# Scraping Data

*Yusuf Ameri, Douglas Sexton*

*December 12, 2016*

## Document Information

The following document describes how we obtained and went through the **ETL** process to create the following files:

- election.txt (use this one)
- participated.txt (use this one)
- party.txt (use this one)
- candidate.txt
- candidate2.txt
- candidate_best.txt (use this one)
- population.txt
- population_best.txt (use this one)
- polls.txt (use this one)
- candidate_info.txt (use this one)

#### Files not made by this document

Some files are not made in this document and are instead made by hand or in other R scrip files such as ...

- population_best.txt
- candidate_best.txt
- candidate_info.txt

## Doug's Scraping for Election, Participated, Party, and Candidate

The following chunk below makes the **election.txt** file with the **election** relation, as well as the **participated.txt** file with the **participated** relation.

- election.txt (use this one)
- participated.txt (use this one)
- party.txt (use this one)
- candidate.txt (NOT USED)

```
library(rvest)
library(readr)
library(dplyr)
library(tidyr)
#url <- "http://2012election.procon.org/view.resource.php?resourceID=004332"
#webpage <- read_html(url)
# Make sure this file is in your working directory!
webpage <- read_html("PresElectHist.html")
p_table <- html_nodes(webpage, 'table')
```

```
p <- p_table[[18]]
g <- html_table(p, fill=TRUE, trim=TRUE)
names(g) <- c("Year", "Candidates", "Parties", "Electoral Votes", "Popular Votes", "VPs")
g <- g[(1:6)] # Get rid of 4 extraneous columns
# Get rows of candidates with open parenthesis followed by number
g <- g[grepl("\\([[:digit:]]", g$Candidates),]

# These lists are used to make a set of the candidates and parties
candList<-list()
partyList<-list()

# The election and participated files
electionFile <- "election.txt"
participatedFile <- "participated.txt"

# drop if tables exist
write("DROP TABLE IF EXISTS `election`;", electionFile)
write("DROP TABLE IF EXISTS `participated`;", participatedFile)

# create tables
write("CREATE TABLE presidential_elections.election
      (year INTEGER NOT NULL, winner VARCHAR(64), num INTEGER);",
      electionFile, append = TRUE)
write("CREATE TABLE presidential_elections.participated
      (year INTEGER NOT NULL,
      candidate VARCHAR(64),
      party VARCHAR(64),
      electoral_vote INTEGER,
      popular_vote INTEGER,
      vice_president VARCHAR(64));",
      participatedFile, append = TRUE)

# Write initial sql text to file
write("INSERT INTO `election`
      (`year`, `winner`, `num`) VALUES ",
      electionFile, append = TRUE)
write("INSERT INTO `participated`
      (`year`, `candidate`, `party`, `electoral_vote`,
      `popular_vote`, `vice_president`) VALUES ",
      participatedFile,
      append = TRUE)

elecNum <- 57 # Hard coded value for number of elections

for(i in 1:nrow(g)) {
  row <- g[i,]
  candidates <- strsplit(row$Candidates, "\n")
  parties <- strsplit(row$Parties, "\n")
  electoralVotes <- strsplit(row$`Electoral Votes`, "\n")
  popularVotes <- strsplit(row$`Popular Votes`, "\n")
  vicePresidents <- strsplit(row$VPs, "\n")
  for (j in 1:length(candidates[[1]])) {
    pop <- ""
```

```
  vp <- ""
  # Popular votes and vp might not exist
  if (j <= length(popularVotes[[1]])) {
    pop <- (popularVotes[[1]][j])
  } else {
    pop <- ("NULL")
  }

  if (j <= length(vicePresidents[[1]])) {
    vp <- (vicePresidents[[1]][j])
  } else {
    vp<- ("NULL")
  }

  # Now clean up names to get rid of parenthesis
  cleanCandidate <-
    strsplit(candidates[[1]][j], " \\(")[[1]][1] # Get rid of end parenthesis
  cleanCandidate <-
    strsplit(cleanCandidate, "\\*")[[1]][1] # Get rid of trailing asterisks
  cleanCandidate <-
    trimws(cleanCandidate) # Get rid of leading and trailing whitespace
  cleanCandidate <-
    gsub("\'", "\\\\\'", cleanCandidate) # Escape ' character for sql

  vp <- strsplit(vp, " \\(")[[1]][1] # Get rid of end parenthesis
