## **Introduction**

## The Cherry Blossom running race started in 1973 as a training run for elite runners who were planning to compete in the Boston Marathon. It has since grown in popularity and in 2012 nearly 17,000 runners ranging in age from 9 to 89 participated. The race has become so popular that entrants are chosen via a lottery or they guarantee a spot by raising $500 for an official race charity. After each year’s race, the organizers publish the results at cherry blossom website [1].

## These data offer a tremendous resource for learning about the relationship between age and performance, and Nolan and Lang are interested in understanding how people’s physical performance changes as they age using this data. This is freely accessible data and it may provide us with insights to our question about performance and age. They used the Cherry Blossom Ten Mile Men and Women Run held in Washington D.C. in early April when the cherry trees are typically in bloom during 1999 and 2012. The publicly available race results from the Cherry Blossom Ten Mile Run can be scraped from the Web and read into R for analysis. But the format of the data is not consistent across all the years. The task of scraping the Web site and formatting the results in a way that can be analyzed in R is a bit challenging because the information reported and the format of this information changes from year to year. Some simple differences in format occur in the table header and the use of footnotes. The tables also include many mistakes, e.g., values that begin in the wrong column, missing headers, and so on. All in all, the acquisition of the data is quite straightforward, but it is an iterative process as we uncover several small errors. This is the story of a “messy” data. The data needs to be cleansed before Nolan and Lang can find a correlation between age and physical performance.

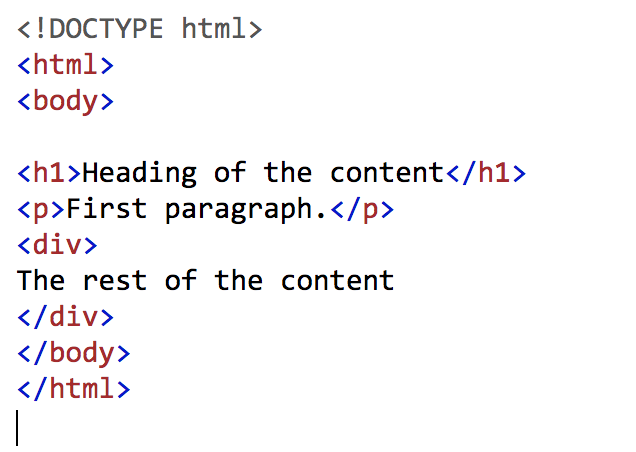
## We would like to read the files for the female runners and then process them using the functions in Section 2.3 to create a data frame for further analysis. As part of our work, we might need to generalize the createDF() and extractVariables functions to handle additional oddities in the raw text files. These operations need extensive investigate of Nolan and Temple work and data investigation.

## **Background**

Nolan and Temple have analyzed the Cherry Blossom running race using different technologies in the R code. Nolan and Lang suggest downloading the men and women race results between 1999 and 2012 from Cherry Blossom website. They extensively use Regular Expressions, XML package in R to address different cleansing needs.

## **Exploratory Data Analysis**

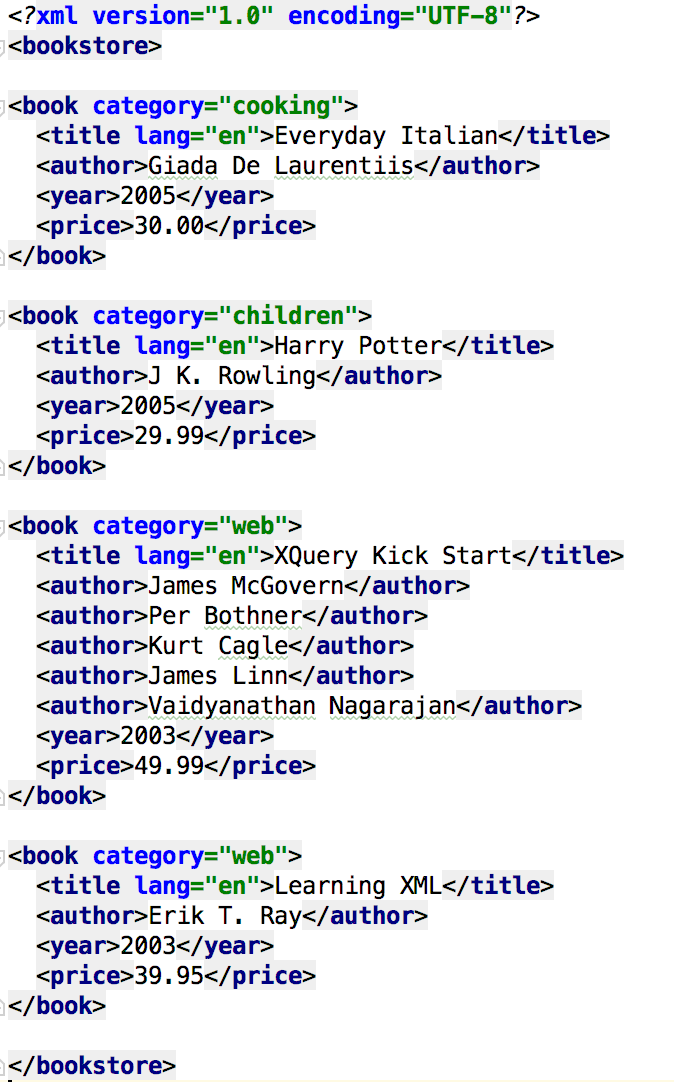
Nolan and Lang suggest downloading the men and women race results between 1999 and 2012 from Cherry Blossom website. The data is in an inconsistent HTML format. Following image depicts a typical skeleton HTML file[2]. The race data can be defined inside any internal structure in the body tag. That is why the data needs to be investigated year by year and a certain technology needs to be used to query the data. Nolan and Lang suggest using XML library in R and Xpath query language. Following section briefly explains these technologies to have a deeper understanding of how data cleansing should be conducted.



**Figure 1: HDML Content Sample**

## **XML and XPath**

In order to cleanse the data, Nolan and Lang suggest to use XML library in R. Behind the scene, this library transforms an html page into a DOM object(DOM is a cross-platform and language-independent application programming interface that treats an XML document as a tree structure wherein each node is an object representing a part of the document) so that we can query the content using Xpath language. XPath uses path expressions to select nodes or node-sets in an XML document. These path expressions look very much like the expressions you see when you work with a traditional computer file system [3]. For instance, if books of a bookstore are explained using xml, the content might look like the following.



