Per Capita Metrics as Predictors of Tuberculosis Infection Rates: Data Acquisition

Author: James Topor

Contents

Introduction

This R Markdown file contains the R code required to collect, cleanse, and load the data required for the **CUNY MSDA 607 Spring 2016 Final Project** into a MySQL database. The output of this R Markdown file will be used for the analysis portion of the Final Project. Specifically, a separate R Markdown file named

"FP_Analysis.Rmd" will rely upon the results of the data acquisition, data cleansing, and database loading tasks performed within this R Markdown file.

Data for the **CUNY MSDA 607 Spring 2016 Final Project** are collected from several distinct sources:

1. A MySQL database table that tells us how many individuals of a given age category ('child', 'adult', or 'elderly') of a given gender were diagnosed with tuberculosis within a given country (100 countries in total) for a given year for the years 1995 - 2013.

2. A CSV file containing population counts for 100 countries for the years 1995 - 2013.

3. The World Bank Data web site, accessible at http://data.worldbank.org/. Four metrics were collected from that web site:

- Life Expectancy at Birth (in years) per Country;
- Health Care Expenditure Per Capita per Country;
- Gross National Income (GNI) Per Capita per Country;
- The percentage of a country's population having access to electricity.

4. The United Nations' Development Program's Human Development Reports web site, accessible at http://hdr.undp.org/. One metric was collected from that web site:

- Average years of Schooling per capita per Country.

These data sources each require a distinct data collection and data cleansing process, each of which will be explained below. Furthermore, each data source makes use of a distinct set of naming conventions for the various countries contained therein. For purposes of this project, the tuberculosis data stored within the MySQL database provides the "master" set of country names, and the country names used within the other data sets are modified to adhere to the naming conventions used within the tuberculosis data set. How these modifications are made is explained below within the respective data collection explanatory writeups.

IMPORTANT: This R code assumes that the user has the required 'tb_prediction' database installed on their own local MySQL server. If the user does not have the 'tb_prediction' database installed, please download the 'tb_db_createscript.sql' and 'tb.csv' files accessible via the following github links and install the database within your local MySQL environment, bearing in mind that you may need to alter the 'LOAD DATA INFILE' command given on Line 34 of that file so that it comports with your specific computing environment. The 'tb.csv' file contains the data that will populate the 'tb' database. The data are automatically loaded into the database via the SQL sript's 'LOAD DATA INFILE' command mentioned above.

Path to 'tb_db_createscript.sql' SQL script file:
https://raw.githubusercontent.com/jtopor/CUNY-MSDA-607/master/Final%20Project/tb_db_createscript.sql

Path to 'tb.csv' data file:
https://raw.githubusercontent.com/jtopor/CUNY-MSDA-607/master/Final%20Project/Data/tb.csv

---

R Packages Used

Several R packages are required for the collection, cleansing, and MySQL components used herein. Specifically, the **tidyr**, **dplyr**, **stringr**, **XML**, and **RODBC** packages must be installed within your local R environment prior to running the code contained within this R Markdown file. The **knitr** package is also required for

purposes of properly formatting the on-screen appearance of portions of the output of the processes contained herein. The packages are loaded as follows:

```
library(knitr)
library(tidyr, warn.conflicts = FALSE, quietly=TRUE)
library(dplyr, warn.conflicts = FALSE, quietly=TRUE)
library(stringr)
library(XML)
library(RODBC)
library(ggplot2)
options(stringsAsFactors = FALSE)
```

Calculating Tuberculosis Infection Rates by Country

As a first step we need to make use of both the population data and the tuberculosis data to derive the rate of tuberculosis infection for each country and for each year for which we have both population figures and tuberculosis infection counts. As mentioned above, we have data for both metrics for the years 1995 - 2013. Both data sets need to be loaded into R so that we can perform the data transformations and calculations required for deriving the desired metric.

Loading the Population Data

Annual population counts for each country are read from a .csv file given at:

https://raw.githubusercontent.com/jtopor/CUNY-MSDA-607/master/Final%20Project/Data/population.csv

This .csv file provides three columns:

1) Country name

2) Year

3) Population

The records within the file are pre-sorted by year and then alphabetically by country name. We start by reading the csv file and sorting the data we've read so that it is ordered by country and then by year:

```
# load the population.csv file from Github into a dataframe
# NOTE: The data has a 1 line header
popcsv <- read.csv("https://raw.githubusercontent.com/jtopor/CUNY-MSDA-607/master/Final%20Project/Data/p

str(popcsv)
```

```
## 'data.frame':    1900 obs. of  3 variables:
##  $ country   : chr  "Afghanistan" "Algeria" "Angola" "Argentina" ...
##  $ year      : int  1995 1995 1995 1995 1995 1995 1995 1995 1995 1995 ...
##  $ population: int  17586073 29315463 12104952 34833168 7770806 119869585 10189075 5985658 7635362 15
```

```
kable(head(popcsv))
```

country	year	population
Afghanistan	1995	17586073
Algeria	1995	29315463
Angola	1995	12104952
Argentina	1995	34833168
Azerbaijan	1995	7770806
Bangladesh	1995	119869585

```
# sort the data frame by country & year using the 'arrange' function from 'plyr'
popcsv <- arrange(popcsv, country, year)
kable(head(popcsv))
```

country	year	population
Afghanistan	1995	17586073
Afghanistan	1996	18415307
Afghanistan	1997	19021226
Afghanistan	1998	19496836
Afghanistan	1999	19987071
Afghanistan	2000	20595360

Loading the Tuberculosis Data from MySQL

The tuberculosis data is assumed to reside within a MySQL database. The R code given below makes use of R's **RODBC** package to provide access to connect to an SQL server and subsequently access a database on that server named **tb**. The tuberculosis data can be sorted and loaded into an R dataframe as follows:

```
# establish connection to local SQL server
# NOTE: Be sure to set the server reference appropriately to comport with your own local
# computing environment.
con <- odbcConnect("local_server")

# select a database to use - in this case, the 'tb_prediction' database
sqlQuery(con, "use tb_prediction")
```

```
## [1] "No Data"
```

```
tb_df <- sqlQuery(con, "SELECT country, year, SUM(child + adult + elderly) FROM tb
                        GROUP BY country, year ORDER BY country, year", stringsAsFactors=F)

# rename third column to meaningful name
colnames(tb_df)[3] <- "cases"
kable(head(tb_df))
```

country	year	cases
Afghanistan	1995	NA

country	year	cases
Afghanistan	1996	NA
Afghanistan	1997	128
Afghanistan	1998	1778
Afghanistan	1999	745
Afghanistan	2000	2666

Ensuring Country Names Match Across Data Sets

As we discovered in Assignment 3 earlier this semester, there is an inconsistency between the data sets for the country names used for the country of Ivory Coast. As such, we need to perform a bit of data cleansing to eliminate that inconsistency:

```
# get distinct country names from population data into char format for comparison
pop_country <- data.frame(unique(popcsv$country))
names(pop_country) <- 'country'

# get distinct country names from tb data
tb_country <- sqlQuery(con, "SELECT DISTINCT country FROM tb", stringsAsFactors=F)
names(tb_country) <- 'country'

# ----------------------------------------
# Now anti-join to check for mismatches
pop_nonmatch <- anti_join(pop_country, tb_country)
```

```
## Joining by: "country"
```

```
pop_nonmatch <- as.character(pop_nonmatch$country)
pop_nonmatch
```

```
## [1] "CÃ´te d'Ivoire"
```

```
tb_nonmatch <- anti_join(tb_country, pop_country)
```

```
## Joining by: "country"
```

```
tb_nonmatch <- as.character(tb_nonmatch$country)
tb_nonmatch
```

```
## [1] "Côte d'Ivoire"
```

```
##############################################
# Now Rename mismatched country names:
##############################################

# change Ivory Coast country name in popcsv to match that of tb data
popcsv$country[popcsv$country == pop_nonmatch] <- tb_nonmatch

# test to ensure there are now no mismatches between the two data sets
```

```
pop_country <- data.frame(unique(popcsv$country))
names(pop_country) <- 'country'
anti_join(pop_country, tb_country)
```

```
## Joining by: "country"
```

```
## [1] country
## <0 rows> (or 0-length row.names)
```

```
# re-sort popcsv to match tb data order
popcsv <- arrange(popcsv, country, year)
kable(head(popcsv))
```

country	year	population
Afghanistan	1995	17586073
Afghanistan	1996	18415307
Afghanistan	1997	19021226
Afghanistan	1998	19496836
Afghanistan	1999	19987071
Afghanistan	2000	20595360

