Data from the Web

Dustin Johnson

December 6, 2014

In this final homework, we will get familiar with extracting data of the web in the following ways:

- Accessing data through pre-wrapped APIs
- Running API queries by hand
- Web scraping

We begin by loading the packages. Be sure to remember that the package `geonames` requires your username to log in, so include `options(geonamesUsername = "your_user_name")` before loading your libraries. As I already have mine loaded, I didn't bother putting it in.

```
suppressPackageStartupMessages({
    library(ggplot2)
    library(plyr)
    library(dplyr)
    library(gapminder)
    library(geonames)
    library(RCurl)
    library(magrittr)
    library(rvest)
    library(jsonlite)
    library(reshape2)
    library(knitr)
})
```

Combine gapminder and data from geonames

In this section, we will combine various sources of web data with our Gapminder data set to determine the relationship between per-capita GDP and the proportion of the population which lives in urban centres. We will soon find that this process is not as easy as it seems...

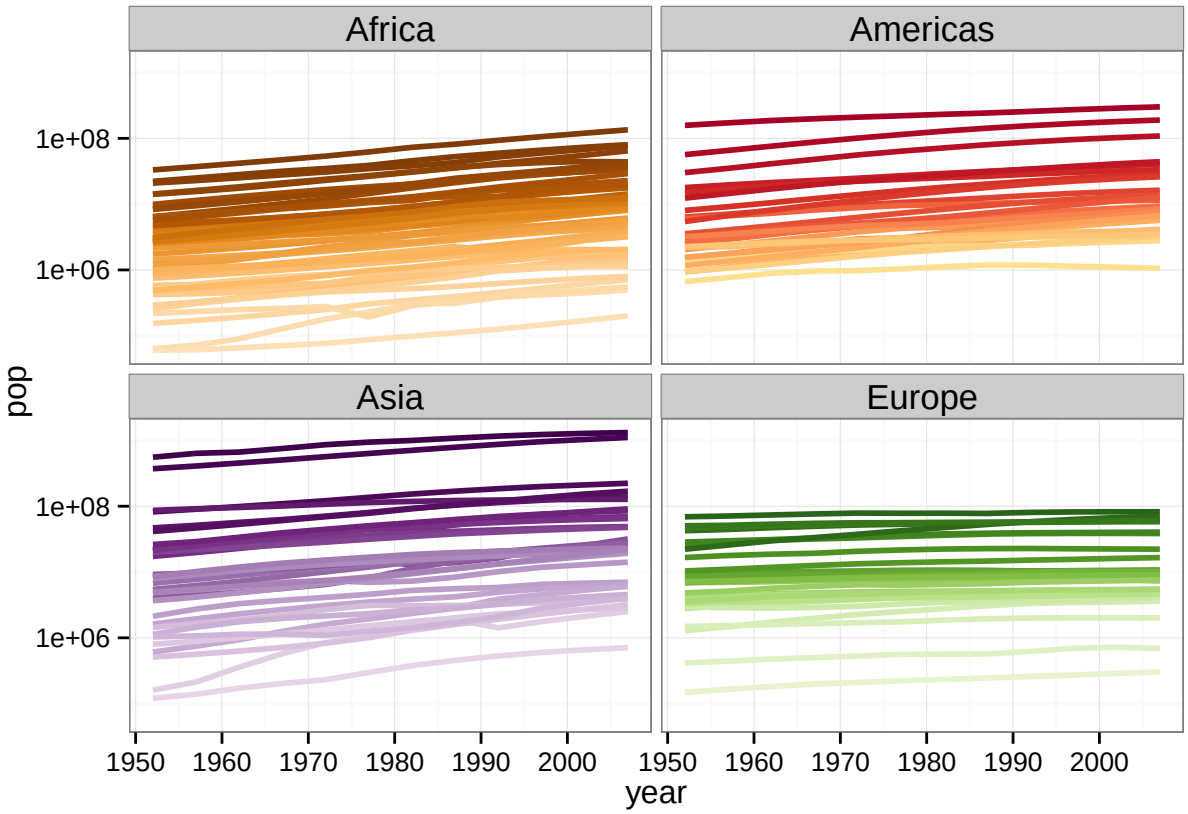
We begin by importing our trusty Gapminder data set that we installed from Jenny's repo. Just to be sure, let's make sure everything looks as it should.

```
# gapminder date
gDat <- gapminder
str(gDat)
```

```
## 'data.frame':    1704 obs. of  6 variables:
##  $ country  : Factor w/ 142 levels "Afghanistan",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ continent: Factor w/ 5 levels "Africa","Americas",..: 3 3 3 3 3 3 3 3 3 3 ...
##  $ year     : num  1952 1957 1962 1967 1972 ...
##  $ lifeExp  : num  28.8 30.3 32 34 36.1 ...
##  $ pop      : num  8425333 9240934 10267083 11537966 13079460 ...
##  $ gdpPercap: num  779 821 853 836 740 ...
```

