Male Age distribution shifts from 1999 - 2012

Jeffery Lancon, Manisha Pednekar, Andrew Walch, David Stroud

MSDS 7333 - Quantifying the World - Case Study # 4

02/05/2019

Abstract

The purpose of this study is to; a) obtain data from website(s) by scraping techniques, b) clean, parse and organize the scraped data into a useable format and c) perform statistical analysis on the dataset. The study is an extension of work previously conducted by the authors of *Modeling Runners' Times in the Cherry Blossom Race (MRCBR),* Nolan and Lang (NTL)[1][2]. We will utilize The Credit Union Cherry Blossom Ten Mile Run and 5K Run-Walk race results for the male runners in the 10-mile races for the years 1999 thru 2012. Results were obtained from the race's official website [3]. Our extended analysis will build a model to investigate and compare the male age distribution shifts throughout the 14-year period between 1999 thru 2012. We are looking at the outcome base statistics such as quantile–quantile plots, boxplots, and density curves to make our analysis.

Introduction

The Credit Union Cherry Blossom Ten Mile Run and 5K Run-Walk is a charity event that focuses on raising money to support Children's Miracle Network Hospitals. The Children's Miracle Network (CMN) is a non-profit international organization that helps treat millions of children across the U.S. and Canada.

Our analysis aims to determine whether or not the male runners in 1999 were older than the runners in 2012. To conduct this analysis, we checked the assumptions of normality by measuring our findings with a density curve, boxplot and quantile-quantile plot. This type of analysis allows us to visualize the type of distribution that we are working with, compare the results of each analysis and ascertain whether or not there is consistency in the three methods.

The compilation of density curves, boxplots and quantile-quantile plots allows one to visually inspect how the distributions change over the years and whether the change was gradual or not. One of the key measures of success is whether or not the results are consistent. You could easily surmise that your conclusions are valid if all three of the visual models are telling you that, in fact, that male runners entering the race were older in 1999 than 2012.

The information below has been broken up into several different sections, web scraping, Analysis and future work/discussions. These sections will answer all the questions from exercise ten and provide insight to the data from The Credit Union Cherry Blossom Ten Mile Run and 5K Run-Walk.

Male Age distribution shifts from 1999 - 2012

Jeffery Lancon, Manisha Pednekar, Andrew Walch, David Stroud

MSDS 7333 - Quantifying the World - Case Study # 4

02/05/2019

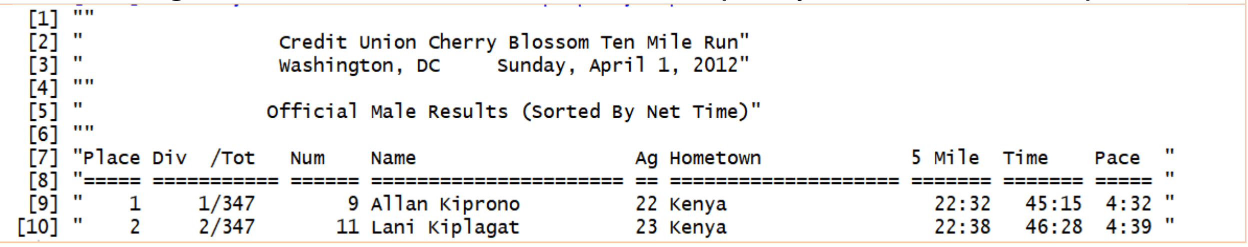
Methods

All methods described in this study are implemented utilizing the R programming language. The methods section will describe reading the tables of the Cherry Blossom race results into R, data cleaning, data formatting, data exploration and data analysis.

Web scraping the data from the Web

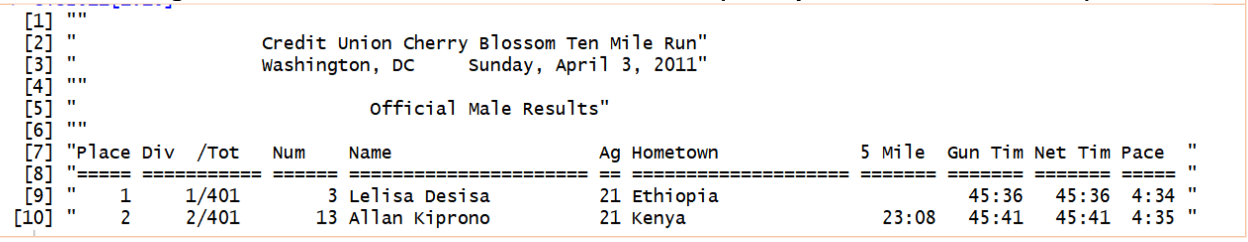
The first step in this study is to scrap the race results data from the Cherry Blossom achieve race results pages. The yearly results are stored in HTML as a block of plain text. We will extract the text blocks of the individual race results in a data format that can be utilized in R. Cherry Blossom Run website contains links to each year's race results from 1999 to 2012. The data set in this case study contains the race times for male and female runners in the Cherry Blossom Ten Mile Run held in Washington DC. The data is from 1999 through 2012 and was obtained from www.cherryblossom.org. The race result text blocks (tables) from the web pages are formatted slightly differently from year to year. Following MRCBR's methodology and experience with this case study, we manually created lists of URLs for the men's race results from 1999 to 2012. Acquisition of data via Web scraping is a trial and error process as Websites are constantly changing and efforts to programmatically scrape the data today might not work tomorrow because of changes/modifications that may occur on the Website. Then we created a custom function named "extractResTable" using "htmlParse()" function of XML package to scrape the data from the Webpages to extract the men's result tables across the years. At this point, we downloaded and stored the scraped data locally in .txt files. There is one text file for each year. These files contain variables such as name, hometown, age, gun time, and net time. Not all files adhere to the same format or contain the same set of variables which are visible. Evidence of this can be seen in Fig 1A and Fig 1B.

Figure 1A - 2012 Men's 10 Mile Race Results (Cherry Blossom Ten Mile Run)



```
 [1] ""
 [2] "                   Credit Union Cherry Blossom Ten Mile Run"
 [3] "                   Washington, DC     Sunday, April 1, 2012"
 [4] ""
 [5] "                  Official Male Results (Sorted By Net Time)"
 [6] ""
 [7] "Place Div  /Tot   Num    Name                   Ag Hometown             5 Mile  Time    Pace  "
 [8] "===== =========== ====== ====================== == ==================== ======= ======= ===== "
 [9] "    1     1/347       9 Allan Kiprono          22 Kenya                  22:32   45:15  4:32 "
[10] "    2     2/347      11 Lani Kiplagat          23 Kenya                  22:38   46:28  4:39 "
```

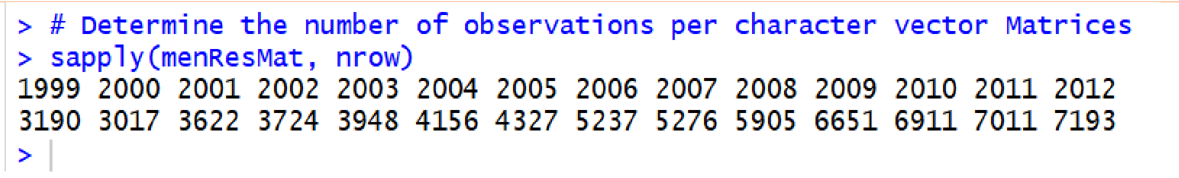
Figure 1B - 2011 Men's 10 Mile Race Results (Cherry Blossom Ten Mile Run)



```
 [1] ""
 [2] "                   Credit Union Cherry Blossom Ten Mile Run"
 [3] "                   Washington, DC     Sunday, April 3, 2011"
 [4] ""
 [5] "                             Official Male Results"
 [6] ""
 [7] "Place Div  /Tot   Num    Name                   Ag Hometown             5 Mile  Gun Tim Net Tim Pace  "
 [8] "===== =========== ====== ====================== == ==================== ======= ======= ======= ===== "
 [9] "    1     1/401       3 Lelisa Desisa          21 Ethiopia                       45:36   45:36  4:34 "
[10] "    2     2/401      13 Allan Kiprono          21 Kenya                  23:08   45:41   45:41  4:35 "
```

We can see from Fig. 1A & 1B. that each year's data files have similar attributes, similar header structure, but they are not identical. We will use these similarities to help us parse the data into meaningful variables. The column names are separated from the data by a row of equal signs. The equal signs have a space in between them where a new column / variable begins. The above Figures depicts how different years contain slightly different attributes, example; 2011 contains a "Net Tim" column, but 2012 does not.
Following MRCBR's original coding and additional exploratory work of our own, we developed a function that can read in all the txt files into a list of character matrices, accounting for their formatting differences. Figure 2 shows a summary of the number of observations(runners) in each year.

Figure 2 – Number of Observations in each Year (Men's 10 Mile Race)



```
> # Determine the number of observations per character vector Matrices
> sapply(menResMat, nrow)
1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012
3190 3017 3622 3724 3948 4156 4327 5237 5276 5905 6651 6911 7011 7193
>
```

Data cleaning

Now, that the web scraping process was successfully completed and the number of observations in each year(matrix) looks reasonable, we can now convert them into a format that lends itself to analysis and perform additional data cleaning as necessary. To do this, we will convert the character matrix into a dataframe and convert each column to a data type that makes sense for each respective variable. We will start by converting the age column to numeric values and checking its validity using "summary" statistics and graphics. We will use a boxplot, Figure 3, to visualize the ages in each year to see if any of the years look anomalous.

Quickly, we identified some problem from the above boxplot for year 2003 and 2006, so we investigate the root cause of the problems. The age header in 2003 is shifted by one character and in 2006, some ages are shifted in the column, so we are only grabbing the 1st digit of Age values. To address these issues, we retrace our steps and modify the code of the parsing function to account for idiosyncrasies of 2003 and 2006, rerun the parsing function and check the age statistics again. Figure 4's box plot shows that the age variable statistics are reasonable. We further removed any observations that had "NA" values for the Age variable, blank lines, and footnote rows.

Figure 3 – Boxplot of Runner's Age by Year (Men's 10 Mile Race) – First Pass

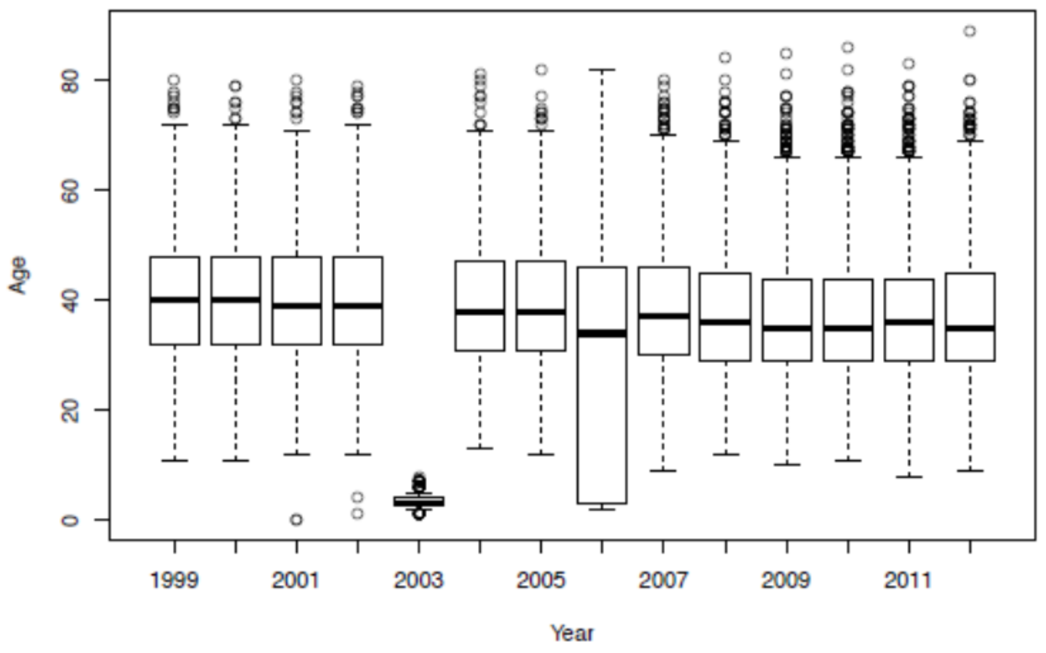
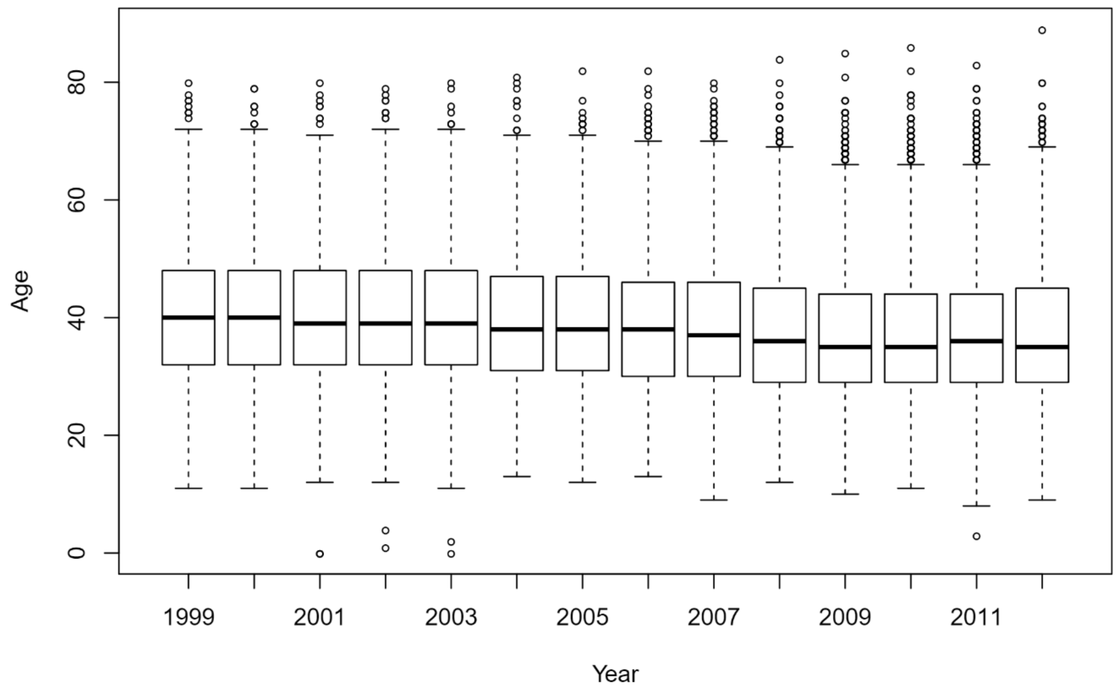
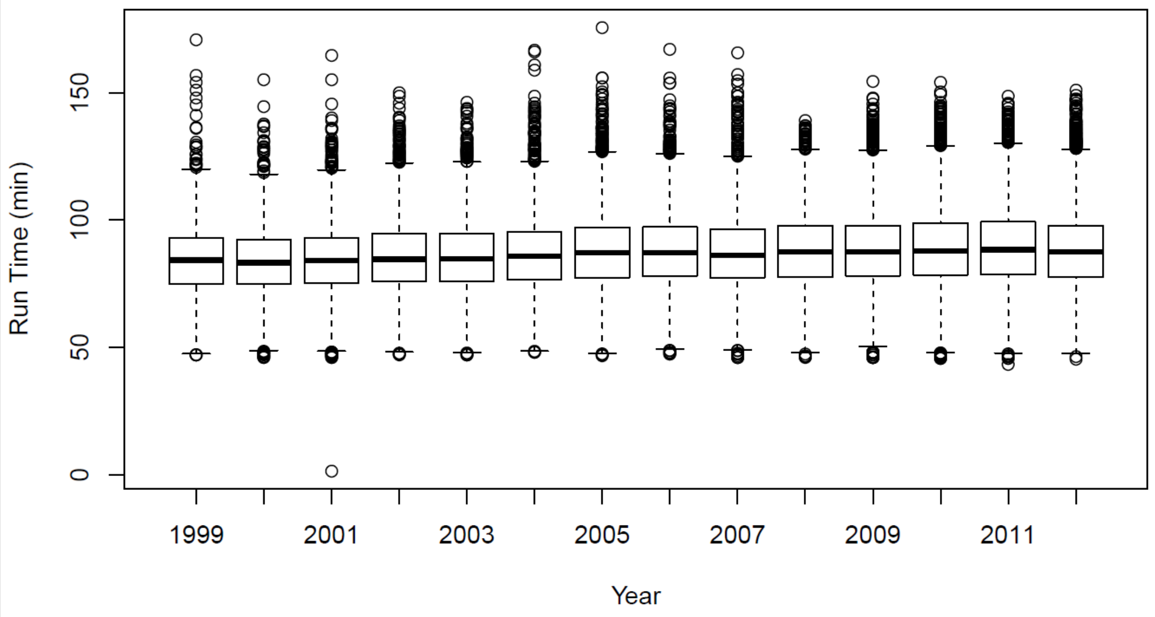