Calculate TB Infection Rates & Store Results in MySQL

With the data cleansing complete we can proceed to calculate the tuberculosis infection rate metric we require and then write the results to a table in our MySQL database for later use:

```
tb_rates <- data.frame(popcsv$country, popcsv$year,
                       round((tb_df$cases / popcsv$population), 5))

# rename columns to meaningful names
names(tb_rates) <- c("Country", "Year", "Rate")

kable(head(tb_rates))
```

Country	Year	Rate
Afghanistan	1995	NA
Afghanistan	1996	NA
Afghanistan	1997	0.00001
Afghanistan	1998	0.00009
Afghanistan	1999	0.00004
Afghanistan	2000	0.00013

Ensure the country names match those used by R's geomapping tools:

```
# Ensure country names match those used by R's geomapping tools:

rmap_df <- sqlQuery(con, "SELECT * FROM rmap_lookup", stringsAsFactors=F)
```

```
for(i in 1:nrow(rmap_df)) {
  # change name of l_exp$country[l_exp$country == wb_country] <- tb_nin$country[i]
  tb_rates$Country[tb_rates$Country == rmap_df$tb_country[i]] <- rmap_df$rmap_country[i]
}
```

Write the dataframe containing the derived metric into a new table in MySQL for later use:

```
# load each item from the 'tb_rates' data frame into the tb_rates table
for(i in 1:nrow(tb_rates)) {

  # insert escape character if country name contains apostrophe
  ins_country <- as.character(gsub("'", "\\\\'", tb_rates$Country[i]) )

  # check to see if rate = NA due to lack of data. If so insert NULL for rate in database
  if (is.na(tb_rates$Rate[i])) {
    # format the required INSERT statement
    sql_stmt <- sprintf("INSERT INTO tb_rates (country, year, rate ) VALUES ('%s', %d, NULL)",
                        ins_country,
                        as.numeric(tb_rates$Year[i]) )
  } else {
    # format the required INSERT statement
    sql_stmt <- sprintf("INSERT INTO tb_rates (country, year, rate ) VALUES ('%s', %d, %f)",
                        ins_country,
                        as.numeric(tb_rates$Year[i]),
                        as.numeric(tb_rates$Rate[i]) )
  } # end if / else

  res <- sqlQuery(con, sql_stmt, errors= TRUE)

} # end for loop
```

Life Expectancy at Birth Data Loading

The life expectancy at birth data can be found at the following web link:

http://data.worldbank.org/indicator/SP.DYN.LE00.IN

The data contained therein can be downloaded in the form of a CSV file by clicking the "DOWNLOAD DATA" button located in the upper righthand side of that page and selecting the "CSV" option. A CSV file is then downloaded to your local machine. A copy of that CSV file is available at the following Github link:

https://raw.githubusercontent.com/jtopor/CUNY-MSDA-607/master/Final%20Project/Data/LifeExpec.csv

We start by loading that CSV file into R:

```
# load csv file
l_e.raw <- read.csv("https://raw.githubusercontent.com/jtopor/CUNY-MSDA-607/master/Final%20Project/Data/

names(l_e.raw)
```

```
##  [1] "Country.Name"   "Country.Code"   "Indicator.Name" "Indicator.Code"
##  [5] "X1960"          "X1961"          "X1962"          "X1963"
##  [9] "X1964"          "X1965"          "X1966"          "X1967"
## [13] "X1968"          "X1969"          "X1970"          "X1971"
## [17] "X1972"          "X1973"          "X1974"          "X1975"
## [21] "X1976"          "X1977"          "X1978"          "X1979"
## [25] "X1980"          "X1981"          "X1982"          "X1983"
## [29] "X1984"          "X1985"          "X1986"          "X1987"
## [33] "X1988"          "X1989"          "X1990"          "X1991"
## [37] "X1992"          "X1993"          "X1994"          "X1995"
## [41] "X1996"          "X1997"          "X1998"          "X1999"
## [45] "X2000"          "X2001"          "X2002"          "X2003"
## [49] "X2004"          "X2005"          "X2006"          "X2007"
## [53] "X2008"          "X2009"          "X2010"          "X2011"
## [57] "X2012"          "X2013"          "X2014"          "X2015"
```

As shown in the output of R's **names** function above, there are more than 50 columns worth of data that have been read in from the CSV file. However, most of those columns are not of interest to us: we are only interested in the country name and the columns containing data for the years 1995 - 2013. As such, we can discard the rest of the columns:

```
# create a new data frame using only those columns that match up to TB data
l_exp <- data.frame(l_e.raw$Country.Name,
                    l_e.raw$X1995,
                    l_e.raw$X1996,
                    l_e.raw$X1997,
                    l_e.raw$X1998,
                    l_e.raw$X1999,
                    l_e.raw$X2000,
                    l_e.raw$X2001,
                    l_e.raw$X2002,
                    l_e.raw$X2003,
                    l_e.raw$X2004,
                    l_e.raw$X2005,
                    l_e.raw$X2006,
                    l_e.raw$X2007,
                    l_e.raw$X2008,
                    l_e.raw$X2009,
                    l_e.raw$X2010,
                    l_e.raw$X2011,
                    l_e.raw$X2012,
                    l_e.raw$X2013)

# apply meaningful column names that will conform with TB data format
names(l_exp)<-c("country", "x1995", "x1996", "x1997", "x1998", "x1999", "x2000",
                           "x2001", "x2002", "x2003", "x2004", "x2005",
                           "x2006", "x2007", "x2008", "x2009", "x2010",
                           "x2011", "x2012", "x2013")
# display
kable(head(l_exp), padding=0)
```

country	x1995	x1996	x1997	x1998	x1999	x2000	x2001	x2002	x2003	x200
Aruba	73.56361	73.58944	73.61480	73.64229	73.67539	73.72061	73.78290	73.86524	73.96856	74.0908

country	x1995	x1996	x1997	x1998	x1999	x2000	x2001	x2002	x2003	x200
Andorra	NA	NA	NA	NA	NA	NA	NA	NA	NA	N
Afghanistan	53.13939	53.60205	54.01783	54.40273	54.76727	55.12588	55.48754	55.85720	56.23529	56.626
Angola	42.05146	42.50163	43.06212	43.71427	44.43900	45.20478	45.97315	46.71173	47.39317	48.003
Albania	72.23602	72.52741	72.89810	73.32659	73.79339	74.27154	74.73256	75.15461	75.52329	75.831
Arab World	66.14346	66.48454	66.81624	67.11727	67.40314	67.65187	67.88562	68.10399	68.31517	68.534

```
# discard original data frame - no longer needed
rm(l_e.raw)
```

Ensuring Country Names Match Those Found in TB Data

As discussed above, the data sources used herein each make use of varying naming conventions for the country names contained within their data sets. Therefore, we need to ensure that any disparate country names are modified to conform with those found within the tuberculosis data set. Unfortunately, no programmatic solution could be identified that was capable of addressing all of the various disparities we can find between the country names in these data sets. As such, we are forced to instead create hard-coded lookup tables within our MySQL database that can be used for purposes of ensuring common country names are applied across our various data sets.

One such lookup table was created for the metrics obtained via the World Bank Data web site. Fortunately, we found that each data set obtained from that web site had the exact same set of inconsistencies in country names when compared to the country names used within the tuberculosis data set.