All seems well! Now let's plot the population of the country by year per continent using the chunk of code provided on the homework page.

```
# plot population of country by year per continent
ggplot(subset(gDat, continent != "Oceania"),
             aes(x = year, y = pop, group = country, color = country)) +
    geom_line(lwd = 1, show_guide = FALSE) + facet_wrap(~ continent) +
    scale_color_manual(values = country_colors) +
    theme_bw() + theme(strip.text = element_text(size = rel(1.1))) + scale_y_log10()
```



We notice that population is increasing for the most part over the years for each country, although Europe appears to look more stagnant than the other continents.

Now we can gather the `GNcountryInfo` retrieved from the geonames API. Our goal is to merge our Gapminder with the geonames data set in such a way that we can include the variable `areaInSqKm` into our analysis for each country.

```
# Gather country info
countryInfo <- GNcountryInfo()
str(countryInfo)
```

```
## 'data.frame':    250 obs. of  17 variables:
##  $ continent    : chr  "EU" "AS" "AS" "NA" ...
##  $ capital      : chr  "Andorra la Vella" "Abu Dhabi" "Kabul" "Saint John's" ...
##  $ languages    : chr  "ca" "ar-AE,fa,en,hi,ur" "fa-AF,ps,uz-AF,tk" "en-AG" ...
##  $ geonameId    : chr  "3041565" "290557" "1149361" "3576396" ...
##  $ south        : chr  "42.4284925987684" "22.6333293914795" "29.377472" "16.996979" ...
##  $ isoAlpha3    : chr  "AND" "ARE" "AFG" "ATG" ...
```

```
##  $ north        : chr  "42.6560438963" "26.0841598510742" "38.483418" "17.729387" ...
##  $ fipsCode     : chr  "AN" "AE" "AF" "AC" ...
##  $ population   : chr  "84000" "4975593" "29121286" "86754" ...
##  $ east         : chr  "1.78654277783198" "56.3816604614258" "74.879448" "-61.672421" ...
##  $ isoNumeric   : chr  "020" "784" "004" "028" ...
##  $ areaInSqKm   : chr  "468.0" "82880.0" "647500.0" "443.0" ...
##  $ countryCode  : chr  "AD" "AE" "AF" "AG" ...
##  $ west         : chr  "1.40718671411128" "51.5833282470703" "60.478443" "-61.906425" ...
##  $ countryName  : chr  "Principality of Andorra" "United Arab Emirates" "Islamic Republic of Afghanis
##  $ continentName: chr  "Europe" "Asia" "Asia" "North America" ...
##  $ currencyCode : chr  "EUR" "AED" "AFN" "XCD" ...
```

Firstly, we notice that the number of observations is far smaller than the 1704 from our Gapminder data set. This is mainly due to the time variable for each country, so let's see how many countries we have for each.

```
count_geo <- length(unique(countryInfo$countryName))
count_gDat <- length(unique(gDat$country))
kable(data.frame(count_geo, count_gDat), format = "markdown")
```

count_geo	count_gDat
250	142

Well, we have many more countries in `countryInfo` . We can also see that many of the countries have different names, as demonstrated below. Only 28 countries share the exact same names, while the rest are spelled differently.

```
uniq_geo <- unique(countryInfo$countryName)
uniq_gDat <- unique(gDat$country)
anti <- length(which(uniq_gDat %in% uniq_geo))
anti <- data.frame(anti)
colnames(anti) <- "countries"
rownames(anti) <- "Total number of common countries"
kable(anti, format = "markdown")
```

