# Modeling Runners' Times in the Cherry Blossom Race

Justin Howard, Stuart Miller, Paul Adams

September 22, 2020

## 1 Introduction

The internet is a vast open resource for data, but internet data can be messy and difficult to wrangle. In this case study we collect data on participants of the Cherry Blossom Race from the event website[1]. We find that there are a number of issues present in the data, including inconsistent web addressing formatting, inconsistent formatting, and missing data. We collect and clean this data to provide the Men's and Women's information in a format suitable for analysis.

## 2 Methods

### 2.1 Data

#### 2.1.1 Data Collection

In this case study, we collected data on participants in the Cherry Blossom Race. We scraped the data for men and women from 1999 to 2012. The data was stored under the base address in the following form, where the year is between 1999 and 2012 and page is a specific page name.

```
http://www.cherryblossom.org/results/<year>/<page>
```

The specific addresses where the data is stored are given in table 1. While the structure follows `results/<year>` consistently, the page naming for each year is not consistent. In some cases, the names for the men's and women's pages are `men` and `women`, respectively. In other cases, a code is used such as `cb99m` and `cb99f` in 1999 for `men` and `women` respectively.

[1]See http://www.cherryblossom.org/.

**Table 1. Web Pages**

| Year | Men's Pages | Women's Pages |
|---|---|---|
| 1999 | `results/1999/cb99m.html` | `results/1999/cb99f.html` |
| 2000 | `results/2000/Cb003m.htm` | `results/2000/Cb003f.htm` |
| 2001 | `results/2001/oof_m.html` | `results/2001/oof_f.html` |
| 2002 | `results/2002/oofm.htm` | `results/2002/ooff.htm` |
| 2003 | `results/2003/CB03-M.HTM` | `results/2003/CB03-F.HTM` |
| 2004 | `results/2004/men.htm` | `results/2004/women.htm` |
| 2005 | `results/2005/CB05-M.htm` | `results/2005/CB05-F.htm` |
| 2006 | `results/2006/men.htm` | `results/2006/women.htm` |
| 2007 | `results/2007/men.htm` | `results/2007/women.ht` |
| 2008 | `results/2008/men.htm` | `results/2008/women.htm` |
| 2009 | `results/2009/09cucb-M.htm` | `results/2009/09cucb-F.htm` |
| 2010 | `results/2010/2010cucb10m-m.htm` | `results/2010/2010cucb10m-F.htm` |
| 2011 | `results/2011/2011cucb10m-m.htm` | `results/2011/2011cucb10m-F.htm` |
| 2012 | `results/2012/2012cucb10m-m.htm` | `results/2012/2012cucb10m-F.htm` |

### 2.1.2 Data Cleaning

*This section refers to code in Appendix A*

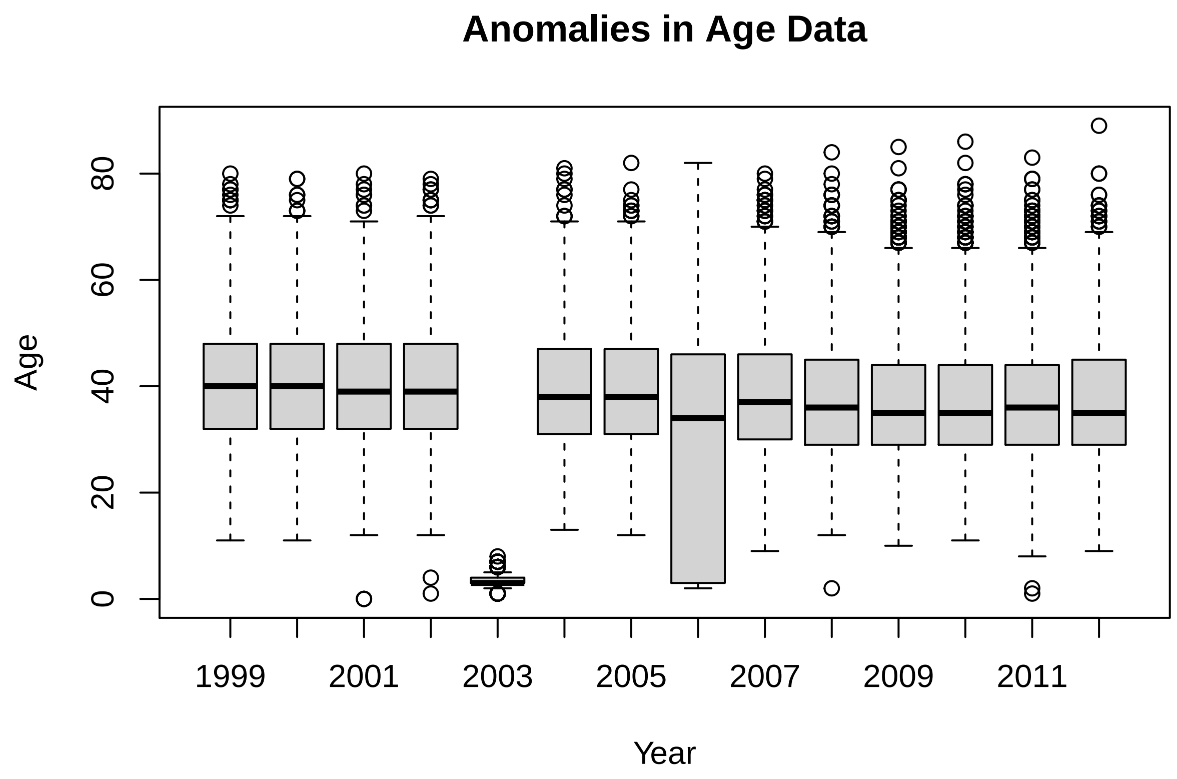
The data are successfully scraped from the urls that are given and we observe that the number of participants in the Cherry Blossom race has increased steadily from 1999 to 2012. We can use the `extractResTable` function to exptract all of the results, or to specify a single set of results.