To address the issue of inconsistent names we first need to identify the country names contained within the tuberculosis data set that do not appear within one of the World Bank data sets. We'll use the life expectancy data since we're in the process of loading it just now.

```
# USE tb_country data frame from above for unique country names

lex_country <- data.frame(l_exp$country)
names(lex_country) <- 'country'

# get names of countries in TB data that ARE NOT in l_exp data
tb_nin <- data.frame(tb_country$country[!(tb_country$country %in% lex_country$country) ] )
names(tb_nin) <- 'country'
tb_nin
```

```
##                                          country
## 1                    Bolivia (Plurinational State of)
## 2                                China, Hong Kong SAR
## 3                                               Congo
## 4                                       Côte d'Ivoire
## 5              Democratic People's Republic of Korea
## 6                   Democratic Republic of the Congo
## 7                                               Egypt
## 8                         Iran (Islamic Republic of)
## 9                                          Kyrgyzstan
## 10                   Lao People's Democratic Republic
## 11                                  Republic of Korea
## 12                                  Republic of Moldova
## 13 United Kingdom of Great Britain and Northern Ireland
```

```
## 14                          United Republic of Tanzania
## 15                             United States of America
## 16                  Venezuela (Bolivarian Republic of)
## 17                                           Viet Nam
## 18                                              Yemen
```

We appear to have 18 country names within the life expectancy data set that do not match the corresponding country names within the TB data set. Unfortunately, as mentioned above, there seems to be no way to fully automate the process of resolving the differences between names. Finding the inconsistencies requires reading each individual country name listed above, extracting the 'most relevant' aspect therein by hand, and using that aspect to locate a similar country name within the life expectancy data.

The R code shown below was used for purposes of that manual process. The outputs of the R code were used to create and populate a lookup table within the SQL script that is described above. The lookup table is automatically created and populated when the script is executed within MySQL Workbench.

```r
# R Code used to facilitate creation of 'cn_lookup' table as defined in the required SQL script

# Find Bolivia
for(i in 1:nrow(l_exp)) {
  if(str_detect(l_exp$country[i], 'Bolivia') == TRUE) print(i)
}

l_exp$country[27]


# Find Hong Kong
for(i in 1:nrow(l_exp)) {
  if(str_detect(l_exp$country[i], 'Hong Kong') == TRUE) print(i)
}

l_exp$country[92]

# Find Congo
for(i in 1:nrow(l_exp)) {
  if(str_detect(l_exp$country[i], 'Congo') == TRUE) print(i)
}

l_exp$country[42]


# find Côte d'Ivoire
for(i in 1:nrow(l_exp)) {
  if(str_detect(l_exp$country[i], "Cote d'Ivoire") == TRUE) print(i)
}
l_exp$country[40]


# Find Democratic People's Republic of Korea
for(i in 1:nrow(l_exp)) {
  if(str_detect(l_exp$country[i], "Korea") == TRUE) print(i)
}
l_exp$country[184]
```

```r
# Find  Democratic Republic of the Congo
for(i in 1:nrow(l_exp)) {
  if(str_detect(l_exp$country[i], "Congo") == TRUE) print(i)
}
l_exp$country[246]

# Find  Egypt
for(i in 1:nrow(l_exp)) {
  if(str_detect(l_exp$country[i], "Egypt") == TRUE) print(i)
}
l_exp$country[64]


# Find  Iran (Islamic Republic of)
for(i in 1:nrow(l_exp)) {
  if(str_detect(l_exp$country[i], "Iran") == TRUE) print(i)
}
l_exp$country[103]


# Find Kyrgyzstan
for(i in 1:nrow(l_exp)) {
  if(str_detect(l_exp$country[i], "Kyrgyz") == TRUE) print(i)
}
l_exp$country[113]


# Find Lao People's Democratic Republic
for(i in 1:nrow(l_exp)) {
  if(str_detect(l_exp$country[i], "Lao") == TRUE) print(i)
}
l_exp$country[121]

# Find Republic of Korea
for(i in 1:nrow(l_exp)) {
  if(str_detect(l_exp$country[i], "Korea") == TRUE) print(i)
}
l_exp$country[117]

# Find Republic of Moldova
for(i in 1:nrow(l_exp)) {
  if(str_detect(l_exp$country[i], "Moldova") == TRUE) print(i)
}
l_exp$country[141]

# Find United Kingdom of Great Britain and Northern Ireland
for(i in 1:nrow(l_exp)) {
  if(str_detect(l_exp$country[i], "United Kingdom") == TRUE) print(i)
}
l_exp$country[78]

# Find United Republic of Tanzania
for(i in 1:nrow(l_exp)) {
```

```
  if(str_detect(l_exp$country[i], "Tanzania") == TRUE) print(i)
}
l_exp$country[229]

# Find United States of America
for(i in 1:nrow(l_exp)) {
  if(str_detect(l_exp$country[i], "United States") == TRUE) print(i)
}
l_exp$country[234]

# Find Venezuela (Bolivarian Republic of)
for(i in 1:nrow(l_exp)) {
  if(str_detect(l_exp$country[i], "Venezuela") == TRUE) print(i)
}
l_exp$country[237]

# Find Viet Nam
for(i in 1:nrow(l_exp)) {
  if(str_detect(l_exp$country[i], "Viet") == TRUE) print(i)
}
l_exp$country[239]

# Find Yemen
for(i in 1:nrow(l_exp)) {
  if(str_detect(l_exp$country[i], "Yemen") == TRUE) print(i)
}
l_exp$country[244]
```

Since the lookup table is created automatically within the MySQL database when the database creation script is executed, we can now perform the required lookup task and assign conforming country names to the 18 non-conforming country names we found within the life expectancy data:

```
dummy <- l_exp

# take 18 country names in tb_nin
for(i in 1:nrow(tb_nin)) {
  # insert escape character if country name contains apostrophe
  s_country <- as.character(gsub("'", "\\\\'", tb_nin$country[i]) )

  # for each tb country name, look up wb_country
  sql_stmt <- sprintf("SELECT wb_country FROM cn_lookup WHERE tb_country = '%s' ", s_country)

  res <- as.character(sqlQuery(con, sql_stmt, errors= TRUE))

  # change name of l_exp$country[l_exp$country == wb_country] <- tb_nin$country[i]
  dummy$country[dummy$country == res] <- tb_nin$country[i]

}
```

Discarding Country Data Not Relevant to Tuberculosis Data Set

The data sets obtained via the World Bank Data web site each appear to contain 248 rows of data. However, we are only interested in data pertaining to the 100 countries specified within the tuberculosis data set.

Therefore, we can discard all other World Bank data from our life expectancy data frame:

```
# discard unused World Bank data
new_df <- data.frame(dummy[ which( dummy$country %in% tb_country$country ), ])
nrow(new_df)
```

```
## [1] 100
```

```
kable(head(new_df))
```

	country	x1995	x1996	x1997	x1998	x1999	x2000	x2001	x2002	x2003	
3	Afghanistan	53.13939	53.60205	54.01783	54.40273	54.76727	55.12588	55.48754	55.85720	56.23529	56
4	Angola	42.05146	42.50163	43.06212	43.71427	44.43900	45.20478	45.97315	46.71173	47.39317	48
8	Argentina	72.66512	72.88734	73.10805	73.32768	73.54427	73.75580	73.96032	74.15578	74.34127	74
14	Azerbaijan	64.57583	64.99400	65.46239	65.92337	66.35549	66.75824	67.15029	67.56107	68.00368	68
15	Burundi	48.80078	49.46261	50.11985	50.69205	51.14924	51.48549	51.72483	51.92827	52.14532	52
17	Benin	54.81595	54.83180	54.84232	54.89015	54.99878	55.18824	55.46649	55.81432	56.20807	56

Ensure the country names match R's geoplotting tools:

```
for(i in 1:nrow(rmap_df)) {
  # Ensure the country names match R's geoplotting tools
  new_df$country[new_df$country == rmap_df$tb_country[i]] <- rmap_df$rmap_country[i]
}
```