Figure 4 – Boxplot of Runner's Age by Year (Men's 10 Mile Race) – Final Pass



We will now clean up the race times. Some runners finished in under an hour, their time values only have minutes and seconds. Other runners' times have hours, minutes, and seconds. Since the race times are not all in the same format, we will convert all times to minutes. We do this by a combination of string splitting, using the ':' as the split character, converting the new character strings to numeric variables, converted them to minutes, and added them back together.
After removing records with null run times and to strip off the special characters from footnoted runtimes, we created a boxplot, Figure 5, to visually inspect the male runner's time data for any obvious anomalies.

Figure 5 – Boxplot of Runner's Times by Year (Men's 10 Mile Race)



With the data set sufficiently cleaned, i.e. runner's time and ages per year are clean, we can move beyond MRCBR's initial analysis and investigate the age distribution shift among Male runners between 1999 and 2012.

Analysis

Measuring male age distribution shifts between 1999 and 2012

In order to measure the age distributions of the male's runners from 1999 until 2012 we used three different comparisons to determine if the runners in 1999 where older than the runners in 2012. The methods we employed were density plots, boxplots and quantile-quantile plots. The aforementioned plots were also used to determine how the distributions changed over the years and if the changes were gradual.

Male Age distribution shifts from 1999 - 2012

Jeffery Lancon, Manisha Pednekar, Andrew Walch, David Stroud
MSDS 7333 - Quantifying the World - Case Study # 4
02/05/2019

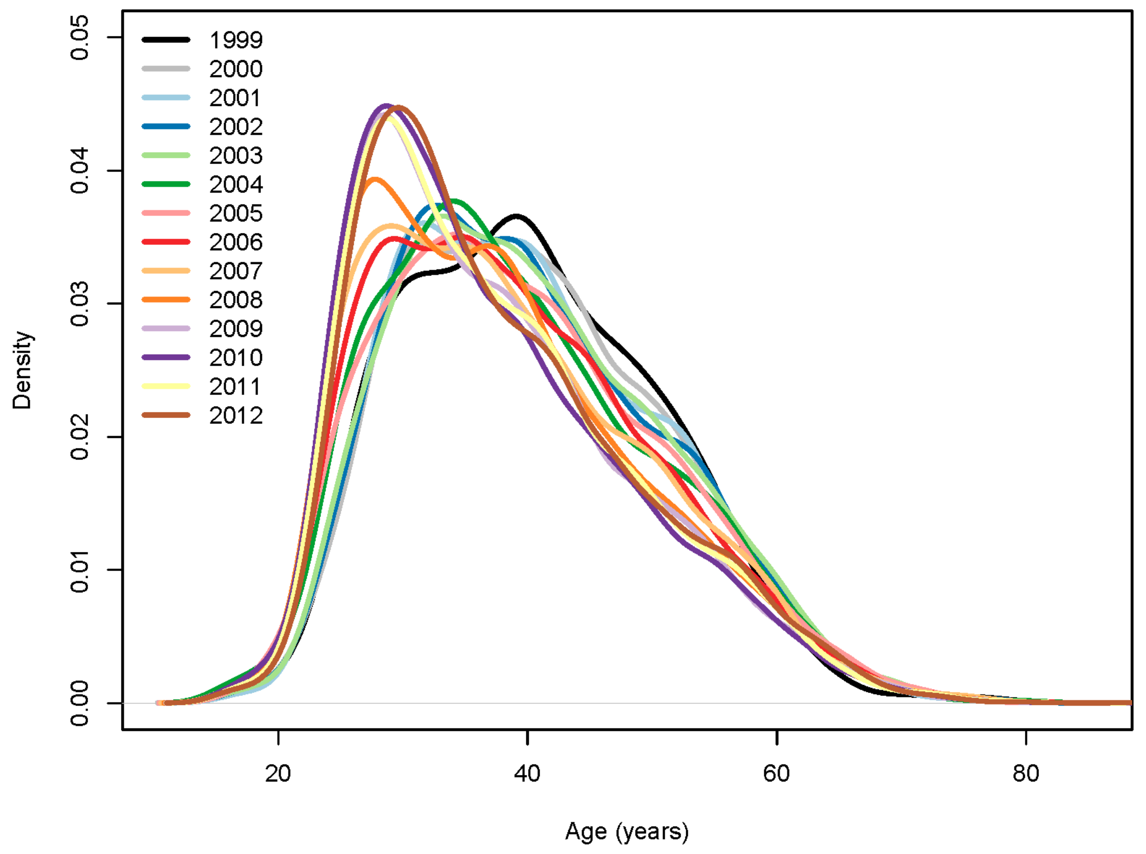
Density Curves

Density cures are used to show probability and the area under the curve is equal to 100 percent of all probabilities. For the purposes of our example, you can consider the total area under the curve as equal to one. Here we are measuring the change in age of the runners from 1999 until 2012.

Density curves are not always symmetrical and often come in many shapes and sizes. The male age distribution has a right-skew, meaning that it is positively skewed. A right-skewed graph will have the mean to the right of the median.

The density curve below shows a clear indication of difference between male runners ages in 1999 verses 2012. The curve is telling us that the average age of the runners was lower in 2012 than in 1999. It appears that this shift began in 2009 and what was a normal distribution shape turned into a right-skewed shape.

Figure 6 – Age Density Curve by Year (Men's 10 Mile Race)



Male Age distribution shifts from 1999 - 2012

Jeffery Lancon, Manisha Pednekar, Andrew Walch, David Stroud
MSDS 7333 - Quantifying the World - Case Study # 4
02/05/2019

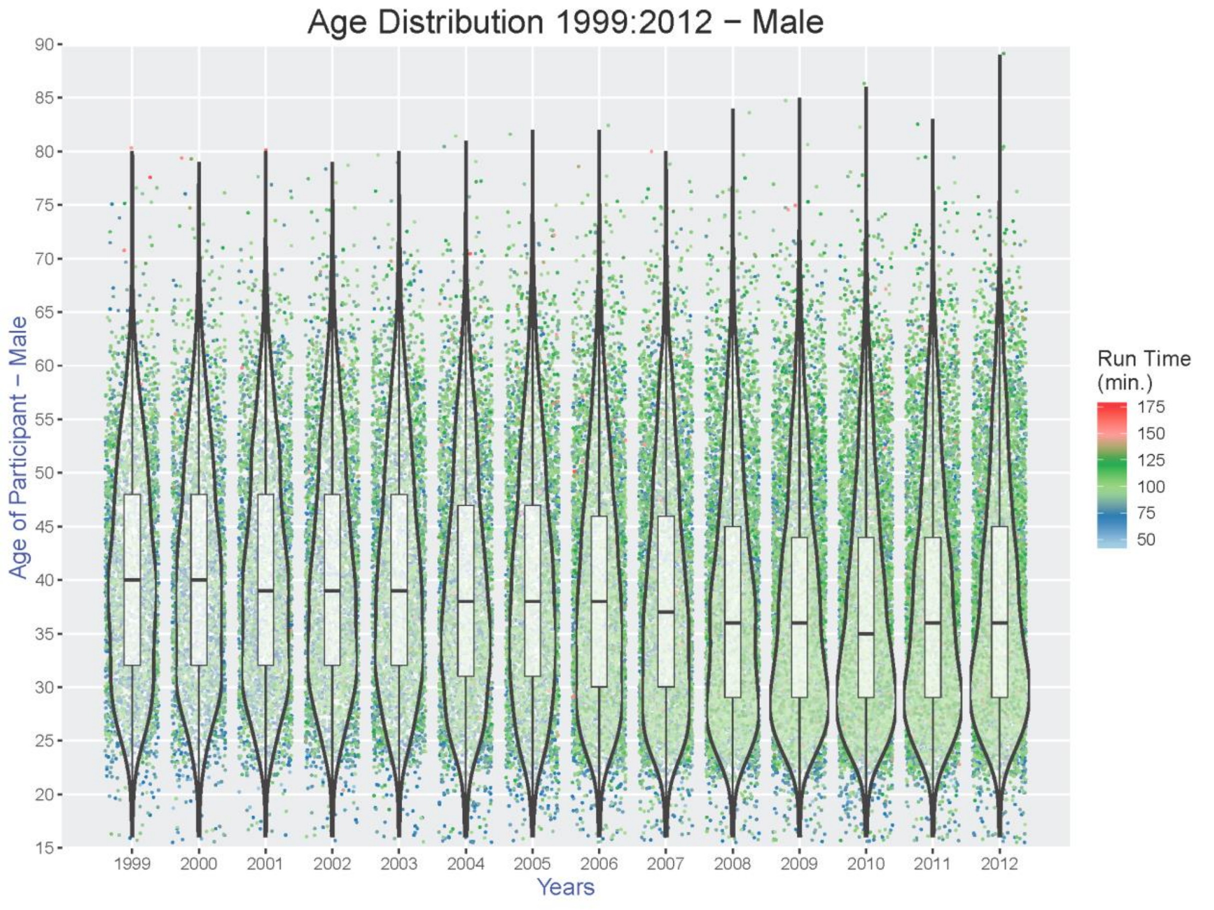
CREDIT UNION CHERRY BLOSSOM TEN MILE RUN

Boxplots

The box plot, Figure 7, is a standardized way of displaying the distribution of data based on the five-number summary: minimum, first quartile, median, third quartile and maximum. The interquartile range, or IQR, is consistent between 1999 and 2012. Thus, approximately fifty percent of the runners were in the twenty-seven to forty-seven year old age group throughout the fourteen years that we analyzed. However, beginning in 2009, you notice that the third quartile slips moves down to forty-five years old from approximately forty-seven years old.

It is also worth noting that each year we have runners that are outliers, or above the maximum quartile. These runners are typically more than seventy years old. Suspected outliers are not uncommon in a larger (greater than 100 data-points), normally distributed dataset. Our analysis shows that the outliers are legitimate, as many people that run ten-mile races are more than sixty-five years old. This information is also valuable to our research because it supplies an accurate frame of reference in determining the rate of change with the density curves.