	countries
Total number of common countries	28

In order to merge these two data sets, we will resort to ISO_3166-1 alpha-2 country codes, which are a generic two-letter way of abbreviating a country. Our hope is to obtain a set of country codes with country names that better resemble our data at hand - like a dictionary. To do this, we will "scrape" the data from a Wikipedia table. This can be done as simply as follows:

```
countryCode <- html("http://en.wikipedia.org/wiki/ISO_3166-1") %>%
    html_nodes(".wikitable") %>%
    html_table %>%
    extract2(1) %>%
    set_colnames(c("country", "Alpha2", "Alpha3", "Numeric", "ISO_3116-2")) %>%
```

```
    select(country, Alpha2)
which(is.na(countryCode$Alpha2))
```

```
## [1] 154
```

```
countryCode$Alpha2[154] <- as.character("NA")
```

As we are only interested in two columns of this data, namely country and Alpha2 (countryCode), we will extract them through `dplyr`. After a quick check of `NA` values, we can quickly notice that R has mistakenly identified the country code `NA` for `Nambia` as an `NA` value. This can easily be fixed through quick coercion.

It is time to see how well our `gDat` and `countryCode` data sets merge. Remember, our goal is to gather the country codes from this "dictionary" set, then merge `gDat` to geonames by reference of country code.

```
gDat_countryCode <- countryCode %>%
    merge(gDat, by = "country", all.y = TRUE)

missingISO <- unique(gDat_countryCode$country[which(is.na(gDat_countryCode$Alpha2))]) %>%
    as.character()
print(missingISO)
```

```
##  [1] "Bolivia"            "Congo, Dem. Rep."   "Congo, Rep."
##  [4] "Cote d'Ivoire"      "Hong Kong, China"   "Iran"
##  [7] "Korea, Dem. Rep."   "Korea, Rep."        "Reunion"
## [10] "Slovak Republic"    "Syria"              "Taiwan"
## [13] "Tanzania"           "United Kingdom"     "United States"
## [16] "Venezuela"          "Vietnam"            "West Bank and Gaza"
## [19] "Yemen, Rep."
```

First off, it seems we are not able to match 17 countries, yet we still require their country codes. The missing entries have been given `NA` values, so we can work with them later. To get around this issue, I bring back a topic that caught my interest in homework 8 - fuzzy string matching. I utilise the fuzzy string matching approach to get an approximation of the string variables in the other data set. Instead of going through the countries one-by-one, I placed my missing strings into a list and examined how many resembled country names in the `countryCode` data set. Let's see how it worked out.

```
fuzzy_string <- sapply(missingISO, agrep, countryCode$country, ignore.case = TRUE)
str(fuzzy_string)
```

```
## List of 19
##  $ Bolivia           : int [1:2] 27 241
##  $ Congo, Dem. Rep.  : int(0)
##  $ Congo, Rep.       : int(0)
##  $ Cote d'Ivoire     : int 55
##  $ Hong Kong, China  : int(0)
##  $ Iran              : int [1:19] 5 16 33 43 64 76 77 105 106 141 ...
##  $ Korea, Dem. Rep.  : int(0)
##  $ Korea, Rep.       : int(0)
##  $ Reunion           : int 181
##  $ Slovak Republic   : int(0)
##  $ Syria             : int [1:2] 15 217
```

```
##  $ Taiwan            : int 218
##  $ Tanzania          : int 220
##  $ United Kingdom    : int 235
##  $ United States     : int [1:2] 236 237
##  $ Venezuela         : int 241
##  $ Vietnam           : int 242
##  $ West Bank and Gaza: int(0)
##  $ Yemen, Rep.       : int(0)
```

It seems I caught quite a few! If you would like to double check for yourself, you will notice that they are indeed matches for the countries I'm interested in. Some countries have multiple possibilities, but it sure helps narrow down your search. By playing around with `agrep()`, you will find that you can narrow your search down even more (as I did with Iran).