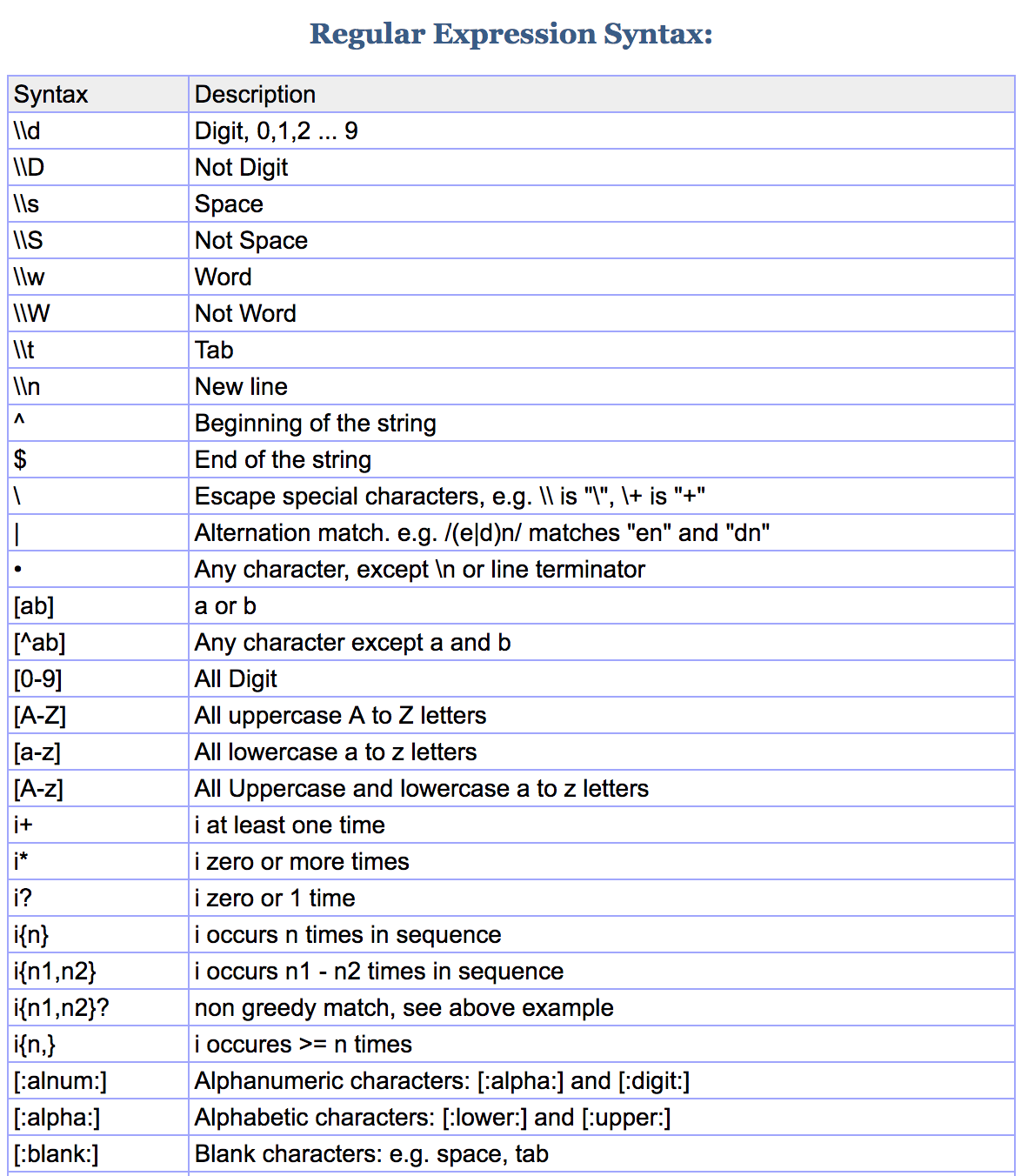
**Figure 2: XML Content Sample**

“/bookstore/book[1]” queries the first book element that is the child of the bookstore element. “title[@lang]” also queries the title elements that have an attribute named lang.

A more comprehensive explanation of Xpath capabilities can be found in 4th edition of Michael Kay’s “XSLT 2.0 and Xpath 2.0 Programmers Reference”[4].

## **Regular Expression**

Regular expression is a sequence of characters that define a search pattern with a standard textual syntax. Nolan and Temple extensively use to find the inconsistencies of different input. The following table can be used to address the Regex needs as we process files.



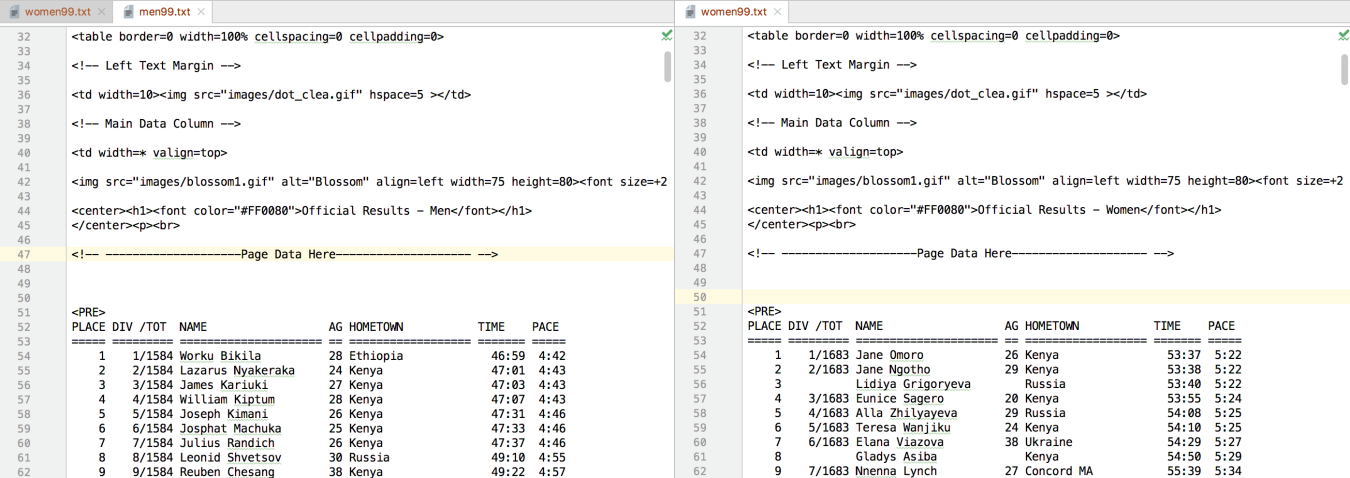
**Figure 3. Regular Expression Syntax**

**Methods**

Not all files are consistent and inconsistency needs to be addressed separately. Nolan and Temple introduce base functionality using single input functions and then expand them using new one that can be applied on an input collection. For that matter, we will initially create two collections for where all input files for different years are located. We introspect the raw data differences. We will then address the issue in the single file input function and then apply it to the input collection of all years. When there is an error, we minimize the years to 1 or 2 and then expand it as we address separate issues. We will also use trycatch blocks and recover debugging capabilities in R to have a better insight when we handle errors.