We observe the formatting of the text data, such as the use of the `=` character can be used to transform the lines of text into a a matrix. We will used this pattern to identify the boundary between the headering and the body of the data. Once we defined the column names, we were free to use the text structure, such as the spacing of column names and data, to identify datapoints. We started with age, which denoted by the `"ag"` identifier. After we sucessfully used the structure of the text data to identify an index location for `"ag"` data points. We expanded the search to inlcude the spacing of text data to identify other data points. The spacing indices were used in a funtion that compiled a matrix of data points across the entire list of tables.

A function `findColLocs` was defined to use these space indices to locate the begin points for data throughout the extracted text data. `selectCols` is a tool that uses `findColLocs` to locate the data points for each column throughout the text for all of the race years. Our functions extracted almost all of the age data fomr the text, which aided in the identification errors within the dataset. To generalize this method, we defined a function, `extractVariables`, that combined the various processes we used to preprocess the data into a more machine readable format.

The execution of the helper functions helped transform the original text data into a much more machine readable format. To perform statistical analyses on the numeric data, we performed datatype transformations, converting relevant data from a character format into a numeric format.

This transformation revealed anomalies in the data for the year 2003. After an examination of the original data table, we found that the spacing in the year 2003 data is not the same as the other years. We adjusted our `selectCols` function to better capture the data for 2003. We also observe anomalies in the formatting of the year 2001. We introduced fixes and re-scraped the data once more to form the list of matrices called `menResMat`.



After these anomalies were corrected, we reformatted the datatypes. The `"ag"` data was changed to a numeric format for analysis, which revealed *more* anomalies. We observed suspiciously young ages between the years 2001 and 2003. These ages were revealed to be *erroneous* upon closer examination. These values were removed from the dataset.

To make the remaining analyses easier, we transformed the data structure from a list of matrices into a dataframe. Combining the data into a single dataframe facilitated the cleaning of the columns containing the time data for each runner. These values were stored as characters and were in an `HR:MIN:SEC` format. We can separate the values and convert them all into a standard time format, minutes. We defined a function that converts the time into a minute format. The end result was a dataframe called `menDF`. With this process complete, we simply re-used the functions on the women's data.

In preparation for further analysis, the values for 'home' needed to be updated. First, 12 runners were identified as having at least one iteration of a bad address. Based on state codes provided with those addresses, alternative addresses used by those runners in other races were applied in place of the bad addresses.

Additionally, because most of the addresses were not in standard format, we converted home names to ISO standards for national naming. For example, we converted 'Rep Of S.africa' to the iso3166-recognized 'South

Africa'. The names were originally given ISO Name values for analysis and reporting, then converted to ISO 3-byte country codes for cross-walking into the map's visual abstraction. All runners with specified nations (whether by ISO code or name) were listed with their nation while all others were listed as 'United States.' For cross-walking locations to states, we identified 7,398 records where a state was given, ending in value such as ' AK',',AK', where 'AK' is 'Alaska'. This information was processed in creating Fig. 1 and Fig. 2.

Our dataset is complete, clean, and ready for further analysis.

```
##   year sex                   name                home age  runTime
## 1 1999   M Worku Bikila          Ethiopia              28 46.98333
## 2 1999   M Lazarus Nyakeraka      Kenya                 24 47.01667
## 3 1999   M James Kariuki         Kenya                 27 47.05000
## 4 1999   M William Kiptum        Kenya                 28 47.11667
## 5 1999   M Joseph Kimani         Kenya                 26 47.51667
## 6 1999   M Josphat Machuka       Kenya                 25 47.55000
```

## 3 Results

### 3.1 Race Results by Geographic Location

As visualized in Fig. 1, Results from analyzing geographic runner information identified that the nations with the fastest run times, represented by average, were from Tanzania, Morocco, Ethiopia, Kenya and Mexico. The nations with the slowest average run times were Ireland, Slovenia, Lebanon, Republic of Korea, and Jordan.

| Top 5 Fastest Nations | Overall Race Completion Time |
|---|---|
| Tanzania | 46.067 |
| Morocco | 46.817 |
| Ethiopia | 46.983 |
| Kenya | 47.017 |
| Mexico | 47.317 |

| Top 5 Slowest Nations | Overall Race Completion Time |
|---|---|
| Ireland | 121.42 |
| Slovenia | 108.25 |
| Lebanon | 103.95 |
| Republic of Korea | 100.733 |
| Jordan | 100.067 |

