## **Introduction**

Data science jobs have been heralded as “the best role in America” by Glassdoor for several years, based on the current industry demand, salary, and job satisfaction. Data scientist has also been called the “sexiest job of the 21st century” by the Harvard Business Review. Most data science practitioners will agree that the term “data science” is incredibly broad and includes many skills like: data engineering, modeling, analysis, feature engineering, machine learning and data visualization. While most outsiders regard data science as “magic” one thing almost always holds true – data scientists spend most of their time finding, cleaning, and reorganizing huge amounts of data. [1]

Armand Ruiz, a contributor to the InfoWorld website, argues that the composition of a data scientist’s work follows the pattern of the 80/20 Pareto principle. That is – that most data scientists spend around 80 percent of their time finding, cleaning, and reorganizing data, while only 20 percent of their time performing analysis, modeling, and gleaning insights. The effect of this in the modern enterprise is often hasty decisions and judgment calls about the data, which leads to an imbalance of quality against time constraints. Data scientists are often tempted to declare data as “good enough” versus further pursuit to acquire data or further prepare data that would have a more optimal result.

In this study, we will demonstrate the amount of data munging and cleaning iterations involved in preparing even a small, simple data set for further analysis. Our analysis requires race results data acquired from the annual Cherry Blossom Ten Mile Run in Washington, D.C. Our case study will address question 17 from the case study *Modeling Runners Times in the Cherry Blossom Race* from the Nolan and Lang text:

In Section 2.7, we discovered that the HTML file for the male 2000 results was so poorly formatted that htmlParse() was unable to fix it to allow us to extract the text table from the <pre> tag. In this exercise, we programmatically edit this HTML file so that we can use htmlParse() as desired. To do this, begin by reading the HTML file located at http://www.cherryblossom.org/cb003m.htm using readLines(). Carefully examine the HTML displayed in Section 2.7 and come up with a plan for correcting it. Consider whether you want to drop <font>s or close them properly. Once you have fixed the problem so that the <pre> tag contains the text table, pass your corrected HTML to htmlParse(). You may want to use a text connection to do this rather than writing the file to disk and reading it in. [1]

## **Background**

The race results for the Cherry Blossom Ten Mile Run are available in HTML format from the [www.cherryblossom.org](http://www.cherryblossom.org) website. We employed a process known as web scraping, which is programmatically loading web content from a web server for the purpose of acquiring data for analysis. This approach often has to be done in the absence of other readily available cohesive collections of the data, for example in a comma-separated values or other text tabular format. Many packages are available for R and Python to deal with scraping web data and manipulating it into a format that is suitable for data analysis.

The complexities with web scraping comes from two primary sources: malformed or improper structure in the HTML document and subtle inconsistencies among a series of documents that are assumed to follow the same structure. Correctly formed HTML documents include appropriate markup to designate various sections of the document, including: headers, tables, lists, and body text. The HTML markup itself must also behave in an expected manner. A common issue with HTML is the improper nesting of hierarchical elements, which means that a closing tag for a given markup tag is occurring outside of another closing tag in the wrong order. Algorithms in web scraping packages expect HTML documents to follow standard markup best practices, so the existence of improper nesting of HTML elements often leads to missed or malformed data when it is parsed. Secondly, while improper HTML structure can be handled when it’s found, most approaches assume that a series of HTML documents containing similar data will have the same formatting. Ideally, analysts will not have to make distinct ingest and parsing decisions on a per document basis as this is both time consuming and error-prone. As we prepared our results data from the Cherry Blossom Ten Mile Run for analysis, we uncovered both of these types of complexities.

The goal of the analysis performed in the case study *Modeling Runners Times in the Cherry Blossom Race* from the Nolan and Lang text was to examine the relationship of age and performance. With the data that is available the authors are able to analyze groups of people by age and gender and their related performance (race time). They also identify race results for individuals who have run multiple races, since the full name and runner location information is available. Therefore, in order to perform both of these analyses, our parsed data set will require full name, hometown, the year the race was completed, age, and the race time. [2]

## **Method**

## The methods used in this case study are based on the analysis outlined in Case Studies in Data Science in R, Chapter 2. The data cleaning approach in our preparation closely follows the approach given by the book, while addressing several special cases that the authors chose not to implement.

First, the data are available on the [www.cherryblossom.org](http://www.cherryblossom.org) website under the Race Results navigation on the left navigation bar. The first task was to understand similarities in the results pages that could be used to parse the data programmatically. We found that the files all included various number of rows in the header that included general information. We also found that the files had rows designating the column name as well as a row that we could use to understand where to get each value per row by character indexing. This spacer row consists of equal signs that align to the column widths of the data below and they are separated with spaces (Figure 1).