Figure 7 – Box, Violin and Scatter Plot Age by Year including Run Times (Men's 10 Mile Race)

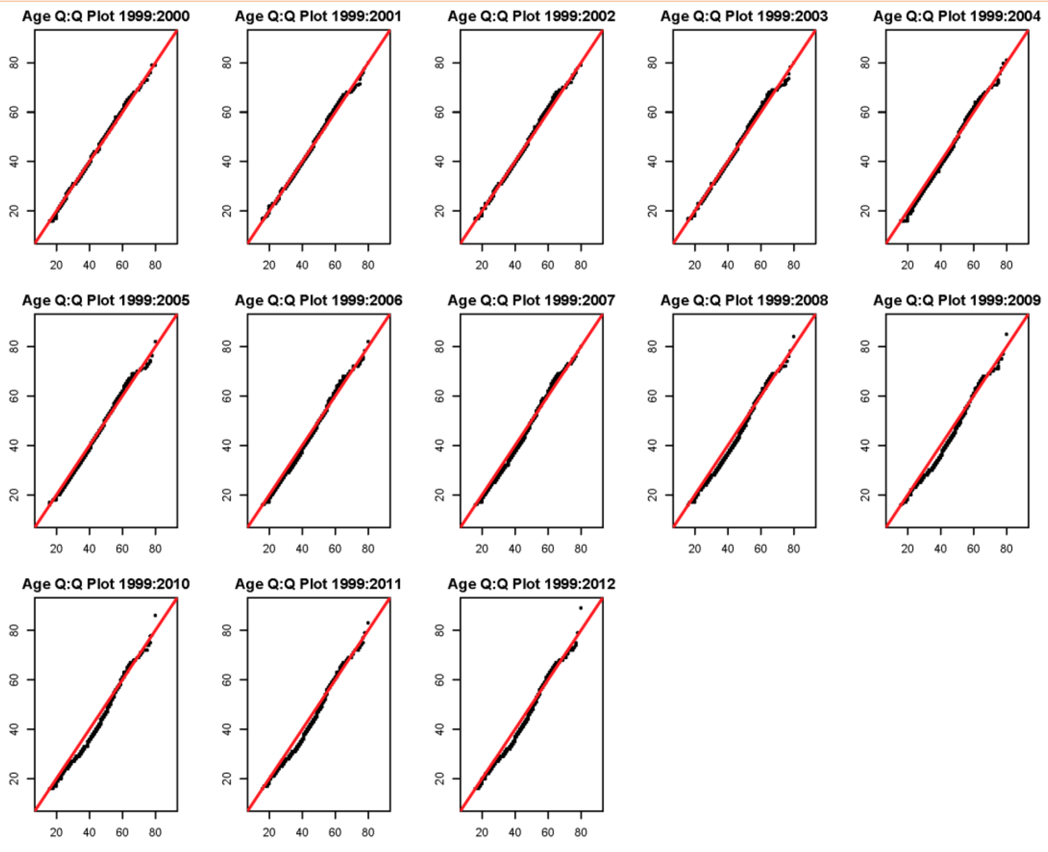


Violin plots were added into our boxplots to determine different age levels and changes in probabilities of their respective ages. Wider sections of the violin plot represent a higher probability that members of the population represent a given value; the skinnier sections represent a lower probability that the population represents a given value. The grouped violin plots illustrate the most compelling evidence that you have a younger group of runners in 2012 versus those in 1999. It also confirms that starting in 2009 you have a higher probability that the members of the population are less than thirty years old.

Quantile–Quantile plots:

Our last plot, Figure 8, is quantile-quantile (q-q) plots which is a graphical technique for determining if two data sets come from populations with a common distribution. We plot the quantiles of the first data set against the quantiles of the second data set. Our analysis shows plots that are a year by year comparison of the race data from 1999 until 2012. By a quantile, we mean the fraction (or percent) of points below the given value. Specifically, the 0.25 (or 25 percent) quantile is the point at which 25 percent of the data fall below and 75 percent fall above that value.

Figure 8 – Q-Q Plot Runner's Ages 1999 vs (2000 thru 2012) (Men's 10 Mile Race)



We are looking to determine if two of these sets come from a population with a different distribution. Using a 45-degree reference line, if two sets come from the same distribution, the points will fall approximately along the reference line. Greater departures from the reference line offer evidence that is conclusive of two data sets that come from a population with different distributions.

Beginning in 2009 you see a greater departure from the reference line in the 25 to 35 age range, as well as those above the age of 60. Note that this is consistent with our previous analysis of the box-plots and density curves. It is also worth mentioning that the departure continues to increase through 2012.

Thus, the data sets from 2009 until 2012 do not appear to come from populations with a common distribution. During these time periods the ages of 25 to 40 and those runners above age 60 have a greater departure from the reference line. You do have slight departures from the reference lines in almost every year for the runners above age 60. However, the departure from the reference line only become apparent for the runners between the ages of 25 to 40 during year 2009 and they continue this trend until 2012.

Summary of the Analysis

Our analysis of using the three methods of density curves, boxplots and quantile-quantile plots support the statement that the runners in 1999 were typically older than the runners in 2012. The density curve below shows a clear indication of difference between male runners ages in 1999 verses 2012. The grouped violin plots confirm that starting in 2009 you have a higher probability that members of the population are less than thirty years old

From 2009 until 2012, the change was significant for runners between the ages of 20 and 40. We found this behavior to be consistent among all three of the methods that we used for measurement. For runners over the age of forty, there was a gradual decline that was uniform with their respective age group as seventy-five percent of the runners are approximately twenty-seven to forty-seven years old throughout the fourteen years that we measure.

Future Work / Discussions

Legal issues surrounding web scraping of data

The scrapping of data from the web can be a convenient, low cost, source of data. The act of scraping data is legal but the way in which someone uses that data may not be legal. With the increase awareness of personal data privacy and increasing regulation of how businesses/entities can gather, access, and use this personal data, increased caution must be exercised when scraping data from the web. The enactment of EU's General Data

Protection Regulation (GDPR), which provides protection of EU citizens regarding the processing and utilization of personal data, brought to the forefront the necessity of properly handling and using data. The GDPR allows for the imposing of non-trivial fines for those who break these rules. Fines can range from €10 million, or 2% of the worldwide annual revenue for lower level violations up to €10 million, or 2% of the worldwide annual revenue for more substantial violations.

Let's investigate the legal issues surrounding our scrapping efforts in obtaining the men's race results for the 'Cherry Blossom 10 Mile Run' between 1999 to 2012. The data is freely presented to the public for review. Each runner probably signed a waiver when they registered for the race, to allow the organizers of the race to publish the results of that race, either in print or digital form. Since the data reaches back beyond the age of internet, the digital form of the data was probably not covered.

Our efforts to scrape the data from 'Cherry Blossom 10 Mile Run' official website would not be a violation of the original waiver but our analysis, using this data for purposes other than the original intent, probably would be considered an 'over extension' of the rights granted by the waiver.

Anonymizing of datasets, removing all identifiable attributes of the datasets, is often employed to extend the usefulness of datasets and reduce the chances of sensitive data being exposed to unwanted groups. With increasingly stringent legal requirements prohibiting use of data for purposes other than what was strictly granted, most of the normal avenues to increase the usefulness of the data beyond it intended original use, i.e. anonymization of data, are no longer adequate protection from possible legal actions.

In our case, using the summary statistics from the 'Cherry Blossom' race, even though it is probably stretching the intent of the original waiver, would usually be acceptable in many legal jurisdictions. Where we would run into legal issues in most jurisdictions is when we switched from summary statistics to identifying and tracking individual runners' performances. This removed the veil of anonymity and specifically targeted individuals by name. From the results we can uniquely identify an individual's name, age, hometown, and their corresponding performance(s).

Extreme caution should be used when scraping data from open websites/databases, where the use/reuse(s) of the data may not be covered within the original framework of the individual and/or entities waivers.

Good practices when scraping data from web

Acquisition of data via Web scraping is a trial and error process as Websites are constantly changing and efforts to programmatically scrape the data today might not work properly tomorrow because of changes/modifications that may occur on the Website. To mitigate this issue, any data scrapped should be stored locally, to ensure access to it in the future.

Parsing data from web scraped data can be a challenge. Data is often received in an unstructured format and/or poorly structured format. A good example of this is the variation in formatting among the results tables of the Cherry Blossom Run. Doing this scraping/parsing programmatically requires additional efforts to customize the code to account for these special cases. Whether the extra effort is worth it is a judgement call. For the sake of repeatability, parsing programmatically is encouraged.

Population Bias

Selection Bias – Self Selection

The population of the male runners studied is biased in that there is an inherent selection bias. These individuals chose to participate in this race, i.e. selective participation. They were not a random sampling of male runners from within the overall population of runners. Broader inference of results beyond the population of the runners participating in this race should not be inferred.

Selection Bias – Race Bias

The mix of professional, athlete, and recreational runners are probably eschewed. Since the Cherry Blossom 10 Mile Run is unofficially considered a primer race for the first major professional marathon of the year, Boston Marathon, one would expect a larger percentage of professional runners would compete in this race than would be expected for other similar size/stature races.

The time of year also may influence the statistical analysis. The race is held in early April every year. This being one of the earlier races in Spring season. Being that winter is usually not an optimal season to train for a run such as this, it would be prudent to take into consideration the fitness levels of the participants. We would anticipate the effect of 'out-of-shape' factor, due to winter conditions be more pronounced on athletes and recreational runners than on professional runners. Professional runners tend to train year-around, often time in other countries where weather is more conducive to training during the winter months. This might widen the performance gaps between the runners, eschewing the results of any analysis.

Selection Bias – Open vs Qualifier Race Bias

The Cherry Blossom 10 Mile Run is considered an 'open race', meaning that anyone who registers for and is selected in the Lottery is eligible to compete in the race. There is no performance qualifications required. Many of the other famous races/marathons i.e. Boston Marathon, NYC Marathon, Chicago Marathon, have qualification standards that must be met before a participant can be considered to participate in the race. This selection criteria would tend to exclude recreational runners much more than athletes and professional runners. This bias would present itself if the results of this analysis are compared to other races with different race participation requirements.

Future Work

Men vs Women's Statistics

The first future work we may intend to investigate is the comparison of the statistical results of the men's results versus the women's results. We would attempt to identify similarities and differences that might present themselves in the analysis. Since we have already obtained the women's data from the Cherry Blossom 10 Mile race for 1999-2012, this will be a straightforward exercise.

Comparison of results with other races- General

Future work may entail obtaining male runner's results from other races, over the same time period (1999-2012), from across the country, to see if the statistical analysis is similar to the results obtained from the Cherry Blossom 10 Mile Run.

Investigate if there is a Time of Year Effect

Being that the Cherry Blossom 10 Mile Run is one of the first races of the season, investigate weather similar races which occur later in the year show any different statistical patterns. The premise being, athletes and recreational runners may not be in optimal shape so early in the season.

Investigate if there is a Length of Race Effect

Do the results of the analysis from the Cherry Blossom Race differ for races that are longer? The Cherry Blossom race is a 10-mile race, which is not a common distance in competition. Most races are 10K, half-marathon, or full marathons. We would like to investigate how the performance statistics vary when considering the distance of the race.

Investigate shift in Demographics of Runners

During our investigation of the demographics of male runners between 1999 thru 2012, a shift downward in the median runner's age. A substantial shift downwards was noticed in 2008-2010 timeframe. It would be worth investigating if this dip in age may be correlated with external economic conditions. The US housing/banking crisis was in full swing during 2008-2010 timeframe. Could this be correlated with less, novice runners participating in the race due to economic hardships?

Conclusion

The study is an extension of Nolan and Lang *Modeling Runners' Times in the Cherry Blossom Race (MRCBR)* case study, which a) obtained data from website(s) by scraping techniques, b) clean, parse and organize the data and c) perform statistical analysis on the dataset. The Credit

Male Age distribution shifts from 1999 - 2012

Jeffery Lancon, Manisha Pednekar, Andrew Walch, David Stroud
MSDS 7333 - Quantifying the World - Case Study # 4
02/05/2019

Union Cherry Blossom Ten Mile Run and 5K Run-Walk race results for the male runners in the 10-mile races for the years 1999 thru 2012 were used.
This study expanded NTL's original analysis by comparing the age distribution of male runners across all 14 years of the race. We discovered that there was indeed a steady downward shift in median age of the male runners from 40 in 1999 to 35 in 2010, with a slight upward trend from 2010 to 2012 to 36 years old. The reasons behind the shift was not evident in the dataset itself. We proposed several additional study topics that might shed additional light on the topic.
Acquisition of and parsing of data via Web scraping is a trial and error process as websites are constantly updating and efforts to programmatically scrape the data can be difficult and tedious. Whether the extra effort required to obtain this data is worth it is a judgement call. The legal issues associated with scrapping data were also discussed. With an increase focus by governments and individuals, regarding the use and/or reuse of data for purposed other than what it was originally intended, web-scrapping data may lead to legal issues for organizations. An abundance of care must be employed when scrapping and/or using data from sources that were not intended for that purpose.