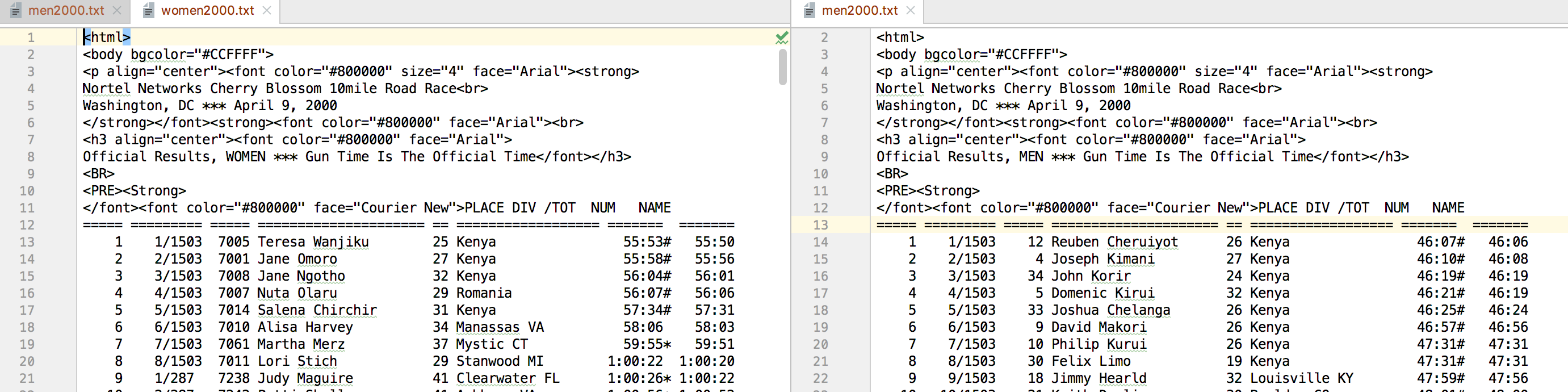
**Input data between 1999 and 2012**

When we investigated the year 1999 data, we observed that the content of it is inside the pre tag.



**Figure 4. Year 1999 Input Data for Male and Female**

Year 2000 is a bit different. The data is located after strong and invalid font tags.



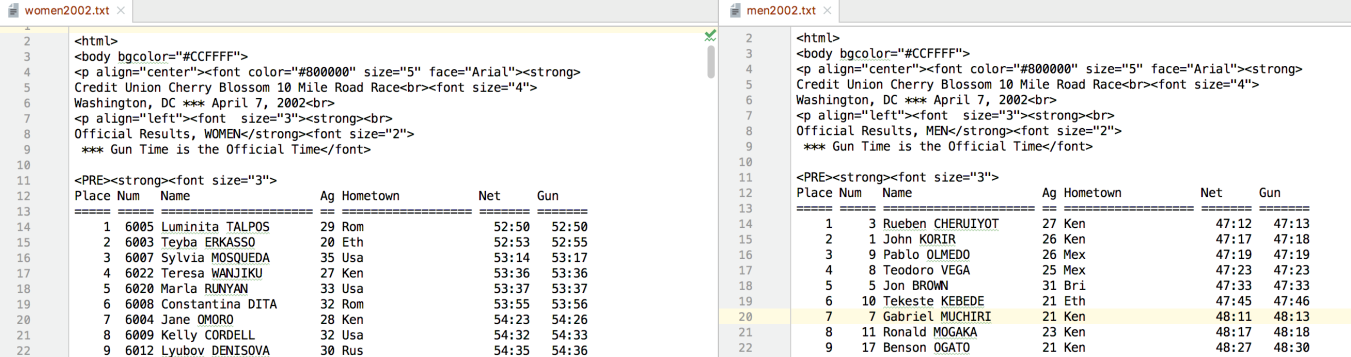
**Figure 5. Year 2000 Input Data for Male and Female**

Year 2001 files hold the data a few lines after the strong and font tags. We also observe an interesting difference. Female data does not have a header.



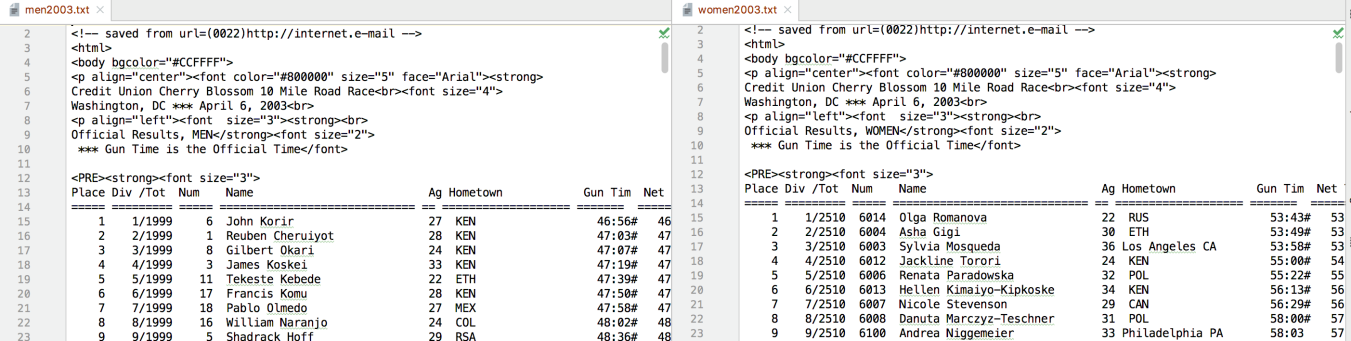
**Figure 6. Year 2001 Input Data for Male and Female**

Year 2002 holds the data inside the pre, strong and font tags but it is not a few lines away.



