## **Abstract**

Data preparation is a critical, often time-consuming, task in any data-centric project. In this paper, we use the programming language R to perform various data cleansing operations on female runner-level data from the Cherry Blossom Ten Mile Run for the years 1999 through 2012. We use several built-in R functions as well as our own custom functions to parse strings and align the data in such a way that is suitable for statistical analysis. We calculate summary statistics on the cleaned data and determine that average age of the female runners across the 14 races is almost 34 years and the average race time is approximately 1 hour, 38 minutes.

## **Introduction**

In an ideal world, all datasets would be structured in such a way that any analysis of interest could be performed on the data in their raw form. However, such is rarely the case in the real world, as most datasets of interest need to be re-organized, trimmed of faulty observations or otherwise modified before any mathematical or statistical models can be run on the data. Indeed, approximately 80% of the work a data scientist does consists of data preparation in some form [1]. In this paper, we analyze a dataset that requires several steps of data cleansing before any meaningful analysis can be done on it.

Our data source contains the ages and race times of runners who ran in the Cherry Blossom Ten Mile Run, which has been held in Washington D.C. in early April of each year since 1973 [2]. Specifically, we analyze these data from the years 1999 through 2012. Our goal is to provide standard summary statistics for the female runners’ age and run time. However, before we can do that, we first perform several manipulations to the datasets for each of the 14 years of interest in order to ensure each year’s data is formatted uniformly and in such a way that statistical analysis can be accurately conducted. We facilitate this data cleaning process by creating several of our own functions in R as well as taking advantage of R’s regular expression capabilities.

The remainder of this paper is structured as described here. The following section gives a detailed explanation of our methods for data cleaning and preparation. Then, in the Results section, we view the data in its re-shaped form and provide summary statistics for the female runners across the years 1999-2012. Lastly, we conclude by mentioning a few ways in which the data can be further analyzed in order to provide meaningful insight.