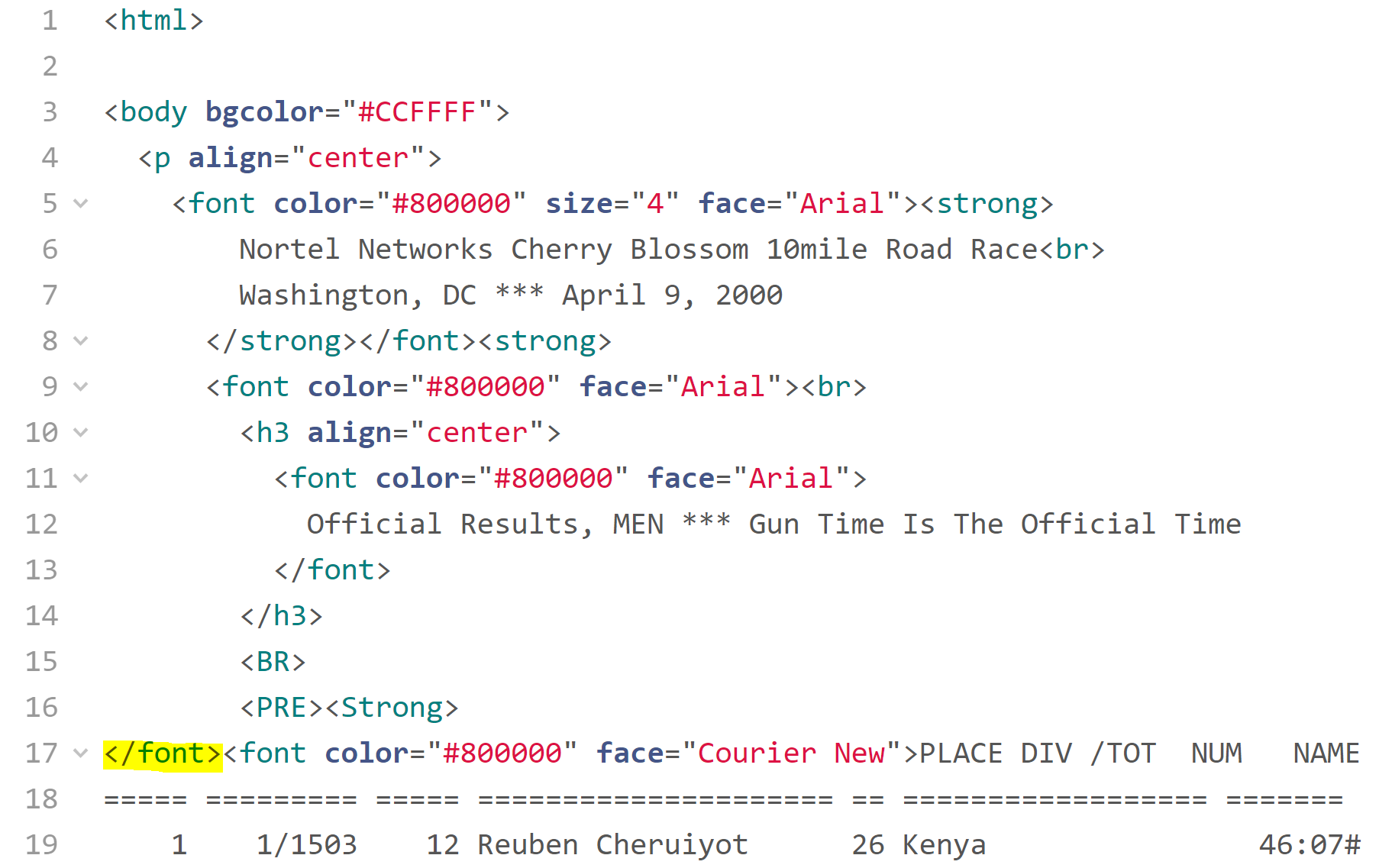
**Figure 1. Raw Text Results Data from the 2012 Men’s Cherry Blossom Race**



Our approach was to find the row containing column names by first finding the row that begins with equal signs – all header rows before the column names and spacer row can be removed. The spacer row can then be analyzed to find the index of each of the spaces as well as taking note of the beginning index and ending index (since those are important, but aren’t spaces). Once these indexes are known, the column names row and each of the participant rows can be reduced to their individual fields by applying the fixed width indexes. Furthermore, the column names row can be interpreted to understand the type of data in each of the columns below. This process was considered our generalized case and was acceptable for parsing most of the files. However, when scraping the data we identified several special cases where the HTML differed from the generalized case.

The first of these cases were the results for both men and women from the year 2000. Upon examination, the HTML in this instance was improperly nested, with one </font> tag occuring in the wrong place, as highlighted in Figure 2 below. To fix the issue programmatically, we removed all </font> tags and exploited a feature of the htmlParse() function that automatically closes unclosed tags. This resulted in the proper nesting. We did this by reading the lines of the HTML file directly, using regular expressions to find and remove the closing tags, and calling htmlParse()on the modified version of the HTML source.

