Water Quality Data Cleanup

Brian High

05/21/2015

Objectives

Get drinking water data from WA DOH, clean, and convert to CSV and TSV formats for import into a database.

Setup

Java is required for package “XLConnect”. Make sure Java is installed.

```
if (system2("java","-version")) {
    stop("Java not found. Install Java first. https://java.com/en/download/")
}
```

Load the required R packages.

```
for (pkg in c("knitr", "hash", "rJava", "XLConnect", "dplyr", "ggmap")) {
    if (! suppressWarnings(require(pkg, character.only=TRUE)) ) {
        install.packages(pkg, repos="http://cran.fhcrc.org", dependencies=TRUE)
        if (! suppressWarnings(require(pkg, character.only=TRUE)) ) {
            stop(paste0(c("Can't load package: ", pkg, "!"), collapse = ""))
        }
    }
}
```

Configure `knitr` options.

```
opts_chunk$set(tidy=FALSE, cache=FALSE)
```

Create the data folder if needed.

```
datadir <- "data"
dir.create(file.path(datadir), showWarnings=FALSE, recursive=TRUE)
```

Water Systems and Sources Data

Get the water systems and sources data from WA DOH Water System Data for Download page (updated 2015-02-09).

```r
# Link title format: Group (A|B) (general|source) data (TXT, n KB)
urlbase='http://www.doh.wa.gov/portals/1/documents/4200/'
for (type in c('general','source')) {
    for (grp in c('a', 'b')) {
        datafile <- paste(c(grp, type, '.txt'), sep='', collapse='')
        datafileout <- paste(c(datadir, '/', grp, type, '.txt'),
                             sep='', collapse='')
        dataurl <- paste(c(urlbase, datafile), sep='', collapse='')
        if (! file.exists(datafileout)) {
            print("Downloading data file...")
            download.file(dataurl, datafileout)
        }
    }
}
```

Import Data

Import the data from text files into `data.frames`.

```r
tsv_import <- function(filename) {
    infile <- paste(c(datadir, '/', filename), sep='', collapse='')
    if (file.exists(infile)) {
        read.delim(infile, stringsAsFactors=FALSE, header=TRUE)
    }
    else {
        stop(paste("Can't find", filename, "in folder", datadir, "!", sep=" "))
    }
}

systema <- tsv_import('ageneral.txt')
systemb <- tsv_import('bgeneral.txt')
systems <- rbind(systema, systemb)
sourcea <- tsv_import('asource.txt')
sourceb <- tsv_import('bsource.txt')
sources <- rbind(sourcea, sourceb)
```

Convert Character Encoding

Use `sapply` and `iconv` to convert character encoding from latin1 to UTF-8.

We "normalize" on a common character set (UTF-8). This makes the resulting output files larger since more bytes will be used per character. The UTF-8 character set is the default for import into phpMyAdmin.

```r
systems <- as.data.frame(sapply(systems,
                                function(x) iconv(x, "latin1", "UTF-8")),
                         stringsAsFactors=FALSE)

sources <- as.data.frame(sapply(sources,
                                function(x) iconv(x, "latin1", "UTF-8")),
                         stringsAsFactors=FALSE)
```

Convert Case

Convert character values to upper-case.

```
systems <- as.data.frame(sapply(systems, toupper), stringsAsFactors=FALSE)
sources <- as.data.frame(sapply(sources, toupper), stringsAsFactors=FALSE)
```

Shorten Zip Codes

Shorten Zip Codes to 5 digits.

```
systems$WSZipCode <- sapply(systems$WSZipCode,
                            function(x) return(substr(x, start=1, stop=5)))
```

Fix Inconsistent City Names

Fix inconsistencies (i.e., typographical errors) in the PWSCity column. As many (over a hundred) were found, we have saved these in a CSV file. We read the search-replace terms as key-value pairs into a `hash` and loop through them to perform the replacements.

```
filename_prefix <- 'wa_doh_dw_'
typo_file <- paste(c(datadir, '/', filename_prefix, 'city_replace.csv'),
                        sep='', collapse='')

if (file.exists(typo_file)) {
    typo_df <- read.csv(typo_file, col.names=c("key", "value"),
                         header=FALSE, stringsAsFactors=FALSE)

    typo_hash <- hash(keys=typo_df$key, values=typo_df$value)

    for (typo in keys(typo_hash)) {
        systems$PWSCity[systems$PWSCity == typo] <- typo_hash[[typo]]
    }
}
```

You can visually inspect the city names and their frequency counts with these R commands:

```
by_city <- filter(systems, WSState=="WA") %>% group_by(PWSCity)
cities_count <- summarize(by_city, count = n())
View(cities_count)
```