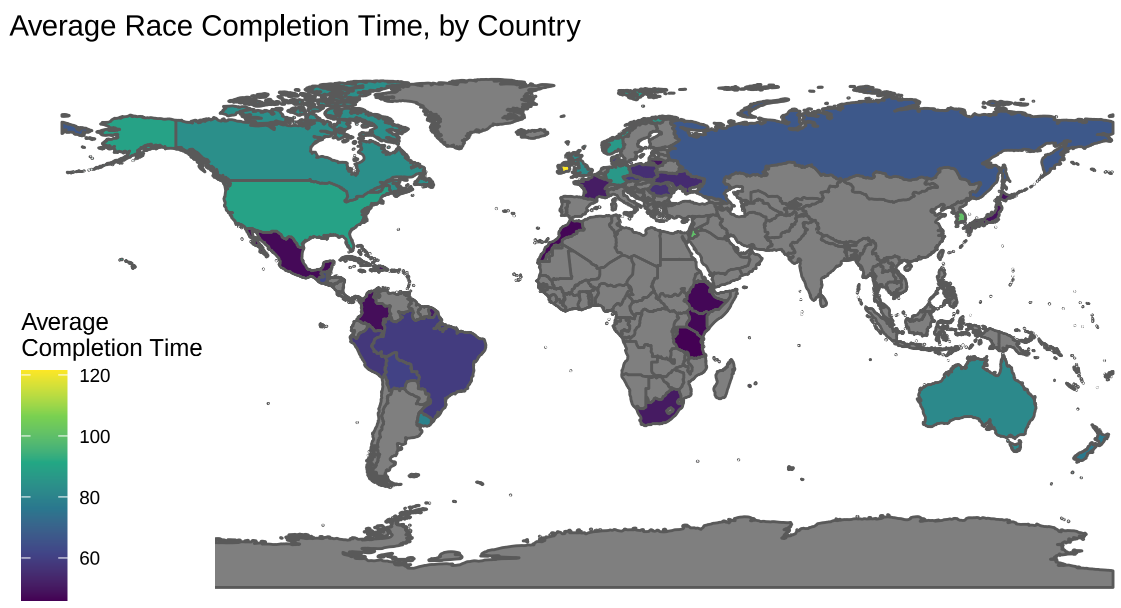
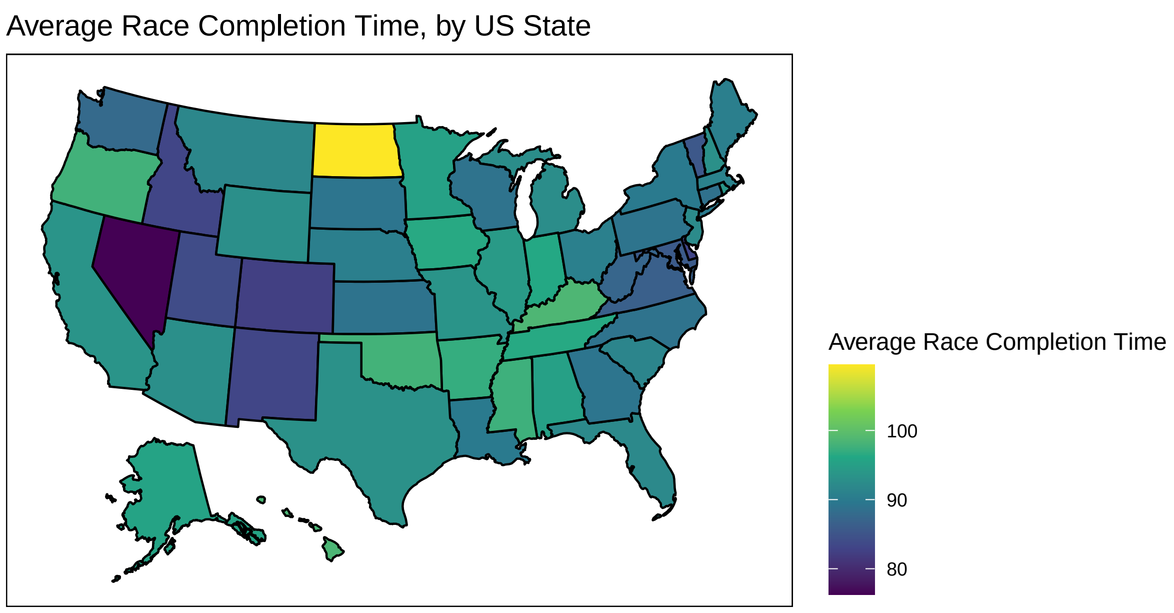


Figure 1:
Global Race Completion Times

As visualized in Fig. 2, the states represented by the fastest overall runners - based on average race completion time - are Nevada, Colorado, Delaware, Idaho, and New Mexico. The states with the slowest average race completion times are North Dakota, Kentucky, Hawaii, Oklahoma, and Oregon.

| Top 5 Fastest States | Overall Race Completion Time |
|---|---|
| Nevada | 76.269 |
| Colorado | 82.400 |
| Delaware | 83.019 |
| Idaho | 83.157 |
| New Mexico | 83.208 |

| Top 5 Slowest States | Overall Race Completion Time |
|---|---|
| North Dakota | 109.517 |
| Kentucky | 98.602 |
| Hawaii | 98.467 |
| Oklahoma | 97.837 |
| Oregon | 97.760 |


Average Race Completion Time, by US State
Average Race Completion Time
100
90
80
Figure 2:
Statewide Race Completion Times

## 3.2 Trends over time

The Cherry Blossom race has experienced growth since the year 2000 as indicated by the linear trend line in Fig. 3, below.

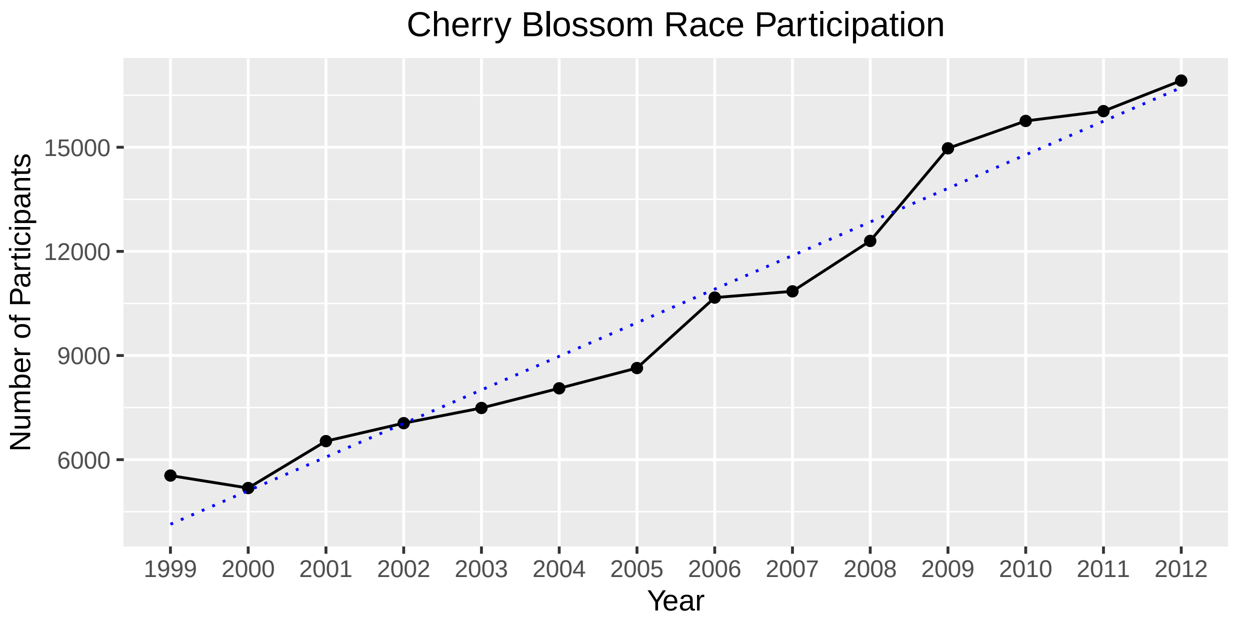


Figure 3: Cherry Blossom race participation with linear trend line

The average race completion time for all ages and genders has also experienced a positive trend. The correlation between race participation and the average race completion time is 92.78%. This is indicated in Fig. 4, below.