References

[1] Nolan, Deborah and Lang, Duncan Temple; from: "Data Science in R: A Case Studies Approach to Computational Reasoning and Problem Solving" Chapman & Hall/CRC ©2015

[2] "Modeling Runners' Times in the Cherry Blossom Race"; from: http://rdatasciencecases.org/CherryBlossom/code.R

[3] "Credit Union Cherry Blossom Ten Mile Run and 5K Run-Walk"; from: http://www.cherryblossom.org/

[4] "Credit Union Cherry Blossom Ten Mile Run and 5K Run-Walk; Race Results 2013-1999": http://www.cherryblossom.org/aboutus/results_list.php

[5] "Boston Marathon"; from: https://en.wikipedia.org/wiki/Boston_Marathon

[6]. https://www.itl.nist.gov/div898/handbook/eda/section3/eda33o.htm

Appendix A

R Code

```
#  MSDS 7333 - Quantifying the World - Case Study #4
#  Web Scraping - Cherry Blossom Race-Modeling Runners
#  Team Members:
#         Jeffery Lancon, Manisha Pednekar, Andrew Walch, David Stroud
#  Date: 02/05/2019
#  Case Study from: Data Science in R: Nolan,Temple,Lang (Ch 2)
#  Initial source code: http://rdatasciencecases.org/code.html

library(XML)
library(rstudioapi)
library(ggplot2)
current_path <- getActiveDocumentContext()$path
setwd(dirname(current_path ))

# ---------------------- Web Scraping Section -------------------
#
#  Searches cheeryblossom.org website and extracts runner's
#  information from the HTML pages for each year 1999 - 20120
#  by gender.  Saves files either to .rda or .txt files for
#  further cleaning/parsing.
#
# ------------------- Section ends on line 270 -------------------

# creating variable for base website url
# to be appended to for each subsequent year
ubase = "http://www.cherryblossom.org/"

# Begin with 2012 dataset
url = paste(ubase, "results/2012/2012cucb10m-m.htm", sep = "")
doc = htmlParse(url)

# create a list of all nodes the contain 'pre'
# in our case, there is only 1
preNode = getNodeSet(doc, "//pre")
preNode

# Extract the txt contents of preNode List
txt = xmlValue(preNode[[1]])

nchar(txt) #690,904 Characters in txt file/dataset

substr(txt, 1, 50) # Displaying first 50 characters of txt
substr(txt, nchar(txt) - 50, nchar(txt)) # Displaying the last 50 characters of txt

# Splitting the txt into individual lines; using '\\r\\n' to signify the end of each
# line
els = strsplit(txt, "\\r\\n")[[1]]

length(els) # 7201 lines

els[1:3] # Displaying first 3 lines of els

els[ length(els) ] # Displaying last line of els


#------------- extractResTable () Version A --------
#----------Extract Race Table Function -------------
# Repeats above steps and puts it into convenient function

extractResTable =
 # Retrieve data from web site, find preformatted text,
 # return as a character vector.
 function(url)
 {
  doc = htmlParse(url)
  preNode = getNodeSet(doc, "//pre")
  txt = xmlValue(preNode[[1]])
  els = strsplit(txt, "\r\n")[[1]]

  return(els)
 }

m2012 = extractResTable(url) #extract

identical(m2012, els) # Results of function are identical to previous stepwise code

# creating variable for base website url
# to be appended to for each subsequent year
ubase = "http://www.cherryblossom.org/"

# # -------------TEST Code ---------------
# # Create list of urls for each year of the race from 1999 - 2012
# urls = paste(ubase, "results/", 1999:2012, "/",
#             1999:2012, "cucb10m-m.htm", sep = "")
#
# # Extract the tables for 1999:2012 for Mens race results
# menTables = lapply(urls, extractResTable)
# # FUnction will not work:  URLs for each year do not follow the same pattern
# # as 2011 & 2012... Will have to manually capture Urls and append to ubase for
# # each year
# options(error = recover)
# menTables = lapply(urls, extractResTable)


# Manually created list of urls(minus ubase) for Mens Results for 1999:2012
# Note: Fixed errors from textbook code for 1999, 2000
menURLs =
  c("results/1999/cb99m.html", "results/2000/Cb003m.htm", "results/2001/oof_m.html",
    "results/2002/oofm.htm", "results/2003/CB03-M.HTM",
    "results/2004/men.htm", "results/2005/CB05-M.htm",
    "results/2006/men.htm", "results/2007/men.htm",
    "results/2008/men.htm", "results/2009/09cucb-M.htm",
    "results/2010/2010cucb10m-m.htm",
    "results/2011/2011cucb10m-m.htm",
    "results/2012/2012cucb10m-m.htm")

# Recreate urls for Men's race results
urls = paste(ubase, menURLs, sep = "")

urls[1:3] # Displaying first 3

# Extract Men's results from website for years 1999:2012
menTables = lapply(urls, extractResTable)

names(menTables) = 1999:2012 # Naming each lists according to its year
#typeof(menTables)

sapply(menTables, length) # List of characters for each year. Appears to be errors
                          # for years 1999, 2000, and 2009

# 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012
# 1    1 3627 3727 3951 4164 4335 5245 5283 5913    1 6919 7019 7201

# ------ extractResTable() Version B-------
# Modified for yr 2000 anomaly
extractResTable =
  # Retrieve data from web site,
  # find the preformatted text,
  # and return as a character vector.
  function(url, year = 1999)
  {
    doc = htmlParse(url)
    if (year == 2000) {
      # Get text from 4th font element
      # File is ill-formed so <pre> search doesn't work.
      ff = getNodeSet(doc, "//font")
      txt = xmlValue(ff[[4]])
    }
    else {
      preNode = getNodeSet(doc, "//pre")
      txt = xmlValue(preNode[[1]])
    }

    els = strsplit(txt, "\r\n")[[1]]
    return(els)
  }

years = 1999:2012
menTables = mapply(extractResTable, url = urls, year = years)
names(menTables) = years
sapply(menTables, length)
# Still have issues with 1999 and 2009.  Need to make additional adjustments
# to function to accommodate the different formats


# ------ extractResTable() Version C-[Final]------
# Modified for yr 1999 & 2009 anomalies
extractResTable =
  #
  # Retrieve data from web site,
  # find the preformatted text,
  # and write lines or return as a character vector.
  #
  function(url = "http://www.cherryblossom.org/results/2009/09cucb-F.htm",
           year = 1999, sex = "male", file = NULL)
  {
    doc = htmlParse(url,encoding = 'utf-8') # encoding needed for Windows users for yr 2009

    if (year == 1999) {
      # Get preformatted text from <pre> elements
      pres = getNodeSet(doc, "//pre")
      txt = xmlValue(pres[[1]])
      els = strsplit(txt, "\n")[[1]]  #Line ends with \n
    }
    else if (year == 2000) {
      # Get preformatted text from 4th font element
      # The top file is ill formed so the <pre> search doesn't work.
      ff = getNodeSet(doc, "//font")
      txt = xmlValue(ff[[4]])
      els = strsplit(txt, "\r\n")[[1]]
    }
    else if (year == 2009 & sex == "male") {
      # Get preformatted text from <div class="Section1"> element
      # Each line of results is in a <pre> element
      div1 = getNodeSet(doc, "//div[@class='Section1']")
      pres = getNodeSet(div1[[1]], "//pre")
      els = sapply(pres, xmlValue)
    }
    else {
      # Get preformatted text from <pre> elements
      pres = getNodeSet(doc, "//pre")
      txt = xmlValue(pres[[1]])
      els = strsplit(txt, "\r\n")[[1]]
    }

    if (is.null(file)) return(els)

    path <- paste0('./',file) # path base is current_path
    # Checking to see if data file path exists, if not, it creates it
    if (dir.exists(path) == FALSE){ dir.create(path,recursive = TRUE)}
    # Write the lines as a text file, in proper subdirectory (MenTxt or WomenTxt)
    writeLines(els, con = paste0(path,'/',year,'.txt'))
  }
#------ End of Function extractResTable() Version C --------


years = 1999:2012 # Added years argument to function call to identify year being parsed

# Switch to Matrix mapply because we are using multiple functions to parse data from web.
# also added 'file' argument to do 2 things, identify if we want to store results in .txt format
# and what subdirectory should the data be placed in. file=NULL means no file is created.
# NOTE: Directory must exists prior to executing code. ####
menTables = mapply(extractResTable, url = urls, year = years, file = 'MenTxt')

#menTables = mapply(extractResTable, url = urls, year = years) #for use when creating .rda files

#-----------------------------------------------------------------
#--------- Code only works if file= NULL (no text files created)
    # Adding names to tables
    names(menTables) = years

    # Sanity Check; Prints out length of each matrix
    sapply(menTables, length)
    # #------------- Test Code ---- For troubleshooting
    # doctmp = htmlParse("http://www.cherryblossom.org/results/1999/cb99m.html")
    # ffx = getNodeSet(doctmp, "//PRE")
    # head(menTables$`2009`,15)
    # tail(menTables$`2009`,10)

    save(menTables, file = "CBMenTextTables.rda") #saves tables as r-data file format
#-------------------------------




#-------------------------- Gathering Data for Women Runners -------
#------------------------------------------------------------------

womenURLs =
  c("results/1999/cb99f.html", "results/2000/Cb003f.htm", "results/2001/oof_f.html",
    "results/2002/ooff.htm", "results/2003/CB03-F.HTM",
    "results/2004/women.htm", "results/2005/CB05-F.htm",
    "results/2006/women.htm", "results/2007/women.htm",
    "results/2008/women.htm", "results/2009/09cucb-F.htm",
    "results/2010/2010cucb10m-f.htm",
    "results/2011/2011cucb10m-f.htm",
    "results/2012/2012cucb10m-f.htm") #Manually collected

# Recreate urls for Women's race results
wurls = paste(ubase, womenURLs, sep = "")

womenTables = mapply(extractResTable, url = wurls,
           year = years, sex = rep("female", 14),file = 'WomenTxt')

#--------------------------------------------------------------
#--------- Code only works if file= NULL (no text files created)

    names(womenTables) = years # Adding names to tables

    sapply(womenTables, length) # Sanity Check; Prints out length of each matrix

    # head(womenTables$`2009`,15)
    # tail(womenTables$`2009`,10)

    save(womenTables, file = "CBWomenTextTables.rda") #saves tables as r-data file format

# ----------------------------------------------------------------
# ----------------------End of Web Scrapping Section ----------------
# -----------------------------------------------------------------
# -------------------------------------------------------------------------
# -------------------- Cleaning Web Scraped Data ------------------------
# -------------------------------------------------------------------------

    #------------------ Sample Code using 2012 as an example -------------
    # Read Table does not work properly for raw/parsed data, due to formatting issues
    m2012 = read.table(file="MenTxt/2012.txt", skip = 8)

    # Read lines is a better option for reading the Runner's txt files
    els = readLines("MenTxt/2012.txt")

    els[1:10] # Sanity check to see if data was properly imported

    els2011 = readLines("MenTxt/2011.txt") # Repeat commands with 2011 data
    els2011[1:10]

    # Location of data varies depending on the year the data was obtained
    # One thing in common is that data begins after the line with equal signs '==='
    # Will use grep function to search through character strings for the index where
    # the header is located
    eqIndex = grep("^===", els)
    eqIndex # Line 8 (2012 data)

      # Alternative to using grep. Use substr.  Looks for line that has '===' within
      # the first 3 characters
      first3 = substr(els, 1, 3)
      which(first3 == "===")

    spacerRow = els[eqIndex] # Setting spacerRow variable equal to index of row containing '==='
    headerRow = els[eqIndex - 1] # Setting headerRow variable equal to index of row containing '===' - 1
    body = els[ -(1:eqIndex) ] # body variable is everything after the spacerRow 7193 Elements

    headerRow = tolower(headerRow) # converting header row to lower case, to avoid issues between
                                   # years because format varies year to year (lower case & upper case)

    ageStart = regexpr("ag", headerRow) #Attempt to identify 'Age' category in header using regex
    ageStart #Starts at index 49 and is 2 characters long

    # extract age variable in character strings in body variable, using AgeStart to identify location
    age = substr(body, start = ageStart, stop = ageStart + 1)
    head(age)
    summary(as.numeric(age))
    # Summary results
    # Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
    #    9.00   29.00   35.00   37.75   45.00   89.00      1

    blankLocs = gregexpr(" ", spacerRow) # locating blanks in spacerRow, signifying separation of different variables
    blankLocs #6 18 25 48 51 72 80 88 94

    searchLocs = c(0, blankLocs[[1]]) # vector of location of blanks in spacerRow

    # Separation of variable in body variable, using searchLocs to distinguish location & length
    Values = mapply(substr, list(body),
                start = searchLocs[ -length(searchLocs)] + 1, #adds 1 to every value of searchLocs: 1  7 19 26 49 52 73
81 89
                stop = searchLocs[ -1 ] - 1)# Subtracts 1 from every value of searchLocs except 1st one: 5 17 24 47 50
71 79 87 93



#---------- findColLocs ()  Find column locations --------
# COnverts all the previous exploratory work to identity column location
# and put it into a single function

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))
}
#-----

#---------- selectCols ()  Rev A Selection of each variable --------
# Converts all the previous exploratory work to select variables and populating
# values. Puts it into a single function
selectCols =
  function(colNames, headerRow, searchLocs)
  {
    sapply(colNames,
           function(name, headerRow, searchLocs)
           {
             startPos = regexpr(name, headerRow)[[1]] # Returns a -1 if regexpr does not match the