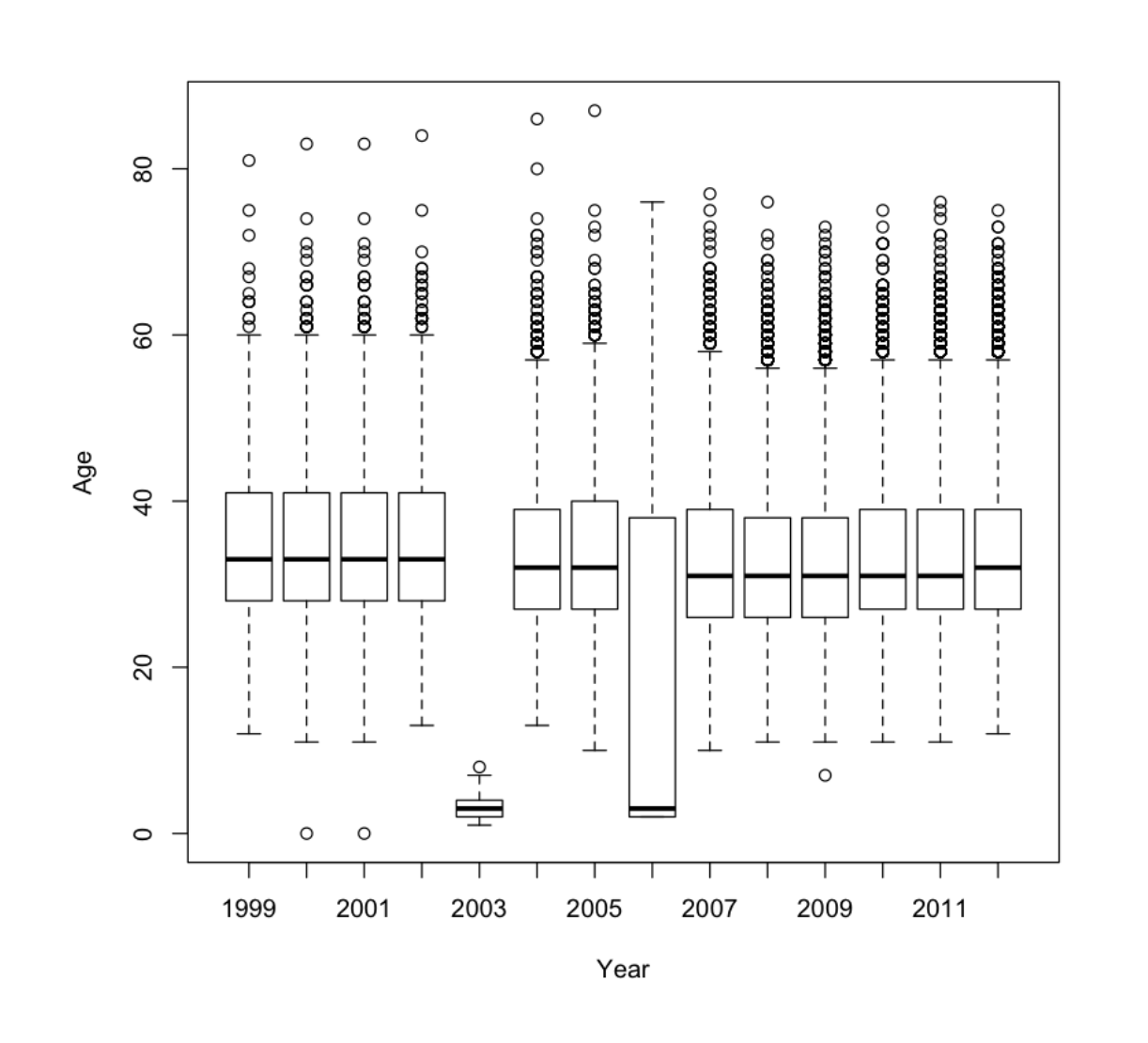
**Methods**

The data were collected from the Credit Union Cherry Blossom Ten Mile Run & 5K Run-Walk website (http://www.cherryblossom.org/). In this section, we describe the functions used in the R code, which is presented in full in the appendix. There are 14 years of data, from 1999 to 2012. For each year, the place, runner number, name, age, hometown, net time and official time were collected on each runner and stored in a text file. All of these text files need to be cleaned in order to perform any type of meaningful analysis. These data require a fully customized approach for cleaning. Not all of the years are formatted the same, and the columns are not identical across all of the years. Therefore, we need to extract the variables from each file accordingly. We will do this by programmatically interpreting the format via our own custom function, *extractVariables()*.

Before explaining what *extractVariables()* does, we will describe two key helper functions. The first helper function, *findColLocs()*, is made to encapsulate the task of finding both the starting and ending positions of the columns. Using the R function *gregexpr()*, we are able to find the spaces between the characters. Then using R’s *substring()* function, we write an *if* statement with the following logic: if a string is not equal to space, “ “, then return the space plus 1, else return the space. This *if* statement will be able to differentiate a string from space and thus will be able to find the columns in our data. The second helper function, *selectCols()*, is created to select the columns with the data. This function will take in three arguments: the names of the columns, the header row that contains the names of the columns and the locations of the spaces. The function will, using the R function *sapply()*, traverse through each column name and grab the starting position of the column name and then index it with the given columns’ name. These two helper functions will make the code modular which will make it easier to follow and be modified.