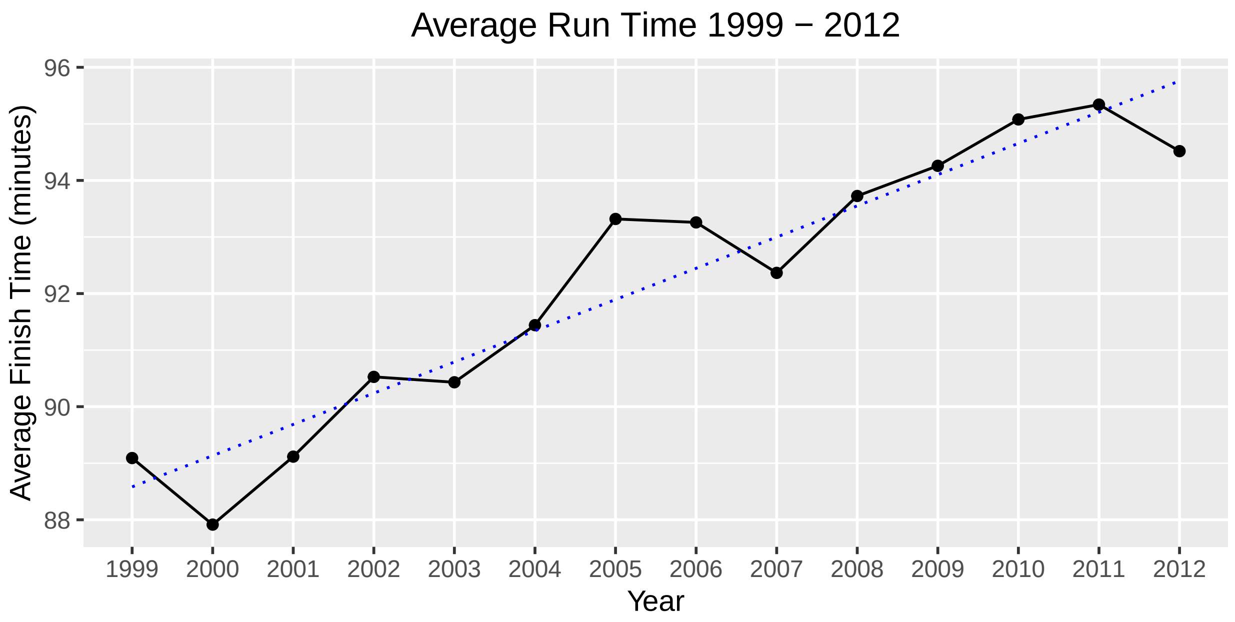


Figure 4: Average race finish time with linear trend line

## 4 Conclusion

These web-scraping methods proved useful for identifying the times of race completion per runner from year 1999 through 2012. From this information we were able to identify average race completion times, by nation and by state. However, documentation methodology and format varied over the years, which presented issues where data was required to be dropped - often as a result of missing information or incorrect information, such as children as young as 1 competing in the race - which impacted the results we were able to identify. National information was relatively reliable, but the state-level information was heavily under-represented.

Because of the level of missing, unrecoverable information, we did not feel inferential analysis based on geographic location would have been useful. However, linear trajectory analysis proved useful when assessing racer count data over time; there is a clear indication race popularity - at least from the measure of race participant count - increased over the time spanning 1999 to 2012. Although with larger variance along the mean over time, there still appears to be a strong indication race completion time - measured in minutes - also been increased over the same years. That is, to say, during the same time period racer participation increased, race completion time increased as well.