  vp <- strsplit(vp, "\\*")[[1]][1] # Get rid of trailing asterisks
  vp <- gsub("\'", "\\\\\'", vp) # Escape ' character for sql
  vp <- trimws(vp)

  # Store candidates and parties in hash
  candList[[length(candList) + 1]] <- cleanCandidate
  if (vp != "NULL") {
    candList[[length(candList) + 1]] <- vp # Make sure if no_vp to not add to list
  }

  # Clean up parties so ' char is escaped for sql
  party <- parties[[1]][j]
  party <- gsub("\'", "\\\\\'", party)
  partyList[[length(partyList) + 1]] <- party

  # Clean up popular votes to get rid of commas in numbers
  pop <- gsub(",", "", pop)

  cat(sprintf("%s %s %s %s %s %s\n",
              row$Year,
              cleanCandidate,
              party,
              electoralVotes[[1]][j],
              pop,
              vp))

  # Write to election file
  if (row$Year != "") {
```

```
    if (j == 1) {
      # Only write if election year and if first candidate in list
      if (elecNum == 1) {
        write(sprintf("('%s', '%s', '%d')",
                      row$Year,
                      cleanCandidate,
                      elecNum),
              electionFile, append=TRUE)
      } else {
        write(sprintf("('%s', '%s', '%d'), ",
                      row$Year,
                      cleanCandidate,
                      elecNum),
              electionFile, append=TRUE)
      }
      elecNum <- elecNum - 1
    }
  }

  # Write to participated file
  if (row$Year != "") {
    # Write a participant unless last candidate
    # population and vp are weird in that they might be null so handle that now
    popa <- pop
    if (popa != "NULL") {
      popa <- paste("'",popa, "'", sep="")
    }
    vpa <- vp
    if (vpa != "NULL") {
      vpa <- paste("'",vpa, "'", sep="")
    }
    if (i == nrow(g) &&  j == length(candidates[[1]])) {
      write(sprintf("('%s', '%s', '%s', '%s', %s, %s)",
                    row$Year,
                    cleanCandidate,
                    party,
                    electoralVotes[[1]][j],
                    popa,
                    vpa),
            participatedFile, append=TRUE)
    } else {
      write(sprintf("('%s', '%s', '%s', '%s', %s, %s), ",
                    row$Year,
                    cleanCandidate,
                    party,
                    electoralVotes[[1]][j],
                    popa,
                    vpa),
            participatedFile, append=TRUE)
    }
  }

}
```

```
}

emptyList<-list()
candList <- union(candList, emptyList) # Set operations to remove duplicates
partyList <- union(partyList, emptyList)

# Now let's write all parties to a file as a sql command
fileConn <- "party.txt"
write("DROP TABLE IF EXISTS `party`;", fileConn)
write("CREATE TABLE presidential_elections.party
      (name VARCHAR(64) NOT NULL);",  fileConn, append = TRUE)
write("INSERT INTO `party` (`name`) VALUES ", fileConn, append = TRUE)
for (k in 1:length(partyList)){
  if (k == length(partyList)) {
    write(sprintf("('%s')", partyList[[k]]), fileConn, append=TRUE)
  } else {
    write(sprintf("('%s'), ", partyList[[k]]), fileConn, append=TRUE)
  }
}

# Now let's write all candidates to a file as a sql command
fileConn <- "candidate.txt"
write("DROP TABLE IF EXISTS `candidate`;", fileConn)
write("CREATE TABLE presidential_elections.candidate
      (name VARCHAR(64) NOT NULL);",  fileConn, append = TRUE)
write("INSERT INTO `candidate` (`name`) VALUES ", fileConn, append = TRUE)
for (k in 1:length(candList)){
  if (k == length(candList)) {
    write(sprintf("('%s')", candList[[k]]), fileConn, append=TRUE)
  } else {
    write(sprintf("('%s'), ", candList[[k]]), fileConn, append=TRUE)
  }
}
```

**Scraping more info about candidates in the candidateScrape.R file**

The candidate.txt file that we generate about is not used in our DB implemenation. Instead, the script **candidateScrape.R** makes the SQL file for candidates called **candidate2.txt**. We however modify this file again by hand and finally use the **candidate_best.txt** file as part of our DB implementation.