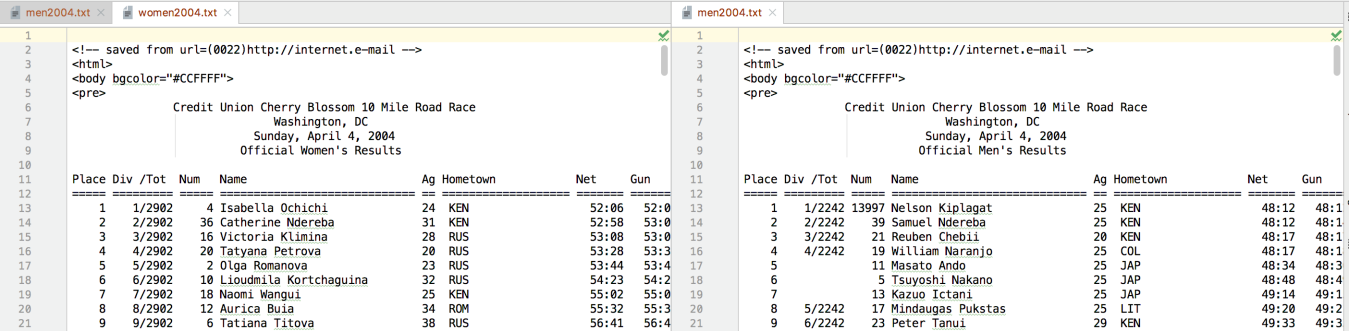
**Figure 7. Year 2002 Input Data for Male and Female**

Year 2003 is similar to the previous year.



**Figure 8. Year 2003 Input Data for Male and Female**

Year 2004 has some additional lines in the pre tag. There are no strong or font tags in the beginning of the data.



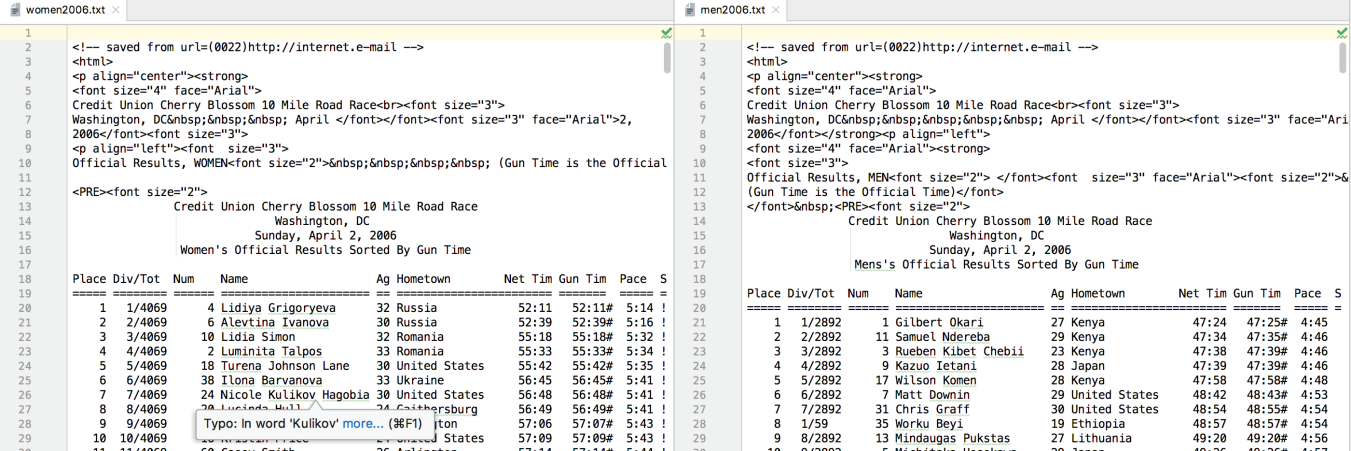
**Figure 9. Year 2004 Input Data for Male and Female**

Year 2005 is similar to 2004 but the data is defined inside pre and font tags.



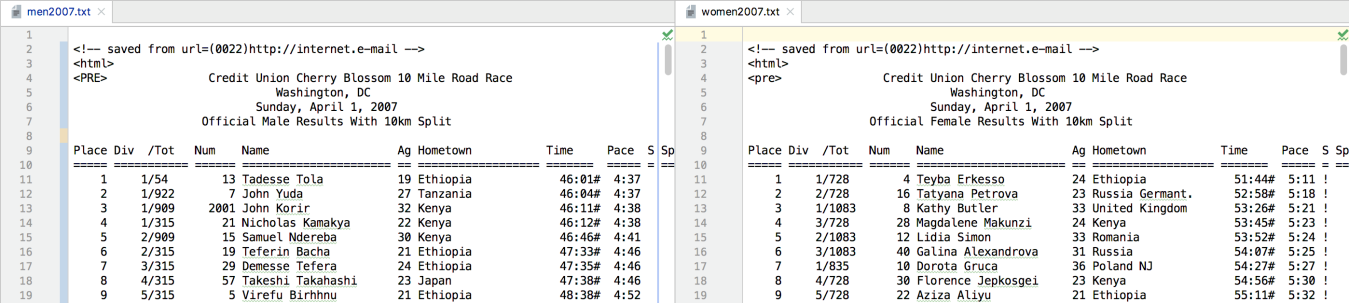
**Figure 10. Year 2005 Input Data for Male and Female**

Year 2006 has only font tag and a few lines of text before the content.



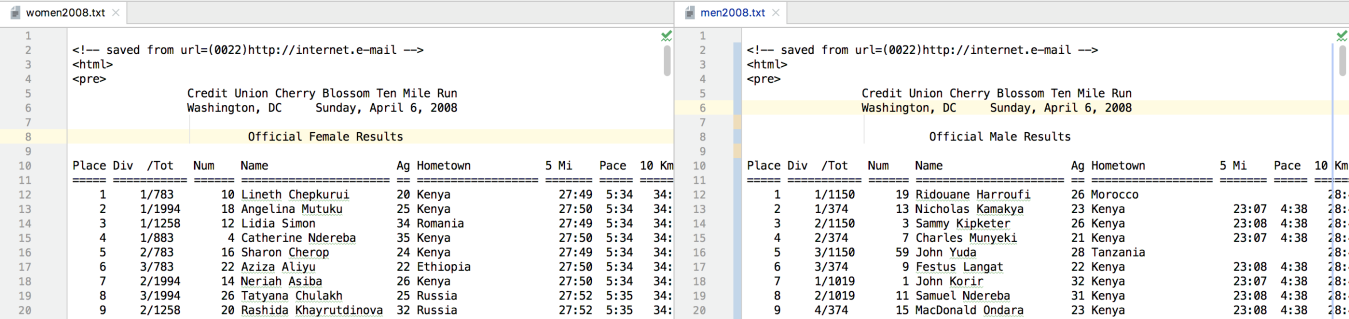
**Figure 11. Year 2006 Input Data for Male and Female**

Year 2007 has a few lines of text before the content but there are no tags in the beginning of the pre tag.