## A Code

```
library(pacman)
p_load(XML, rnaturalearth,rnaturalearthdata,countrycode,tidyverse,ggplot2,ggthemes,viridis,rgeos,usmap,
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")

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

extractResTable =
  # takes a list of websites from the cherry blossom race
  # a list of years corresponding to the year the result is for
  # and the gender of the participant
  # Retrieve data from web site,
  # find the preformatted text,
  # and write lines or return as a character vector.
  # returns a list of strings corrsponding to lines in the web url
  function(url = "http://www.cherryblossom.org/results/2009/09cucb-F.htm",
           year = 1999, sex = "male", file = NULL)
  {
    doc = htmlParse(url)

    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)
      els = gsub("Â", "", els)
    }
    else if (year == 1999) {
      # Get preformatted text from <pre> elements
      pres = getNodeSet(doc, "//pre")
      txt = xmlValue(pres[[1]])
      els = strsplit(txt, "\n")[[1]]
    }
    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)
  }

# turn the text data into a list of matrices
years = 1999:2012
menTables = mapply(extractResTable, url = urls, year = years)
names(menTables) = years
sapply(menTables, length)

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

# extractResTable can gather results for a single year, or multiple
men_2009<-extractResTable(url = urls[11], year = 2009, sex = 'male')
men_2009[1:10]

# find the row index where '===' can be found
equals_idx<- grep('^===', men_2009)
equals_idx

equals_idx2<- substr(men_2009, 1,3)
which(equals_idx2 == '===')

#use the location of '===' to identify the spacer and header
spacerRow<- men_2009[equals_idx]
headerRow<- men_2009[equals_idx -1]
body<- men_2009[-(1:equals_idx)] # working with 2009 body
```

```
head(body)

# transform the header to lower case to standardize it
headerRow<- tolower(headerRow)
headerRow

# looks for first match of given sequence
ageStart<- regexec('ag', headerRow, fixed = TRUE)
#ageStart

# extract age data from the text
age<- substr(body, start = 49, stop = 50)
head(age)

summary(as.numeric(age))

#gregexpr looks for all matches of a given sequence
blankLocs<- gregexpr(' ', spacerRow)
blankLocs

# dientify search indices to find data
searchLocs<- c(0,blankLocs[[1]])
searchLocs

# extract data using the helper functions
values<- mapply(substr, list(body), start = searchLocs[-length(searchLocs)] +1, stop = searchLocs[-1] -
length(values)
head(values)

# a function that finds the column locations using the
# previously defined spacing tools
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))
}

locs_2012<- findColLocs(spacerRow)
length(locs_2012)
locs_2012
```

```
# extracts column data from text
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 )
}

# the findColLocs function can be used to find a specific
# type of data, such as age

searchLocs = findColLocs(spacerRow)
ageLoc = selectCols("ag", headerRow, searchLocs)
ages = mapply(substr, list(body),
              start = ageLoc[1,], stop = ageLoc[2, ])
summary(as.numeric(ages))

# the tested functions can now be used to form matrices
shortColNames = c("name", "home", "ag", "gun", "net", "time")
locCols = selectCols(shortColNames, headerRow, searchLocs)

#mapply forms a matrix after completion
Values = mapply(substr, list(body), start = locCols[1, ],
                stop = locCols[2, ])

class(Values)

colnames(Values) = shortColNames
head(Values)
tail(Values)[ , 1:3]

# a function to extract all variables from the menTables 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) ]

       # 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)
}

# assign names and extract the data into a list of matrices
years = 1999:2012
menTables = mapply(extractResTable, url = urls, year = years)
names(menTables) = years

# menResMat is a list of matrices containing race data
menResMat = lapply(menTables, extractVariables)
length(menResMat)

sapply(menResMat, nrow)

# transforming the age to a numeric format
age = as.numeric(menResMat[['2012']][ , 'ag'])

tail(age)
# applying the transformation to all matrices
age = sapply(menResMat,
             function(x) as.numeric(x[ , 'ag']))



# visualizing 'age' to check for anomalies
boxplot(age, ylab = "Age", xlab = "Year",
        main = 'Anomalies in Age Data')

# adjusting the selectCols function to catch more age data
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 )
}

# examining the data for 2001
age2001 = age[["2001"]]

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

badAgeIndex = which(is.na(age2001)) + 5

# adjusting the extractVariables function to make it more performant
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(menTables, extractVariables)

# applying datatype transformations again
age<- sapply(menResMat, function(x) as.numeric(x[, 'ag']))

# counting na values
sapply(age, function(x) sum(is.na(x)))

# visualizing results to check for anomalies
boxplot(age, ylab = "Age", xlab = "Year")

# removing young outliers under 10 years old
young<- menResMat[['2001']][,'ag'] < 10
menResMat[['2001']]<- menResMat[['2001']][-c(young),]
y2002<- menResMat[['2002']][,'ag'] < 10
y2003<- menResMat[['2003']][,'ag'] < 10
menResMat[['2002']]<- menResMat[['2002']][-c(y2002),]
menResMat[['2003']]<- menResMat[['2003']][-c(y2003),]
y2002<- menResMat[['2002']][,'ag'] < 10
y2003<- menResMat[['2003']][,'ag'] < 10

# transforming the list of matrices into a dataframe
menResDF = lapply(menResMat, data.frame)

# creating rows to identify each year
col_1999 = rep(1999, nrow(menResDF$`1999`))

col_2000 = rep(2000, nrow(menResDF$`2000`))

col_2001 = rep(2001, nrow(menResDF$`2001`))

col_2002 = rep(2002, nrow(menResDF$`2002`))

col_2003 = rep(2003, nrow(menResDF$`2003`))

col_2004 = rep(2004, nrow(menResDF$`2004`))
```

```
col_2005 = rep(2005, nrow(menResDF$‘2005‘))

col_2006 = rep(2006, nrow(menResDF$‘2006‘))

col_2007 = rep(2007, nrow(menResDF$‘2007‘))

col_2008 = rep(2008, nrow(menResDF$‘2008‘))

col_2009 = rep(2009, nrow(menResDF$‘2009‘))

col_2010 = rep(2010, nrow(menResDF$‘2010‘))

col_2011 = rep(2011, nrow(menResDF$‘2011‘))

col_2012 = rep(2012, nrow(menResDF$‘2012‘))

cols = c(col_1999, col_2000, col_2001, col_2002, col_2003, col_2004, col_2005, col_2006, col_2007, col_

# building dataframe with year column
menResFinal = data.frame()
for (i in 1:length(names(menResDF))) {

  df<- menResDF[[i]]
  menResFinal = rbind(menResFinal, df)
}
menResFinal$Date<- cols

# transforming the age column to numeric
menResFinal$ag = as.numeric(menResFinal$ag)
sum(is.na(menResFinal$ag))

# identifying a method for transforming the times data
times = menResFinal$time
timePieces = strsplit(as.character(times), ":")

tail(timePieces)

timePieces = sapply(timePieces, as.numeric)
head(timePieces)

# a function to transform the time data into a standard minute format
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)

# generalizing the tiem transformation process and applying it
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
                      })
}

# a function creating the final dataframe
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)
}

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

```
# checking menDF for anomalies
sapply(menDF, function(x) sum(is.na(x$runTime)))
# fixing spacing anomalies in the 2006 data
separatorIdx = grep("^===", menTables[["2006"]])
separatorRow = menTables[['2006']][separatorIdx]
separatorRowX = paste(substring(separatorRow, 1, 63), " ",
                      substring(separatorRow, 65, nchar(separatorRow)),
                      sep = "")
menTables[['2006']][separatorIdx] = separatorRowX
# applying the spacing fix to recreate menResMat
menResMat = sapply(menTables, extractVariables)
# recreating menDF
menDF = mapply(createDF, menResMat, year = 1999:2012,
               sex = rep("M", 14), SIMPLIFY = FALSE)