## Scrape Population/Census Table

**The next section will create population.txt and population_best.txt.** We will be using **population_best.txt** for our database. We scrape census population data for every 10 years starting from 1610. The following data can be found from this wikipedia article here.

```
url <-
  'https://en.wikipedia.org/wiki/Demographic_history_of_the_United_States#Historical_population'
xpath <- '//*[@id="mw-content-text"]/table[2]'

census <- xml2::read_html(url)
```

```
census_table <- census %>% html_node(xpath = xpath) %>% html_table()

# remove last row
census_table <- census_table[2:nrow(census_table),]

# pdf source: http://www2.census.gov/prod2/statcomp/documents/CT1970p2-13.pdf
```

### Stip commas from numbers

We need to strip the commas from the population data before we can store it via sql

```
census_table$Population <- gsub(",","", census_table$Population)
```

Now lets write the table as an SQL entry file

```
# Now let's write the census to a file as a sql command
fileConn <- "population.txt"
write("INSERT INTO `population` (`year`, `population`) VALUES ", fileConn)
for (k in 1:nrow(census_table)){
  if (k < nrow(census_table)) {
    write(sprintf("('%s', '%s'), ", census_table[k,1],census_table[k,2]), fileConn, append=TRUE)
  } else {
    write(sprintf("('%s', '%s')", census_table[k,1],census_table[k,2]), fileConn, append=TRUE)
  }
}
```

### Manually adding population data for election years.

For election years, we will manually add the folling data for population. We only had census data for every 10 years before but we would like to have population for years where elections took place as well. The text below is appended to the **population.txt** file and saved into a new file called **population_best.txt**

```
('1789', '3804342'),
('1792', '4172906'),
('1796', '4706570'),
('1804', '6010005'),
('1808', '6804234'),
('1812', '7666314'),
('1816', '8596011'),
('1824', '10818875'),
('1828', '12143863'),
('1832', '13614425'),
('1836', '15244369'),
('1844', '19295948'),
('1848', '21812961'),
('1852', '24647586'),
('1856', '27838857'),
('1864', '34116504'),
('1868', '37016949'),
('1872', '40645925'),
('1876', '45166214'),
```

```
('1884', '54959988'),
('1888', '60184258'),
('1892', '65408007'),
('1896', '70548968'),
('1904', '82166000'),
('1908', '88710000'),
('1912', '95335000'),
('1916', '101961000'),
('1924', '114109000'),
('1928', '120509000'),
('1932', '124840471'),
('1936', '128053180'),
('1944', '138397345'),
('1948', '146631302'),
('1952', '157552740'),
('1956', '168903031'),
('1964', '191888791'),
('1968', '200706052'),
('1972', '209896021'),
('1976', '218035164'),
('1984', '235824902'),
('1988', '244498982'),
('1992', '254994517'),
('1996', '265189794'),
('2004', '293045739'),
('2008', '304374846'),
('2012', '314100000');
```

## Scraping Polling Data

**Original API sources for csv files.**

We are getting our sources from pollyvote.com. Below is the list of URLs for API calls to get the csv for our data. Note that data for polls is only available from the 1940 election (that is when polling data became widely available and popular). This is expected and the best data we can get.

```
# national expectation polls 2016
nep16 <-
  'pollyvote.com/wp-content/plugins/pollyvote/data/index.php?time=current&type=expectations&format=csv'
# national intention polls 2016
nip16 <-
  'pollyvote.com/wp-content/plugins/pollyvote/data/index.php?time=current&type=polls&format=csv'

# national expectation polls historical (1940 - 2012)
neph <-
  'pollyvote.com/wp-content/plugins/pollyvote/data/index.php?type=expectations&format=csv'

# national intention polls historical (1940 - 2012)
niph <-
  'pollyvote.com/wp-content/plugins/pollyvote/data/index.php?type=polls&format=csv'
```

### Import/Download the csv from these URLs

We make a function to download given a URL and make a dataframe as well as a .csv file.

```
# filename: the name you want to download and save the csv as.
# df.name: the name of the dataframe for the csv you want
# source.URL: the url source that we download from
download_polls <- function(filename, df.name, source.URL) {
  if (!file.exists(filename)) {
    download.file(source.URL, destfile=filename)
  }
  assign(x = df.name, value = read_csv(filename), envir = .GlobalEnv)
}
```