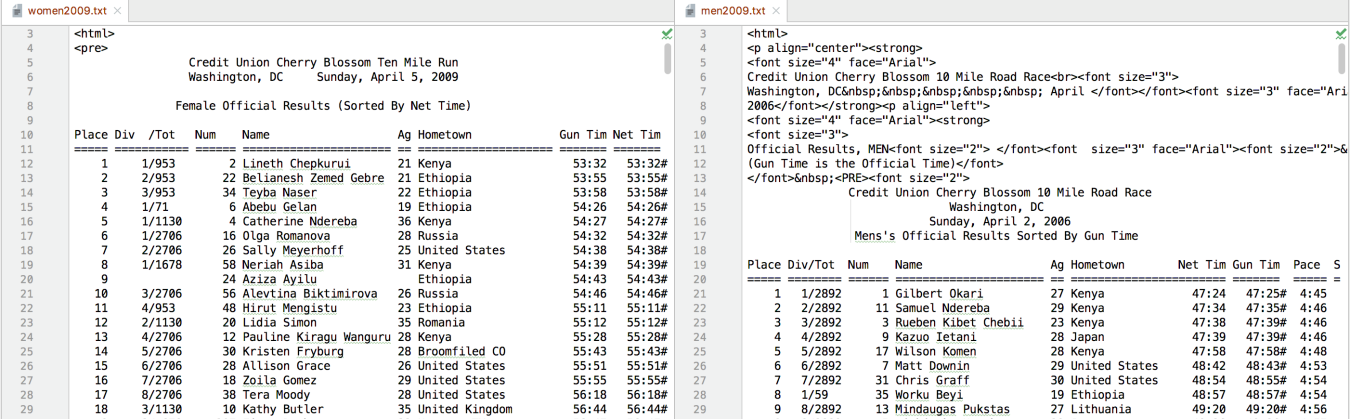
**Figure 12. Year 2007 Input Data for Male and Female**

Year 2008 has no additional tags in the beginning of the pre tag. It has a few lines which might be unique to its year.



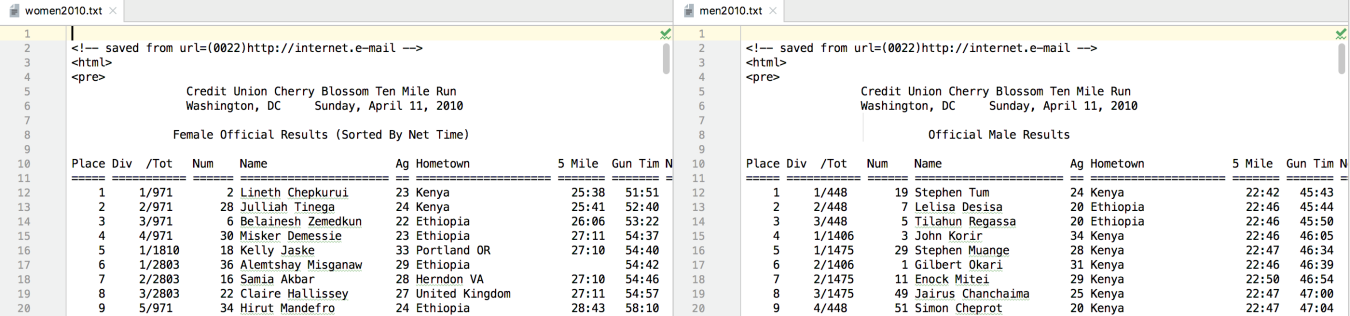
**Figure 13. Year 2008 Input Data for Male and Female**

Year 2009 has a few lines after pre and font tag for men but the data is under pre tag for women.



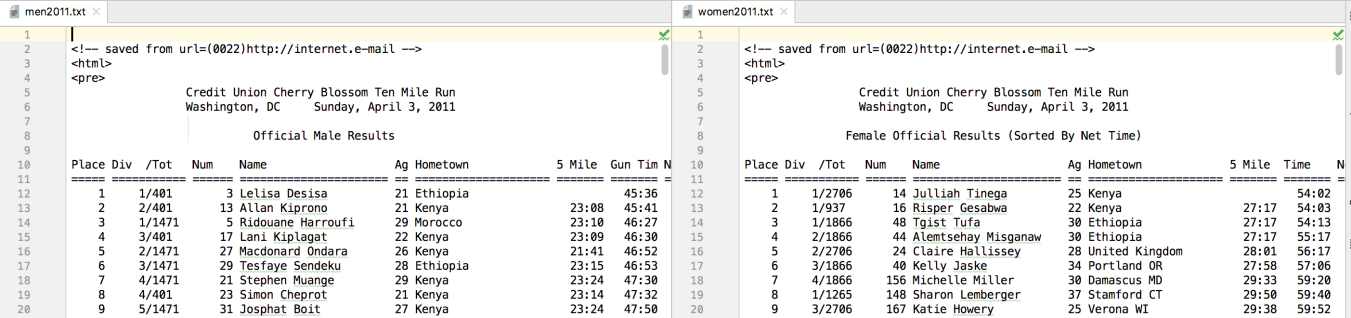
**Figure 14. Year 2009 Input Data for Male and Female**

Year 2010 has no additional tags in the beginning of the pre tag. The content starts after a few lines of text.



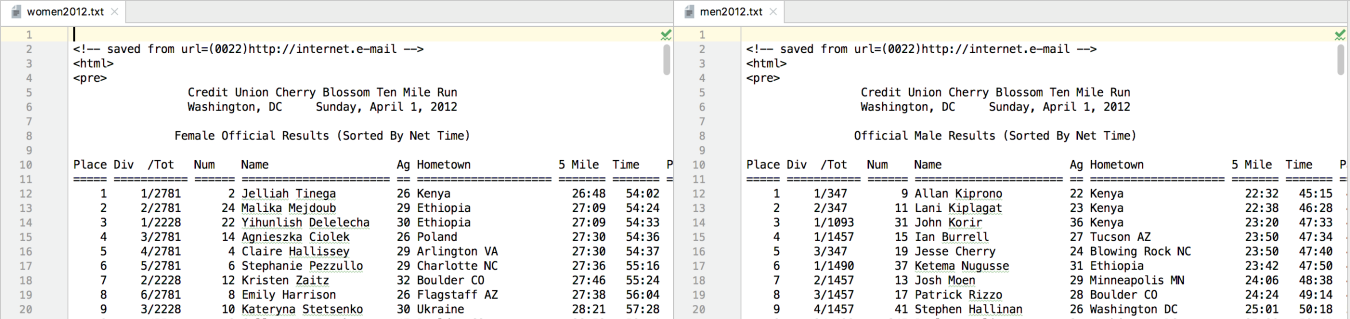
**Figure 15. Year 2010 Input Data for Male and Female**

Year 2011 is similar to the previous year. Number of lines of text is the same in the pre tag before the content starts but the content is different for men and women.