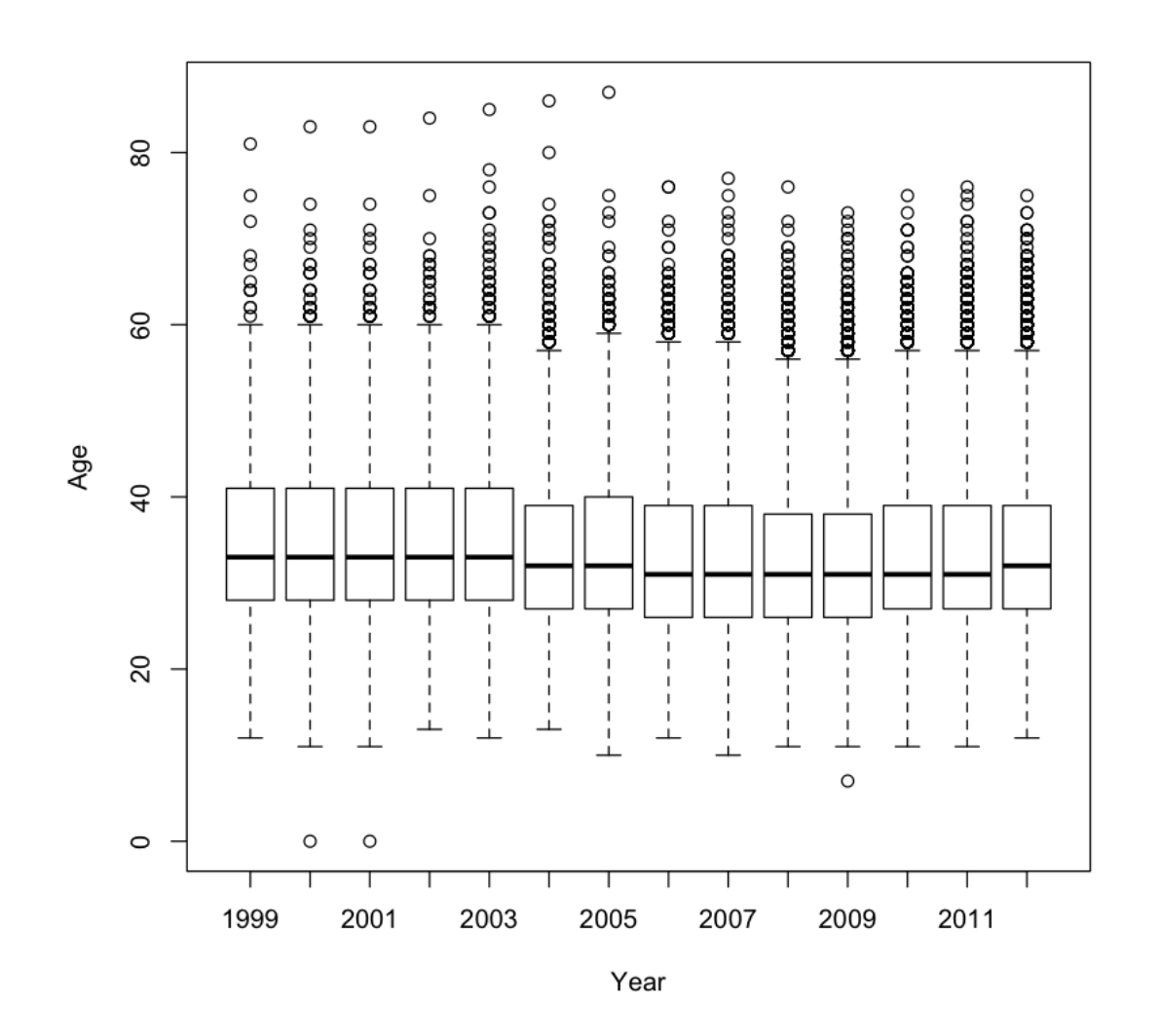
Our main function, *extractVariables()*, will take two arguments: a file name and column names. It will perform three major tasks for cleaning the data. First, each file starts with a description of what the file is and ends with a line of equal signs (=). The data of interest is after the equal signs. Therefore, we find the index of the rows with the equal signs and use an R function call *grep()* to discard the equal sign and everything preceding it. Secondly, we need to extract the two key rows and the data. Finally, using our two helper functions described in the previous paragraph, we find the column names in the data, then select those columns and index them with the given input name.

Now, the data is in a workable format. However, more cleaning still needs to be done. The main function, *extractVariables()*, outputs the data into a matrix. We will create another function, *createDF()*, to convert the matrix into a data frame and then convert each column to a data type that makes sense for each respective variable. Before we perform anymore cleaning, the data is explored. Figure 1, which displays side-by-side boxplots of age for each race year, shows a few problems with the data for 2003 and 2006. The runners in these years are unrealistically young.