**Figure 2. Improperly Nested HTML for 2000 Men’s Race Results**

****

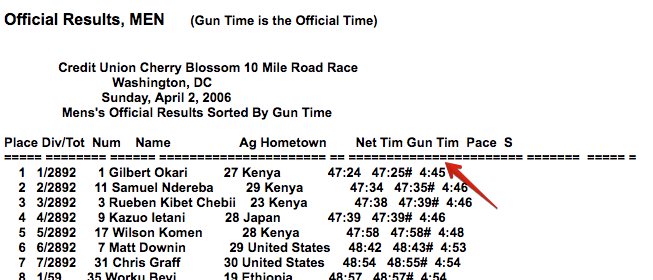
The men’s results for 2009 presented a different challenge, since the file was formatted quite differently from the other years. There were many additional tags to accommodate CSS formatting in this year’s file, and the main challenge was the use of a <div> tag within the <body> tag. The getNodeSet()function could not find the tags it was looking for because it was “masked” inside the wrapping <div> tag. To fix this, we looked for the data in the <div> tag, and then we were able to feed that to a second call of getNodeSet() to extract the <pre> tags. Because each line was stored in its own <pre> tag, we used a different method of extracting each line than for the other files as well. See the highlighted tags in Figure 3 below to examine this special case. The 2009 women’s results did not have the same issue as discussed above for the men, so that file was able to use the generalized process.

**Figure 3. Additional Nesting and <pre> Tags in HTML for 2009 Men’s Race Results**



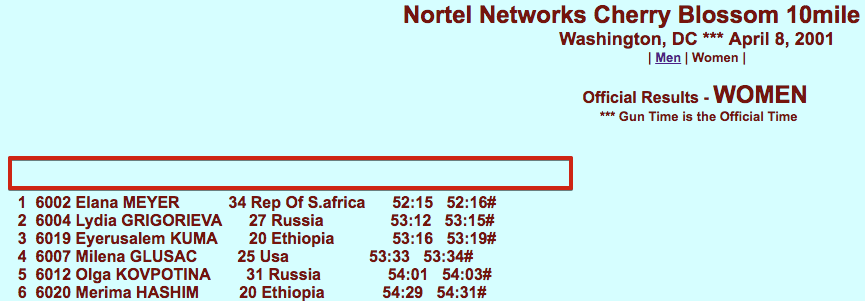
The results from 2006 also presented a new issue. We found that data was being improperly parsed because a space character did not exist between hometown and age in the spacer row. This issue was consistent with the men’s and the women’s files. In order to fix the problem we programmatically inserted a space character at the 64th character of the spacer row. See Figure 4 to examine where the new space character was inserted.

**Figure 4. 2006 Men’s and Women’s Results Missing Space Between Hometown and Net Time**



Finally, the women’s results for 2001 did not include the expected column names or spacer row (Figure 5). We found that the men’s file results from that year was processed correctly and the column names and spacer row were consistent with the women’s file field spacing. Thus we were able to copy the column name and spacer row explicitly from the men’s file to read in the women’s results for 2001.

**Figure 5. 2001 Women’s Results Missing Column Names and Spacer Row**



Once the data was scraped from the web, we needed to ensure the data was the correct types for analysis, and we focused on time. We uncovered that the files use different versions of time measurement, including net time, gun time, and total time. Based on the authors recommendation we would prefer net time when available. We also must convert the string format time hh:mm:ss (hours:minutes:seconds) into a format that can be used for analysis. To handle this conversion, we built a function to break the time apart by colons and convert total time into minutes and convert to a numerical data type.

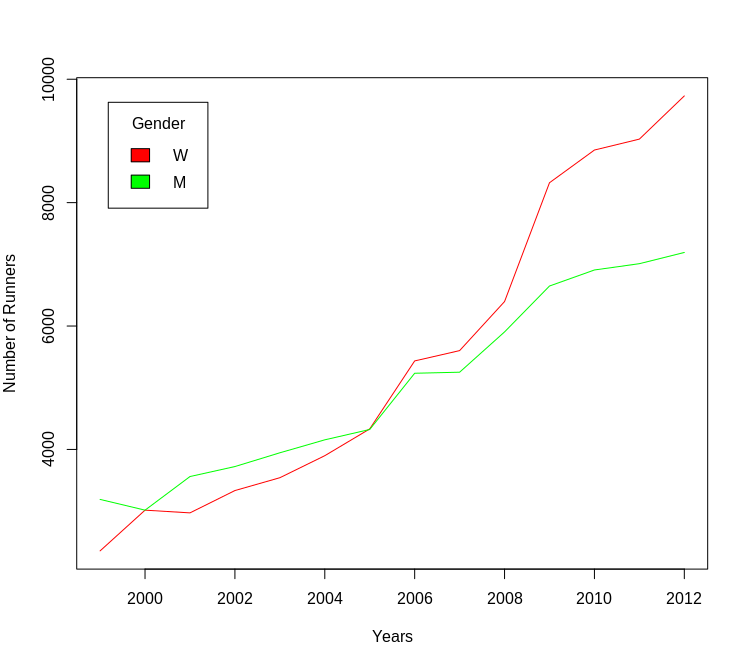
Because some years included different information, a subset of the available variables was chosen for creating the final datasets. These were variables that were available across all of the years, and were comparable to one another. They are: race year, gender, name, age, and run time. Run time represents the amount of time that each racer spent on the course, no matter what the master clock said when they crossed the starting line.

## **Results**

With the data prepared through the steps described above we utilized this cleaned data set to analyze the relationship between age and race times. This analysis followed a similar structure to what is covered in Section 2.4 of Data Science in R.