Data Tidying

The World Bank data is presented in "wide" format: each row represents a single country and the columns contain the life expectancy metrics for the years 1995 - 2013. As such, we need to transform this data to "long" format where each row represents a single instance of a life expectancy metric: each row needs to have only a country name, year, and corresponding life expectancy estimate. We therefore make use of R's **gather** function to rearrange the data, and then remove the leading "x" characters from each year value:

```
# gather the 'year' columns
le_df2 <- gather(new_df, year, life_exp, x1995:x2013, na.rm = FALSE)
kable(head(le_df2))
```

country	year	life_exp
Afghanistan	x1995	53.13939
Angola	x1995	42.05146
Argentina	x1995	72.66512
Azerbaijan	x1995	64.57583
Burundi	x1995	48.80078
Benin	x1995	54.81595

```
# clean the "Year" column of the letter 'x'
le_df2$year <- str_extract(le_df2$year, "[[:digit:]]{4}")
```

```
kable(head(le_df2))
```

country	year	life_exp
Afghanistan	1995	53.13939
Angola	1995	42.05146
Argentina	1995	72.66512
Azerbaijan	1995	64.57583
Burundi	1995	48.80078
Benin	1995	54.81595

```
# sort the data so it is ordered by country then year
le_df2 <- arrange(le_df2, country, year)
kable(head(le_df2))
```

country	year	life_exp
Afghanistan	1995	53.13939
Afghanistan	1996	53.60205
Afghanistan	1997	54.01783
Afghanistan	1998	54.40273
Afghanistan	1999	54.76727
Afghanistan	2000	55.12588

Loading the Refined Life Expectancy Data into MySQL

Now that the data has been transformed and tidied we can load it into a pre-defined table within MySQL for future use:

```
# load each item from the 'le_df2' data frame into the life_exp table
for(i in 1:nrow(le_df2)) {

  # insert escape character if country name contains apostrophe
  ins_country <- as.character(gsub("'", "\\\\'", le_df2$country[i]) )

  # check to see if rate = NA due to lack of data. If so insert NULL for rate in database
  if (is.na(le_df2$life_exp[i])) {
    # format the required INSERT statement
    sql_stmt <- sprintf("INSERT INTO life_exp (country, year, life_exp )
                        VALUES ('%s', %d, NULL)",
                        ins_country,
                        as.numeric(le_df2$year[i]) )
  } else {
    # format the required INSERT statement
    sql_stmt <- sprintf("INSERT INTO life_exp (country, year, life_exp )
                        VALUES ('%s', %d, %f)",
                        ins_country,
                        as.numeric(le_df2$year[i]),
                        as.numeric(le_df2$life_exp[i]) )
  } # end if / else
```

```
  res <- sqlQuery(con, sql_stmt, errors= TRUE)

} # end for loop
```

And finally we'll do some memory cleanup as follows:

```
# how to remove unneeded objects in R

rm(life_exp)
```

```
## Warning in rm(life_exp): object 'life_exp' not found
```

Health Care Expenditure Per Capita Data Loading

The healtch care expenditure per capita data can be found at the following web link:

http://data.worldbank.org/indicator/SH.XPD.PCAP

The data contained therein can be downloaded in the form of a CSV file by clicking the "DOWNLOAD DATA" button located in the upper righthand side of that page and selecting the "CSV" option. A CSV file is then downloaded to your local machine. A copy of that CSV file is available at the following Github link:

https://raw.githubusercontent.com/jtopor/CUNY-MSDA-607/master/Final%20Project/Data/HC_Spending.csv

The process used for reading, transforming, tidying, and loading the health care expenditure data is very similar to that used for the life expectancy data discussed above. Therefore, we will not repeat the narrative portion of that writeup here.

```
# load csv file
options(stringsAsFactors = FALSE)
hc.raw <- read.csv("https://raw.githubusercontent.com/jtopor/CUNY-MSDA-607/master/Final%20Project/Data/

names(hc.raw)
```

```
##  [1] "Country.Name"   "Country.Code"   "Indicator.Name" "Indicator.Code"
##  [5] "X1960"          "X1961"          "X1962"          "X1963"
##  [9] "X1964"          "X1965"          "X1966"          "X1967"
## [13] "X1968"          "X1969"          "X1970"          "X1971"
## [17] "X1972"          "X1973"          "X1974"          "X1975"
## [21] "X1976"          "X1977"          "X1978"          "X1979"
## [25] "X1980"          "X1981"          "X1982"          "X1983"
## [29] "X1984"          "X1985"          "X1986"          "X1987"
## [33] "X1988"          "X1989"          "X1990"          "X1991"
## [37] "X1992"          "X1993"          "X1994"          "X1995"
## [41] "X1996"          "X1997"          "X1998"          "X1999"
## [45] "X2000"          "X2001"          "X2002"          "X2003"
## [49] "X2004"          "X2005"          "X2006"          "X2007"
## [53] "X2008"          "X2009"          "X2010"          "X2011"
## [57] "X2012"          "X2013"          "X2014"          "X2015"
```

Discard unneeded columns:

```
# create a new data frame using only those columns that match up to TB data
hc.df <- data.frame(hc.raw$Country.Name,
                        hc.raw$X1995,
                        hc.raw$X1996,
                        hc.raw$X1997,
                        hc.raw$X1998,
                        hc.raw$X1999,
                        hc.raw$X2000,
                        hc.raw$X2001,
                        hc.raw$X2002,
                        hc.raw$X2003,
                        hc.raw$X2004,
                        hc.raw$X2005,
                        hc.raw$X2006,
                        hc.raw$X2007,
                        hc.raw$X2008,
                        hc.raw$X2009,
                        hc.raw$X2010,
                        hc.raw$X2011,
                        hc.raw$X2012,
                        hc.raw$X2013)

# apply meaningful column names that will conform with TB data format
names(hc.df)<-c("country", "x1995", "x1996", "x1997", "x1998", "x1999", "x2000",
                           "x2001", "x2002", "x2003", "x2004", "x2005",
                           "x2006", "x2007", "x2008", "x2009", "x2010",
                           "x2011", "x2012", "x2013")
# display
kable(head(hc.df), padding=0)
```

country	x1995	x1996	x1997	x1998	x1999	x2000	x2001	x2002
Aruba	NA	NA	NA	NA	NA	NA	NA	NA
Andorra	1282.52259	1386.66115	1395.52150	1831.20947	1369.16128	1242.21098	1268.63335	1450.79647
Afghanistan	NA	NA	NA	NA	NA	NA	NA	15.48190
Angola	18.00618	13.75622	19.27912	15.14206	14.23259	16.89917	30.91774	28.19060
Albania	51.48380	72.42533	52.94162	61.25168	75.95312	74.02882	78.66820	89.91616
Arab World	85.72369	84.10694	89.95608	91.81946	99.32991	105.97744	108.30153	109.02299

```
# discard original data frame - no longer needed
rm(hc.raw)
```

Rename countries not matching between data sets:

```
dummy <- hc.df

# take 18 country names in tb_nin
for(i in 1:nrow(tb_nin)) {
  # insert escape character if country name contains apostrophe
  s_country <- as.character(gsub("'", "\\\\'", tb_nin$country[i]) )
```

```
  # for each tb country name, look up wb_country
  sql_stmt <- sprintf("SELECT wb_country FROM cn_lookup WHERE tb_country = '%s' ", s_country)

  # res <- as.character(sqlQuery(con, sql_stmt, errors= TRUE))
  res <- sqlQuery(con, sql_stmt, errors= TRUE, stringsAsFactors = FALSE)

  # change name of l_exp$country[l_exp$country == wb_country] <- tb_nin$country[i]
  dummy$country[dummy$country == res$wb_country] <- tb_nin$country[i]

}
```

Create a new data frame that discards all non-matched country names from hc.df:

```
# Create a new data frame that discards all non-matched country names from hc.df:

new_df <- data.frame(dummy[ which( dummy$country %in% tb_country$country ), ])
nrow(new_df)
```

```
## [1] 100
```

```
kable(head(new_df))
```

	country	x1995	x1996	x1997	x1998	x1999	x2000	x2001	x2002
3	Afghanistan	NA	NA	NA	NA	NA	NA	NA	15.48190
4	Angola	18.006184	13.75622	19.279119	15.142060	14.232591	16.899173	30.917745	28.19060
8	Argentina	612.958243	616.81447	683.687951	704.460658	724.111277	706.904994	672.652327	223.90609
14	Azerbaijan	18.019157	24.77839	27.266537	30.267665	30.495468	30.297526	31.169348	33.61684
15	Burundi	8.610735	7.57429	7.651652	7.582356	6.577176	6.403948	6.261189	5.94115
17	Benin	17.009609	17.53351	15.374934	16.447469	16.936695	14.728821	16.325491	16.18551