You will notice that some countries turn up with no matches - `int(0)`. To find these ones, I just reused `agrep()` with the few countries remaining. Just as simple as that, out popped the indices for the countries in question!

```
fuzzy_string_missing <- sapply(c("Yemen", "Congo", "Hong Kong", "Korea", "Slovak"),
                                                                agrep,countryCode$country, ignore.case = TR
str(fuzzy_string_missing)
```

```
## List of 5
##  $ Yemen    : int 247
##  $ Congo    : int [1:3] 51 52 148
##  $ Hong Kong: int 100
##  $ Korea    : int [1:2] 118 119
##  $ Slovak   : int 202
```

You notice that Congo has 3 (51 and 52 are Congo), because there are two Congos. The same follows for Korea.

To expedite this process, I placed the indices and countries of ascending order in two separate lists and integrated them directly into my merged data set, thereby replacing the `NA`'s with country codes. As you will notice, the order comes in handy. I can simply arrange my data according to year.

```
fuzzy_match <- c(27, 51, 52, 55, 100, 105, 118, 119, 181, 202, 217, 218, 220, 241, 242, 247)
fuzzy_country <- c("Bolivia", "Congo, Rep.", "Congo, Dem. Rep.", "Cote d'Ivoire",
                                    "Hong Kong, China", "Iran", "Korea, Dem. Rep.", "Korea, Rep.",
                                    "Reunion", "Slovak Republic", "Syria", "Taiwan", "Tanzania",
                                    "Venezuela", "Vietnam", "Yemen, Rep.")

gISO <- gDat_countryCode %>%
    arrange(year)

gISO$Alpha2[which(gISO$country %in% fuzzy_country)] <- countryCode$Alpha2[fuzzy_match]

gISO <- gISO %>%
    arrange(country)

missing_remainder <- unique(gISO$country[which(is.na(gISO$Alpha2))])
print(missing_remainder)
```

```
## [1] "United Kingdom"     "United States"      "West Bank and Gaza"
```

One last check leaves us with the outlier - a country that doesn't seem to fit no matter how hard I search. I'm sure there is a country corresponding to "West Bank and Gaza", but it is like trying to find a needle in a haystack. For the time being, I will remove this little problem from `gDat`.

```
set.seed(21)

gClean <- gISO %>%
    filter(country != "West Bank and Gaza") %>%
    droplevels

kable(gClean[sample(nrow(gClean), 10), ], format = "markdown")
```