### Next, we call the function above on all the polling data we want

```
# national expectation polls 2016
download_polls(filename = '2016_national_expectation_polls.csv',
               df.name = 'national_expectation_polls.2016',
               source.URL = nep16)

# national intention polls 2016
download_polls(filename = '2016_national_intention_polls.csv',
               df.name = 'national_intention_polls.2016',
               source.URL = nip16)

# national expectation polls historical (1940 - 2012)
download_polls(filename = 'historic_national_expectation_polls.csv',
               df.name = 'national_expectation_polls.historic',
               source.URL = neph)

# national intention polls historical (1940 - 2012)
download_polls(filename = 'historic_national_intention_polls.csv',
               df.name = 'national_intention_polls.historic',
               source.URL = niph)
```

### Only select the columns we care about

The original CSV gives us an astounding **43** columns of data. A lot of these columns have empty (i.e. NULL) data and are not interesting and usefull to us. We will only select the relevant ones below

- electionyear
- fcdate
- daystoelection
- fcrepvs
- fcdemvs
- fcerror
- absolutefcerror
- lastsurveyday
- firstsurveyday
- released
- sample
- moe

- questiontxt
- surveyorg
- surveysponsor
- intmethod
- sampledesc
- state

```
# selects only relevant/good columns from
# the initial list of 43 columns that we get from the csv
selectGoodColumns <- function(df.name, list.of.columns) {
      df.name %>% select(c(electionyear,
                           fcdate,              # forecast date
                           daystoelection,      # the number of days until the election
                           fcrepvs,             # republican voter share
                           fcdemvs,             # democratic voter share
                           fcerror,             # forecase error (Forecast - Actual)
                           absolutefcerror,     # absolute forecast error
                           lastsurveyday,       # last day survey was conducted
                           firstsurveyday,      # first day survey was conducted
                           released,            # date survey was released
                           sample,              # sample size of survey
                           moe,                 # margin of error of survey
                           questiontxt,         # question text description of survey
                           surveyorg,           # survey organization
                           surveysponsor,       # survey sponsor
                           intmethod,           # interview method
                           sampledesc,          # information about the sample of respondents
                           state))
}

# national expectation polls 2016
national_expectation_polls.2016 <- selectGoodColumns(national_expectation_polls.2016)
national_expectation_polls.2016 <-
  national_expectation_polls.2016 %>% mutate(pollType = "expectation")

# national intention polls 2016
national_intention_polls.2016 <- selectGoodColumns(national_intention_polls.2016)
national_intention_polls.2016 <-
  national_intention_polls.2016 %>% mutate(pollType = "intention")

# national expectation polls historical (1940 - 2012)
national_expectation_polls.historic <- selectGoodColumns(national_expectation_polls.historic)
national_expectation_polls.historic <-
  national_expectation_polls.historic %>% mutate(pollType = "expectation")

# national intention polls historical (1940 - 2012)
national_intention_polls.historic <- selectGoodColumns(national_intention_polls.historic)
national_intention_polls.historic <-
  national_intention_polls.historic %>% mutate(pollType = "intention")
```

### Convert some columns before sql insertion

Some of the original columns have dates stored with '.' seperating the month, day, and year. We need to convert these to dashes. We do this below for each of the dataframes using **readr::type_convert** function.

```
# national expectation polls 2016
national_expectation_polls.2016 <-
  national_expectation_polls.2016 %>% readr::type_convert(col_types = cols(
    fcdate = col_datetime(format="%d.%m.%Y"),
    lastsurveyday = col_datetime(format="%d.%m.%Y"),
    firstsurveyday = col_datetime(format="%d.%m.%Y"),
    released = col_datetime(format="%d.%m.%Y")
))

# national intention polls 2016
national_intention_polls.2016 <-
  national_intention_polls.2016 %>% readr::type_convert(col_types = cols(
    fcdate = col_datetime(format="%d.%m.%Y"),
    lastsurveyday = col_datetime(format="%d.%m.%Y"),
    firstsurveyday = col_datetime(format="%d.%m.%Y"),
    released = col_datetime(format="%d.%m.%Y")
))

# national expectation polls historical (1940 - 2012)
national_expectation_polls.historic <-
  national_expectation_polls.historic %>% readr::type_convert(col_types = cols(
    fcdate = col_datetime(format="%d.%m.%Y"),
    lastsurveyday = col_datetime(format="%d.%m.%Y"),
    firstsurveyday = col_datetime(format="%d.%m.%Y"),
    released = col_datetime(format="%d.%m.%Y")
))

# national intention polls historical (1940 - 2012)
national_intention_polls.historic <-
  national_intention_polls.historic %>% readr::type_convert(col_types = cols(
    fcdate = col_datetime(format="%d.%m.%Y"),
    lastsurveyday = col_datetime(format="%d.%m.%Y"),
    firstsurveyday = col_datetime(format="%d.%m.%Y"),
    released = col_datetime(format="%d.%m.%Y")
))
```

