SPEC Lab R Resources: Appending County IDs

Miriam Barnum (with minor revisions by Ben Graham)

June 9, 2020

Setup

Before beginning a prep script, you will need to store several file paths as variables. This allows other users to run your code with file paths that work on their own computers. For folks working on the IPE/Talus team or anyone else tasked with running actual data preps for the lab, these file paths should be set to the Master IPE Data folder and its subfolders. For those of you doing this in training, you should set these file paths to the "0 Training Data" folder and its subfolders.

In a **separate** script, run code similar to the following. Save this script so that you can reload these file paths every time you prep data in the IPE Data Resource folders.

We save this mini R script as its own separate file so that any lab member can run any lab prep script – nobody's individual file path should be hard-coded into the script. Save this file-path R script to your local machine. You then run this script once before you get to work on data preps on a given day. Once you have run it, your filepaths are available in R to be called during the prep script (you will see these objects in your Global Environment window in R Studio).

Note that it is very important that you don't leave off the final / at the end of your file path.

An example for the IPE/Talus Team:

```
rawdata <- "/Volumes/GoogleDrive/My Drive/Master IPE Data/SUMMER 2020/rawdata/"
preppeddata <- "/Volumes/GoogleDrive/My Drive/Master IPE Data/SUMMER 2020/prepped/"
prepscripts <- "/Volumes/GoogleDrive/My Drive/Master IPE Data/SUMMER 2020/scripts/"
```

An example for use during training:

```
rawdata <- "/Volumes/GoogleDrive/My Drive/SPEC Summer 2020/Training Data Science/0 Training Data/"
preppeddata <- "/Volumes/GoogleDrive/My Drive/SPEC Summer 2020/Training Data Science/0 Training Data/tra
prepscripts <- "/Volumes/GoogleDrive/My Drive/SPEC Summer 2020/Training Data Science/0 Training Data/app
```

Load Data

Begin a prep script by loading the data, using your `rawdata` file path. Don't forget to give the dataset object a meaningful name (i.e., *not* `df` , `data`, etc.)

The `paste` function simply puts two strings together. So what you are doing here is calling the data file example_RAWDATA_BCG_2011.dta, which is located in a folder specified by the file path that you have saved as the variable `rawdata`.

```
library(foreign)
bcg <- read.dta(paste(rawdata,"example_RAWDATA_BCG_2011.dta",sep=""))
```

Prep

Nearly all of the work to prep a dataset should be done before you append country IDs. This includes things like reshaping the data, removing extraneous variables, creating any variables that need to be created, and renaming variables appropriately. The dataset we're working on here is in pretty good shape already — all we need to do is select only the variables we need, and rename the country variable.

*Note: If you have run libary(tidyverse) already at the top of the script, you don't need the dplyr:: part of the code below. The dplyr:: part just tells R where to go find the select command if the package isn't already loaded into your library. You will see this coding convention in some lab scripts – it is a way to make the code extra robust.

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
bcg <- dplyr::select(bcg, country = ctryname, year, under, undertype)
```

Checks

We're almost ready to add IDs, but there are a few things we need to check first, in order to ensure that the `append_ids` function works properly.

First, do we have country and year columns in the data?

```
names(bcg)
```

```
## [1] "country"   "year"      "under"     "undertype"
```

Yes! (We already knew this, because we just selected these variables, but it doesn't hurt to check.)

Next, is the year column numeric?

```
str(bcg$year)
```

```
##  num [1:6860] 1970 1971 1972 1973 1974 ...
```

In this case it is. If it wasn't, we could change it, using either of the following lines of code:

```
bcg <- mutate(bcg, year = as.numeric(year))

bcg$year <- as.numeric(bcg$year)
```

Finally, are there any duplicate country-years? To check this, we make a cross-tabulation of country and year. Then we check if any of the country-year combos occur more than once in the dataset.

```
n_occur <- data.frame(table(bcg$country, bcg$year))
print(n_occur[n_occur$Freq > 1,])
```

```
##      Var1 Var2 Freq
## 1         1970   23
## 193       1971   19
## 385       1972   19
## 577       1973   19
## 769       1974   19
## 961       1975   17
## 1153       1976   16
## 1345       1977   16
## 1537       1978   15
## 1729       1979   15
## 1921       1980   15
## 2113       1981   14
## 2305       1982   14
## 2497       1983   14
## 2689       1984   14
## 2881       1985   14
## 3073       1986   14
## 3265       1987   14
## 3457       1988   14
## 3649       1989   14
## 3841       1990   14
## 4033       1991    4
## 4225       1992    4
## 4417       1993    2
## 4609       1994    2
## 4801       1995    2
## 4993       1996    2
## 5185       1997    2
## 5377       1998    2
## 5569       1999    2
## 5761       2000    2
## 5953       2001    2
## 6145       2002    2
## 6337       2003    2
## 6529       2004    2
## 6721       2005    2
```

What's going on? It looks like some observations just have a blank string for the country name. We want to discard these. If we leave them in the data, the `append_ids` function will filter them out for us, but we can also do that ourselves before running `append_ids`.

In this filter command, we keep only observations for which `country` is not blank. Note: With string variables, you always put the value in quotation marks so "" is how you refer to a blank – quotes around nothing.

```
bcg <- filter(bcg, country != "")
```

Appending IDs

We're finally ready to add county IDs to the data!

First, we need to load the `append_ids` function, using the `prepscripts` file path. **DO NOT** open or edit the append IDs script yourself — everyone uses it, so if you accidentally break something, you're breaking it for everyone. If you have a problem using the script, ask your team lead for help.

If you are doing this as a member of the IPE Talus team or are doing an actual prep for the lab, your code will look like this, because you will be using our master version of the append_ids script.