**Figure 16. Year 2011 Input Data for Male and Female**

Year 2012 is similar to the previous year.



**Figure 17. Year 2012 Input Data for Male and Female**

Considering all the differences among the tags and lines of content, xpath query language that is available through the XML package in R will be very useful to fix our problem.

**Downloading the input**

The above issues should be handled case by case. We addressed them by try and error (and by modifying options to introspect the inner functions) on Nolan and Lang single year code and then extended the extractResTable function. We used xpath extensively to address common issues among all files. For instance, “//pre” and “//font” xpath queries will help us find the specific tag in the document. All we need to do is to address different years separately as following.



**Figure 18. ExtractResTable Function Addressing Input Differences**

**Extracting Columns**

If you go back to the Input data section and check the positioning of different columns, you will observe that there is still a long way to go before we can transform the data into data frame. Nolan and Temple have introduced functions like “selectCols” and “findColLocs” that enable us to generically read columns. The different data in most of the years has a key line (“===== =========== ====== ====================== == ==================== ======= ======= ===== =”) that can help us locate the start of each column. The function “findColLocs” will enable the “selectCols” to load the columns in the data. “name”, “Age”, “hometown”, “gun”, “net”, and “time” columns are considered to be loaded.

**Age Column**

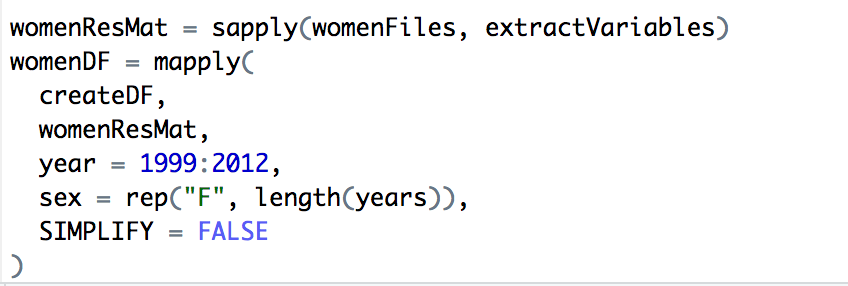
After some data cleansing for 2009 data, the new revision of “extractVariables” will use the new functionality to load the columns data. It loads the downloaded files into separate lists for men and women. We should prepare some functionality to so that the “Age” column can be casted into numeric type. Missing age data will also be addressed by the new revision of “extractVariable” functionality.

**Time Column**

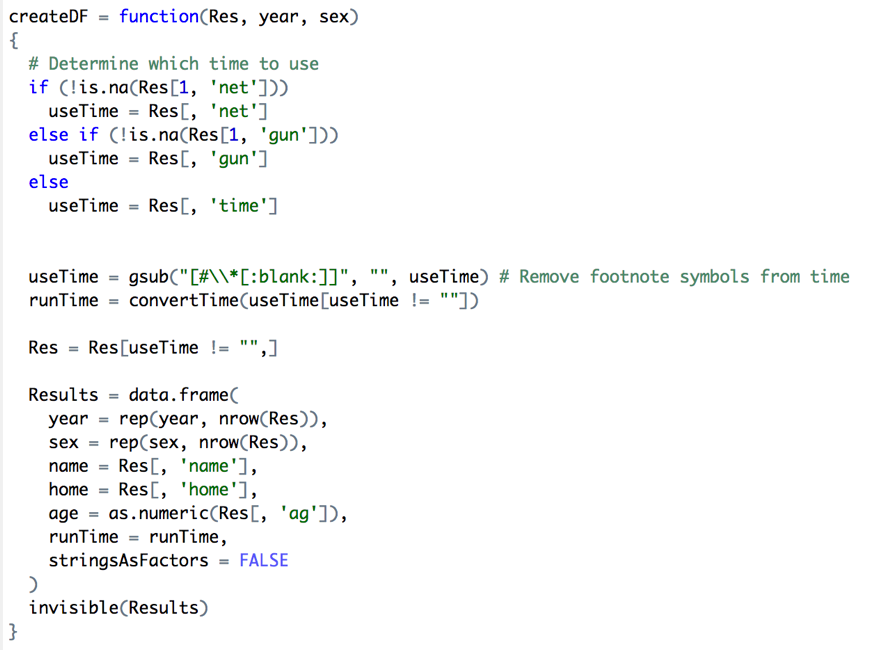
Time columns need specific attention. After initial casting to numeric type, we should split minute and second into time pieces (e.g. “54:36” will be transformed into c(54, 36)). Then we should convert the time into a single unit. We prepare this functionality to be used during data frame conversion.

**Result**

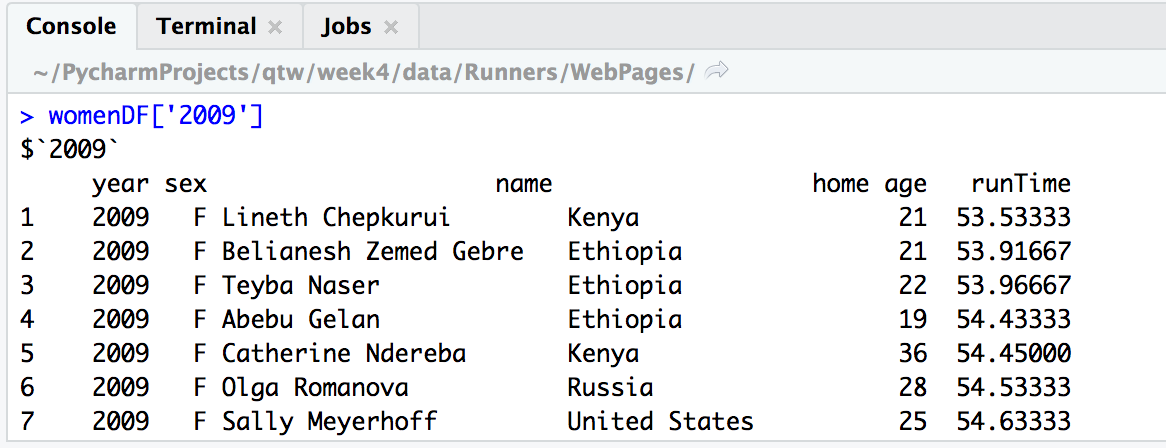
At this point, “womenResMat” is a list that contains the raw information of all rows of all years in 14 separate objects. The conversion functionality of the mentioned columns along with the suggested regular expression to remove footnotes in section 2.3 will be used to transform the data into data frame as following. We will use one of the ‘net’, ‘gun’ and ‘time’ (when they are not null) to address missing data problem. As they are all of type time, the conversion works for all of them.