# importing the data for women
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
urls = paste(ubase, womenURLs, sep = "")
# saving extracted text data
womenTables = mapply(extractResTable, url = urls, year = years, sex='female')
names(womenTables) = years
sapply(womenTables, length)

# applying the same fixes to the women's data as was done to the men
separatorIdx = grep("^===", womenTables[["2006"]])
separatorRow = womenTables[['2006']][separatorIdx]
separatorRowX = paste(substring(separatorRow, 1, 63), " ",
                      substring(separatorRow, 65, nchar(separatorRow)),
                      sep = "")
womenTables[['2006']][separatorIdx] = separatorRowX

womenTables[['2001']][2:3] = womenTables[['2002']][2:3]
# making list of matrices of women's data
```

```
womenResMat = sapply(womenTables, extractVariables)
# transforming list of matrices into dataframe
womenDF = mapply(createDF, womenResMat, year = 1999:2012,
               sex = rep("W", 14), SIMPLIFY = FALSE)

# checking age for na values
sapply(womenDF, function(x) sum(is.na(x$age)))

# checking runTime for na values
sapply(womenDF, function(x) sum(is.na(x$runTime)))

# removing na values because there are so few
menDF_complete = na.omit(menDF)
womenDF_complete = na.omit(womenDF)

# copying the men's data dataframe into a single dataframe
cherryblossom = data.frame()
for (i in 1:14) {

  df<- menDF_complete[[i]]
  cherryblossom = rbind(cherryblossom, df)
}
head(cherryblossom)

# copying the women's data into the cherryblossom dataframe
for (i in 1:14) {

  df<- womenDF_complete[[i]]
  cherryblossom = rbind(cherryblossom, df)
}
tail(cherryblossom)

# checking for na values
sum(is.na(cherryblossom))

# removing na values
cherryblossom = na.omit(cherryblossom)
sum(is.na(cherryblossom))

head(cherryblossom)

library(pacman)
# loading all packages. The first 3 are for ISO codes and the usmap and map are for the literal maps
# also loading sqldf. Because of the volume of the data, sqldf was extremely faster than updating with 
# the time difference was roughly 5 seconds (sqldf) vs. 5 minutes (conditional for-loop)
p_load(rnaturalearth,rnaturalearthdata,countrycode,tidyverse,ggplot2,ggthemes,viridis,rgeos,usmap, sqld

# These runners were all missing locations or had bad locations. We replaced them with what they had li
# from previous races:
cherryblossom[trimws(cherryblossom$name, "both", "[ \t\r\n]")=="Marla Hallacy",]$home = "Washington DC"
cherryblossom[trimws(cherryblossom$name, "both", "[ \t\r\n]")=="Matthew Rogers",]$home = "Arlington VA"
cherryblossom[trimws(cherryblossom$name, "both", "[ \t\r\n]")=="Adri Jayaratne",]$home = "Washington DC
cherryblossom[trimws(cherryblossom$name, "both", "[ \t\r\n]")=="Marla Hallacy",]$home = "Washington DC"
cherryblossom[trimws(cherryblossom$name, "both", "[ \t\r\n]")=="Eleanor Rathbone",]$home = "Washington 
cherryblossom[trimws(cherryblossom$name, "both", "[ \t\r\n]")=="Emily Naden",]$home = "Washington DC"
cherryblossom[trimws(cherryblossom$name, "both", "[ \t\r\n]")=="Grace L Chen",]$home = "Washington DC"
cherryblossom[trimws(cherryblossom$name, "both", "[ \t\r\n]")=="Stephanie Rogers",]$home = "Aldie VA"
cherryblossom[trimws(cherryblossom$name, "both", "[ \t\r\n]")=="Sarah Garner",]$home = "    Arlington V
cherryblossom[trimws(cherryblossom$name, "both", "[ \t\r\n]")=="Maria Solomon",]$home = "Arlington VA"
cherryblossom[trimws(cherryblossom$name, "both", "[ \t\r\n]")=="Amy Iacopi",]$home = "San Francisco CA"
cherryblossom[trimws(cherryblossom$name, "both", "[ \t\r\n]")=="Sarah Watkins",]$home = "Washington DC"
cherryblossom[trimws(cherryblossom$name, "both", "[ \t\r\n]")=="Alissa Havens",]$home = "Washington DC"

cherryblossomX = cherryblossom
# the home column needed leading and trailing trimming
cherryblossomX$home = trimws(cherryblossomX$home, "both", "[ \t\r\n]")

# Standardizing countries to names where only codes were present. Although we revert them to codes for 
# it is easier to work with the countries by name when coding
cherryblossomX[cherryblossomX$home =="AUS",]$home = "Australia"
cherryblossomX[cherryblossomX$home =="CAN",]$home = "Canada"
cherryblossomX[cherryblossomX$home =="COL",]$home = "Colombia"
cherryblossomX[cherryblossomX$home =="CZE",]$home = "Czech Republic"
cherryblossomX[cherryblossomX$home =="Fra",]$home = "France"
cherryblossomX[cherryblossomX$home =="GER",]$home = "Germany"
cherryblossomX[cherryblossomX$home =="IRE",]$home = "Ireland"
cherryblossomX[cherryblossomX$home =="JAP",]$home = "Japan"
cherryblossomX[cherryblossomX$home =="KOR",]$home = "Republic of Korea"
cherryblossomX[cherryblossomX$home =="LIT",]$home = "Lithuania"
cherryblossomX[cherryblossomX$home =="PER",]$home = "Peru"
cherryblossomX[cherryblossomX$home =="POL",]$home = "Poland"
cherryblossomX[cherryblossomX$home =="RSA",]$home = "South Africa"

cherryblossomX[cherryblossomX$home %in% c("ETH","Eth"),]$home = "Ethiopia"
cherryblossomX[cherryblossomX$home %in% c("KEN","Ken"),]$home = "Kenya"
cherryblossomX[cherryblossomX$home %in% c("MEX","Mex"),]$home = "Mexico"
```