                                          # name with a value in the headerRow (matches variable
                                          # to header row value, if not there, does not
                                          # populate variable)
             if (startPos == -1)
```

```
          return( c(NA, NA) )

         index = sum(startPos >= searchLocs)
         c(searchLocs[index] + 1, searchLocs[index + 1] - 1)
        },
        headerRow = headerRow, searchLocs = searchLocs )
  }
#-----

   #------------------ Sample Code using 2012 as an example -------------
   # To confirm functions work as anticipated
   searchLocs = findColLocs(spacerRow)  #0  6 18 25 48 51 72 80 88 94
   ageLoc = selectCols("ag", headerRow, searchLocs)
   ages = mapply(substr, list(body),
              start = ageLoc[1,], stop = ageLoc[2, ])

   summary(as.numeric(ages)) #   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
                         #   9.00   29.00   35.00   37.75   45.00   89.00       1
   #-------------------------------------------------------------------

   #------------------ Sample Code using 2012 as an example -------------
   shortColNames = c("name", "home", "ag", "gun", "net", "time") #short name must be enough to uniquely
identify
                                                   # the column without over specifying it

   locCols = selectCols(shortColNames, headerRow, searchLocs) #Creates location index for all shortColNames
                                                # variables that exist in 2012 dataset
                                                #       name home ag gun net time
                                                # [1,]   26   52 49  NA  NA   81  (Start)
                                                # [2,]   47   71 50  NA  NA   87  (Stop)

   # Mapping variable values to matrix, using parsing functions
   Values = mapply(substr, list(body), start = locCols[1, ],
               stop = locCols[2, ])
   class(Values) #Matrix

   colnames(Values) = shortColNames #Adding column name to matrix variables (7193 observations)
    head(Values)
    tail(Values)[ , 1:3]
    #-------------------------------------------------------------------


#---------- extractVariables ()  REv A Extract variables for each observation --------
# Parses the web scrapped data into a list of matrices, using grep to id location
# and variable name: 'name' 'home' 'age' 'gun' 'net', and 'time'.
# If field does not exists, populates with NA. Puts it into a single function
```

```
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
   # returns values variable
   invisible(Values)
 }
#--------------

#----------- Parsing of txt files for results from Cherry Blossom Race
#----------- For years 1999 - 2012:  Using
mfilenames = paste("MenTxt/", 1999:2012, ".txt", sep = "") #Creating Path
menFiles = lapply(mfilenames, readLines) # Reading lines of the 14 .txt files as character vector and
                              # creating a list of Char vectors(one for each row in txt file),
                              #  storiing them in menFiles
names(menFiles) = 1999:2012 # Naming list
sapply(menFiles, length)

# Creating Matrix of men's results from web-scrapped data, using extractVariables function
menResMat = lapply(menFiles, extractVariables)
class(menResMat) # A list
sapply(menResMat,class) # Each entry in menResMat is a matrix
length(menResMat) # List of 14 character vector Matrices 1999:2012
      menResMat$`1999`[2,'home']

# Determine the number of observations per character vector Matrices
sapply(menResMat, nrow)
# 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012
# 3190 3017 3622 3724 3948 4156 4327 5237 5276 5905 6651 6911 7011 7193

      ### The 2001 results for women are missing the === and the column names.
      ### Can we pick it up from the 2001 men? YES! Make an exercise
```

```
    #wfilenames = paste("WomenTxt/", 1999:2012, ".txt", sep = "")
    #womenTables = lapply(wfilenames, readLines)

    #womenTables[[3]][1:5]

    #names(womenTables) = 1999:2012
    #womenResMat = lapply(womenTables, extractVariables)
    #head(womenResMat[[3]], 10)
    #tail(womenResMat[[3]], 10)

# ---------------------------------------
# -------------- Data Cleaning 2.3 -------

   #------------------ Sample Code using 2012 as an example;
   # --- to confirm results of function----------
   age = as.numeric(menResMat[['2012']][ , 'ag'])

   tail(age) # 41 39 56 35 NA 48
   #-----

# Extract men's Age data from yrs 2009:2012.
# using as.numeric since all data is character data (Since only 1 type of variable is allowed
# in a matrix, we have to extract it from the matrix before converting)
age = sapply(menResMat,
        function(x) as.numeric(x[ , 'ag']))
# NOTE: Waring messages means that some values could not be converted to numeric, resulting in NA values
#      Warning messages:
#       1: In FUN(X[[i]], ...) : NAs introduced by coercion (x3)


   #------- Figure 2.4 - Boxplot Ages by yr ----
   # Problems identified through the use of boxplots 2003,2006
pdf("./Figures/FIg_2.4_CB_BoxplotAgeByYr.pdf", width = 8, height = 5)
oldPar = par(mar = c(4.1, 4.1, 1, 1))

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

par(oldPar)
dev.off()
#-----

    #------------------ Sample Code using 2003, 2006 as an example;
    # --- to confirm results of function----------
    head(menFiles[['2003']])
    menFiles[['2006']][2200:2205]
```

```
# Notice Age header in 2003 is shifted by 1 character, so we are only grabbing the 1st digit
# In 2006, some ages are shifted in the column
# solve both problems by modifying the index for the end of each variable by +1 (include space)

#---------- selectCols () Rev B  Selection of each variable --------
# Modified previous function to includ spacer modification
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]) #removed -1
  }, headerRow = headerRow, searchLocs = searchLocs )
}
#-------

# Rerun menResMat using updated selectCols function
menResMat = lapply(menFiles, extractVariables)
#womenResMat = lapply(womenFiles, extractVariables)

#ReRun Age extraction
age = sapply(menResMat,
             function(x) as.numeric(x[ , 'ag']))
# NOTE: Waring messages means that some values could not be converted to numeric, resulting in NA values
#     Warning messages:
#       1: In FUN(X[[i]], ...) : NAs introduced by coercion (x4)


#------- Figure 2.5 - Boxplot Ages by yr ----
# Recreating boxplot for male ages by year
pdf("./Figures/Fig_2.5_CB_BoxplotAgeByYrRevised.pdf", width = 8, height = 5)
oldPar = par(mar = c(4.1, 4.1, 1, 1))
boxplot(age, ylab = "Age", xlab = "Year")
par(oldPar)
dev.off()
#----

# Determining number of NA in age variable
sapply(age,  function(x) sum(is.na(x)))
# 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012
# 1    1   61    3    2    0   13    2    5    1    2    6    0    1

# Further investigating year 2001 (61 NA)
age2001 = age[["2001"]]

# Determine the line === appears in original .txt file for 2001
```

```
grep("^===", menFiles[['2001']]) # 5

#Determine the location of NA values (post extraction of data) after removal of top 5 rows
badAgeIndex = which(is.na(age2001)) + 5

menFiles[['2001']][ badAgeIndex ] #Displaying all lines where age NA
# All lines are blank except for last entry which is an annotation line
badAgeIndex
#[1] 1756 1757 1758 1759 1760 1761 1762 1763 1814 1815 1816 1817 1818 1819 1820 1821 1872 1873 1874 1875
1876
#[22] 1877 1878 1879 1930 1931 1932 1933 1934 1935 1936 1937 2538 2539 2540 2541 2542 2543 2544 2545
2546 2897
#[43] 2898 2899 2900 2901 2902 2903 2904 2955 2956 2957 3008 3009 3010 3011 3012 3013 3014 3015 3627


#---------- extractVariables ()  Rev B -[FINAL] Extract variables for each observation --------
# Modified to eliminate blank lines and remove footnotes
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) ]
    # Remove footnotes and blank rows
    footnotes = grep("^[[:blank:]]*(\\*|\\#)", body)
    if ( length(footnotes) > 0 ) body = body[ -footnotes ]
    blanks = grep("^[[:blank:]]*$", body)
    if (length(blanks) > 0 ) body = body[ -blanks ]


    # 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

    return(Values)
  }
#----
```

```
# Rerun menResMat using updated extractVariables function
menResMat = lapply(menFiles, extractVariables)
#womenResMat = lapply(womenFiles, extractVariables)

which(age2001 < 5) # look at 2001 where age < 5 (3 lines 1377 3063 3112)

menFiles[['2001']][ which(age2001 < 5) + 5 ] # look at these lines manually [Ages listed as 0]



#------ Working with Time variable ------------------
#-----------------------------------------------------

   #------------------ Sample Code using 2012 as an example -------------
   # Extraction and manipulaton of time character variable.
   charTime = menResMat[['2012']][, 'time']
   class(charTime) # Character class
   head(charTime, 5) # "  45:15 " "  46:28 " "  47:33 " "  47:34 " "  47:40 "

   tail(charTime, 5) # "2:27:11 " "2:27:20 " "2:27:30 " "2:28:58 " "2:30:59 "

   # Splitting hh:mm:ss using ':" as the split parameter
   timePieces = strsplit(charTime, ":")
   typeof(timePieces) #list of character vectors
   timePieces[[1]] # "  45" "15 "

   tail(timePieces, 1) #  "2"   "30"  "59 "

   # Converting timePieces from character to numeric
   timePieces = sapply(timePieces, as.numeric)
   timePieces[[1]] #  45 15

   # Combine timePieces character vectors into a numeric variable
   # in minutes 60*hrs + min + sec/60
   runTime = sapply(timePieces,
            function(x) {
             if (length(x) == 2) x[1] + x[2]/60
             else 60*x[1] + x[2] + x[3]/60
            })

   summary(runTime)
   # Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   # 45.25   77.57   87.47   88.43   97.78  150.98

#---------- convertTime ()  Rev A Extracts time variable for each observation --------
```

```
# --------- splits time, using ':' then converts timePieces to numeric and
#---------- coverts and records results in minutes
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
  })
}

#---------- createDF ()  Rev A Crates DataFrame from menResMat --------
# Creating a dataframe with the data from menResMat
# Dataframe will include new variables to track 'sex' and 'year',
# which are currently not part of the data within the matrices
createDF =
  function(Res, year, sex)
  {
    # Determine which time to use 'net','gun','time'
    useTime = if( !is.na(Res[1, 'net']) )  # Net time gets top priority if it exists
      Res[ , 'net']
    else if( !is.na(Res[1, 'gun']) ) # gun time gets second priority
      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)
  }

# Creating list of dataframe with: year, sex,name,home,age,runtime, using createDF function
menDF = mapply(createDF, menResMat, year = 1999:2012,
               sex = rep("M", 14), SIMPLIFY = FALSE)
# There were 50 or more warnings (use warnings() to see the first 50)
typeof(menDF) #list with 14 elements
sapply(menDF,class) # 14 DataFrames

warnings()[ c(1:2, 49:50) ]
```

```
# 1: In lapply(X = X, FUN = FUN, ...) : NAs introduced by coercion
# 2: In lapply(X = X, FUN = FUN, ...) : NAs introduced by coercion
# 3: In lapply(X = X, FUN = FUN, ...) : NAs introduced by coercion
# 4: In lapply(X = X, FUN = FUN, ...) : NAs introduced by coercion

# Checking to see if NAs are coming from time conversion runTime.
# Errors coming from yr 2006, 2007,2009,2010
sapply(menDF, function(x) sum(is.na(x$runTime)))
# 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012
# 0    0    0    0    0    0    0 5232   83    0   72   68    1    0


#---------- createDF ()  Rev B - [FINAL] Crates DataFrame from menResMat --------
# -- modifies original function to strip tramp characters from time '#','*',' [[blanks]]'
# -- Drops observations with no time
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)
}

# Rerun updated createDF Function
menDF = mapply(createDF, menResMat, year = 1999:2012,
               sex = rep("M", 14), SIMPLIFY = FALSE)

sapply(menDF, function(x) sum(is.na(x$runTime)))
# 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012
# 0    0    0    0    0    0    0 5232    0    0    0    0    0    0
```

```
#Displaying 2006 runTime data.  (home variable include home and time;  time variable includes NA)
menDF$`2006`[1:3,c('home','runTime')]
# home runTime
# 1 Kenya         47:24     NA
# 2 Kenya         47:34     NA
# 3 Kenya         47:38     NA

#Looking back at original file to determine issue
menFiles[['2006']][6:10]
# [6] ""
# [7] "Place Div/Tot  Num    Name                 Ag Hometown        Net Tim Gun Tim  Pace  S "
# [8] "===== ======== ====== ===================== == ====================== =======  ===== = "
# [9] "    1   1/2892      1 Gilbert Okari         27 Kenya           47:24   47:25#  4:45   "
# [10] "    2   2/2892     11 Samuel Ndereba        29 Kenya           47:34   47:35#  4:46   "

# Notice that Net Time header is joined with Hometown, so it does not recognize that this is 2 variables
# and not 1 variable.  We will need to split 'home' variable up into 2 variables

# Manually adjusting time data for year 2006
separatorIdx = grep("^===", menFiles[["2006"]]) # Line 8
separatorRow = menFiles[['2006']][separatorIdx] # Extracting Separator Row
      separatorRow
      nchar(separatorRow) #89
      substr(separatorRow,40,80)

separatorRowX = paste(substring(separatorRow, 1, 63), " ",
             substring(separatorRow, 65, nchar(separatorRow)), # Manually imparting a ' ' character
             sep = "")                                   # in separatorRow @ chr 64, so