**Figure 19. Women Dataframe Creation by Applying CreateDF function on womenResMat Data**



**Figure 20. CreateDF Function**



**Figure 21. Women Dataframe Result on R Console for Year 2009**

**Discussion**

It takes a lot of effort to cleanse inconsistent html data. Our solution works for this particular problem, but it does not necessarily scale for larger problems with more than certain number of html content. In such cases, importing the input data might become a different project at the initial stage. But our exercise is finished at this point, and the data is ready for further processing.

**Future Work**

During the conversion phase, ‘net’, ‘gun’ and ‘time’ columns can be investigated further. Null values can be replaced with default average value.

Age distribution of the runners across all 14 years of the races can be compared for both men and women. Quantile–quantile plots, boxplots, and density curves can be used to support comparisons. The change of distribution over the years can be analyzed to clarify whether this was a gradual change or not.

## **References**

1. <http://www.rdatasciencecases.org/exercises.html#chap:CherryBlossom>
2. <https://www.amazon.com/Head-First-HTML-CSS-Standards-Based/dp/0596159900/ref=sr_1_9?keywords=HTML+book&qid=1559366177&s=gateway&sr=8-9>
3. <https://www.w3schools.com/xml/xml_xpath.asp>
4. <https://www.amazon.com/XSLT-2-0-XPath-Programmers-Reference/dp/0470192747/ref=sr_1_2?crid=32B3U4XIK0WM7&keywords=xpath&qid=1559364893&s=gateway&sprefix=Xpath%2Caps%2C222&sr=8-2>
5. Nolan, D. and Lang, D. T. “Data Science in R.” CRC Press, 2015 (Chapter 2).
6. <https://www.amazon.com/Mastering-Regular-Expressions-Jeffrey-Friedl/dp/0596528124/ref=sr_1_1?crid=1F45R06QUYNY5&keywords=mastering+regular+expressions&qid=1559504602&s=gateway&sprefix=mastering+regular%2Caps%2C195&sr=8-1>
7. <https://www.regular-expressions.info/>
8. <https://www.amazon.com/Functional-Programming-Advanced-Statistical-Analysis/dp/148422745X>

## **Appendix - R Code**

### Case Study

# install.packages("XML")

library(XML)

shons\_path <- ''

maryams\_path <- ''

ramins\_path <- '~/PycharmProjects/qtw/week4/data/Runners/WebPages/'

current\_path <- ramins\_path

setwd(current\_path)

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

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"

)

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"

)

maleUrls = paste(ubase, menURLs, sep = "")

femaleUrls = paste(ubase, womenURLs, sep = "")

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)

if (year == 1999) {

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

txt = xmlValue(pres[[1]])

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

}

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)

# Write the lines as a text file.

writeLines(els, con = file)

}

years = 1999:2012

menTables = mapply(extractResTable,

url = maleUrls,

year = years,

sex = "male")

womenTables = mapply(extractResTable,

url = femaleUrls,

year = years,

sex = "female")

names(menTables) = years

names(womenTables) = years

invisible(sapply(menTables, length))

invisible(sapply(womenTables, length))

save(menTables, file = "CBMenTextTables.rda")

save(womenTables, file = "CBWomenTextTables.rda")

womenTables$'2001'[2:3] <- womenTables$'2002'[2:3]

sapply(c("MenTxt", "WomenTxt"), function(dir)

{

dir.create(file.path(getwd(), dir))

})

write(x = menTables$'2011', file = "MenTxt/2011.txt")

write(x = menTables$'2012', file = "MenTxt/2012.txt")

write(x = menTables$'2010', file = "MenTxt/2010.txt")

write(x = menTables$'2009', file = "MenTxt/2009.txt")

write(x = menTables$'2008', file = "MenTxt/2008.txt")

write(x = menTables$'2007', file = "MenTxt/2007.txt")

write(x = menTables$'2006', file = "MenTxt/2006.txt")

write(x = menTables$'2005', file = "MenTxt/2005.txt")

write(x = menTables$'2004', file = "MenTxt/2004.txt")

write(x = menTables$'2003', file = "MenTxt/2003.txt")

write(x = menTables$'2002', file = "MenTxt/2002.txt")

write(x = menTables$'2001', file = "MenTxt/2001.txt")

write(x = menTables$'2000', file = "MenTxt/2000.txt")

write(x = menTables$'1999', file = "MenTxt/1999.txt")

write(x = womenTables$'2011', file = "WomenTxt/2011.txt")

write(x = womenTables$'2012', file = "WomenTxt/2012.txt")

write(x = womenTables$'2010', file = "WomenTxt/2010.txt")

write(x = womenTables$'2009', file = "WomenTxt/2009.txt")

write(x = womenTables$'2008', file = "WomenTxt/2008.txt")

write(x = womenTables$'2007', file = "WomenTxt/2007.txt")

write(x = womenTables$'2006', file = "WomenTxt/2006.txt")

write(x = womenTables$'2005', file = "WomenTxt/2005.txt")

write(x = womenTables$'2004', file = "WomenTxt/2004.txt")

write(x = womenTables$'2003', file = "WomenTxt/2003.txt")

write(x = womenTables$'2002', file = "WomenTxt/2002.txt")

write(x = womenTables$'2001', file = "WomenTxt/2001.txt")

write(x = womenTables$'2000', file = "WomenTxt/2000.txt")

write(x = womenTables$'1999', file = "WomenTxt/1999.txt")

# 2012 example race logs

els = readLines("WomenTxt/2012.txt")

# Identify line index for header-data break

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

eqIndex

first3 = substr(els, 1, 3)

which(first3 == "===")

# Ignore rows above header

spacerRow = els[eqIndex]

headerRow = els[eqIndex - 1]

body = els[-(1:eqIndex)]