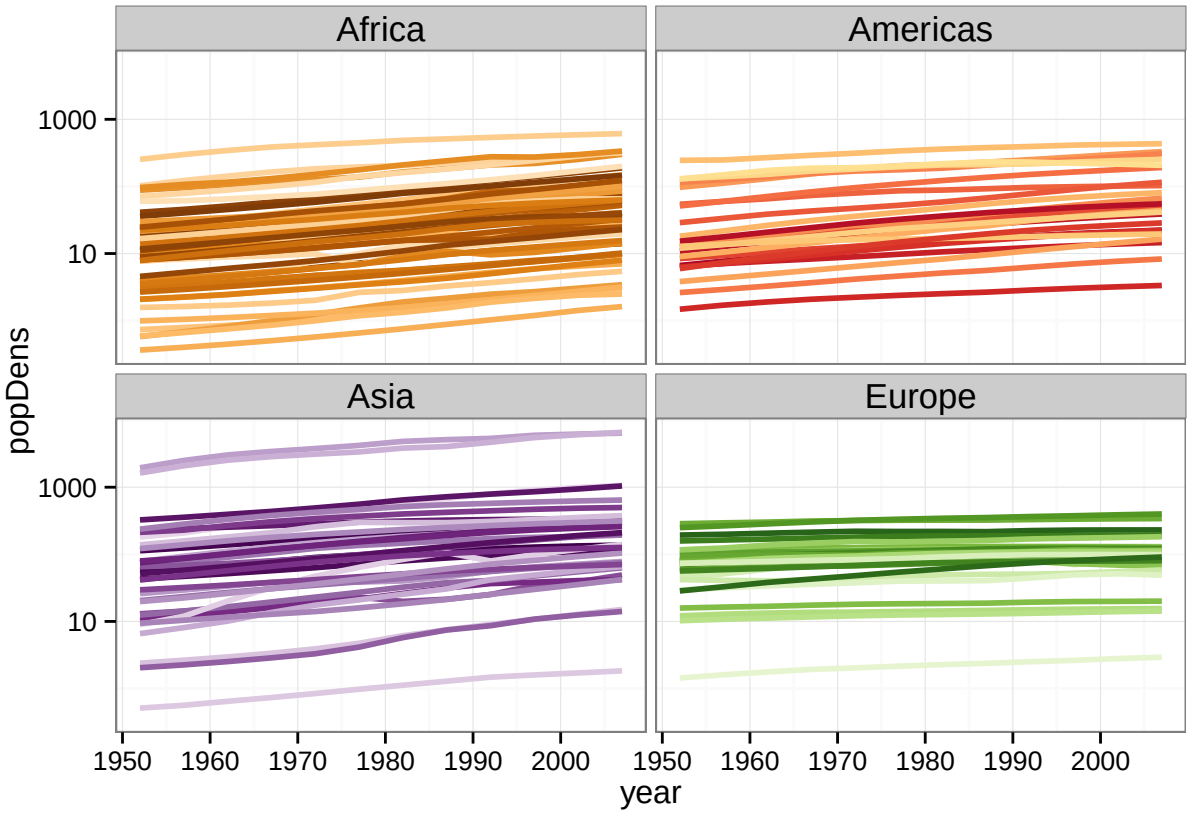
	country	Alpha2	continent	year	lifeExp	pop	gdpPercap
1331	Senegal	SN	Africa	2002	61.60000	10870037	1519.635
427	Djibouti	DJ	Africa	1982	48.81200	305991	2879.468
1182	Panama	PA	Americas	1977	68.68100	1839782	5351.912
312	Colombia	CO	Americas	2007	72.88900	44227550	7006.580
1620	United States	NA	Americas	2007	78.24200	301139947	42951.653
1550	Trinidad and Tobago	TT	Americas	1957	61.80000	764900	4100.393
172	Brazil	BR	Americas	1967	57.63200	88049823	3429.864
291	China	CN	Asia	1962	44.50136	665770000	487.674
1661	Yemen, Rep.	YE	Asia	1972	39.84800	7407075	1265.047
1430	Sri Lanka	LK	Asia	1957	61.45600	9128546	1072.547

A sample of rows taken from `gClean` is shown in the above table with each country and its corresponding code. Finally, we are at our final merge! All of our countries in `gDat` have a country code, which we can connect with our geonames data set, `countryInfo`. Of course, `countryInfo` comes with a whole slew of information, so we select just what we need. Using `mutate()`, we can incorporate the population density and we are nearly there.

```
gDat_popDens <- gClean %>%
    set_colnames(c("country", "countryCode", "Continent", "year",
                                   "lifeExp", "pop", "gdpPercap")) %>%
    left_join(countryInfo, by = "countryCode") %>%
    arrange(country, year) %>%
    mutate(popDens = pop/as.numeric(areaInSqKm),
                  country = as.factor(country)) %>%
    select(country, Continent, countryCode, year, popDens)
```

The plot of the population density of country by year per continent is provided below.

```
# plot population density of country by year per continent
ggplot(subset(gDat_popDens, Continent != "Oceania"),
             aes(x = year, y = popDens, group = country, color = country)) +
    geom_line(lwd = 1, show_guide = FALSE) + facet_wrap(~ Continent) +
    scale_color_manual(values = country_colors) +
    theme_bw() + theme(strip.text = element_text(size = rel(1.1))) + scale_y_log10()
```



We notice that countries are less spread out when it comes to population density, but the same mild increasing trend is apparent. There is one straggler in the Americas that has a declining population density over the years, which paints a different picture than the prior.

The World Bank API Query

The World Bank have a massive source of public data covering the financial sector, economy, environment, and agriculture of countries around the world. Fortunately, they have an API not yet provided on rOpenSci that we can work with. Until recently it was necessary to use an API key to make calls to its API, but now we no longer need an API key. XML, JSON and JSONP formats are supported, although we will be focusing our attention on JSON.