Ensure country names match R's geoplotting tools:

```
for(i in 1:nrow(rmap_df)) {
  # Ensure country names match R's geoplotting tools
  new_df$country[new_df$country == rmap_df$tb_country[i]] <- rmap_df$rmap_country[i]
}
```

Transform data to long format:

```
# gather the 'year' columns
hc.df2 <- gather(new_df, year, percap_hc, x1995:x2013, na.rm = FALSE)
kable(head(hc.df2))
```

country	year	percap_hc
Afghanistan	x1995	NA
Angola	x1995	18.006184
Argentina	x1995	612.958243
Azerbaijan	x1995	18.019157
Burundi	x1995	8.610735
Benin	x1995	17.009609

```
# clean the "Year" column of the letter 'x'
hc.df2$year <- str_extract(hc.df2$year, "[[:digit:]]{4}")
kable(head(hc.df2))
```

country	year	percap_hc
Afghanistan	1995	NA
Angola	1995	18.006184
Argentina	1995	612.958243
Azerbaijan	1995	18.019157
Burundi	1995	8.610735
Benin	1995	17.009609

```
# sort the data so it is ordered by country then year
hc.df2 <- arrange(hc.df2, country, year)
kable(head(hc.df2))
```

country	year	percap_hc
Afghanistan	1995	NA
Afghanistan	1996	NA
Afghanistan	1997	NA
Afghanistan	1998	NA
Afghanistan	1999	NA
Afghanistan	2000	NA

Load refined data into pre-defined MySQL table:

```
# load each item from the 'hc.df2' data frame into the percap_hc table
for(i in 1:nrow(hc.df2)) {

  # insert escape character if country name contains apostrophe
  ins_country <- as.character(gsub("'", "\\\\'", hc.df2$country[i]) )

  # check to see if rate = NA due to lack of data. If so insert NULL for rate in database
  if (is.na(hc.df2$percap_hc[i])) {
    # format the required INSERT statement
    sql_stmt <- sprintf("INSERT INTO percap_hc (country, year, percap_hc )
                        VALUES ('%s', %d, NULL)",
                        ins_country,
                        as.numeric(hc.df2$year[i]) )
  } else {
    # format the required INSERT statement
    sql_stmt <- sprintf("INSERT INTO percap_hc (country, year, percap_hc )
                        VALUES ('%s', %d, %f)",
                        ins_country,
                        as.numeric(hc.df2$year[i]),
                        as.numeric(hc.df2$percap_hc[i]) )
  } # end if / else

  res <- sqlQuery(con, sql_stmt, errors= TRUE)
```

```
} # end for loop

# cleanup data frames
rm(hc.df)
rm(hc.df2)
```

Gross National Income (GNI) Per Capita Data Loading

The gross national income per capita data can be found at the following web link:

http://data.worldbank.org/indicator/NY.GNP.PCAP.PP.CD

The data contained therein can be downloaded in the form of a CSV file by clicking the "DOWNLOAD DATA" button located in the upper righthand side of that page and selecting the "CSV" option. A CSV file is then downloaded to your local machine. A copy of that CSV file is available at the following Github link:

https://raw.githubusercontent.com/jtopor/CUNY-MSDA-607/master/Final%20Project/Data/GNI.csv

The process used for reading, transforming, tidying, and loading the gross national income data is very similar to that used for the life expectancy data discussed above. Therefore, we will not repeat the narrative portion of that writeup here.

```
# load csv file
options(stringsAsFactors = FALSE)
d.raw <- read.csv("https://raw.githubusercontent.com/jtopor/CUNY-MSDA-607/master/Final%20Project/Data/GI

names(d.raw)
```

```
##  [1] "Country.Name"   "Country.Code"   "Indicator.Name" "Indicator.Code"
##  [5] "X1960"          "X1961"          "X1962"          "X1963"
##  [9] "X1964"          "X1965"          "X1966"          "X1967"
## [13] "X1968"          "X1969"          "X1970"          "X1971"
## [17] "X1972"          "X1973"          "X1974"          "X1975"
## [21] "X1976"          "X1977"          "X1978"          "X1979"
## [25] "X1980"          "X1981"          "X1982"          "X1983"
## [29] "X1984"          "X1985"          "X1986"          "X1987"
## [33] "X1988"          "X1989"          "X1990"          "X1991"
## [37] "X1992"          "X1993"          "X1994"          "X1995"
## [41] "X1996"          "X1997"          "X1998"          "X1999"
## [45] "X2000"          "X2001"          "X2002"          "X2003"
## [49] "X2004"          "X2005"          "X2006"          "X2007"
## [53] "X2008"          "X2009"          "X2010"          "X2011"
## [57] "X2012"          "X2013"          "X2014"          "X2015"
```

Discard unneeded columns:

```
# create a new data frame using only those columns that match up to TB data
d.df <- data.frame(d.raw$Country.Name,
                       d.raw$X1995,
                       d.raw$X1996,
```

```
                        d.raw$X1997,
                        d.raw$X1998,
                        d.raw$X1999,
                        d.raw$X2000,
                        d.raw$X2001,
                        d.raw$X2002,
                        d.raw$X2003,
                        d.raw$X2004,
                        d.raw$X2005,
                        d.raw$X2006,
                        d.raw$X2007,
                        d.raw$X2008,
                        d.raw$X2009,
                        d.raw$X2010,
                        d.raw$X2011,
                        d.raw$X2012,
                        d.raw$X2013)

# apply meaningful column names that will conform with TB data format
names(d.df)<-c("country", "x1995", "x1996", "x1997", "x1998", "x1999", "x2000",
                          "x2001", "x2002", "x2003", "x2004", "x2005",
                          "x2006", "x2007", "x2008", "x2009", "x2010",
                          "x2011", "x2012", "x2013")
# display
kable(head(d.df), padding=0)
```

country	x1995	x1996	x1997	x1998	x1999	x2000	x2001	x2002	x2003	x2004
Aruba	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
Andorra	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
Afghanistan	NA	NA	NA	NA	NA	NA	NA	890.00	940.000	940.00
Angola	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
Albania	2860.000	3210.000	2930.000	3270.000	3770.00	4160.000	4650.0	4900.00	5320.000	5790.00
Arab World	7416.151	7717.187	8051.912	8512.477	8701.42	9033.871	9284.4	9248.64	9523.662	10523.04

```
# discard original data frame - no longer needed
rm(d.raw)
```

Rename countries not matching between data sets:

```
# Rename countries not matching between data sets
dummy <- d.df

# take 18 country names in tb_nin
for(i in 1:nrow(tb_nin)) {
  # insert escape character if country name contains apostrophe
  s_country <- as.character(gsub("'", "\\\\'", tb_nin$country[i]) )

  # for each tb country name, look up wb_country
  sql_stmt <- sprintf("SELECT wb_country FROM cn_lookup WHERE tb_country = '%s' ", s_country)

  res <- as.character(sqlQuery(con, sql_stmt, errors= TRUE))
```

```
  # change name of l_exp$country[l_exp$country == wb_country] <- tb_nin$country[i]
  dummy$country[dummy$country == res] <- tb_nin$country[i]

}
```

Create a new data frame that discards all non-matched country names from d.df:

```
# Create a new data frame that discards all non-matched country names from d.df
new_df <- data.frame(dummy[ which( dummy$country %in% tb_country$country ), ])
nrow(new_df)
```

```
## [1] 100
```

```
kable(head(new_df))
```

	country	x1995	x1996	x1997	x1998	x1999	x2000	x2001	x2002	x2003	x2004	x2005	x2006	x2
3	Afghanistan	NA	NA	NA	NA	NA	NA	NA	890	940	940	1040	1100	1
4	Angola	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	
8	Argentina	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	
14	Azerbaijan	2410	2420	2630	2890	3110	3340	3710	4120	4650	5100	6280	8570	10
15	Burundi	600	550	550	570	560	570	570	580	560	580	590	630	
17	Benin	1070	1100	1160	1190	1240	1300	1350	1390	1410	1470	1500	1560	1