Remove Thousands Separator

Remove commas from numeric columns (used for “thousands” separator).

```
remove_commas <- function(x) {
    as.numeric(gsub(",", "", x))
}

numeric_columns <- c("ResPop", "ResConn", "TotalConn", "ApprovSvcs")
systems[numeric_columns] <- lapply(systems[numeric_columns],
                                   function(x) remove_commas(x))
```

Replace Slashes

Replace back-slashes with forward-slashes in source name so it will not appear like an escape.

```r
sources$Src_Name <- gsub("([\\])","/", sources$Src_Name)
```

Convert Date Format

Convert dates to YYYY-MM-DD format.

```r
# Note: May have to convert in SQL after import, e.g.:
# SELECT "PWSID","SystemName","Group","County","OwnerTypeDesc","ResPop",
#        "ResConn","TotalConn","ApprovSvcs",
#        CAST([EffectiveDate] AS DATE) AS EffectiveDate,
#        "PWSAddress1","PWSCity","WSState","WSZipCode"
#        FROM [high@washington.edu].[table_wa_doh_dw_systems.tsv]
systems$EffectiveDate <- as.Date(systems$EffectiveDate, "%m/%d/%Y")
sources$Src_EffectieDate <- as.Date(sources$Src_EffectieDate, "%m/%d/%Y")
sources$SRC_InactiveDate <- as.Date(sources$SRC_InactiveDate, "%m/%d/%Y")
```

Define Export Functions

These helper functions will allow for cleaner, reusable export code. They create a CSV for import into phpMyAdmin and a TSV for import into SQLShare. The CSV is zipped to work around a file size limit with phpMyAdmin. phpMyAdmin takes a zipped CSV as an allowed file format with no extra effort on our part.

```r
# Function: Export to CSV and ZIP for phpMyAdmin (MySQL) import
# NOTE: *** Windows users will need RTools installed first. ***
#       See: http://cran.r-project.org/bin/windows/Rtools/index.html
ex_csv_zip <- function(df, filename) {
    # Write data to CSV, using \ to escape " (not per RFC 4180!) using
    # qmethod="escape" so that phpMyAdmin can import the CSV correctly.
    write.table(df, file = filename, fileEncoding="UTF-8",
                row.names=FALSE, quote=TRUE, sep=",", qmethod="escape")

    # Zip the CSV so that it will not exceed phpMyAdmin's upload size limit
    zip(paste(c(filename, '.zip'), sep='', collapse=''), filename)
}

# Function: Export to TSV for SQLShare import
ex_tsv <- function(df, filename) {
    # Write data to TSV, using \ to escape
    write.table(df, file = filename, fileEncoding="UTF-8",
                row.names=FALSE, quote=FALSE, sep="\t", qmethod="escape")
}
```

Export Systems and Sources data

We may use the CSV with phpMyAdmin or the TSV with SQLShare, so we will export both types of output.

```r
# Zipped CSV for pypMyAdmin
ex_csv_zip(systems, paste(c(datadir, '/', filename_prefix, 'systems.csv'),
                          sep='', collapse=''))
ex_csv_zip(sources, paste(c(datadir, '/', filename_prefix, 'sources.csv'),
                          sep='', collapse=''))

# TSV for SQLShare
ex_tsv(systems, paste(c(datadir, '/', filename_prefix, 'systems.tsv'),
                      sep='', collapse=''))
ex_tsv(sources, paste(c(datadir, '/', filename_prefix, 'sources.tsv'),
                      sep='', collapse=''))
```

Cleanup Fluoride Data

This is handled separately from the "systems" and "sources" since the the data file format is completely different.

Download Fluoride Data

Get the "Lists of Water Systems with fluoride (Excel, 06/13)" XLSX file from WA DOH Fluoride in Drinking Water page.

```r
datafile <- 'fluoride-data.xlsx'
datafileout <- paste(c(datadir, '/', datafile), sep='', collapse='')
dataurl <- paste(c(urlbase, datafile),sep='', collapse='')

if (! file.exists(datafileout)) {
    print("Downloading data file...")
    download.file(dataurl, datafileout, mode="wb")
}
```