                                                        # extractVariable function will
                                                        # work properly
menFiles[['2006']][separatorIdx] = separatorRowX # overwriting original speratorRow with modified one

# Rerun extractVariables function, utilizing modified separator for Yr 2006
menResMat = sapply(menFiles, extractVariables)
menDF = mapply(createDF, menResMat, year = 1999:2012,
         sex = rep("M", 14), SIMPLIFY = FALSE)

#separatorIdx = grep("^===", womenFiles[["2006"]])
#separatorRow = womenFiles[['2006']][separatorIdx]
#separatorRowX = paste(substring(separatorRow, 1, 63), " ",
#              substring(separatorRow, 65, nchar(separatorRow)),
#              sep = "")
#womenFiles[['2006']][separatorIdx] = separatorRowX

#womenResMat = sapply(womenFiles, extractVariables)
#womenDF = mapply(createDF, womenResMat, year = 1999:2012,
#           sex = rep("W", 14), SIMPLIFY = FALSE)



#------- Figure 2.5b - Boxplot time by yr (Men)----
# Preliminary view of data, to discover any problems with data
# Time data looks fine, no obvious issues
pdf("./Figures/FIg_2.5B_CB_BoxplotTimeByYr.pdf", width = 8, height = 5)
boxplot(sapply(menDF, function(x) x$runTime),
     xlab = "Year", ylab = "Run Time (min)")
dev.off()
#----

# Since we have added the yr and sex variables, we can convert
# the yearly dataframes into 1 single dataframe and save them to .rda
# and csv files
cbMen = do.call(rbind, menDF) # 70070 observations x 6 variables

path <- paste0('./MenTxt') # path base is current_path
# Checking to see if data file path exists, if not, it creates it
if (dir.exists(path) == FALSE){ dir.create(path,recursive = TRUE)}

# Write dataframe as a csv file, in proper subdirectory (MenTxt or WomenTxt)
write.csv(cbMen, file = paste0(path,'/cbMen.csv'))
# Write dataframe as R data file
save(cbMen, file = "./MenTxt/cbMen.rda")

dim(cbMen) # 70070 observations x 6 variables

#####################################################################################################
########
#####################################################################################################
########
################          2.4 Data Exploration
################



# Loading Men's data file from hard-drive.
load("./MenTxt/cbMen.rda")

#------- Figure 2.6 - ScatterPlot Age vs Run Time (Men)----
# Preliminary view of men's data, to discover any problems with data
# Ylimit were introduced, to screen out erroneous Run times
pdf("./Figures/Fig_2.6_CB_Overplot.pdf", width = 8, height = 6)
oldPar = par(mar = c(4.1, 4.1, 1, 1))
```

```
plot(runTime ~ age, data = cbMen, ylim = c(40, 180),
    xlab = "Age (years)", ylab = "Run Time (minutes)")

par(oldPar)
dev.off()
#----

# To improve the appearance of graphics(colors), we will load the RColorBrewer package
library(RColorBrewer)
     ls("package:RColorBrewer")
     display.brewer.all()
     dev.off()

Purples8 = brewer.pal(9, "Purples")[8] # Selecting the 8 color in the Purple pallett
Purples8 # RGB hex code for the color: #54278F   54 Red, 27 Blue, 8F Green
Purples8A = paste(Purples8, "14", sep = "") # appending the Alpha value to the RGB value, for transparency


#------- Figure 2.7 - ScatterPlot Age vs Run Time (Men)----
# Improved graphics for view of men's data, using colors, transparency,
# reduced symbol size and shape, and add jitter to age
pdf("./Figures/Fig_2.7_CB_OverplotTransparent.pdf", width = 8, height = 6)
oldPar = par(mar = c(4.1, 4.1, 1, 1))
plot(runTime ~ jitter(age, amount = 0.5),
    data = cbMen,
    pch = 19,cex = 0.25, col = Purples8A,
    ylim = c(45, 165), xlim = c(15, 85),
    xlab = "Age (years)", ylab = "Run Time (minutes)")
par(oldPar)
dev.off()
#-----

#?smoothScatter()
#?colorRampPalette()
#------- Figure 2.8 - ScatterPlot Age vs Run Time (Men)----
# Improved graphics for view of men's data, using smoothScatter(),
# color is determined by density of data points within a small region
# around that point. 2D kernel density
pdf("./Figures/FIg_2.8_CB_SmoothScatter.pdf", width = 8, height = 6)
oldPar = par(mar = c(4.1, 4.1, 1, 1))

smoothScatter(y = cbMen$runTime, x = cbMen$age,
            ylim = c(40, 165), xlim = c(15, 85),
            colramp = colorRampPalette(c("ivory", Purples8A)),
            xlab = "Age (years)", ylab = "Run Time (minutes)")
```

```
par(oldPar)
dev.off()
#-----

# Extracting Men's run times (>30min) and runner's age (>15)
cbMenSub = cbMen[cbMen$runTime > 30 &
                !is.na(cbMen$age) & cbMen$age > 15, ]
max(cbMenSub$age)
# Creating Age categories for runners [15:75] in 10yr steps and max Age
ageCat = cut(cbMenSub$age, breaks = c(seq(15, 75, 10), max(cbMenSub$age)))
# (15,25] (25,35] (35,45] (45,55] (55,65] (65,75] (75,89]
# 5804   25432   20535   12212   5001     751     69

table(ageCat) # Creating a table of age categories

#------- Figure 2.9 - BoxPlot Run Time by Age Grouping (Men)----
#
pdf("./Figures/Fig_2.9_CB_Boxplots.pdf", width = 8, height = 6)
oldPar = par(mar = c(4.1, 4.1, 1, 1))

plot(cbMenSub$runTime ~ ageCat,
   xlab = "Age (years)", ylab = "Run Time (minutes)")

par(oldPar)
dev.off()
#-----

# Creation of a linear model for the runtime vs age data
lmAge = lm(runTime ~ age, data = cbMenSub)

lmAge$coefficients
#Linear Model Coefficients
# (Intercept)        age
# 78.757076    0.225285

summary(lmAge)
    # Call:
    #   lm(formula = runTime ~ age, data = cbMenSub)
    #
    # Residuals:
    #   Min      1Q  Median      3Q     Max
    # -40.333 -10.221  -0.952   9.103  82.425
    #
    # Coefficients:
    #   Estimate Std. Error t value Pr(>|t|)
    # (Intercept) 78.75708    0.20771  379.17   <2e-16 ***
    #   age         0.22529    0.00517   43.58   <2e-16 ***
    #   ---
    #   Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
    #
    # Residual standard error: 14.77 on 69802 degrees of freedom
    # Multiple R-squared:  0.02648, Adjusted R-squared:  0.02647
    # F-statistic:  1899 on 1 and 69802 DF,  p-value: < 2.2e-16

class(lmAge) # "lm"

# To access how well the lm model fits the data, wer will plot the residuals
# versus age

#------- Figure 2.10 - Linear Model Residual scatter Plot with overlays(Men)----
# Smoothed Scatter plot; Includes horizontal line overlay at y=0 and local smooth polynomial
# fitted line of residuals (for the fit at point x, the fit is made using points
# in a neighbourhood of x, weighted by their distance from x)these have tricubic
# weighting (proportional to (1 - (dist/maxdist)^3)^3)

pdf("./Figures/Fig_2.10_CB_ResidSimpleLM.pdf", width = 8, height = 6)
oldPar = par(mar = c(4.1, 4.1, 1, 1))

smoothScatter(x = cbMenSub$age, y = lmAge$residuals,
        xlab = "Age (years)", ylab = "Residuals",
        colramp = colorRampPalette(c("ivory", Purples8A)))
abline(h = 0, col = "darkred", lwd = 3) # Horizontal line y=0

# Local weighted averages of the residuals. Contains fitted values
# for the local weighted avg of residuals
resid.lo = loess(resids ~ age,
          data = data.frame(resids = residuals(lmAge),
                    age = cbMenSub$age))
#?loess()
age20to80 = 20:80

# Using the values of the locally weighted residuals (redid.lo),
# then using predict() function to predict the loess value for each
# age. Thus, creating a dataframe of predicted loess of residuals vs age
resid.lo.pr =
  predict(resid.lo, newdata = data.frame(age = age20to80)) # Predictions of value of line

# Creating a line using the resid.lo.pr values
lines(x = age20to80, y = resid.lo.pr, col = "green", lwd = 2)
par(oldPar)
dev.off()
#--------
```

```
# It appears that a simple linear model tends to underestimate the finish times for male
# runners over the age of 60


##--------------------------------------------------------
# Since linear model is not an ideal predictor for this dataset, we want to switch
# gears somewhat and attempt to find another solution the better models the dataset
# 2 options are:
# A) Piecewise Linear function
# B) Loess Curve
##-------------------------------------


# To investigate these 2 options, we will the two models vs runner's age

####### Loess Curve
# Calculate the locally weighted average for the runner's time by age
menRes.lo = loess(runTime ~ age, cbMenSub)

# Using the values of the locally weighted run times (menRes.lo),
# then using predict() function to predict the loess value for each
# age. Thus, creating a dataframe of predicted loess run times vs age
menRes.lo.pr = predict(menRes.lo, data.frame(age = age20to80))



####### Piecewise Linear Model

    #-------- Sample code for testing purposes --------
    #Creating a piecewise LM segment from 50+ yo runners
    #Determine if runner was over 50, if so, by how many years
    over50 = pmax(0, cbMenSub$age - 50) # takes maximum value of age-50 or 0, produces a vector

    # Create a linear model of runners over age of 50
    lmOver50 = lm(runTime ~ age + over50, data = cbMenSub)

    summary(lmOver50)
    # From the summary, formula (a - 50c) + (b+c)age
    #  a = 82.75
    #  b = 0.105 slope of line
    #  c = 0.563 change in slope after 50 yrs old
    # so, for every year above 50, times increase by 0.67 minutes a year
    # before 50, time decrease by 0.106 minutes for every year below 50

    # Call:
```

```
    #   lm(formula = runTime ~ age + over50, data = cbMenSub)
    #
    # Residuals:
    #   Min      1Q  Median      3Q     Max
    # -40.265 -10.099  -0.881   9.061  79.044
    #
    # Coefficients:
    #   Estimate Std. Error t value Pr(>|t|)
    # (Intercept) 82.755465   0.265049  312.23   <2e-16 ***
    #   age         0.105681   0.007147   14.79   <2e-16 ***
    #   over50      0.563889   0.023372   24.13   <2e-16 ***
    #   ---
    #   Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
    #
    # Residual standard error: 14.71 on 69801 degrees of freedom
    # Multiple R-squared:  0.03453,            Adjusted R-squared:  0.03451
    # F-statistic:  1248 on 2 and 69801 DF,  p-value: < 2.2e-16

#########
#Creating a piecewise LM segments from 30+ yo runners, in increments of 10 yrs

decades = seq(30, 60, by = 10) # Creating decades vector
# Creating 4 vectors (one for each decade) and subtracting the runner's age from value
overAge = lapply(decades,
         function(x) pmax(0, (cbMenSub$age - x)))
names(overAge) = paste("over", decades, sep = "")
overAge = as.data.frame(overAge)
tail(overAge)
#       over30  over40 over50 over60
# 69799     36     26     16      6
# 69800     11      1      0      0
# 69801      9      0      0      0
# 69802     26     16      6      0
# 69803      5      0      0      0
# 69804     18      8      0      0
#By comparison, Ages of last 6 runners in dataset
# 66
# 41
# 39
# 56
# 35
# 48

# Creating a piecewise linear model with breaks at 30,40,50,60 yr age vs runTime
lmPiecewise = lm(runTime ~ . ,
         data = cbind(cbMenSub[, c("runTime", "age")],
                    overAge))
summary(lmPiecewise)
# Residuals:
#   Min      1Q  Median      3Q     Max
# -40.921 -10.119  -0.885   9.023  78.965
#
# Coefficients:
#             Estimate    Std. Error  t value   Pr(>|t|)
# (Intercept)   74.227662  0.915265  81.100  < 2e-16 ***
#   age         0.424331   0.033208  12.778  < 2e-16 ***
#   over30     -0.477114   0.047779  -9.986  < 2e-16 ***
#   over40      0.221650   0.040667   5.450 5.04e-08 ***
#   over50      0.494407   0.052933   9.340  < 2e-16 ***
#   over60     -0.003592   0.077656  -0.046    0.963
# ---
#   Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# Residual standard error: 14.7 on 69798 degrees of freedom
# Multiple R-squared:  0.03593,       Adjusted R-squared:  0.03586
# F-statistic: 520.3 on 5 and 69798 DF,  p-value: < 2.2e-16




# When we want to plot the piecewise linear funtion, we need to
# invoke the predict() function, to provide the fitted values for
# each age value (20-80). We first need to create a dataframe with
# all the covariates (ie. dataframe for each age in range).
# so predict can use these values in its predictions
overAge20 = lapply(decades, function(x) pmax(0, (age20to80 - x)))
names(overAge20) = paste("over", decades, sep = "")
overAgeDF = cbind(age = data.frame(age = age20to80), overAge20)
head(overAgeDF,5)
# age over30 over40 over50 over60
# 1  20      0      0      0      0
# 2  21      0      0      0      0
# 3  22      0      0      0      0
# 4  23      0      0      0      0
# 5  24      0      0      0      0
tail(overAgeDF,5)
# age over30 over40 over50 over60
# 57  76     46     36     26     16
# 58  77     47     37     27     17
# 59  78     48     38     28     18
# 60  79     49     39     29     19
# 61  80     50     40     30     20
# Calling predict function, using results of previous lmPiecewise function
# and the 'standard' overAgeDF covariates DF
predPiecewise = predict(lmPiecewise, overAgeDF)

#------- Figure 2.11 - Piecewise Linear & Loess Model RunTime vs Age (Men)----
# Piecewise linear with inflection points at 30,40,50,60 yrs
pdf("./Figures/Fig_2.11_CB_PiecewiseLoessCurves.pdf", width = 8, height = 6)
oldPar = par(mar = c(4.1, 4.1, 1, 1))
plot(predPiecewise ~ age20to80,
    type = "l", col = "#984ea3", lwd = 3,
    #   type = "l", col = "purple", lwd = 2,
    xlab = "Age (years)", ylab = "Run Time Prediction")

lines(x = age20to80, y = menRes.lo.pr,
     col = "#4daf4a", lwd = 3, lty = 2)
legend("topleft", col = c("#984ea3", "#4daf4a"), lty = c(1, 2), lwd = 3,
      legend = c("Piecewise Linear", "Loess Curve"), bty = "n")
par(oldPar)
dev.off()
#------


#------- Figure 2.12 - Number of Runners by Year (Men)----
# To get a feeling of how participation varies over the years
pdf("./Figures/Fig_2.12_CB_NumRunnersLinePlot.pdf", width = 8, height = 6)
oldPar = par(mar = c(4.1, 4.1, 1, 1))