Ensure country names match R's geoplotting tools:

```
for(i in 1:nrow(rmap_df)) {
  # Ensure country names match R's geoplotting tools
  new_df$country[new_df$country == rmap_df$tb_country[i]] <- rmap_df$rmap_country[i]
}
```

Transform Data to long format:

```
# gather the 'year' columns
d.df2 <- gather(new_df, year, percap_gni, x1995:x2013, na.rm = FALSE)
kable(head(d.df2))
```

country	year	percap_gni
Afghanistan	x1995	NA
Angola	x1995	NA
Argentina	x1995	NA
Azerbaijan	x1995	2410
Burundi	x1995	600
Benin	x1995	1070

```
# clean the "Year" column of the letter 'x'
d.df2$year <- str_extract(d.df2$year, "[[:digit:]]{4}")
kable(head(d.df2))
```

country	year	percap_gni
Afghanistan	1995	NA
Angola	1995	NA
Argentina	1995	NA
Azerbaijan	1995	2410
Burundi	1995	600
Benin	1995	1070

```
# sort the data so it is ordered by country then year
d.df2 <- arrange(d.df2, country, year)
kable(head(d.df2))
```

country	year	percap_gni
Afghanistan	1995	NA
Afghanistan	1996	NA
Afghanistan	1997	NA
Afghanistan	1998	NA
Afghanistan	1999	NA
Afghanistan	2000	NA

Load refined data into pre-defined MySQL table:

```
# load each item from the d.df2 data frame into the percap_gni table
for(i in 1:nrow(d.df2)) {

  # insert escape character if country name contains apostrophe
  ins_country <- as.character(gsub("'", "\\\\'", d.df2$country[i]) )

  # check to see if rate = NA due to lack of data. If so insert NULL for rate in database
  if (is.na(d.df2$percap_gni[i])) {
    # format the required INSERT statement
    sql_stmt <- sprintf("INSERT INTO percap_gni (country, year, percap_gni )
                         VALUES ('%s', %d, NULL)",
                         ins_country,
                         as.numeric(d.df2$year[i]) )
  } else {
    # format the required INSERT statement
    sql_stmt <- sprintf("INSERT INTO percap_gni (country, year, percap_gni )
                         VALUES ('%s', %d, %f)",
                         ins_country,
                         as.numeric(d.df2$year[i]),
                         as.numeric(d.df2$percap_gni[i]) )
  } # end if / else

  res <- sqlQuery(con, sql_stmt, errors= TRUE)

} # end for loop

# cleanup data frames
rm(d.df)
rm(d.df2)
```

Access to Electricity Data Loading

Access to electricity is defined as the percentage of a country's population having access to electricity.

NOTE: Metrics for access to electricity are limited to 3 years: 2000, 2010, 2012

The access to electricity data can be found at the following web link:

http://data.worldbank.org/indicator/EG.ELC.ACCS.ZS?page=1

The data contained therein can be downloaded in the form of a CSV file by clicking the "DOWNLOAD DATA" button located in the upper righthand side of that page and selecting the "CSV" option. A CSV file is then downloaded to your local machine. A copy of that CSV file is available at the following Github link:

https://raw.githubusercontent.com/jtopor/CUNY-MSDA-607/master/Final%20Project/Data/ElecAccess.csv

The process used for reading, transforming, tidying, and loading the gross national income data is very similar to that used for the life expectancy data discussed above. Therefore, we will not repeat the narrative portion of that writeup here.

```
# load csv file
options(stringsAsFactors = FALSE)
d.raw <- read.csv("https://raw.githubusercontent.com/jtopor/CUNY-MSDA-607/master/Final%20Project/Data/E

names(d.raw)
```

```
##  [1] "Country.Name"   "Country.Code"   "Indicator.Name" "Indicator.Code"
##  [5] "X1960"          "X1961"          "X1962"          "X1963"
##  [9] "X1964"          "X1965"          "X1966"          "X1967"
## [13] "X1968"          "X1969"          "X1970"          "X1971"
## [17] "X1972"          "X1973"          "X1974"          "X1975"
## [21] "X1976"          "X1977"          "X1978"          "X1979"
## [25] "X1980"          "X1981"          "X1982"          "X1983"
## [29] "X1984"          "X1985"          "X1986"          "X1987"
## [33] "X1988"          "X1989"          "X1990"          "X1991"
## [37] "X1992"          "X1993"          "X1994"          "X1995"
## [41] "X1996"          "X1997"          "X1998"          "X1999"
## [45] "X2000"          "X2001"          "X2002"          "X2003"
## [49] "X2004"          "X2005"          "X2006"          "X2007"
## [53] "X2008"          "X2009"          "X2010"          "X2011"
## [57] "X2012"          "X2013"          "X2014"          "X2015"
```

Discard unneeded columns:

```
# create a new data frame using only those columns that match up to TB data
d.df <- data.frame(d.raw$Country.Name,
                       d.raw$X2000,
                       d.raw$X2010,
                       d.raw$X2012)
```

```
# apply meaningful column names that will conform with TB data format
names(d.df)<-c("country", "x2000", "x2010", "x2012")

# display
kable(head(d.df), padding=0)
```

country	x2000	x2010	x2012
Aruba	85.41120	87.87328	90.87544
Andorra	100.00000	100.00000	100.00000
Afghanistan	37.45584	41.00000	43.00000
Angola	31.05585	34.60000	37.00000
Albania	100.00000	100.00000	100.00000
Arab World	79.53695	84.34222	86.27075

```
# discard original data frame - no longer needed
rm(d.raw)
```

Rename countries not matching between data sets:

```
# Rename countries not matching between data sets
dummy <- d.df

# take 18 country names in tb_nin
for(i in 1:nrow(tb_nin)) {
  # insert escape character if country name contains apostrophe
  s_country <- as.character(gsub("'", "\\\\'", tb_nin$country[i]) )

  # for each tb country name, look up wb_country
  sql_stmt <- sprintf("SELECT wb_country FROM cn_lookup WHERE tb_country = '%s' ", s_country)

  res <- as.character(sqlQuery(con, sql_stmt, errors= TRUE))

  # change name of l_exp$country[l_exp$country == wb_country] <- tb_nin$country[i]
  dummy$country[dummy$country == res] <- tb_nin$country[i]

}
```

Create a new data frame that discards all non-matched country names from d.df:

```
# Create a new data frame that discards all non-matched country names from d.df
new_df <- data.frame(dummy[ which( dummy$country %in% tb_country$country ), ])
nrow(new_df)
```

```
## [1] 100
```

```
kable(head(new_df))
```

	country	x2000	x2010	x2012
3	Afghanistan	37.45584	41.0	43.0
4	Angola	31.05585	34.6	37.0
8	Argentina	92.34713	94.0	99.8
14	Azerbaijan	95.95585	99.5	100.0
15	Burundi	3.90000	5.3	6.5
17	Benin	25.40000	27.9	38.4

Ensure country names match R's geoplotting tools:

```
for(i in 1:nrow(rmap_df)) {
  # make sure country names match
  new_df$country[new_df$country == rmap_df$tb_country[i]] <- rmap_df$rmap_country[i]
}
```

Transform data to long format:

```
# gather the 'year' columns

d.df2 <- gather(new_df, year, perc_e_acc, x2000:x2012, na.rm = FALSE)
kable(head(d.df2))
```

country	year	perc_e_acc
Afghanistan	x2000	37.45584
Angola	x2000	31.05585
Argentina	x2000	92.34713
Azerbaijan	x2000	95.95585
Burundi	x2000	3.90000
Benin	x2000	25.40000

```
# clean the "Year" column of the letter 'x'
d.df2$year <- str_extract(d.df2$year, "[[:digit:]]{4}")
kable(head(d.df2))
```

country	year	perc_e_acc
Afghanistan	2000	37.45584
Angola	2000	31.05585
Argentina	2000	92.34713
Azerbaijan	2000	95.95585
Burundi	2000	3.90000
Benin	2000	25.40000

```
# sort the data so it is ordered by country then year
d.df2 <- arrange(d.df2, country, year)
kable(head(d.df2))
```

country	year	perc_e_acc
Afghanistan	2000	37.45584
Afghanistan	2010	41.00000
Afghanistan	2012	43.00000
Algeria	2000	98.00000
Algeria	2010	99.30000
Algeria	2012	100.00000