We would like to make a query to obtain a data.frame of an indicator based on the country and time-frame in question.

```
# Function for API query
WorldBank <- function(countries = "", indicators = "",
                                          date = "2000:2014", format = "") {
    require(plyr); require(dplyr)
    baseURL <- "http://api.worldbank.org/countries/"
    params <- c("", "/indicators/", "?date=", "&format=")
    values <- c(countries, indicators, date, format)
    param_values <- Map(paste0, params, values)
    args <- paste0(param_values, collapse = "")
    finalURL <- paste0(baseURL, args)

    request <- RCurl::getURL(finalURL)
    data <- data.frame(fromJSON(request))
```

```
    data$country <- as.character(data$country[[1]])
    data$value <- as.numeric(data$value)
    data$date <- as.integer(data$date)
    select(data, indicator, country, value, date)
}
```

We have constructed our function for our query. We can now obtain a time-series data.frame of countless indicators. The indicators can be obtained from the World Development Indicators. ISO 3166-1 alpha-2 country codes are used, just like our previous section.

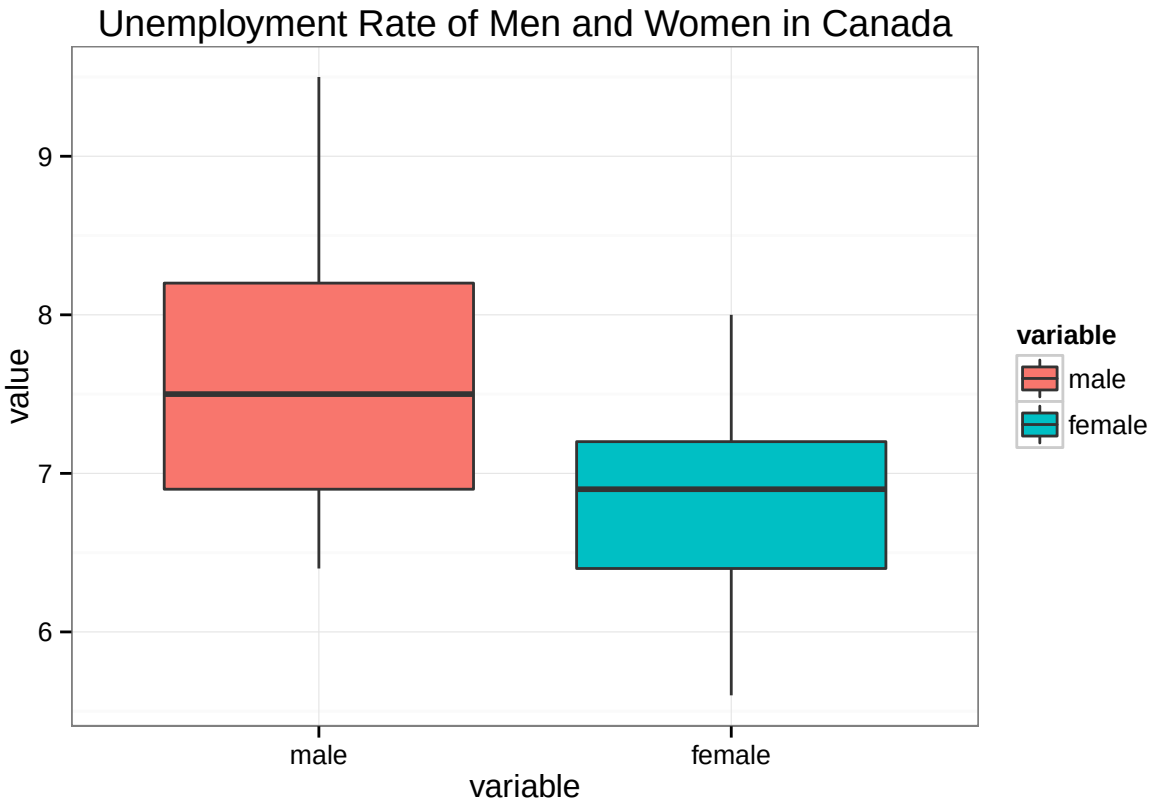
Gender Indicator: Unemployment The table below provides the unemployment rate pertaining to men and women. I have taken out the first column, as it merely provides the indicator information. If you would like to reconfirm the information provided, as well as its units, leave this column in.

```
unemploymentFemale_ca <- WorldBank("ca", "SL.UEM.TOTL.FE.ZS", "1998:2010", "json")[,-1]
colnames(unemploymentFemale_ca) <- c("country", "female", "year")
unemploymentMale_ca <- WorldBank("ca", "SL.UEM.TOTL.MA.ZS", "1998:2010", "json")[,-1]
colnames(unemploymentMale_ca) <- c("country", "male", "year")
unemployment <- data.frame(melt(merge(unemploymentMale_ca, unemploymentFemale_ca,
                                                                    by = c("year", "country")),
                                                          id.vars = c("country", "year")))

kable(head(unemployment), format = "markdown")
```