numRunners = with(cbMen, tapply(runTime, year, length))
plot(numRunners ~ names(numRunners), type="l", lwd = 2,
    xlab = "Years", ylab = "Number of Runners")
par(oldPar)
dev.off()
#------

# Comparing the performance between earliest and latest years (Men)
# Note: World record time: 44:24
summary(cbMenSub$runTime[cbMenSub$year == 1999])
# Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
# 46.98   74.82   84.29   84.35   93.06  170.83

summary(cbMenSub$runTime[cbMenSub$year == 2012])
# Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
# 45.25   77.57   87.47   88.44   97.78  150.98

# The min time has dropped but the 1stQTR Med and 3rdQTR have increased through the years.
# Could this be because the average age had increased and/or make-up of runners has changed (pro vs
recreational)


#------- Figure 2.13 - Age Density comparison 1999 vs 2012 by Year (Men)----
# To get a feeling of how age of participation varies between 1999 & 2012
pdf("./Figures/Fig_2.13_CB_AgeDensity99vs12.pdf", width = 8, height = 6)
oldPar = par(mar = c(4.1, 4.1, 1, 1))

age1999 = cbMenSub[ cbMenSub$year == 1999, "age" ]
age2012 = cbMenSub[ cbMenSub$year == 2012, "age" ]

plot(density(age1999, na.rm = TRUE),
    ylim = c(0, 0.05), col = "purple",
    lwd = 3,  xlab = "Age (years)",  main = "")
lines(density(age2012, na.rm = TRUE),
     lwd = 3, lty = 2, col="green")
legend("topleft", col = c("purple", "green"), lty= 1:2, lwd = 3,
      legend = c("1999", "2012"), bty = "n")

par(oldPar)
dev.off()
#-----


# Initializing a list (ageList) that will contain a list of dataframes by year of
# male runner's ages
ageList <- list()
years<-c(1999:2012)

# Parsing the cbMenSub data to create ageList - list of dataframes
for (p in years){
  ageList[[paste0('age',p)]] <- cbMenSub[ cbMenSub$year == p, "age" ]}

# Creating a color palette with 14 colors
library(RColorBrewer)
cp <- palette(c('black','grey', brewer.pal(n=12, name='Paired')))

#------- Figure 2.Custom_1 - Age Density comparison 1999 through 2012 by Year (Men)----
# To show how age of participation varies between 1999 & 2012
pdf("./Figures/Fig_2.Custom_1_AgeDensity_1999thru2012.pdf", width = 8, height = 6)
oldPar = par(mar = c(4.1, 4.1, 1, 1))
lc = 1
plot(density(age1999, na.rm = TRUE),
    ylim = c(0, 0.05), col = cp[lc],
    lwd = 3,  xlab = "Age (years)",  main = "")
```

```
for (j in years){
  if (j != 1999){
    lc = lc + 1 # Loop count
    lines(density(ageList[[as.character(paste0('age',j))]], na.rm = TRUE),
          lwd = 3, lty = 1, col=cp[lc])
  }
}

legend("topleft", col = cp, lty= 1, lwd = 3,
       legend = c(years), bty = "n")

par(oldPar)
dev.off()
dev.off()
#-----

# The results of the grpah show something peculiar.  The age density for 2012 shows that
# there were more younger runners than in 1999.  Which should have produced lower statistics
# for 2012 vs 1999 (Younger runners run faster...).  This must mean that the demographics of
# the runners has changed (less pro racers vs amateurs????)



library(extrafont)
library(ggplot2)
#font_import()    # May need to run these functions the first time you run it
#fonts()          # May need to run these functions the first time you run it
#loadfonts()      # May need to run these functions the first time you run it

#------- Figure 2.Custom2 - Box and Violin Plot runner's age by year (Men)----
# To show how age of participation varies between 1999 & 2012
#Box and Violin plot for runner's age distribution by year
# included jittered scatter plot of runners by age colored by Run times.
pdf("./Figures/Fig_2.Custom_2_AgeDistribution_1999thru2012.pdf", width = 10, height = 7.5)
oldPar = par(mar = c(4.1, 4.1, 1, 1))
b <- ggplot(data=cbMenSub, aes(x=year, y=age, group=year))

Final <- b + geom_jitter(aes(color=runTime),position=position_jitter(width=.4, height=.5),cex=.4) +

  scale_color_distiller(palette = 'Paired',direction = 1)+

  geom_violin(size=.8,alpha=0.5, trim = TRUE) +
  geom_boxplot(size=.4,alpha=.7, width=.25,outlier.shape=NA) +
  scale_x_discrete(limits=c(1999:2012))+
  scale_y_discrete(limits=c(seq(15,90,5)))+
  xlab("Years") +
  ylab("Age of Participant - Male") +
  ggtitle('Age Distribution 1999:2012 - Male')+
  theme(#text = element_text(family='Comic Sans MS'),
        axis.title.x=element_text(colour='Blue',size=14),
        axis.title.y=element_text(colour='Blue',size=14),
        axis.text.x = element_text(size=10),
        axis.text.y = element_text(size=10),

        legend.title = element_text(size=13),
        legend.text = element_text(size=10),

        plot.title = element_text(colour='Black',
                                  size=20,
                                  hjust = 0.5)
        # Alternate for setting font
        # text = element_text(family='Comic Sans MS')
  )
Final$labels$colour = "Run Time\n(min.)"
Final

par(oldPar)
dev.off()
#-----



# ------ QQ Plot age of Male racers yr 1999 vs 2012----- NOT Plotted to PDF
qqplot(age1999, age2012, pch = 19, cex = 0.5,
       ylim = c(10,90), xlim = c(10,90),
       xlab = "Age in 1999 Race",
       ylab = "Age in 2012 Race",
       main = "Quantile-quantile plot of male runner's age")
abline(a =0, b = 1, col="red", lwd = 2)
#-------



#------- Figure 2.Custom_3 - Q-Q Plots 1999 runner's age vs all other year (Men)----
# Grid plot to look at QQ plot over the years to see if there is a change in population
dev.off()
pdf("./Figures/Fig_2_Custom_3_Q-Q_Plots_1999_thru_2012.pdf", width = 10, height = 8)
par(mfrow = c(3, 5), mar = c(3, 3, 2, 2))
invisible(
  lapply(years[-1], function(years){qqplot(ageList[['age1999']], ageList[[as.character(paste0('age',years))]], pch = 19,
cex = 0.5,
                    ylim = c(10,90), xlim = c(10,90),
                    xlab = "Age in 1999 Race",
                    ylab = paste0("Age in ",years," Race"),
                    main = paste0("Age Q:Q Plot 1999:",years))
   abline(a =0, b = 1, col="red", lwd = 2)}
  ) )
par(oldPar)
dev.off()




#------- Figure 2.14 - Loess Model RunTime vs Age (Men) for yrs 1999 & 2012 only----
# Similar to Fig 2.11 but only looking at 2 year of data
mR.lo99 = loess(runTime ~ age, cbMenSub[ cbMenSub$year == 1999,])
mR.lo.pr99 = predict(mR.lo99, data.frame(age = age20to80))

mR.lo12 = loess(runTime ~ age, cbMenSub[ cbMenSub$year == 2012,])
mR.lo.pr12 = predict(mR.lo12, data.frame(age = age20to80))

pdf("./Figures/FIg_2.14_CB_Loess99vs12.pdf", width = 8, height = 6)
oldPar = par(mar = c(4.1, 4.1, 1, 1))

plot(mR.lo.pr99 ~ age20to80,
   type = "l", col = "#984ea3", lwd = 3,
   xlab = "Age (years)", ylab = "Prediction (minutes)")
lines(x = age20to80, y = mR.lo.pr12, col="#4daf4a", lty = 2, lwd = 3)
legend("topleft", col = c("#984ea3", "#4daf4a"), lty = 1:2, lwd = 3,
     legend = c("1999", "2012"), bty = "n")
par(oldPar)
dev.off()
#-----


# Creating a variable quantifying the difference between Loess prediction
# modes for 1999 and 2012
gap14 = mR.lo.pr12 - mR.lo.pr99


#------- Figure 2.15 - Difference between Loess Model RunTime vs Age (Men) for yrs 1999 & 2012 only----
pdf("./Figures/Fig_2.15_CB_DifferenceInFittedCurves.pdf", width = 8, height = 6)
oldPar = par(mar = c(4.1, 4.1, 1, 1))

plot(gap14 ~ age20to80, type = "l" , xlab = "Age (years)",
   ylab = "Difference in Fitted Curves(2012 - 2009) (minutes)", lwd = 2)
par(oldPar)
```

```
dev.off()
# -----

# The Fig 2.15 graph shows that the finish times for 2012, for runners less than 50 yrs old were slower
# than those of 1999.  Does this mean that more amature runners are participating in the race vs the
# first few years?



#------- Figure NA - FastestMan by Age actual & Loess prediction----
pdf("./Figures/Fig_NA_FastestManLoessPredictedCurves.pdf", width = 8, height = 6)

# Determines fastest runner's time for each age using all years(men)
fastestMan = tapply(cbMenSub$runTime, cbMenSub$age, min, na.rm = TRUE)
# Example fastestMan
#  16       17       18       19 .....
# 55.91667  55.50000  56.16667  45.96667 ....
oldPar = par(mar = c(4.1, 4.1, 1, 1))
plot(fastestMan ~ names(fastestMan), type ="l", xlim = c(20, 90), col = 'blue')

ageFM = as.numeric(names(fastestMan)) #List of ages with fastestMan data
mR.loF = loess(fastestMan ~ ageFM) # Using FastestMan data to loess model, (only data is fastest for each
                    #  age )
mR.lo.prF = predict(mR.loF, data.frame(age = ageFM), se = FALSE) # Predicts the time for each year based on
                                    # the loess model created from FastestMan dataset
# Sameple output of mR.lo.prF
# 1        2        3        4
#50.97034  50.43490  49.93866  49.48353

# Plotting loess prediction of FastestMan dataset
lines(x = ageFM, y = mR.lo.prF, col = "purple", lwd = 2)
legend("topleft", col = c("blue", "purple"), lty = 1:1, lwd = 3,
    legend = c("Fastest Man", "Loess Predition"), bty = "n")
par(oldPar)
dev.off()
#------




timeNorm = cbMenSub$runTime / mR.lo.prF[as.character(cbMenSub$age)]
# mR.lo.prF (first 4 values) - predicted values of fastest runner for each age group
# 28       24       27       28        ....
# 52.23179 50.28427 51.77801 52.23179   ....
# cbMenSub[1:4,c('age','runTime')]
# Index    age    runTime
# 1999.1   28   46.98333
# 1999.2   24   47.01667
# 1999.3   27   47.05000
# 1999.4   28   47.11667
#
# timeNorm[1:4]
# 28        24        27        28
# 0.8995161 0.9350175 0.9086869 0.9020688

time99Norm = timeNorm[cbMenSub$year == 1999]
time12Norm = timeNorm[cbMenSub$year == 2012]
summary(time99Norm)
#   Min.   1st Qu.  Median   Mean 3rd Qu.    Max.    NA's
#  0.7285  1.2399  1.4056  1.4228  1.5908  2.5189       9

summary(time12Norm)
#   Min.   1st Qu.  Median   Mean 3rd Qu.    Max.    NA's
#  0.6967  1.3118  1.5126  1.5315  1.7255  2.7663      13

#------- Figure NA - Time Distribution of Runners vs predicted fastest time for each Age group ----
pdf("./Figures/Fig_NA_TimeDistribution_vs_PredictedFastestTimePerAgeGroup.pdf", width = 10, height = 6)
plot(density(100*time99Norm, na.rm = TRUE),
     ylim = c(0, 0.016),
     col = "purple",
     lwd = 3,  xlab = "Time (percentage)",
     main = "Time Distribution for 1999 and 2012 Runners\n Percentage of the fastest runner for that age")
lines(density(100*time12Norm, na.rm = TRUE),
      lwd = 3, col = "green")
legend("topleft", fill = c("purple", "green"),
       legend = c("1999", "2012"), bty = "n")
par(oldPar)
dev.off()
#-----



#------------------- Section 2.5 -------------------------------------------
#-------------------  Switching over to individuals runners across years ----

# Want to start tracking runners that participated in race more than 1
# year.  Will use name, age, and hometown to identify these individuals
# (Was no unique identifier supplied to each individual)

# start by trimming the blanks from the name variables

# Create a function that removes blanks at the beginning,
# end, and multiple blanks between first and last names
trimBlanks = function(charVector) {
  nameClean = gsub("^[[:blank:]]+", "", charVector)# beginning
  nameClean = gsub("[[:blank:]]+$", "", nameClean) # end
  nameClean = gsub("[[:blank:]]+", " ", nameClean) #multiple between, replaced with 1 blank
}

# Create a vector of cleaned names
nameClean = trimBlanks(cbMenSub$name)
# [1] "Worku Bikila"       "Lazarus Nyakeraka"    "James Kariuki"       "William Kiptum"

length(nameClean) #69804
length(unique(nameClean)) #42882

# how many times a name appears in
table(table(nameClean))
# 1      2     3     4    5     6     7     8     9   10  11  12  13  14  15  17  18  19  30
#29291  7716  2736  1386  712   417   249   149   92  56  44  19  7   3   1   1   1   1   1

# Most common name
head( sort(table(nameClean), decreasing = TRUE), 1)
#Michael Smith
#30

# Selecting all Michael Smith's and seeing where they live
mSmith = cbMenSub[nameClean == "Michael Smith", ]
head(unique(mSmith$home))
#1] "Annapolis MD      "  "Bethesda MD       "  " Annapolis MD      " " Chevy Chase MD    " " Annandale VA

#To aid in cleaning the home variable, switch all characters to lowercase
nameClean = tolower(nameClean)
head( sort(table(nameClean), decreasing = TRUE), 1) # now we have 33 unique entries for Michael Smith

nameClean = gsub("[,.]", "", nameClean) # removing , . from home

#How many times a name appears in a given year
# Creates a table with every name in database, for all years, and how many times it appears in each year
tabNameYr = table(cbMenSub$year, nameClean)
max(tabNameYr) # 5 runners with same name in 1 yr

class(tabNameYr) # table
mode(tabNameYr) # numeric
names(attributes(tabNameYr)) #"dim"    "dimnames" "class"

dim(tabNameYr)#14 39133  14
head(colnames(tabNameYr), 3) #"8illiam maury"    "a gudu memon"     "a miles simmons"

which( tabNameYr == max(tabNameYr) ) #356496
which( tabNameYr == max(tabNameYr), arr.ind = TRUE )
#      Row Col
# 2012 14   25464

indMax = which( tabNameYr == max(tabNameYr), arr.ind = TRUE )
colnames(tabNameYr)[indMax[2]] #"michael brown"

# add cleaned name to our men's dataframe as an additional variable
cbMenSub$nameClean = nameClean

# Creating a year of birth variable by subtracting runner's age from the year of competition
cbMenSub$yob = cbMenSub$year - cbMenSub$age

# Fix home in a similar way and add it to the dataframe
homeClean = trimBlanks(tolower(cbMenSub$home))
cbMenSub$homeClean = gsub("[,.]", "", homeClean)

# exploring dataset variable subsetted for viewing all 'michael brown's'
vars = c("year", "homeClean", "nameClean", "yob",  "runTime")
mb = which(nameClean == "michael brown")
birthOrder = order(cbMenSub$yob[mb])
cbMenSub[mb[birthOrder], vars]