Load refined data into pre-defined MySQL table:

```
# load each item from the d.df2 data frame into the elec_acc table
for(i in 1:nrow(d.df2)) {

  # insert escape character if country name contains apostrophe
  ins_country <- as.character(gsub("'", "\\\\'", d.df2$country[i]) )

  # check to see if rate = NA due to lack of data. If so insert NULL for rate in database
  if (is.na(d.df2$perc_e_acc[i])) {
    # format the required INSERT statement
    sql_stmt <- sprintf("INSERT INTO perc_e_acc (country, year, perc_e_acc )
                         VALUES ('%s', %d, NULL)",
                         ins_country,
                         as.numeric(d.df2$year[i]) )
  } else {
    # format the required INSERT statement
    sql_stmt <- sprintf("INSERT INTO perc_e_acc (country, year, perc_e_acc )
                         VALUES ('%s', %d, %f)",
                         ins_country,
                         as.numeric(d.df2$year[i]),
                         as.numeric(d.df2$perc_e_acc[i]) )
  } # end if / else

  res <- sqlQuery(con, sql_stmt, errors= TRUE)

} # end for loop

# cleanup data frames
rm(d.df)
rm(d.df2)
```

Average years of Schooling Data Loading

The average years of schooling data can be found at the following web link:

http://hdr.undp.org/en/content/mean-years-schooling-adults-years

PLEASE NOTE: Years of Schooling Data are available for years 2000, 2005 - 2012 ONLY

The process used for reading, transforming, tidying, and loading the average years of schooling data is somewhat similar to that used for the life expectancy data discussed above. The primary areas in which

the process varies from that of the World Bank data is in how the data set is loaded into R and the need to construct yet another country name lookup table. Explanatory narratives for these specific areas of variance are provided below.

Loading the Data into R

The years of schooling data is not available via a CSV download. As such, the data set is acquired via scraping of the source web page using the **XML** package's **readHTMLTable** function. The output of that function is then stripped of its non-relevant rows and columns so that we are left with a data frame that is similar in structure to those of the data frames we've used with the World Bank data.

```
# scrape web page
d.raw <- readHTMLTable("http://hdr.undp.org/en/content/mean-years-schooling-adults-years", stringsAsFac
d.raw <- data.frame(d.raw)

# remove first 4 rows
d.raw <- d.raw[5:nrow(d.raw),]

# Make new dataframe containing only rows 2, 6 - 14
d.df <- data.frame(d.raw[,2],d.raw[,6:14] )

# apply meaningful column names
names(d.df)<-c("country", "x2000", "x2005", "x2006", "x2007", "x2008",
               "x2009", "x2010", "x2011", "x2012")

# replace all empty column items with NA
d.df[d.df == '..'] <- NA

# discard original data frame - no longer needed
rm(d.raw)
```

Unfortunately, as mentioned above, the UN's data makes use of yet another set of naming conventions for country names, and as with the World Bank data we are forced to identify those inconsistencies and create a lookup table that can be used for purposes of changing any non-conforming country names contained within the schooling data to match the country names used within the tuberculosis data. The process used is nearly identical to that used with the World Bank data (see above).

First, we identify country names within the TB data that are not contained within the schooling data:

```
# get names of countries in TB data that ARE NOT in schooling data
tb_nin <- data.frame(tb_country$country[!(tb_country$country %in% d.df$country) ] )

names(tb_nin) <- 'country'

tb_nin
```

```
##                                          country
## 1                                China, Hong Kong SAR
## 2               Democratic People's Republic of Korea
## 3                    Democratic Republic of the Congo
## 4                                  Republic of Korea
## 5                                 Republic of Moldova
## 6 United Kingdom of Great Britain and Northern Ireland
```

```
## 7                          United Republic of Tanzania
## 8                             United States of America
```

We appear to have 8 country names within the schooling data set that do not match the corresponding country names within the TB data set. Finding the inconsistencies requires reading each individual country name listed above, extracting the 'most relevant' aspect therein, and using that aspect to locate a similar country name within the years of schooling data. This requires a great deal of manual "brute force" effort, the results of which we can use to construct a lookup table within our MySQL database for use going forward.

The R code shown below was used for purposes of that manual process. The outputs of the R code were used to create and populate a lookup table within the SQL script that is described above. The lookup table is automatically created and populated when the script is executed within MySQL Workbench.

```r
# R Code used to facilitate creation of 'hdr_c_lookup' table as defined in the required SQL script

# Find Hong Kong
for(i in 1:nrow(d.df)) {
  if(str_detect(d.df$country[i], 'Hong Kong') == TRUE) print(i)
}

d.df$country[15]


# Find Democratic People's Republic of Korea
for(i in 1:nrow(d.df)) {
  if(str_detect(d.df$country[i], "Korea") == TRUE) print(i)
}
d.df$country[194]


# Find  Democratic Republic of the Congo
for(i in 1:nrow(d.df)) {
  if(str_detect(d.df$country[i], "Congo") == TRUE) print(i)
}
d.df$country[185]


# Find Republic of Korea
for(i in 1:nrow(d.df)) {
  if(str_detect(d.df$country[i], "Korea") == TRUE) print(i)
}
d.df$country[14]

# Find Republic of Moldova
for(i in 1:nrow(d.df)) {
  if(str_detect(d.df$country[i], "Moldova") == TRUE) print(i)
}
d.df$country[113]

# Find United Kingdom of Great Britain and Northern Ireland
for(i in 1:nrow(d.df)) {
  if(str_detect(d.df$country[i], "United Kingdom") == TRUE) print(i)
}
d.df$country[13]
```

```
# Find United Republic of Tanzania
for(i in 1:nrow(d.df)) {
  if(str_detect(d.df$country[i], "Tanzania") == TRUE) print(i)
}
d.df$country[158]

# Find United States of America
for(i in 1:nrow(d.df)) {
  if(str_detect(d.df$country[i], "United States") == TRUE) print(i)
}
d.df$country[4]
```

Since the lookup table is created automatically within the MySQL database when the database creation script is executed, we can now perform the required lookup task and assign conforming country names to the 8 non-conforming country names we found within the years of schooling data:

```
dummy <- d.df

# take 8 country names in tb_nin
for(i in 1:nrow(tb_nin)) {
  # insert escape character if country name contains apostrophe
  s_country <- as.character(gsub("'", "\\\\'", tb_nin$country[i]) )

  # for each tb country name, look up wb_country
  sql_stmt <- sprintf("SELECT hdr_country FROM hdr_c_lookup WHERE tb_country = '%s' ", s_country)

  res <- as.character(sqlQuery(con, sql_stmt, errors= TRUE))

  # change name of l_exp$country[l_exp$country == wb_country] <- tb_nin$country[i]
  dummy$country[dummy$country == res] <- tb_nin$country[i]

}
```

Create a new data frame that discards all non-matched country names from d.df:

```
# Create a new data frame that discards all non-matched country names from d.df
new_df <- data.frame(dummy[ which( dummy$country %in% tb_country$country ), ])
nrow(new_df)
```

```
## [1] 100
```

```
kable(head(new_df))
```

	country	x2000	x2005	x2006	x2007	x2008	x2009	x2010	x
8	United States of America	12.7	12.8	12.8	12.9	12.9	12.9	12.9	1
9	Germany	10.5	12.4	12.8	12.8	12.9	12.9	12.9	1
17	United Kingdom of Great Britain and Northern Ireland	11.6	12.2	12.2	12.2	12.3	12.3	12.3	1
18	Republic of Korea	10.6	11.4	11.4	11.5	11.6	11.7	11.8	1
19	China, Hong Kong SAR	8.7	9.4	9.5	9.7	9.8	9.9	10.0	1
20	Japan	10.8	11.1	11.2	11.3	11.3	11.4	11.5	1

Ensure country names match R's geoplotting tools:

```
for(i in 1:nrow(rmap_df)) {
  # Ensure country names match R's geoplotting tools
  new_df$country[new_df$country == rmap_df$tb_country[i]] <- rmap_df$rmap_country[i]
}
```