country	year	variable	value
CA	1998	male	8.5
CA	1999	male	7.8
CA	2000	male	6.9
CA	2001	male	7.5
CA	2002	male	8.2
CA	2003	male	8.0

The table output looks just as we hoped it to. The `head` does not show us too much though... Let's examine the box-plot.

```
ggplot(unemployment, aes(variable, value)) + geom_boxplot(aes(fill = variable)) +
    theme_bw() + theme(strip.text = element_text(size = rel(1.1))) +
    ggtitle("Unemployment Rate of Men and Women in Canada")
```



We notice that unemployment rate of males is indeed larger on average that the unemployment rate of females over the period of 1998 to 2010.

Population Indicator: Birth Rates What we have done here is select a few countries to examine their birth rates over a period from 1960 to 2010. The table below provides a random sample of rows to make sure our data looks good.

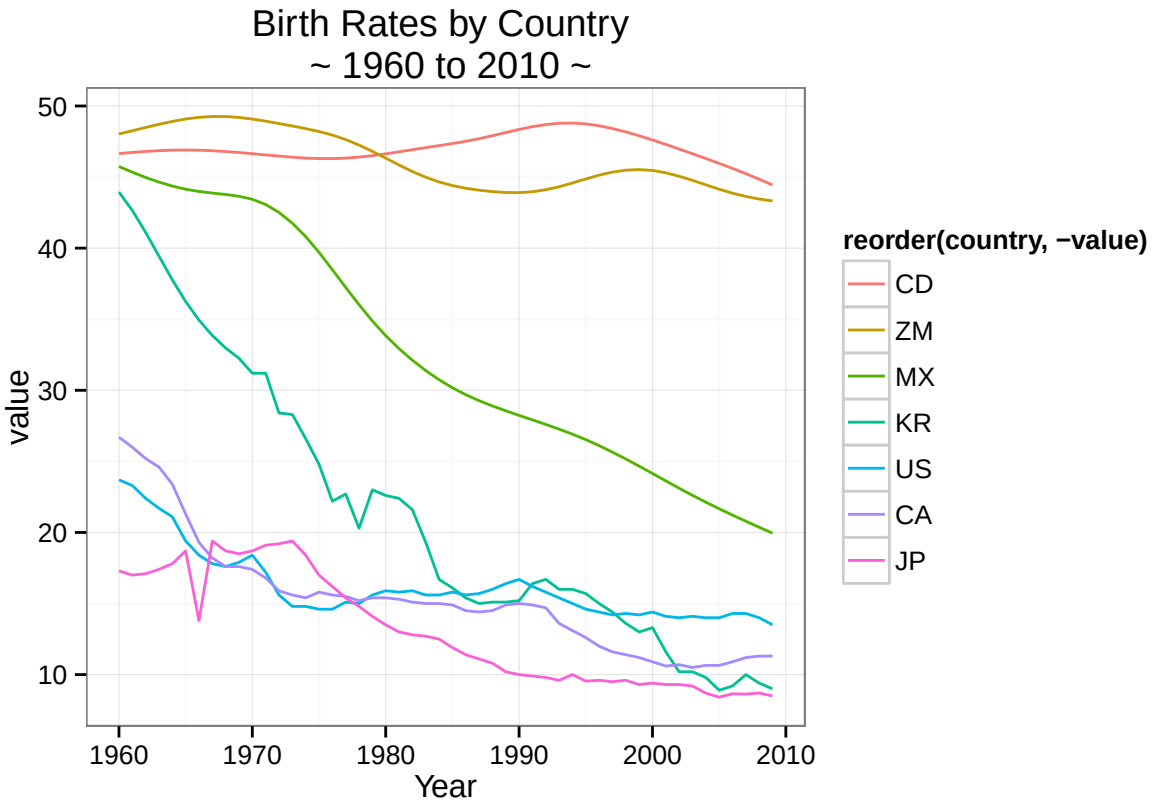
```
birthRate_ca <- WorldBank("ca", "SP.DYN.CBRT.IN", "1960:2010", "json")[,-1]
birthRate_us <- WorldBank("us", "SP.DYN.CBRT.IN", "1960:2010", "json")[,-1]
birthRate_mx <- WorldBank("mx", "SP.DYN.CBRT.IN", "1960:2010", "json")[,-1]
birthRate_jp <- WorldBank("jp", "SP.DYN.CBRT.IN", "1960:2010", "json")[,-1]
birthRate_cd <- WorldBank("cd", "SP.DYN.CBRT.IN", "1960:2010", "json")[,-1]
birthRate_kr <- WorldBank("kr", "SP.DYN.CBRT.IN", "1960:2010", "json")[,-1]
birthRate_zm <- WorldBank("zm", "SP.DYN.CBRT.IN", "1960:2010", "json")[,-1]

birthRate <- data.frame(rbind(birthRate_ca, birthRate_us, birthRate_mx,
                                                          birthRate_jp, birthRate_cd, birthRate_kr,
                                                          birthRate_zm))

set.seed(123)
birthRate %>%
    sample_n(10) %>%
    kable(format = "markdown")
```