Figure 1. Original side-by-side boxplots of age by race year

We see that in the 2003 data, the ages are all shifted one space to the left and in the 2006 data, the ages are shifted one space to the right. This is what is causing the erroneous values, as only one of the two digits is being read in as the age. The function, *extractVariables()*, is updated to fix this issue. Figure 2 displays the correct distribution of ages for each race for each year.

Figure 2. Corrected side-by-side boxplots of age by race year



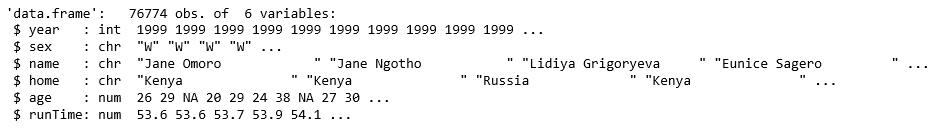
All of the other variables are fine and do not need to be manipulated in any way. Now the data can be converted into a data frame. The function, *createDF()*, to convert the matrix into a data frame takes three arguments: res, year and sex. It will determine which time variable to use out of the three given in the data based on the NA values each has. Next, the function will convert the time variables into a readable time, so we can understand it better when performing our analysis. We created a function to perform the time conversion, *convertTime()*. Each variable separates the time by a colon, “:”. Using R’s function *strsplit()*, we split each instance of time by the colon and then convert the instances into numeric data types. Finally, using *sapply()*, we traverse through each instance and divide it by 60. Now that the time is chosen and converted, all that there is to do is have the function, *createDF()*, create the data frame using R’s built-in function *data.frame()*. At the end of this function, we call *invisible()* because we do not wish to print the values of the data frame when they are assigned.

Once the data is in a data frame, we notice that there are three years that have a huge number of NA values in them: 2007, 2009 and 2010. We find that these are caused by runners who completed half the race but have no final times and also by runners who have a footnote after their time. We have to modify our function that converts the matrix into a data frame to eliminate the footnote symbols (# and \*). Now the data is merged together and ready for analysis.

**Results**

Once the raw data are imported into R, cleaned and formatted, we investigate our data frame at a high level to ensure no issues are still lingering in our data. The data frame has 6 attributes – *year*, *sex*, *name*, *home*, *age* and *runtime* –as shown in Figure 3. We have 3 numeric variables in this data – *year*, *age* and *runtime* –whereas the other 3 – *sex*, *name* and *home* – are characters. The total number of records is 76,774 which represents the total female participation in the Cherry Blossom race between 1999 and 2012. Note that the female participation was slightly higher than total male participation (70,070) over the 14 years considered.

Figure 3. Data-frame for female runners



A sample of records are displayed in Table 1.

Table 1. Sample records

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **year** | **sex** | **name** | **home** | **age** | **runTime** |
| 1999 | W | Jane Omoro | Kenya | 26 | 53.61667 |
| 1999 | W | Jane Ngotho | Kenya | 29 | 53.63333 |
| 1999 | W | Lidiya Grigoryeva | Russia | NA | 53.66667 |
| 1999 | W | Eunice Sagero | Kenya | 20 | 53.91667 |
| 1999 | W | Alla Zhilyayeva | Russia | 29 | 54.13333 |
| 1999 | W | Teresa Wanjiku | Kenya | 24 | 54.16667 |

Now, we run summary statistics for all the variables as shown in Table 2 and Table 3. These summary statistics confirm the presence of all 14 years (1999 - 2012) in the data. The figure also shows that the length of all of the character variables are 76,774 which indicates that there are no missing observations in these attributes. On the contrary, the attributes – *age* and *runtime* – have 24 and 5,434 missing observations, respectively. The oldest lady who participated in the Cherry Blossom race was 87 years of age. The average age of participants is 34 years old and 75% are younger than 39 years. That being said, the oldest women to participate during the 14-year period was 87 years of age.

The time to finish the womens’ ten-mile race was wide spread. The average time taken to cross the finish line was 98.07 minutes, whereas 25% of the participants finished it below 88.50 minutes and 75% completed in less than 106.77 minutes. The fastest time recorded in the female ten-mile run over 14 years is 51.73 minutes while the slowest time is 177.52 minutes. Based on the male run statistics available from *Case Studies in Data Science with R* (Nolan and Duncan Temple Lang, 2015) we observe that overall the female runners were slightly slower than the male runners as in the 14 years records.