Transform data to long format:

```
# gather the 'year' columns
d.df2 <- gather(new_df, year, yrs_school, x2000:x2012, na.rm = FALSE)
kable(head(d.df2))
```

country	year	yrs_school
USA	x2000	12.7
Germany	x2000	10.5
UK	x2000	11.6
South Korea	x2000	10.6
Hong Kong	x2000	8.7
Japan	x2000	10.8

```
# clean the "Year" column of the letter 'x'
d.df2$year <- str_extract(d.df2$year, "[[:digit:]]{4}")
kable(head(d.df2))
```

country	year	yrs_school
USA	2000	12.7
Germany	2000	10.5
UK	2000	11.6
South Korea	2000	10.6
Hong Kong	2000	8.7
Japan	2000	10.8

```
# sort the data so it is ordered by country then year
d.df2 <- arrange(d.df2, country, year)
kable(head(d.df2))
```

country	year	yrs_school
Afghanistan	2000	2.1
Afghanistan	2005	2.5
Afghanistan	2006	2.6
Afghanistan	2007	2.8
Afghanistan	2008	2.9
Afghanistan	2009	3.1

Load refined data into pre-defined MySQL table:

```
# load each item from the 'd.df2' data frame into the yrs_school table
for(i in 1:nrow(d.df2)) {

  # insert escape character if country name contains apostrophe
  ins_country <- as.character(gsub("'", "\\\\'", d.df2$country[i]) )

  # check to see if rate = NA due to lack of data. If so insert NULL for rate in database
  if (is.na(d.df2$yrs_school[i])) {
    # format the required INSERT statement
    sql_stmt <- sprintf("INSERT INTO yrs_school (country, year, yrs_school )
                         VALUES ('%s', %d, NULL)",
                         ins_country,
                         as.numeric(d.df2$year[i]) )
  } else {
    # format the required INSERT statement
    sql_stmt <- sprintf("INSERT INTO yrs_school (country, year, yrs_school )
                         VALUES ('%s', %d, %f)",
                         ins_country,
                         as.numeric(d.df2$year[i]),
                         as.numeric(d.df2$yrs_school[i]) )
  } # end if / else

  res <- sqlQuery(con, sql_stmt, errors= TRUE)

} # end for loop

# cleanup data frames
rm(d.df)
rm(d.df2)

# close the database connection
odbcClose(con)
```

Appendix: Ensuring all Country Names Conform to R's Country Naming Conventions

While we've been able to harmonize the country names across all of the various data sets we've collected, it appears there is yet one additional country name standardization task we need to attend to. Unfortunately, R's geoplotting packages make use of yet ***another*** set of country naming conventions that do not match up to any of the data sets we've collected. As such, we once again are required to try to resolve difference in country names since we hope to be able to make use of R's geoplotting capabilities within our analysis work for this project.

```
library(rworldmap)
temp_map = getMap(resolution='coarse')
temp_map@data

temp_map@data[['NAME']]
```

```
# Get map data for world
world_map <- map_data("world")
world_map

tbr_df <- sqlQuery(con, "SELECT * FROM tb_rates", stringsAsFactors=F)


wm_uniq <- data.frame(unique(world_map$region))
colnames(wm_uniq)[1] <- "region"

tb_uniq <- data.frame(unique(tbr_df$country))
colnames(tb_uniq)[1] <- "country"

# get names of countries in TB data that ARE NOT in world_map data
tb_nin <- data.frame(tb_uniq$country[!(tb_uniq$country %in% wm_uniq$region) ] )
names(tb_nin) <- 'country'
tb_nin
```

We appear to have 16 country names within R's geoplotting tools that do not match the corresponding country names used in our data sets. Unfortunately, finding the inconsistencies requires reading each individual country name listed above, extracting the 'most relevant' aspect therein by hand, and using that aspect to locate a similar country name within the TB data. (NOTE: Use of R's string processing functions is not possible since we can't know what country name we might be looking for without first manually inspecting each non-matching country name).

The R code shown below was used for purposes of that manual process. The outputs of the R code were used to create and populate a lookup table within the SQL script that is described above. The lookup table is automatically created and populated when the script is executed within MySQL Workbench.

```
# R Code used to facilitate creation of 'rmap_lookup' table as defined in the required SQL script

# Find Bolivia
for(i in 1:nrow(wm_uniq)) {
  if(str_detect(wm_uniq$region[i], 'Bolivia') == TRUE) print(i)
}

wm_uniq$region[32]


# Find Hong Kong - APPARENTLY DOES NOT EXIST IN R WORLD MAP  ?????
for(i in 1:nrow(wm_uniq)) {
  if(str_detect(wm_uniq$region[i], 'HongKong') == TRUE) print(i)
}

wm_uniq$region[]

# Find Congo
for(i in 1:nrow(wm_uniq)) {
  if(str_detect(wm_uniq$region[i], 'Congo') == TRUE) print(i)
}

wm_uniq$region[45]
wm_uniq$region[46]
```

```
# find Côte d'Ivoire
for(i in 1:nrow(wm_uniq)) {
  if(str_detect(wm_uniq$region[i], "Ivory Coast") == TRUE) print(i)
}
wm_uniq$region[43]


# Find Democratic People's Republic of Korea
for(i in 1:nrow(wm_uniq)) {
  if(str_detect(wm_uniq$region[i], "Korea") == TRUE) print(i)
}
wm_uniq$region[185]


# Find  Iran (Islamic Republic of)
for(i in 1:nrow(wm_uniq)) {
  if(str_detect(wm_uniq$region[i], "Iran") == TRUE) print(i)
}
wm_uniq$region[107]


# Find Lao People's Democratic Republic
for(i in 1:nrow(wm_uniq)) {
  if(str_detect(wm_uniq$region[i], "Lao") == TRUE) print(i)
}
wm_uniq$region[128]


# Find Republic of Korea
for(i in 1:nrow(wm_uniq)) {
  if(str_detect(wm_uniq$region[i], "Korea") == TRUE) print(i)
}
wm_uniq$region[125]


# Find Republic of Moldova
for(i in 1:nrow(wm_uniq)) {
  if(str_detect(wm_uniq$region[i], "Moldova") == TRUE) print(i)
}
wm_uniq$region[142]


# Find Russian Federation
for(i in 1:nrow(wm_uniq)) {
  if(str_detect(wm_uniq$region[i], "Russia") == TRUE) print(i)
}
wm_uniq$region[194]


# Find  Syrian Arab Republic
for(i in 1:nrow(wm_uniq)) {
  if(str_detect(wm_uniq$region[i], "Syria") == TRUE) print(i)
}
```

```
wm_uniq$region[220]


# Find United Kingdom of Great Britain and Northern Ireland
for(i in 1:nrow(wm_uniq)) {
  if(str_detect(wm_uniq$region[i], "UK") == TRUE) print(i)
}
wm_uniq$region[81]

# Find United Republic of Tanzania
for(i in 1:nrow(wm_uniq)) {
  if(str_detect(wm_uniq$region[i], "Tanzania") == TRUE) print(i)
}
wm_uniq$region[234]

# Find United States of America
for(i in 1:nrow(wm_uniq)) {
  if(str_detect(wm_uniq$region[i], "USA") == TRUE) print(i)
}
wm_uniq$region[238]

# Find Venezuela (Bolivarian Republic of)
for(i in 1:nrow(wm_uniq)) {
  if(str_detect(wm_uniq$region[i], "Venezuela") == TRUE) print(i)
}
wm_uniq$region[243]

# Find Viet Nam
for(i in 1:nrow(wm_uniq)) {
  if(str_detect(wm_uniq$region[i], "Viet") == TRUE) print(i)
}
wm_uniq$region[245]

# Find Yemen
for(i in 1:nrow(l_exp)) {
  if(str_detect(l_exp$country[i], "Yemen") == TRUE) print(i)
}
l_exp$country[244]
```

The lookup table (named "rmap_lookup") is created automatically within the MySQL database when the database creation script is executed. The contents of that table were used above in conjunction with the loading of each of the respective data sets.