```
cherryblossomX[cherryblossomX$home %in% c("RUS","Rus"),]$home = "Russian Federation"
cherryblossomX[cherryblossomX$home %in% c("ROM","Rom"),]$home = "Romania"
cherryblossomX[cherryblossomX$home %in% c("Us","Usa","Ny","MD"),]$home = "United States"
cherryblossomX[cherryblossomX$home %in% c("Bri","SCO"),]$home = "United Kingdom of Great Britain and Nor

# dropping non-locational values
dropVars <- c("", "Lee","M")
cherryblossomX <- cherryblossomX[!(cherryblossomX$home %in% dropVars ),]

# Under the assumption US states have either leading spaces or commas (such as in Phoenix AZ/Phoenix,AZ/
# We recode states to 'United States' as follows in the SQL below. Also, Canadian states are recoded to
# the remaining nations being recoded to ISO standards. These were identified through manual, direct ana
# with a code of LE was given for a city in Leicestershire. Finally, all remaining countries in the ISO
# included are recorded to United States as it was specified in the documentation that American cities w
# location (they, at best, included only state codes)
summary_table = sqldf("
      select
      case when substr( trim(home), length(trim(home))-2, length(trim(home)) ) in (' AL',' AK',' AZ',' A
           when trim(home) in ('California') then 'United States'
           when trim(home) in ('Indiana') then 'United States'
           when trim(home) in ('Wyoming') then 'United States'
           when substr( trim(home), length(trim(home))-2, length(trim(home)) ) in (' CN',' ON',' NO',' S
           when substr( trim(home), length(trim(home))-3, length(trim(home)) ) in (' ONT',',ONT') then
           when substr( trim(home), length(trim(home))-2, length(trim(home)) ) in (' UK',',UK',' LE',',L
           when substr( trim(home), length(trim(home))-2, length(trim(home)) ) in (' PR',',PR') then 'Pu
           when substr( trim(home), length(trim(home))-2, length(trim(home)) ) in (' GE',',GE') then 'Ge
           when trim(home) in ('Rep Of S.africa') then 'South Africa'
           when trim(home) in ('Scotland') then 'United Kingdom of Great Britain and Northern Ireland'
           when trim(home) in ('NewZealand') then 'New Zealand'
           when trim(home) in ('Russia') then 'Dominican Republic'
           when trim(home) in ('Dominican Republi') then 'Russian Federation'
           when trim(home) in ('Trinidad') then 'Trinidad and Tobago'
           when trim(home) in ('Tanzania') then 'Tanzania, United Republic of'
           when trim(home) in ('Bolivia') then 'Bolivia, Plurinational State of'
           when trim(home) not in ('Australia','Bolivia, Plurinational State of','Brazil','United Kingdo
          else trim(home) end as home,
          runTime as runTime
      from cherryblossomX
      group by home
      ")

# getting the ISO codes from package 'maps'
dat <- iso3166
```

```
cherryblossom_country_runtimes <- sqldf("select d.a3, d.mapname, avg(x.runTime) as runTime from summary_

map <- ne_countries(scale = "medium", returnclass = "sf")
# Joining cross-walked ISO names to the map through the code names
runners_map <- merge(map, cherryblossom_country_runtimes, by.x = "adm0_a3", by.y = "a3", all = TRUE)

# global map
assoc_graph <- ggplot(data = runners_map) +
  geom_sf(aes(fill = runTime),
          position = "identity") +
  labs(fill='Average\nCompletion Time', caption = paste(
      'Figure 1:\n',
      'Global Race Completion Times', sep=''))  +
  scale_fill_viridis_c(option = "viridis") +
   ggtitle('Average Race Completion Time, by Country')

assoc_graph + theme_map()