### Build the SQL file

In the code block below, we create the **polls.txt** file and create the **polls** relation. We define the column names and

```
pollsFile <- "polls.txt"
write("DROP TABLE IF EXISTS `polls`;", pollsFile)
write("CREATE TABLE presidential_elections.polls (election_year VARCHAR(64) NOT NULL,
                                                  fc_date DATE,
                                                  days_to_election DATE,
                                                  fc_repvs FLOAT,
                                                  fc_demvs FLOAT,
```

```
                                                fc_error FLOAT,
                                                fc_absolute_error FLOAT,
                                                last_survey_day DATE,
                                                first_survey_day DATE,
                                                release_day DATE,
                                                sample INTEGER,
                                                moe FLOAT,
                                                question TEXT,
                                                survey_org VARCHAR(64),
                                                survey_sponser VARCHAR(128),
                                                intmethod VARCHAR(30),
                                                sample_desc TEXT,
                                                state VARCHAR(40),
                                                poll_type VARCHAR(15));",
pollsFile,
append = TRUE)

write("INSERT INTO polls (                      election_year,
                                                fc_date,
                                                days_to_election,
                                                fc_repvs,
                                                fc_demvs,
                                                fc_error,
                                                fc_absolute_error,
                                                last_survey_day,
                                                first_survey_day,
                                                release_day,
                                                sample,
                                                moe,
                                                question,
                                                survey_org,
                                                survey_sponser,
                                                intmethod,
                                                sample_desc,
                                                state,
                                                poll_type) VALUES ",
pollsFile,
append=TRUE)
```

**Write the poll tuples**

We will now loop through the four dataframes we have created convert the rows into mysql insert statements as well as define/create the relation for the table we will need (polls). **Just a note: at the end of the text file you should remove the last comma and replace it with a semicolon.**

```
# writeFD: function to write the tuples of each dataframe (df) to the polls.txt file
writeFD <- function(df) {
    for(i in 1:nrow(df)) {
      electionyear <-   df$electionyear
      fcdate <-         df$fcdate
      daystoelection <- df$daystoelection
      fcrepvs <-        df$fcrepvs
      fcdemvs <-        df$fcdemvs
```

```
      fcerror <-        df$fcerror
      absolutefcerror <-df$absolutefcerror
      lastsurveyday <-  df$lastsurveyday
      firstsurveyday <- df$firstsurveyday
      released <-       df$released
      sample <-         df$sample
      moe <-            df$moe
      questiontxt <-    gsub(x = df$questiontxt, pattern = "\'", replacement = "")
      surveyorg <-      gsub(x = df$surveyorg, pattern = "\'", replacement = "")
      surveysponsor <-  gsub(x = df$surveysponsor, pattern = "\'", replacement = "")
      intmethod <-      gsub(x = df$intmethod, pattern = "\'", replacement = "")
      sampledesc <-     gsub(x = df$sampledesc, pattern = "\'", replacement = "")
      state <-          df$state
      pollType <-       df$pollType

      # replace the NAs with NULL and sprintf into apporpriate format
      write(
      gsub(x = sprintf(
      "('%s','%s','%s','%s','%s','%s','%s','%s','%s','%s',
      '%s','%s','%s','%s','%s','%s','%s','%s','%s'),",
                    electionyear[i],
                    fcdate[i],
                    daystoelection[i],
                    fcrepvs[i],
                    fcdemvs[i],
                    fcerror[i],
                    absolutefcerror[i],
                    lastsurveyday[i],
                    firstsurveyday[i],
                    released[i],
                    sample[i],
                    moe[i],
                    questiontxt[i],
                    surveyorg[i],
                    surveysponsor[i],
                    intmethod[i],
                    sampledesc[i],
                    state[i],
                    pollType[i]),
      pattern="NA",
      replacement = "NULL"),
      pollsFile, append=TRUE)
    }
}

writeFD(national_expectation_polls.2016)
writeFD(national_intention_polls.2016)
writeFD(national_expectation_polls.historic)
writeFD(national_intention_polls.historic)
```

## Creating the candidate info file.

The **candidate_info.txt** file was manually created to fill in information for candidates which became president through the death or resignation of the previous president.

## Web crawler for candidates

In order to extract candidate birth dates, death dates, and photos. A web crawler was written which outputs the **candidate2.txt** file.