Table 2. Summary statistics for character variables for female runners

|  |  |  |  |
| --- | --- | --- | --- |
| **Statistic** | **Year** | **Age** | **Runtime** |
| **Length** | 76,774 | 76,774 | 76,774 |

Table 3. Summary statistics for numeric variables for female runners

|  |  |  |  |
| --- | --- | --- | --- |
| **Statistic** | **Year** | **Age** | **Runtime** |
| **Minimum** | 1999 | 0.00 | 51.73 |
| **1st Quartile** | 2005 | 27.00 | 88.50 |
| **Median** | 2008 | 32.00 | 97.30 |
| **Mean** | 2007 | 33.84 | 98.07 |
| **3rd Quartile** | 2010 | 39.00 | 106.77 |
| **Maximum** | 2012 | 87.00 | 177.52 |
| **NA's** |  | 24 | 5,434 |

## **Possible Future Work**

Since the data are now in the correct format, one may begin a more thorough analysis. This analysis could go in many directions. Here are three ideas for future work that we propose. The first idea is to explore the relationship between age and run time. This can be examined by creating plots of the runners’ ages against their runtimes. Another interesting analysis could involve fitting models to average the performance of runners. This model may be used to predict the run time based on age. The third idea is to model how running time changes as an individual ages. This model could consist of a short time series that are at most 14 years long and relies on the cross-sectional aspect of the data. There are many ways to go about analyzing these data. We have provided three ideas that could be further investigated.

## **References**

1. G. Press, "Cleaning Big Data: Most Time-Consuming, Least Enjoyable Data Science Task, Survey Says". Forbes. 2016. https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says/#3e9fbcbe6f63
2. Deborah Nolan and Duncan Temple Lang, “Case Studies in Data Science with R”. University of California, Berkeley and University of California, Davis. 2015, pp. 45-63. http://www.rdatasciencecases.org

## **Appendix – R Code**

## Bring the data files in

femalefilenames = paste("/home/science/dan\_sudip\_tim/MSDS-7333-Quantifying-the-World/Unit 8 Case Study/Code", 1999:2012, ".txt", sep = "")

femaleFiles = lapply(femalefilenames, readLines)

names(femaleFiles) = 1999:2012

## Create a function to find the columns

findColLocs = function(spacerRow) {

spaceLocs = gregexpr(" ", spacerRow)[[1]]

rowLength = nchar(spacerRow)

if (substring(spacerRow, rowLength, rowLength) != " ")

return( c(0, spaceLocs, rowLength + 1))

else return(c(0, spaceLocs))

}

## Create a function to select the columns

selectCols = function(shortColNames, headerRow, searchLocs) {

sapply(shortColNames, function(shortName, headerRow, searchLocs){

startPos = regexpr(shortName, headerRow)[[1]]

if (startPos == -1) return( c(NA, NA) )

index = sum(startPos >= searchLocs)

c(searchLocs[index] + 1, searchLocs[index + 1])

}, headerRow = headerRow, searchLocs = searchLocs )

}

## Create a function that uses the two function above

## to extract the variables from each years txt file

extractVariables =

function(file, varNames =c("name", "home", "ag", "gun",

"net", "time"))

{

# Find the index of the row with =s

eqIndex = grep("^===", file)

# Extract the two key rows and the data

spacerRow = file[eqIndex]

headerRow = tolower(file[ eqIndex - 1 ])

body = file[ -(1 : eqIndex) ]

# Obtain the starting and ending positions of variables

searchLocs = findColLocs(spacerRow)

locCols = selectCols(varNames, headerRow, searchLocs)

Values = mapply(substr, list(body), start = locCols[1, ],

stop = locCols[2, ])

colnames(Values) = varNames

invisible(Values)

}

## Call the function

# This function call errors out because of the years 2000 and 2001.

# They do not have the ==== at the begning of the file. But the mens

# files do. So we will take the == from a mens file and slap it into

# the females 2000 and 2001 fiels.

femaleResMat = lapply(femaleFiles, extractVariables)

length(femaleResMat)