	country	value	date
101	MX	19.949000	2009
276	KR	16.700000	1984

	country	value	date
143	MX	43.875000	1967
307	ZM	44.771000	2003
326	ZM	44.657000	1984
16	CA	13.100000	1994
182	JP	14.800000	1978
347	ZM	48.709000	1963
189	JP	19.100000	1971
156	JP	8.693605	2004

From the table, we cannot make any conclusions, but we know that it is in a format ready to be used by `ggplot`.

```
ggplot(birthRate, aes(date, value, colour = reorder(country, -value))) +
    geom_line() +
    theme_bw() + theme(strip.text = element_text(size = rel(1.1))) +
    xlab("Year") + ggtitle("Birth Rates by Country \n ~ 1960 to 2010 ~")
```



By examining the plot, we notice that the birth rates of Congo (CD) and Zambia (ZM) fluctuate over the years, but remain relatively high. Little decline is shown throughout the years. On the other hand, Japan (JP) is declining quite rapidly, reinforcing their fear of insufficient labour supply in the future.

We can query any indicator for any country at any point in time! Check out the indicators you can examine or country codes here:

- Indicators
- Country Codes

Reflection

Wow, this homework took a surprising amount of time. I found it difficult to merge Gapminder and the geonames data set, especially when it came down to coordinating all the proper country codes. This took me a considerable amount of time, as I was stuck with many tools, but I wasn't sure which ones to use. At first I tried simple `grep()` commands, but it didn't seem to provide me with what I wanted - I was stuck dealing with one string at a time, with the limitations of what piece I should search for next. I resorted to fuzzy string matching, and surprisingly it worked quite well for the task at hand. I must admit, majority of my time was spent trying to figure out a clever way to do this.

I also had a tad of trouble getting multiple countries in one query using the World Bank's API logical structure with the semi-colon ";". I have a feeling it is due to my slightly hacky syntax `c("", "/indicators/", "?date=", "&format=")`, although I tried my best to make this query happen. Any insight on cleaning this up into a more "proper" form would be delightful. I would have liked to try the XML form as well with the World Bank API, but maybe another time.

I used quite a few resources to get me through this, including:

1. R-bloggers
2. Wiki ISO data
3. Stack Overflow
4. Andrew and Jenny's Walkthrough
5. Partial String Matching
6. stringdist

Gathering data from the web is a fascinating and sophisticated topic. Being completely new to APIs (sometimes using them when I didn't even know I was) and web scraping, I can immediately see the use in this. In the past, I've resorted to downloading `.csv` files and loading them in externally, but access through an API enables anyone to streamline the process and open their sights to the plethora of data available. I have to say, that I am the most excited about web scraping. Although I wasn't able to use it in a way I would have liked for this assignment, the sheer ability to obtain data from the web becomes limitless. I have always wanted specific data from sources such as the New York Times, etc, and now I am able to.

Merry Christmas!