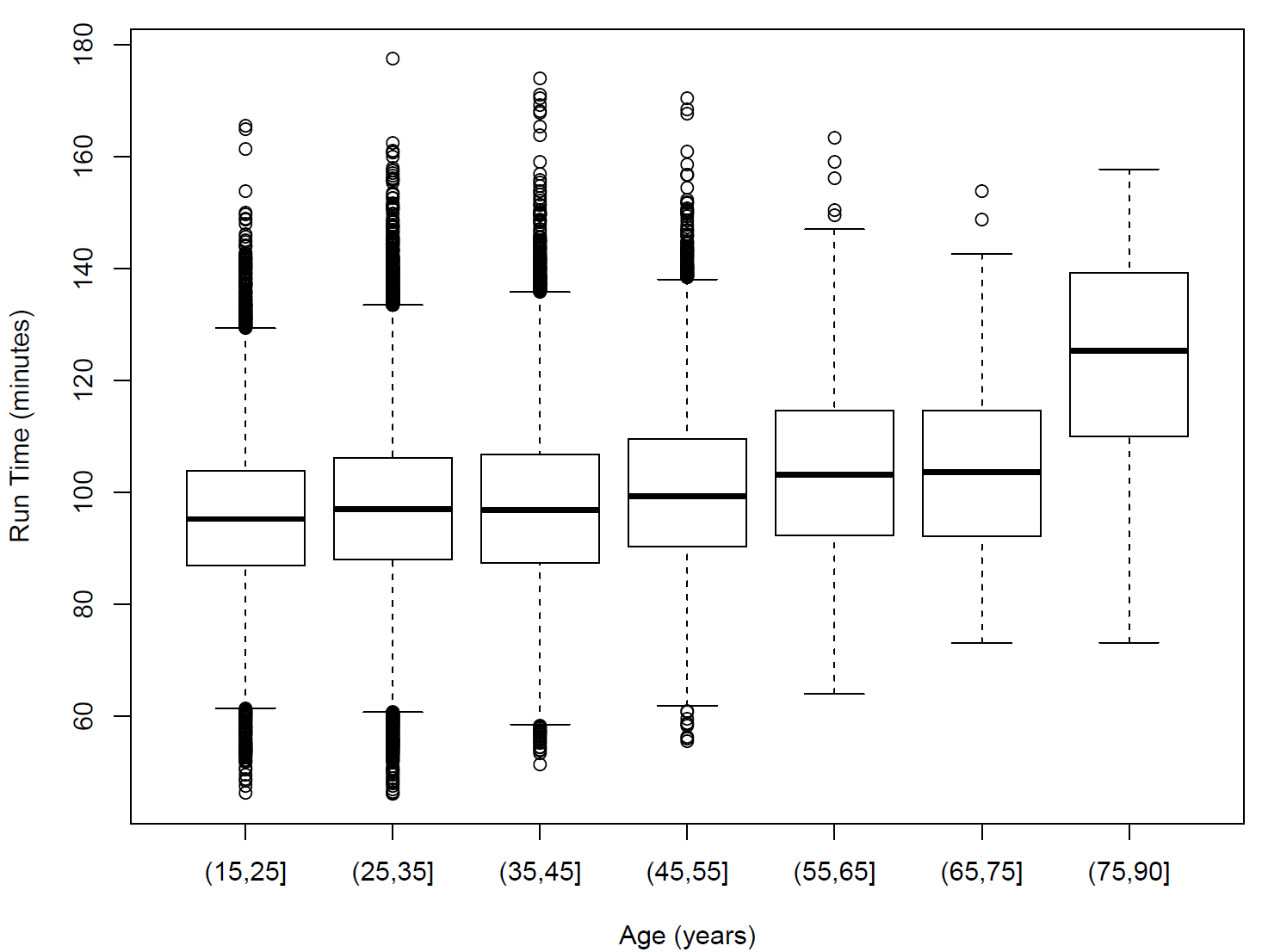
First we look at our overall participation in the race over the years between men and women, shown below in Figure 6. It clearly shows that the overall participation in this race has more than doubled between 1999 - 2012, and that the increase in participation among women has been greater than men [3]. This also demonstrates that our cleaning was able to address the original problem around parsing the HTML correctly for the year 2000 & 2009. Previously the green line for 2000 & 2009 would have fallen down to one due to this issue.