cherryblossomStates = cherryblossom

# Recoding the states to a unified location.
states_summary_table = sqldf("
      select
      case when substr( trim(home), length(trim(home))-2, length(trim(home)) ) in (' AL',',AL') then 'AL
when substr( trim(home), length(trim(home))-2, length(trim(home)) ) in (' AK',',AK') then 'AK'
when substr( trim(home), length(trim(home))-2, length(trim(home)) ) in (' AZ',',AZ') then 'AZ'
when substr( trim(home), length(trim(home))-2, length(trim(home)) ) in (' AR',',AR') then 'AR'
when substr( trim(home), length(trim(home))-2, length(trim(home)) ) in (' CA',',CA') then 'CA'
when substr( trim(home), length(trim(home))-2, length(trim(home)) ) in (' CO',',CO') then 'CO'
when substr( trim(home), length(trim(home))-2, length(trim(home)) ) in (' CT',',CT') then 'CT'
when substr( trim(home), length(trim(home))-2, length(trim(home)) ) in (' DE',',DE') then 'DE'
when substr( trim(home), length(trim(home))-2, length(trim(home)) ) in (' FL',',FL') then 'FL'
when substr( trim(home), length(trim(home))-2, length(trim(home)) ) in (' GA',',GA') then 'GA'
when substr( trim(home), length(trim(home))-2, length(trim(home)) ) in (' HI',',HI') then 'HI'
when substr( trim(home), length(trim(home))-2, length(trim(home)) ) in (' ID',',ID') then 'ID'
when substr( trim(home), length(trim(home))-2, length(trim(home)) ) in (' IL',',IL') then 'IL'
when substr( trim(home), length(trim(home))-2, length(trim(home)) ) in (' IN',',IN') then 'IN'
when substr( trim(home), length(trim(home))-2, length(trim(home)) ) in (' IA',',IA') then 'IA'
when substr( trim(home), length(trim(home))-2, length(trim(home)) ) in (' KS',',KS') then 'KS'
when substr( trim(home), length(trim(home))-2, length(trim(home)) ) in (' KY',',KY') then 'KY'
when substr( trim(home), length(trim(home))-2, length(trim(home)) ) in (' LA',',LA') then 'LA'
when substr( trim(home), length(trim(home))-2, length(trim(home)) ) in (' ME',',ME') then 'ME'
when substr( trim(home), length(trim(home))-2, length(trim(home)) ) in (' MD',',MD') then 'MD'
when substr( trim(home), length(trim(home))-2, length(trim(home)) ) in (' MA',',MA') then 'MA'
when substr( trim(home), length(trim(home))-2, length(trim(home)) ) in (' MI',',MI') then 'MI'
when substr( trim(home), length(trim(home))-2, length(trim(home)) ) in (' MN',',MN') then 'MN'
when substr( trim(home), length(trim(home))-2, length(trim(home)) ) in (' MS',',MS') then 'MS'
when substr( trim(home), length(trim(home))-2, length(trim(home)) ) in (' MO',',MO') then 'MO'
when substr( trim(home), length(trim(home))-2, length(trim(home)) ) in (' MT',',MT') then 'MT'
when substr( trim(home), length(trim(home))-2, length(trim(home)) ) in (' NE',',NE') then 'NE'
when substr( trim(home), length(trim(home))-2, length(trim(home)) ) in (' NV',',NV') then 'NV'
when substr( trim(home), length(trim(home))-2, length(trim(home)) ) in (' NH',',NH') then 'NH'
when substr( trim(home), length(trim(home))-2, length(trim(home)) ) in (' NJ',',NJ') then 'NJ'
when substr( trim(home), length(trim(home))-2, length(trim(home)) ) in (' NM',',NM') then 'NM'
when substr( trim(home), length(trim(home))-2, length(trim(home)) ) in (' NY',',NY') then 'NY'
when substr( trim(home), length(trim(home))-2, length(trim(home)) ) in (' NC',',NC') then 'NC'
when substr( trim(home), length(trim(home))-2, length(trim(home)) ) in (' ND',',ND') then 'ND'
when substr( trim(home), length(trim(home))-2, length(trim(home)) ) in (' OH',',OH') then 'OH'
when substr( trim(home), length(trim(home))-2, length(trim(home)) ) in (' OK',',OK') then 'OK'
when substr( trim(home), length(trim(home))-2, length(trim(home)) ) in (' OR',',OR') then 'OR'
when substr( trim(home), length(trim(home))-2, length(trim(home)) ) in (' PA',',PA') then 'PA'
when substr( trim(home), length(trim(home))-2, length(trim(home)) ) in (' RI',',RI') then 'RI'
when substr( trim(home), length(trim(home))-2, length(trim(home)) ) in (' SC',',SC') then 'SC'
when substr( trim(home), length(trim(home))-2, length(trim(home)) ) in (' SD',',SD') then 'SD'
when substr( trim(home), length(trim(home))-2, length(trim(home)) ) in (' TN',',TN') then 'TN'
when substr( trim(home), length(trim(home))-2, length(trim(home)) ) in (' TX',',TX') then 'TX'
when substr( trim(home), length(trim(home))-2, length(trim(home)) ) in (' UT',',UT') then 'UT'
when substr( trim(home), length(trim(home))-2, length(trim(home)) ) in (' VT',',VT') then 'VT'
when substr( trim(home), length(trim(home))-2, length(trim(home)) ) in (' VA',',VA',' VI',',VI') then '
when substr( trim(home), length(trim(home))-2, length(trim(home)) ) in (' WA',',WA') then 'WA'
when substr( trim(home), length(trim(home))-2, length(trim(home)) ) in (' WV',',WV') then 'WV'
when substr( trim(home), length(trim(home))-2, length(trim(home)) ) in (' WI',',WI') then 'WI'
when substr( trim(home), length(trim(home))-2, length(trim(home)) ) in (' WY',',WY') then 'WY'
when trim(home) = 'Wyoming' then 'WY'
when trim(home) = 'California' then 'CA'
when trim(home) = 'Indiana' then 'IN'
          else NULL end as home,
          runTime as runTime
      from cherryblossomStates
      group by home
      ")

states_summary_table1 = na.omit(states_summary_table)

cherryblossom_sates_runtimes <- sqldf("select home as state, avg(runTime) as runTime from states_summar
# plotting the state-wide data. The only requirement is that the states have a column name 'state' fill
plot_usmap(data = cherryblossom_sates_runtimes, values = "runTime",  labels=FALSE)   +
  scale_fill_viridis_c(option = "viridis") +
  theme(legend.position = "right") +
  theme(panel.background = element_rect(colour = "black")) +
  labs(fill='Average Race Completion Time', title = "Average Race Completion Time, by US State", captio
      'Figure 2:\n',
      'Statewide Race Completion Times', sep=''))

# making runTime subset
just_runTime = cherryblossom[c(1,6)]
# aggregating runTimes by year using the mean value
runTime_by_year = aggregate(just_runTime, by = list(just_runTime$year), FUN = mean)
# aggregating the rowcount by year
participation = aggregate(just_runTime, by = list(just_runTime$year), FUN = length)

# linear trend lines
participation_lm = lm(year ~ Group.1, data = participation)
runTime_lm = lm(runTime ~ year, data = runTime_by_year)

#visualizing particiaption
library(ggplot2)
p = ggplot(participation, aes(Group.1, runTime)) +
  labs(title = 'Cherry Blossom Race Participation', caption = 'Figure 3: Cherry Blossom race participat
  xlab('Year') + ylab('Number of Participants') +
  theme(plot.title = element_text(hjust = .5), plot.caption = element_text(hjust = 0)) +
  scale_x_discrete(name = 'Year', limits = c(1999:2012))
p + geom_line() + geom_point() +
  geom_line(aes(y= participation_lm$fitted.values), color = 'blue', linetype = 'dotted')

# visualizing race completion times
p = ggplot(runTime_by_year, aes(year, runTime)) +
  labs(title = 'Average Run Time 1999 - 2012', caption = 'Figure 4: Average race finish time with linea
  xlab('Year') + ylab('Average Finish Time (minutes)') +
  theme(plot.title = element_text(hjust = .5), plot.caption = element_text(hjust = 0)) +
  scale_x_discrete(name = 'Year', limits = c(1999:2012))
p + geom_line() + geom_point() +
  geom_line(aes(y = runTime_lm$fitted.values), color = 'blue', linetype = 'dotted')

# quantifying the corrleation between participation and completion time
cor(runTime_by_year$runTime, participation$runTime)
```