  percent<-length (distroStorage)/length(inputVector)
  return(percent)
}
print (Distribution(dfStates$Jul2011, mean(dfStates$Jul2011)))
```