**Figure 6. Race Participants by Year**



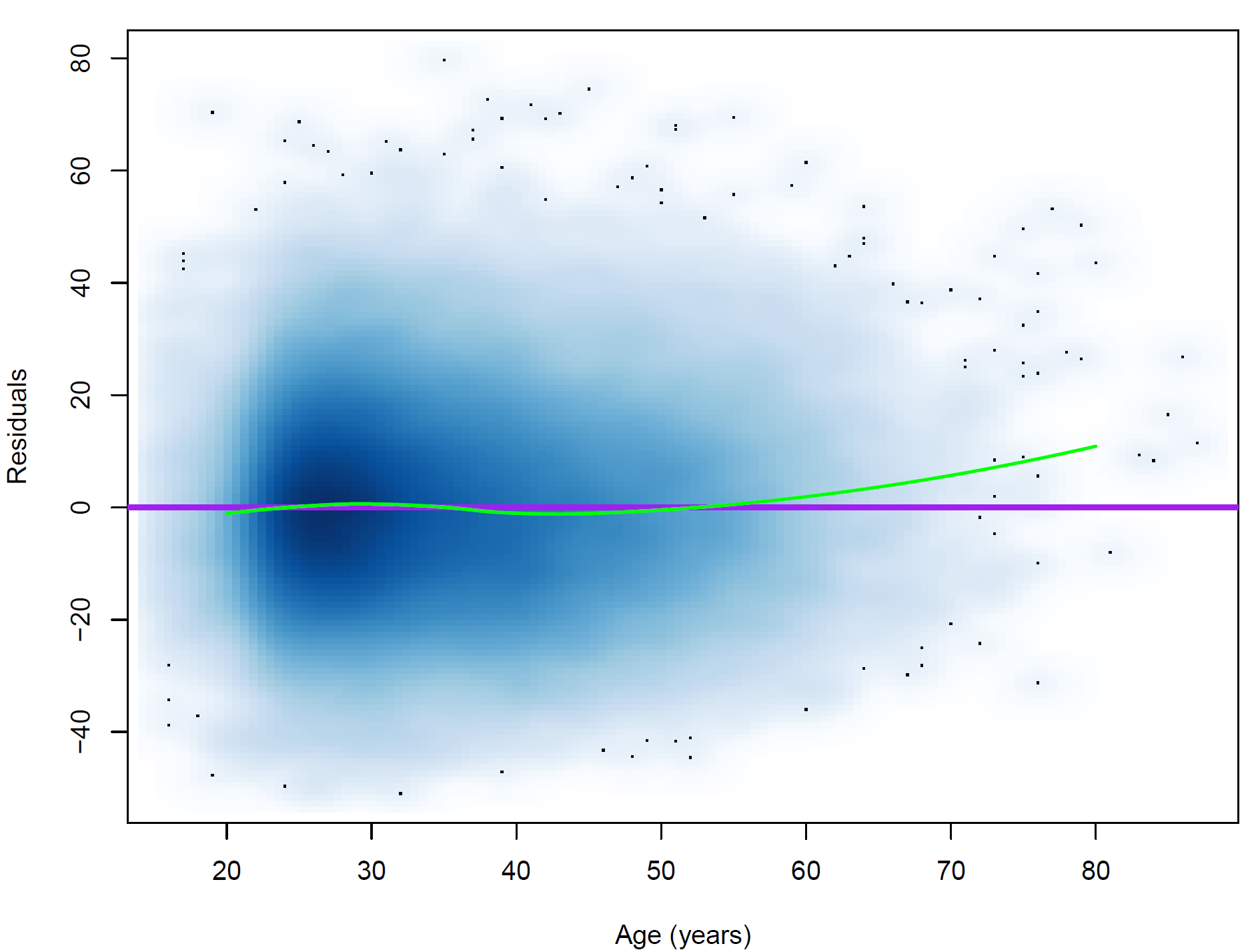
As we began to look specifically at the age versus time relationship, we shifted our focus to the female runners to be develop an comparable analysis to what is done in the book. To get a basic understanding of the distribution of the times relative age we looked at a box-plots by various age groups shown in Figure 7.