Import Worksheet

We need to get each sheet individually, excluding the first and last few rows. Since we import the worksheets ignoring the header, we will need to specify the column names manually. We name them consistently so that all sheet have the same column names, filling empty columns with NA. Then we combine all rows into a single table, including a new column indicating the treatment type.

```r
# Import worksheet
if (file.exists(datafileout)) {
    # Treated: systems which adjust fluoride in water for dental benefits
    fluoride.trt <- readWorksheetFromFile(datafileout, sheet=1, header=FALSE,
                                          startRow=2, endRow=51)
    colnames(fluoride.trt) <- c("County", "PWSID", "SystemName", "Group", "ResPop")
    fluoride.trt <- mutate(fluoride.trt, County, PWSID, SystemName, CllctDate=NA,
                        mgL=NA, ResPop, Group, Treatment="treated")

    # Intertied: systems which receive fluoridated only, but do not adjust
    fluoride.tie <- readWorksheetFromFile(datafileout, sheet=2, header=FALSE,
                                          startRow=2, endRow=116)
    colnames(fluoride.tie) <- c("County", "PWSID", "SystemName", "Group", "ResPop")
    fluoride.tie <- mutate(fluoride.tie, County, PWSID, SystemName, CllctDate=NA,
                         mgL=NA, ResPop, Group, Treatment="intertied")

    # Mixed: unadjusted systems reeceiving fluoridated and unfluoridated water
    fluoride.mix <- readWorksheetFromFile(datafileout, sheet=3, header=FALSE,
                                          startRow=3, endRow=25)
    colnames(fluoride.mix) <- c("PWSID", "SystemName", "ResPop")
    fluoride.mix <- mutate(fluoride.mix, County=NA, PWSID, SystemName, CllctDate=NA,
                         mgL=NA, ResPop, Group=NA, Treatment="mixed")

    # Natural: unadjusted natural systems with fluoride levels >= 0.6 mg/L
    fluoride.nat <- readWorksheetFromFile(datafileout, sheet=4, header=FALSE,
                                          startRow=3, endRow=420)
    # Fix merged cell in last row, correcting an error in original worksheet
    if (fluoride.nat[418, 5] == "0.6           0") {
        fluoride.nat[418, 3] <- 'YAK CO - RAPTOR LANE WATER'
        fluoride.nat[418, 5] <- '0.6'
        fluoride.nat[418, 6] <- '0'
    }
    colnames(fluoride.nat) <- c("County", "PWSID", "SystemName", "CllctDate",
                         "mgL", "ResPop")
    fluoride.nat <- mutate(fluoride.nat, County, PWSID, SystemName, CllctDate,
                         mgL, ResPop, Group=NA, Treatment="natural")
    fluoride <- as.data.frame(rbind(fluoride.trt, fluoride.tie, fluoride.mix, fluoride.nat))
}
```

Apply Character Encoding

Use `sapply` and `iconv` to convert character encoding from latin1 to UTF-8.

```
fluoride <- as.data.frame(sapply(fluoride,
                               function(x) iconv(x, "latin1", "UTF-8")),
                        stringsAsFactors=FALSE)
```

Convert to Upper-Case

Convert character values to upper-case.

```
fluoride <- as.data.frame(sapply(fluoride, toupper), stringsAsFactors=FALSE)
```

Convert Numeric Columns

Convert numeric columns to the numeric data type.

```
numeric_columns <- c("mgL","ResPop")
fluoride[numeric_columns] <- lapply(fluoride[numeric_columns],
                                    function(x) as.numeric(x))
```

Export Fluoride

We may use the CSV with phpMyAdmin or the TSV with SQLShare, so we will export both types of output.

```
# Export data to Zipped CSV for import into phpMyAdmin
ex_csv_zip(fluoride, paste(c(datadir, '/', filename_prefix, 'fluoride.csv'),
                           sep='', collapse=''))

# Export data to TSV for import into UW SQLShare
ex_tsv(fluoride, paste(c(datadir, '/', filename_prefix, 'fluoride.tsv'),
                       sep='', collapse=''))
```

Geocode Water Systems

Get water system location coordinates from Google with `geocode` from the ggmap package. Since Google will not allow a non-commercial user to look up more than 1500 locations a day, we will not include the system address in our search. Instead, we will only search by city, state, and zipcode. Export as TSV and zipped CSV as with other data sources.

```
if (! file.exists(paste(c(datadir, '/', filename_prefix, 'locations.tsv'),
                        sep='', collapse=''))) {
    # Get locations from water systems data.frame and format as string
    locations <- filter(systems, WSState=="WA") %>%
        select(PWSCity, WSState, WSZipCode) %>% unique() %>%
        mutate(location = paste(PWSCity, WSState, WSZipCode, sep = ", "))

    # Get geocodes from Google's Geocode Service using `geocode` function
    lonlat <- geocode(locations$location, messaging=FALSE)
    locations <- data.frame(locations$PWSCity, locations$WSState,
                            locations$WSZipCode, lonlat$lon, lonlat$lat,
                            stringsAsFactors=FALSE)

    # Set column names on locations data.frame
    names(locations) <- c("PWSCity", "WSState", "WSZipCode", "lon", "lat")

    # Export data to Zipped CSV for import into phpMyAdmin
    ex_csv_zip(locations, paste(c(datadir, '/', filename_prefix,
                                  'locations.csv'), sep='', collapse=''))

    # Export data to TSV for import into UW SQLShare
    ex_tsv(locations, paste(c(datadir, '/', filename_prefix, 'locations.tsv'),
                            sep='', collapse=''))
}
```