## Lets explore those years

femaleFiles[['2000']][1:15]

femaleFiles[['2001']][1:15]

## The years above are missing the header with the =====

men2001 = readLines("/Users/timmcwilliams/Documents/DataScience/QTW/Case Study Unit 8/men2001.txt")

men2001[1:15]

## Assign the === to the female files

femaleFiles[['2000']][9:10] = men2001[12:13]

femaleFiles[['2001']][9:10] = men2001[12:13]

## Now call the extraction function again

femaleResMat = lapply(femaleFiles, extractVariables)

length(femaleResMat)

## Check the data for each year

sapply(femaleResMat, nrow)

## Lets check the data by looking at age

age = as.numeric(femaleResMat[['2012']][ , 'ag'])

age = sapply(femaleResMat,

function(x) as.numeric(x[ , 'ag']))

boxplot(age, ylab = "Age", xlab = "Year")

## Check for NAs

sapply(age, function(x) sum(is.na(x)))

## Convert time

convertTime = function(time) {

timePieces = strsplit(time, ":")

timePieces = sapply(timePieces, as.numeric)

sapply(timePieces, function(x) {

if (length(x) == 2) x[1] + x[2]/60

else 60\*x[1] + x[2] + x[3]/60

})

}

## Create a function to convert the data from a matrix to a df

createDF =

function(Res, year, sex)

{

# Determine which time to use

useTime = if( !is.na(Res[1, "net"]) )

Res[ , "net"]

else if( !is.na(Res[1, "gun"]) )

Res[ , "gun"]

else

Res[ , "time"]

runTime = convertTime(useTime)

Results = data.frame(year = rep(year, nrow(Res)),

sex = rep(sex, nrow(Res)),

name = Res[ , "name"],

home = Res[ , "home"],

age = as.numeric(Res[, "ag"]),

runTime = runTime,

stringsAsFactors = FALSE)

invisible(Results)

}

## Call the function

femaleDF = mapply(createDF, femaleResMat, year = 1999:2012,

sex = rep("W", 14), SIMPLIFY = FALSE)

# There are a large number of NAs in 2007, 2009, and 2010, and it appears

# that all of the run time values for 2006 are NA. Let’s begin by examining

# a few of the records in 2007, 2009, and 2010 that have an NA in run time.

# We find that these are caused by runners who completed half the race but

# have no final times and by runners who have a footnote after their time.

# We have to modify our createDF() fucnton to eliminate the footnote

# smybols(# and \*) .

createDF = function(Res, year, sex)

{

# Determine which time to use

if ( !is.na(Res[1, 'net']) ) useTime = Res[ , 'net']

else if ( !is.na(Res[1, 'gun']) ) useTime = Res[ , 'gun']

else useTime = Res[ , 'time']

# Remove # and \* and blanks from time

useTime = gsub("[#\\\*[:blank:]]", "", useTime)

runTime = convertTime(useTime[ useTime != "" ])

# Drop rows with no time

Res = Res[ useTime != "", ]

Results = data.frame(year = rep(year, nrow(Res)),

sex = rep(sex, nrow(Res)),

name = Res[ , 'name'], home = Res[ , 'home'],

age = as.numeric(Res[, 'ag']),

runTime = runTime,

stringsAsFactors = FALSE)

invisible(Results)

}

## Call the function

femaleDF = mapply(createDF, femaleResMat, year = 1999:2012,

sex = rep("W", 14), SIMPLIFY = FALSE)

sapply(femaleDF, function(x) sum(is.na(x$runTime)))

## Now we have fixed the NAs

# Finally, we combine the race results for all years and females into

# one data frame using the do.call() function to call rbind() with the list

# of data frames as input. The do.call() function is very convenient when

# we have the individual arguments to a function as elements of a list.

cbFemale = do.call(rbind, femaleDF)

save(cbFemale, file = "cbFemale.rda")

dim(cbFemale)

head(cbFemale)

str(cbFemale)

summary(cbFemale)