**Figure 7. Run Times by Age Group**

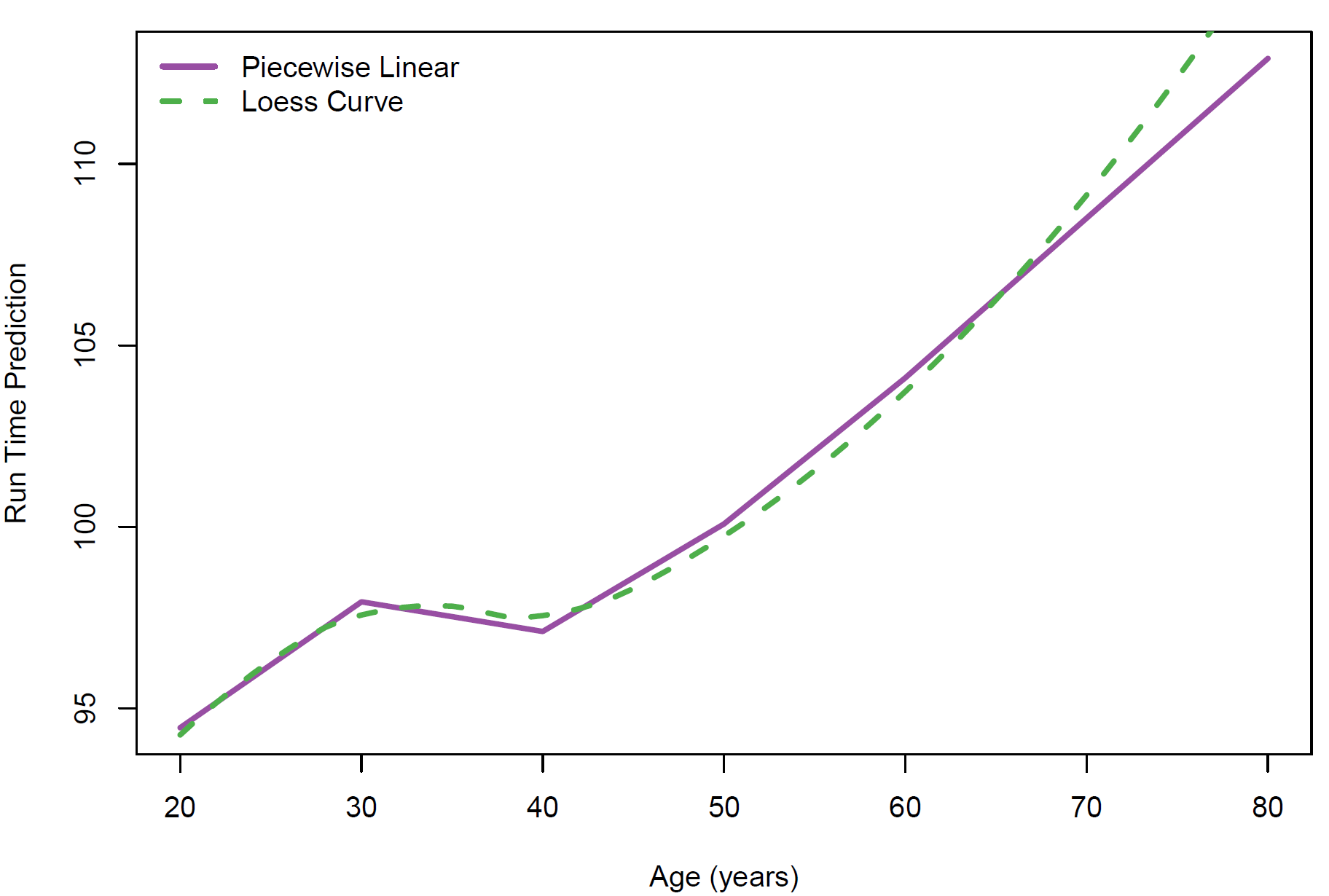


From this plot we are able to see a positive relationship between age and run time. Although, we do notice how the run times begin to increase at a higher rate in the higher age groups. In order to better understand if a linear model would be appropriate, we fit a simple linear model to our data. The resulting model indicated a significant positive relationship between age & time. Our estimates show that for every 1 year increase in age a male runners time would increase by 0.162 minutes with a 95% confidence interval of (0.151, 0.173) . However, this result can be slightly misleading as a review of the the residuals from that model, in Figure 8, show some indication that linear model is perhaps not ideal for this problem.With the residuals as well as a loess curve we are able to very clearly see that our simple linear model begins to deviate away from 0 after the age of about 55. This is provides some insights that a linear model, while shown to be statistically significant, is not the best model for higher age participants.

**Figure 8. Residuals from Simple Linear Model**



Generating a piecewise linear regression model, treating each decade change prior to Age 50 as a point of a change, does a relatively good job of approximating the shape of the loess curve (Figure 9). Even the piecewise linear model does not fully capture the increase seen in the oldest runner’s ages. Additional analysis to identify specific changepoints may help in creating the most descriptive model that accounts for the increase in time by runners over age 55.

**Figure 9. Piecewise Linear Regression vs. Loess Curve**

## **Conclusions**

Once the data are cleaned, there are plenty of additional insights that can be uncovered from analyzing the data. With this particular dataset, we are were interested in looking for evidence that the performance of runners changes as runners age. The plot of the age box plot, as well as the residuals plot, show that there appears to be a relationship between increasing age and increasing time to complete the race.

In the process of building the dataset necessary to come to these conclusions, we’ve demonstrated several ways to modify a web scraping routine to account for differences, as well as how to transform data into an appropriate format for analysis. Despite all of the additional effort to pull and clean the web-scraped data, this effort was still small compared to a manual effort of extracting the data directly from the published pages.

## **References**

1. Ruiz, Armand. “The 80/20 data science dilemma.” InfoWorld, 26 September 2017, <https://www.infoworld.com/article/3228245/the-80-20-data-science-dilemma.html>.
2. Nolan, D. and Lang, D. T. “Data Science in R.” CRC Press, 2015 (Chapter 2).
3. Bachman, Rachel. “How Women took Over the World of Running.” *Wall Street Journal* 26 May 2016: [https://www.wsj.com/articles/how-women-took-over-the-world-of- running-146341598](https://www.wsj.com/articles/how-women-took-over-the-world-of-running-1463415987) .