# year      homeClean     nameClean  yob   runTime
# 2000.2526 2000      tucson az michael brown 1939  96.88333
# 2010.4241 2010  north east md michael brown 1953  92.26667
# 2011.3026 2011  north east md michael brown 1953  85.95000
# 2012.3800 2012  north east md michael brown 1953  88.43333
# 2009.5246 2009      oakton va michael brown 1957  99.73333
# 2008.3896 2008     ashburn va michael brown 1958  93.73333
# 2009.3509 2009     ashburn va michael brown 1958  88.56667
# 2010.5309 2010     ashburn va michael brown 1958  99.75000
# 2012.4078 2012      reston va michael brown 1958  89.95000
# 2006.2631 2006    chevy chase michael brown 1966  84.56667
# 2010.1907 2010 chevy chase md michael brown 1966  79.35000
# 2012.5089 2012 chevy chase md michael brown 1966  95.81667
# 2004.998  2004  berryville va michael brown 1978  76.31667
# 2008.2501 2008   arlington va michael brown 1984  84.68333
# 2010.6307 2010    new york ny michael brown 1984 110.88333
# 2011.2274 2011   arlington va michael brown 1984  81.70000
# 2012.881  2012   arlington va michael brown 1984  70.93333
# 2012.3084 2012     clifton va michael brown 1988  84.88333

# Attempt to create a unique identifer for each runner
cbMenSub$ID = paste(nameClean, cbMenSub$yob, sep = "_")
head(cbMenSub$ID,3) #"worku bikila_1971" "lazarus nyakeraka_1975" "james kariuki_1972"
```

```
# Determine how many times an 'ID' appears in data-frame (We are ignoring the hometown info for now)
races = tapply(cbMenSub$year, cbMenSub$ID, length)

# creating a vector with ID that have appeared in at least 8 races
races8 = names(races)[which(races >= 8)]
length(races8) # 480
# Creating a subset of mens dataframe to include on those ID that appeared more than 8 times
men8 = cbMenSub[ cbMenSub$ID %in% races8, ] #4575 obs x 10 vars

# Reorder dataframe by ID then by Year
orderByRunner = order(men8$ID, men8$year)
men8 = men8[orderByRunner, ]
# year sex                 name            home age   runTime        nameClean  yob
# 1999.1644 1999   M       Aaron Glahe          Arlington VA       25  84.73333        aaron glahe 1974
# 2001.752  2001   M       Aaron GLAHE          Arlington VA       27  74.21667        aaron glahe 1974
# 2002.1188 2002   M       Aaron GLAHE          Arlington VA       28  79.70000        aaron glahe 1974
# 2003.2307 2003   M  Aaron Glahe               Reston VA          29  88.90000        aaron glahe 1974
# 2004.1151 2004   M Aaron Glahe                Reston VA          30  77.86667        aaron glahe 1974
# 2005.2247 2005   M       Aaron Glahe          Reston VA          31  90.68333        aaron glahe 1974
# 2006.3273 2006   M       Aaron Glahe          Reston             32  91.71667        aaron glahe 1974
# 2008.4557 2008   M       Aaron Glahe          Reston VA          34  98.76667        aaron glahe 1974
# 1999.2640 1999   M       Abiy Zewde           Gaithersburg MD    32  96.51667         abiy zewde 1967
# 2000.2616 2000   M       Abiy Zewde           Montgomery Vill MD   33  96.63333        abiy zewde 1967
# 2001.2276 2001   M       Abiy ZEWDE           Montgomery Vill MD   34  89.10000        abiy zewde 1967
# 2002.3684 2002   M       Abiy ZEWDE           Montgomery Vill MD   35 123.00000        abiy zewde 1967
length(unique(men8$ID)) #480

# Alternative approach for data organization. We will store the data as a list with an element
# for each ID in races8 and each list element will contain a dataframe for the results associated
# with that ID
men8L = split(men8, men8$ID)
names(men8L) = races8
men8L[1]
# $`aaron glahe_1974`
# year sex                 name            home        age  runTime   nameClean  yob    homeClean          ID
# 1999.1644 1999   M       Aaron Glahe         Arlington VA       25 84.73333 aaron glahe 1974 arlington va aaron
glahe_1974
# 2001.752  2001   M       Aaron GLAHE         Arlington VA       27 74.21667 aaron glahe 1974 arlington va aaron
glahe_1974
# 2002.1188 2002   M       Aaron GLAHE         Arlington VA       28 79.70000 aaron glahe 1974 arlington va aaron
glahe_1974
# 2003.2307 2003   M  Aaron Glahe              Reston VA          29 88.90000 aaron glahe 1974    reston va aaron
glahe_1974
# 2004.1151 2004   M Aaron Glahe               Reston VA          30 77.86667 aaron glahe 1974    reston va aaron
glahe_1974
```

```
# 2005.2247 2005   M       Aaron Glahe           Reston VA         31 90.68333 aaron glahe 1974    reston va aaron
glahe_1974
# 2006.3273 2006   M        Aaron Glahe              Reston        32 91.71667 aaron glahe 1974      reston aaron
glahe_1974
# 2008.4557 2008   M       Aaron Glahe           Reston VA         34 98.76667 aaron glahe 1974    reston va aaron
glahe_1974
length(men8L) #480

# to attempt to weed out some of the IDs that have different individuals within
# them, we look to performance variation yr-to-yr that vary too much
# We arbitrarily chose a differnece of 20 minutes
# Creating a logical (T/F) variable to record gaptimes greater than 20
gapTime = tapply(men8$runTime, men8$ID,
             function(t) any(abs(diff(t)) > 20))
# optional approach
gapTime = sapply(men8L, function(df)
  any(abs(diff(df$runTime)) > 20))

# Counting the number of gapTImes that are true
sum(gapTime) #49 (suspect pairings)

lapply(men8L[ gapTime ][1:2], function(df) df[, vars])
# $`abiy zewde_1967`
# year          homeClean  nameClean  yob   runTime
# 1999.2640 1999      gaithersburg md abiy zewde 1967  96.51667
# 2000.2616 2000   montgomery vill md abiy zewde 1967  96.63333
# 2001.2276 2001   montgomery vill md abiy zewde 1967  89.10000
# 2002.3684 2002   montgomery vill md abiy zewde 1967 123.00000
# 2003.3301 2003      gaithersburg md abiy zewde 1967  97.68333
# 2004.3579 2004   montgomery vill md abiy zewde 1967 100.36667
# 2006.4839 2006        gaithersburg abiy zewde 1967 108.40000
# 2008.4562 2008   montgomery vill md abiy zewde 1967  98.78333
# 2009.5072 2009 montgomery villag md abiy zewde 1967  98.50000
# 2010.5330 2010 montgomery villag md abiy zewde 1967  99.91667
# 2011.6493 2011 montgomery villag md abiy zewde 1967 113.10000
# 2012.3085 2012 montgomery villag md abiy zewde 1967  84.88333
#
# $`adam hughes_1978`
# year     homeClean   nameClean  yob   runTime
# 2005.1916 2005 washington dc adam hughes 1978  80.38333
# 2006.2279 2006    washington adam hughes 1978  85.16667
# 2007.1367 2007 washington dc adam hughes 1978  77.78333
# 2008.1028 2008 washington dc adam hughes 1978  74.23333
# 2009.5980 2009 washington dc adam hughes 1978 108.06667
# 2010.5641 2010 washington dc adam hughes 1978 103.06667
# 2011.1484 2011 washington dc adam hughes 1978  77.11667
# 2012.1836 2012 washington dc adam hughes 1978  77.76667


# We will further clean up the home variable by extracting the state name and
# create a new variable for the state ID
# Create a variable for the length of each home variable
homeLen = nchar(cbMenSub$homeClean)
# Create a state variable, using the last 2 characters of the home variable
cbMenSub$state = substr(cbMenSub$homeClean,
                    start = homeLen - 1, stop = homeLen)

# Since we know that 2006 does not list states, we will convert them to NA
cbMenSub$state[cbMenSub$year == 2006] = NA

# Modifying the runner's unique ID, adding the State ID to it.
cbMenSub$ID = paste(cbMenSub$nameClean, cbMenSub$yob,
                cbMenSub$state, sep = "_")

# Reselect the ID's that appear at least 8 times (like before)
numRaces = tapply(cbMenSub$year, cbMenSub$ID, length)
races8 = names(numRaces)[which(numRaces >= 8)]
length(races8) #306 IDs
men8 = cbMenSub[ cbMenSub$ID %in% races8, ]
orderByRunner = order(men8$ID, men8$year)
men8 = men8[orderByRunner, ]


#Alternative
men8L = split(men8, men8$ID)
names(men8L) = races8

# We now have 306 runners that have the same name, yob, and state that have completed 8 races



#####################################################################
##### 2.6 Change in Running time for individuals over time
####  Exploration of data
####################################################################

# We will begin by taking the 306 runners and splitting them
# into 9 groups, so we can display them better
groups = 1 + (1:length(men8L) %% 9)
# Create a function to add the runners to the groups,
# create a color index and line type, within group,
# for each runner
```

```
addRunners = function(listRunners, colors, numLty)
{
  numRunners = length(listRunners)
  colIndx = 1 + (1:numRunners) %% length(colors)
  ltys = rep(1:numLty, each = length(colors), length = numRunners)

  mapply(function(df, i) {
    lines(df$runTime ~ df$age,
          col = colors[colIndx[i]], lwd = 2, lty = ltys[i])
  }, listRunners, i = 1:numRunners)
}
#---

# The Fig 2.17 graph shows that the finish times for 2012, for runners less than 50 yrs old were slower
# than those of 1999.  Does this mean that more amature runners are participating in the race vs the
# first few years?


#------- Figure 2.17 - Run Times for Multiple Races ----
# Fig 2.17 graph shows the run times for individuals who participated
# in at least 8 races.  306 individuals are spread across 9 separate
# plots, with different colors and linetypes

# Colors used for color Index
colors = c("#e41a1c", "#377eb8","#4daf4a", "#984ea3",
           "#ff7f00", "#a65628")
pdf("./Figures/Fig_2.17_RunTimesMultipleRaces.pdf", width = 10, height = 6)
par(mfrow = c(3, 3), mar = c(2, 2, 1, 1))
invisible(
  sapply(1:9, function(grpId){
    plot( x = 0, y = 0, type = "n",
          xlim = c(20, 80), ylim = c(50, 130),
          xlab = "Age (years)", ylab = "Run Time (minutes)")

    addRunners(men8L[ groups == grpId ], colors, numLty = 6)
  }) )
par(oldPar)
dev.off()
#-------

#---------- fitOne () Crates lm() for Runners --------
# We want to go ahead and fit a linear regression model to each of the
# runners in a particular group
# We create a function (fitOne) to fit a lm model for each runner, store an
# index value, and coefficients (agecoeff, median Age, Predicted run time)
#
```

```
fitOne = function(oneRunner, addLine = FALSE, col = "grey") {
 lmOne = lm(runTime ~ age, data = oneRunner)
 if (addLine)
  lines(x = oneRunner$age, y = predict(lmOne),
     col = col, lwd = 2, lty = 1)

 ind = floor( (nrow(oneRunner) + 1) / 2)
 res = c(coefficients(lmOne)[2], oneRunner$age[ind],
     predict(lmOne)[ind])
 names(res) = c("ageCoeff", "medAge", "predRunTime")
 return(res)
}
#----

#------- Figure 2.18 - Linear Run Times fit for Individual Runners ----
# Fig 2.18 graph shows the run times for individuals who participated
# in at least 8 races and are part of Group 9.  Figure also display
# predicted linear fit model for each runner (black dashed)
pdf("./Figures/Fig_2.18_RunTimesMultipleRacesLinearFit.pdf", width = 10, height = 6)
par(mfrow = c(1, 1), mar = c(5, 4, 1, 1))
# For this plot, we will display results for group 9 only
plot( x = 0, y = 0, type = "n",
   xlim = c(20, 80), ylim = c(50, 130),
   xlab = "Age (years)", ylab = "Run Time (minutes)")

addRunners(men8L[ groups == 9 ], colors, numLty = 6)
lapply(men8L[groups == 9], fitOne, addLine = TRUE, col = "black")
par(oldPar)
dev.off()
#-----

# We now go ahead and fit a lm() to all 306 runners who have run more than 8 races
# store the coefficients, median age, and predicted run time for median age
men8LongFit = lapply(men8L, fitOne)
head(men8LongFit,4)
# $`abiy zewde_1967_md`
# ageCoeff     medAge    predRunTime
# -0.08965587 37.00000000 99.98437922
#
# $`adam knapp_1977_va`
# ageCoeff     medAge    predRunTime
# -0.2706019  31.0000000 104.5979167
#
# $`adam stolzberg_1976_va`
# ageCoeff     medAge    predRunTime
# -0.8427656  29.0000000  67.6580433
```

```
#
# $`al navidi_1960_md`
# ageCoeff     medAge   predRunTime
# 0.09554381 45.00000000 76.13213746

coeffs = sapply(men8LongFit, "[", "ageCoeff" ) #extracted named coefficients for each of the runners
head(coeffs,3)
# abiy zewde_1967_md.ageCoeff     adam knapp_1977_va.ageCoeff adam stolzberg_1976_va.ageCoeff
# -0.08965587                   -0.27060185                    -0.84276557

ages = sapply(men8LongFit, "[", "medAge") #extracted named median ages for each of the runners
head(ages,3)
# abiy zewde_1967_md.medAge     adam knapp_1977_va.medAge adam stolzberg_1976_va.medAge
# 37                        31                          29


# Create a linear model for all of the 306 runners
longCoeffs = lm(coeffs ~ ages)

summary(longCoeffs)
# Summary Statistics
# Call:
#   lm(formula = coeffs ~ ages)
#
# Residuals:
#   Min      1Q  Median      3Q     Max
# -4.4026 -0.6375 -0.0246  0.5645  3.3541
#
# Coefficients:
#   Estimate Std. Error t value Pr(>|t|)
# (Intercept) -1.958440   0.305487  -6.411 5.51e-10 ***
#   ages        0.055263   0.006175   8.949  < 2e-16 ***
#   ---
#   Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# Residual standard error: 1.01 on 304 degrees of freedom
# Multiple R-squared:  0.2085,        Adjusted R-squared:  0.2059
# F-statistic: 80.09 on 1 and 304 DF,  p-value: < 2.2e-16

#------- Figure 2.19 - Coefficients of Longitudinal Analysis of Athletes ----
# Fig 2.19 graph shows how the coefficients of the individual runners
# varies with age (neg values meas they are incresing with age, + values
# mean they are slowing down).
# Fitted both a lm() and Loess() model to the data and displayed both
pdf("./Figures/Fig_2.19_CB_LongCoeffs.pdf", width = 10, height = 7)
oldPar = par(mar = c(4.1, 4.1, 1, 1))
```

```
plot(coeffs ~ ages, cex=.8, pch=20,xlab = "Median Age (years)",
    ylab = "Coefficient (minutes per race / year)")
abline(longCoeffs, col = "#984ea3", lwd = 3)
abline(h = 0, col="blue", lwd = 3)
loCoeffs = loess(coeffs ~ ages) #Fitting loess line
ageV = min(ages):max(ages)
predV = predict(loCoeffs, new = data.frame(ages = ageV))
lines(x = ageV, y = predV, lwd = 3, lty = 2, col = "#4daf4a")
par(oldPar)
dev.off()
#----
```