**First we write the beginning of the file and read from a csv file containing all the candidates from candidates.txt**

```
library(stringr)
candidates <- read.csv("candidate.csv", header=F, sep=",", stringsAsFactors=FALSE)
```

```
## Warning in read.table(file = file, header = header, sep = sep, quote =
## quote, : incomplete final line found by readTableHeader on 'candidate.csv'
```

```
baseUrl <- "http://www.google.com/search?q="

candidateFile <- "candidate2.txt" # The candidate file to write
write("DROP TABLE IF EXISTS `candidate`;", candidateFile)
write("CREATE TABLE presidential_elections.candidate
      ( name VARCHAR(64) NOT NULL, birth_date DATE, death_date DATE,
      image_url VARCHAR(128) );", candidateFile, append=TRUE)
write("INSERT INTO candidate (name, birth_date, death_date, image_url)
      VALUES ", candidateFile, append=TRUE)
```

**Now we run through each candidate**

First we do a google search for "(candidate name) politician". From the google search we grab the first wikipedia link and visit the wikipedia page. From the wikipedia page the birth date, death date, and image url are extracted.

```
for (i in 1:length(candidates)) {
  webQueryString <- gsub(" ", "+", candidates[[1]])
  url <- sprintf("%s%s%s", baseUrl, webQueryString, "+politician/")
  print(sprintf("%s %d", candidates[[i]], i))
  print(url)
  html <- paste(readLines(url, encoding='UTF-8'), collapse="\n")

  # Because R is not good for web crawling, need to sleep or it will randomly fail frequently
  Sys.sleep(1)

  matched <-
    str_match_all(html, "en.wikipedia.org/wiki/([:alpha:]*\\.?_?\\(?\\)?%?[:digit:]?'?)*")
  wikiUrl <-
```

```
    sprintf("%s%s", "http://", matched[[1]][, 1][1]) # Get a url to wikipedia page
  # Handle the John Fremont outlier case which gets messed up because of character encoding
  wikiUrl <-
    gsub("%.*9", "%e9", wikiUrl)
  print(wikiUrl)
  wikihtml <- paste(readLines(wikiUrl), collapse="\n")
  # Get the birthday string if it exists
  bdaymatch <-
    str_match_all(wikihtml, "span class=\"bday\">(..........)</span")
  # Get the birthday string if it exists
  deathdaymatch <-
    str_match_all(wikihtml, "span class=\"dday deathdate\">(..........)</span")
  print(bdaymatch[[1]][, 2][1]) # Great these are already in database format
  print(deathdaymatch[[1]][, 2][1])
  #.*src=\"(.*)\"</span") # Get the main photo if it exists
  photoUrl <-
    str_match_all(wikihtml, "src=\"(//upload.*.jpg)\"")
  photo <- photoUrl[[1]][, 2][1]
  # If jpg extension has a slash after replace extension with @ symbol
  photo <- gsub("jpg/.*", "@", photo)
  photo <- gsub("@", "jpg", photo) # Replace @ symbol back with jpg
  photo <- gsub("/thumb", "", photo) # Remove thumb directory to get actual image link
  print(photo)

  # Some Final Sanitizing for putting into SQL format
  cleanCandidate <- gsub("\'", "\\\\\'", candidates[i]) # Escape ' character for sql
  if (!is.na(bdaymatch[[1]][, 2][1])) {
    bday <- paste("'",bdaymatch[[1]][, 2][1], "'", sep="")
  } else {
    bday <- "NULL"
  }
  if (!is.na(deathdaymatch[[1]][, 2][1])) {
    dday <- paste("'",deathdaymatch[[1]][, 2][1], "'", sep="")
  } else {
    dday <- "NULL"
  }
  if (!is.na(photo)) {
    photo <- paste("'",photo, "'", sep="")
  } else {
    photo <- "NULL"
  }

  # Now do the file writing
  if (i == length(candidates)) {
    write(sprintf("('%s', %s, %s, %s)",
                  cleanCandidate, bday, dday, photo), candidateFile, append=TRUE)
  } else {
    write(sprintf("('%s', %s, %s, %s),",
                  cleanCandidate, bday, dday, photo), candidateFile, append=TRUE)
  }

}
```

**The candidate2.txt file is manually edited into candidate_best.txt**

Some entries need to be edited because the dates were not found in parsing and tif images are replaced with jpgs from other sources.