## **Appendix - R Code**

library(XML)

extractResTable =

#

# Retrieve data from web site,

# find the preformatted text,

# and write lines or return as a character vector.

#

function(url, year = 1999, sex = "male", file = NULL)

{

doc = htmlParse(url, encoding='UTF-8')

if (year == 1999) {

ff = getNodeSet(doc, "//pre")

txt = xmlValue(ff[[1]])

els = strsplit(txt, "\n")[[1]] ## the '99 files use \n not \r\n

}

else if (year == 2000) {

# Revise the HTML by line to remove the /font tags and force the parse to correctly infer where they belong

html2000 = readLines("http://www.cherryblossom.org/results/2000/cb003m.htm")

newhtml2000 = sub("</font>","",html2000)

doc = htmlParse(newhtml2000)

pres = getNodeSet(doc,"//pre")

txt = xmlValue(pres[[1]])

els = strsplit(txt, "\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 if (year == 2001 & sex == "female") {

# Get preformatted text from <pre> elements

pres = getNodeSet(doc, "//pre")

txt = xmlValue(pres[[1]])

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

els = els[-(1:3)]

fix2001 = c("PLACE NUM NAME AG HOMETOWN NET GUN",

"===== ===== ===================== == ================== ======= =======")

els = c(fix2001, els)

}

else if (year == 2006) {

# THIS FIXES ISSUE WITH 2006 FILES AS NO SEPARATOR

# IN SPACER ROW BETWEEN HOMETOWN AND NET TIME

# OUTLINED ON PG. 62 - fixed for both gender

pres = getNodeSet(doc, "//pre")

txt = xmlValue(pres[[1]])

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

separatorIdx = grep("^===", els)

separatorRow = els[separatorIdx]

separatorRowX = paste(substring(separatorRow, 1, 63), " ", substring(separatorRow, 65, nchar(separatorRow)), sep = "") # insert space

els[separatorIdx] = separatorRowX # replace row

}

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)

} else {

# Write the lines as a text file.

fileConn = file(file)

writeLines(els, con = fileConn)

close(fileConn)

return(els)

}

}

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")

# create URLs for getting all men's data

ubase = "http://www.cherryblossom.org/"

men\_urls = paste(ubase, menURLs, sep = "")

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")

years = 1999:2012

women\_urls = paste(ubase, womenURLs, sep = "")

menFiles = sapply(years, function(x){paste(x, "m.txt", sep="")})

womenFiles = sapply(years, function(x){paste(x, "w.txt", sep="")})

mensTables = mapply(extractResTable, url = men\_urls, year = years, sex = "male", file = menFiles)

womensTables = mapply(extractResTable, url = women\_urls, year = years, sex = "female", file = womenFiles)

names(mensTables) = years

names(womensTables) = years

#save(mensTables, file = "CBMenTextTables.rda")

#save(womensTables, file = "CBWomenTextTables.rda")

findColLocs = function(spacerRow) {

spaceLocs = gregexpr(" ", spacerRow)[[1]] # returns the index of spaces

rowLength = nchar(spacerRow) # returns the length of the spacer row

if (substring(spacerRow, rowLength, rowLength) != " ") # if the last character of the spacerRow isn't a space

return( c(0, spaceLocs, rowLength + 1)) # return the spacer locations with an additional location 1 greater than the space row

else return(c(0, spaceLocs)) # else, space exists so return the spaceLocs as is

}

selectCols = function(colNames, headerRow, searchLocs)

{

sapply(colNames,

function(name, headerRow, searchLocs)

{

startPos = regexpr(name, headerRow)[[1]] # find the starting position of the name in the headerRow

if (startPos == -1) # if startPos = -1 (name doesn't exist) return NAs

return( c(NA, NA) )

index = sum(startPos >= searchLocs) # find the first Loc that contains the name

c(searchLocs[index] + 1, searchLocs[index + 1]) # return position that is one greater than the space containing the name, and one less than next space // then modified to fix issues on pg. 57

},

headerRow = headerRow, searchLocs = searchLocs )

}

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 and headerRow

spacerRow = file[eqIndex]

headerRow = tolower(file[ eqIndex - 1 ])

body = file[ -(1 : eqIndex) ]

# Remove footnotes and blank rows // this is added per cleaning on pages 58-59

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

invisible(Values) # change the print mode to invisble

}

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 = 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)

}

menResMat = sapply(mensTables, extractVariables)

womenResMat = sapply(womensTables, extractVariables)

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

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

cbMen = do.call(rbind, menDF)

cbWomen = do.call(rbind, womenDF)

#Plots!

pdf("CB\_NumRunnersLinePlot\_MenVWomen.pdf", width = 8, height = 6)

oldPar = par(mar = c(4.1, 4.1, 1, 1))

numRunners\_m= with(cbMen, tapply(runTime, year, length))

numRunners\_w = with(cbWomen, tapply(runTime, year, length))

plot(names(numRunners\_m),numRunners\_w, type="l",col="red", xlab = "Years", ylab = "Number of Runners")

lines(names(numRunners\_m),numRunners\_m, col="green")

legend("topleft", inset=.05, title="Gender",c("W","M"), fill=c("red","green"))

par(oldPar)

dev.off()

cbMenSub = cbMen[cbMen$runTime > 30 &

!is.na(cbMen$age) & cbMen$age > 15, ]

cbWomenSub = cbWomen[cbWomen$runTime > 30 &

!is.na(cbWomen$age) & cbWomen$age > 15, ]