# Extract runners' age

headerRow = tolower(headerRow)

ageStart = regexpr("ag", headerRow)

ageStart

age = substr(body, start = ageStart, stop = ageStart + 1)

head(age)

summary(as.numeric(age))

blankLocs = gregexpr(" ", spacerRow)

blankLocs

searchLocs = c(0, blankLocs[[1]])

Values = mapply(substr, list(body),

start = searchLocs[-length(searchLocs)] + 1,

stop = searchLocs[-1] - 1)

#Find locations of all blanks in line of equal characters and extract columns

findColLocs = function(spacerRow) {

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

rowLength = nchar(spacerRow)

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

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

else

return(c(0, spaceLocs))

}

# Extract columns

selectCols = function(colNames, headerRow, searchLocs)

{

sapply(colNames,

function(name, headerRow, searchLocs)

{

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

if (startPos == -1)

return(c(NA, NA))

index = sum(startPos >= searchLocs)

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

},

headerRow = headerRow, searchLocs = searchLocs)

}

# Testing findColLocs and selectCols functions

searchLocs = findColLocs(spacerRow)

ageLoc = selectCols("ag", headerRow, searchLocs)

ages = mapply(substr, list(body), start = ageLoc[1, ], stop = ageLoc[2,])

summary(as.numeric(ages))

# Create shortened column identifiers and account for when some tables missing columns

shortColNames = c("name", "home", "ag", "gun", "net", "time")

locCols = selectCols(shortColNames, headerRow, searchLocs)

Values = mapply(substr, list(body), start = locCols[1,], stop = locCols[2,])

colnames(Values) = shortColNames

head(Values)

tail(Values)[, 1:3]

# Build wrapper function for column extraction

extractVariables = function(file, varNames = c("name", "home", "ag", "gun", "net", "time"))

{

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

# Extract the two key rows and the real data in the xml

spacerRow = file[eqIndex]

headerRow = tolower(file[eqIndex - 1])

body = file[-(1:eqIndex)]

# Obtain the starting and ending positions of variables

searchLocs = findColLocs(spacerRow)

locCols = selectCols(varNames, headerRow, searchLocs)

Values = mapply(substr, list(body), start = locCols[1,], stop = locCols[2,])

colnames(Values) = varNames

invisible(Values)

}

# Read table lines into R

mfilenames = paste("MenTxt/", 1999:2012, ".txt", sep = "")

menFiles = lapply(mfilenames, readLines)

names(menFiles) = 1999:2012

menFiles[['2009']] <- gsub("�", "", menFiles[['2009']])

# Create list of character matrices containing the column contents for each of the 14 years of data

menResMat = lapply(menFiles, extractVariables)

sapply(menResMat, nrow)

# Read table lines into R

wfilenames = paste("WomenTxt/", 1999:2012, ".txt", sep = "")

womenFiles = lapply(wfilenames, readLines)

names(womenFiles) = 1999:2012

womenFiles[['2009']] <- gsub("�", "", womenFiles[['2009']])

# Create list of character matrices containing the column contents for each of the 14 years of data

womenResMat = lapply(womenFiles, extractVariables)

sapply(womenResMat, nrow)

maleAge = sapply(menResMat, function(x)

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

femaleAge = sapply(womenResMat, function(x)

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

head(menFiles[['2003']])

menFiles[['2006']][2200:2205]

# Updated selectCols takes care of the offset in age column

selectCols = function(shortColNames, headerRow, searchLocs) {

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

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

if (startPos == -1)

return(c(NA, NA))

index = sum(startPos >= searchLocs)

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

}, headerRow = headerRow, searchLocs = searchLocs)

}

menResMat = lapply(menFiles, extractVariables)

womenResMat = lapply(womenFiles, extractVariables)

maleAge = sapply(menResMat, function(x)

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

femaleAge = sapply(womenResMat, function(x)

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

sapply(maleAge, function(x)

sum(is.na(x)))

sapply(femaleAge, function(x)

sum(is.na(x)))

# Example

maleAge2001 = maleAge[["2001"]]

femaleAge2001 = femaleAge[["2001"]]

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

grep("^===", womenFiles[['2001']])

badAgeIndex = which(is.na(maleAge2001)) + 5

femaleBadAgeIndex = which(is.na(femaleAge2001)) + 5

# new revision of extractVariables handles missing age data

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)

}

menResMat = lapply(menFiles, extractVariables)

womenResMat = lapply(womenFiles, extractVariables)

maleCharTime = menResMat[['2012']][, 'time']

femaleCharTime = womenResMat[['2012']][, 'time']

maleTimePieces = strsplit(maleCharTime, ":")

femaleTimePieces = strsplit(femaleCharTime, ":")

maleTimePieces = sapply(maleTimePieces, as.numeric)

femaleTimePieces = sapply(femaleTimePieces, as.numeric)

maleRunTime = sapply(maleTimePieces,

function(x) {

if (length(x) == 2)

x[1] + x[2] / 60

else

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

})

femaleRunTime = sapply(femaleTimePieces,

function(x) {

if (length(x) == 2)

x[1] + x[2] / 60

else

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

})

# Split and process times

convertTime = function(time) {

maleTimePieces = strsplit(time, ":")

maleTimePieces = sapply(maleTimePieces, as.numeric)

sapply(maleTimePieces, 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

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

Res[, 'net']

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

Res[, 'gun']

else

Res[, 'time']

runTime = convertTime(useTime)

Results = data.frame(

year = rep(year, nrow(Res)),

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

name = Res[, 'name'],

home = Res[, 'home'],

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

runTime = runTime,

stringsAsFactors = FALSE

)

invisible(Results)

}

menDF = mapply(

createDF,

menResMat,

year = 1999:2012,

sex = rep("M", 14),

SIMPLIFY = FALSE

)

sapply(menDF, function(x)

sum(is.na(x$maleRunTime)))

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']

useTime = gsub("[#\\\*[:blank:]]", "", useTime) # Remove # and \* and blanks from time

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)

}

menDF = mapply(

createDF,

menResMat,

year = 1999:2012,

sex = rep("M", 14),

SIMPLIFY = FALSE

)

sapply(menDF, function(x)

sum(is.na(x$runTime)))

# Fix missing runTime data issues for 2006

separatorIdx = grep("^===", menFiles[["2006"]])

separatorRow = menFiles[['2006']][separatorIdx]

separatorRowX = paste(substring(separatorRow, 1, 63),

" ",

substring(separatorRow, 65, nchar(separatorRow)),

sep = "")

menFiles[['2006']][separatorIdx] = separatorRowX

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", length(years)),

SIMPLIFY = FALSE

)