ageCatm = cut(cbMenSub$age, breaks = c(seq(15, 75, 10), 90))

ageCatw = cut(cbWomenSub$age, breaks = c(seq(15, 75, 10), 90))

table(ageCatm)

table(ageCatw)

pdf("CB\_Boxplots\_Men.pdf", width = 8, height = 6)

oldPar = par(mar = c(4.1, 4.1, 1, 1))

plot(cbMenSub$runTime ~ ageCatm,

xlab = "Age (years)", ylab = "Run Time (minutes)")

par(oldPar)

dev.off()

pdf("CB\_Boxplots\_Women.pdf", width = 8, height = 6)

oldPar = par(mar = c(4.1, 4.1, 1, 1))

plot(cbWomenSub$runTime ~ ageCatw,

xlab = "Age (years)", ylab = "Run Time (minutes)")

par(oldPar)

dev.off()

lmAge\_m = lm(runTime ~ age, data = cbMenSub)

lmAge\_m$coefficients

summary(lmAge\_m)

confint(lmAge\_m, level=0.95)

lmAge\_w = lm(runTime ~ age, data = cbWomenSub)

lmAge\_w$coefficients

summary(lmAge\_w)

confint(lmAge\_w, level=0.95)

pdf("CB\_ResidSimpleLM\_Men.pdf", width = 8, height = 6)

oldPar = par(mar = c(4.1, 4.1, 1, 1))

smoothScatter(x = cbMenSub$age, y = lmAge\_m$residuals,

xlab = "Age (years)", ylab = "Residuals")

abline(h = 0, col = "purple", lwd = 3)

resid.lo = loess(resids ~ age,

data = data.frame(resids = residuals(lmAge\_m),

age = cbMenSub$age))

age20to80 = 20:80

resid.lo.pr =

predict(resid.lo, newdata = data.frame(age = age20to80))

lines(x = age20to80, y = resid.lo.pr, col = "green", lwd = 2)

par(oldPar)

dev.off()

pdf("CB\_ResidSimpleLM\_Women.pdf", width = 8, height = 6)

oldPar = par(mar = c(4.1, 4.1, 1, 1))

smoothScatter(x = cbWomenSub$age, y = lmAge\_w$residuals,

xlab = "Age (years)", ylab = "Residuals")

abline(h = 0, col = "purple", lwd = 3)

resid.lo = loess(resids ~ age,

data = data.frame(resids = residuals(lmAge\_w),

age = cbWomenSub$age))

resid.lo.pr =

predict(resid.lo, newdata = data.frame(age = age20to80))

lines(x = age20to80, y = resid.lo.pr, col = "green", lwd = 2)

par(oldPar)

dev.off()

menRes.lo = loess(runTime ~ age, cbMenSub)

menRes.lo.pr = predict(menRes.lo, data.frame(age = age20to80))

over50\_m = pmax(0, cbMenSub$age - 50)

lmOver50\_m = lm(runTime ~ age + over50\_m, data = cbMenSub)

summary(lmOver50\_m)

womenRes.lo = loess(runTime ~ age, cbWomenSub)

womenRes.lo.pr = predict(womenRes.lo, data.frame(age = age20to80))

over50\_w = pmax(0, cbWomenSub$age - 50)

lmOver50\_w = lm(runTime ~ age + over50\_w, data = cbWomenSub)

summary(lmOver50\_w)

decades = seq(30, 60, by = 10)

overAge\_m = lapply(decades,

function(x) pmax(0, (cbMenSub$age - x)))

names(overAge\_m) = paste("over", decades, sep = "")

overAge\_m = as.data.frame(overAge\_m)

overAge\_w = lapply(decades,

function(x) pmax(0, (cbWomenSub$age - x)))

names(overAge\_w) = paste("over", decades, sep = "")

overAge\_w = as.data.frame(overAge\_w)

lmPiecewise\_m = lm(runTime ~ . ,

data = cbind(cbMenSub[, c("runTime", "age")],

overAge\_m))

summary(lmPiecewise\_m)

lmPiecewise\_w = lm(runTime ~ . ,

data = cbind(cbWomenSub[, c("runTime", "age")],

overAge\_w))

summary(lmPiecewise\_w)

overAge20 = lapply(decades, function(x) pmax(0, (age20to80 - x)))

names(overAge20) = paste("over", decades, sep = "")

overAgeDF = cbind(age = data.frame(age = age20to80), overAge20)

predPiecewise\_m = predict(lmPiecewise\_m, overAgeDF)

predPiecewise\_w = predict(lmPiecewise\_w, overAgeDF)

pdf("CB\_PiecewiseLoessCurves\_Men.pdf", width = 8, height = 6)

plot(predPiecewise\_m ~ 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")

dev.off()

pdf("CB\_PiecewiseLoessCurves\_Women.pdf", width = 8, height = 6)

plot(predPiecewise\_w ~ age20to80,

type = "l", col = "#984ea3", lwd = 3,

# type = "l", col = "purple", lwd = 2,

xlab = "Age (years)", ylab = "Run Time Prediction")

lines(x = age20to80, y = womenRes.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")

dev.off()