```
#source(paste(prepscripts,"append_ids.R",sep=""))
```

If you are doing this for training, your code will look like this, because you will be using a training version of the script (which is exactly the same, but with a different file name to keep it distince from the master version).

```r
source(paste(prepscripts,"append_ids_training.R",sep=""))
```

You should see several new functions in the environment pane of RStudio. Call the `append_ids` function. Note that this code stays the same regardless of whether you are doing this in training or for real.

```r
bcg <- append_ids(bcg, breaks = F)
```

```
## [1] "The following countries were not given gwno codes: "
## [1] NA
## character(0)
```

It prints out a list of country names that did not recive GWNO country codes (and were therefore removed from the datasets). In this case, the only observations that did not recieve country codes are those where the country name was missing (`NA`). There are a variety of reasons that observations may not recieve country codes:

1. Missing or blank values (as seen above), or strings that arent intended to be country names (for instance, if someone typed notes about the data at the bottom of an excel sheet).
2. Aggregate categories (e.g., "World", "Asia", "Non-OECD").
3. Areas that are not independent, internationally recognized countries (e.g., "Western Sahara", "U.S. Virgin Islands").
4. Areas that are microstates with less than 250,000 people (e.g. "Vatican City" or "Nauru").
5. Misspelled country names, or variations/abbreviations of country names that we havent encountered before (e.g., "Bostwana" instead of "Botswana", "DP Rep. Korea" instead of "North Korea")

The only one of these that is a problem is (5). If there are country names listed that you think correspond to actual countries that need to be included in the dataset, copy and paste those names, and send them to the IPE/Talus team lead (Kaitlyn King as of Summer 2020), who will make sure those spellings are added to the master country IDs file. Once they give you the go-ahead, re-run your prep script from the beginning, and check that those countries now recieve IDs.

Check for duplicates

Some duplicates could have been added by the `append_ids` function. This is most likely to occur when countries enter or exit the international state system. For instance the Gleditsch-Ward system includes Czechoslovakia until 1992 and the Czech Republic and Slovakia beginning in 1993. However, some datasets may include observations for the Czech Republic AND Czechoslovakia in 1992, resulting in duplicates when we standardize country names.

```r
n_occur <- data.frame(table(bcg$country, bcg$year))
print(n_occur[n_occur$Freq > 1,])
```

```
## [1] Var1 Var2 Freq
## <0 rows> (or 0-length row.names)
```

No duplicates here!

BUT, if there were, we would want to delete one of these duplicates, so that we have one unique observation for each country year. The original country names for each variable will have been retained as the `countryname_raw` variable. Some rules of thumb for choosing which to delete:

1. If one of the rows contains meaningful data, but the other does not (i.e., it contains mostly missing values, or zeros for anything other than a binary variable), keep the one with meaningful data.
2. Look up the official Gleditsch-Ward start and end dates for the countries in question: <http://ksgleditsch.com/data/iisystem.dat>. Keep the observation where the raw counrtyname existed for the greatest proportion of the year in question.

3. Load the MasterGWNO.RDATA file from the ‘rawdata‘ folder, and look at the min and max years for the relevant country names.
4. If you’re still not sure, ask your team lead or the person who asked you to prep the dataset.

After you have decided which observation to delete, VERY THOROUGHLY document why you made that choice. Someone else may want to understand or second-guess that decision later.

Adding Suffixes

Now we can add variable labels and suffixes. We add suffixes to keep track of what component dataset a variable comes from. Scholars and organizations put years of work into collecting these data and we don’t want them to lose credit for their work. Also, it is important that we know where to go in order to find the original codebook with full details on how a variable was created. We always need to know where data come from!

We add labels because variable names themselves are too short to give us even a basic idea of what each variable is. Good labels include information on the concept being measureed, the units it is in, and the suffix associated with the original dataset the variable came from.

Labels can be added using the `Hmisc` package.

```r
library(Hmisc)
```

```
## Loading required package: lattice

## Loading required package: survival

## Loading required package: Formula

## Loading required package: ggplot2

##
## Attaching package: 'Hmisc'

## The following objects are masked from 'package:dplyr':
##
##     src, summarize

## The following objects are masked from 'package:base':
##
##     format.pval, units
```

```r
label(bcg$under) <- "Whether a country was under an IMF agreement [BCG]"
label(bcg$undertype) <- "Type of IMF agreement a country is under [BCG]"
```

Suffixes can be added using `add_suffix` function (loaded along with `append_ids`). If you are updating a dataset that has been prepped in the lab before, it is imperative that we use the same suffix that was used previously. If this is a new dataset, then we need a new suffix. It is important to choose a suffix not already assigned to another dataset, so check the codebook for the IPE Data Resource before choosing your suffix.

Note that the append_suffix command also runs the same regardless of whether you are doing this for training or for real.

```r
bcg <- append_suffix(bcg,"BCG")
```

Save Prepped Data

We’re finally ready to save the prepped data, using the `preppeddata` file path. The file name should have the following format: “PREPPED_[dataset suffix]*[your initials]*[today’s date (ddmmyyyy)].RDATA”. Here “example_” is included to distinguish this training data from actual prepped datsets.

If you are just training, the `preppedata` file path should be be set to the "SPEC Summer 2020/Training Data Science/0 Training Data/training output/".

```
save(bcg, file=paste(preppeddata,"example_PREPPED_BCG_MB_08062020.RDATA",sep=""))
```

What if there's no country name variable?

Some raw data files may not include country names, instead including some other type of country identifier, such as COW (Correlates of War) country codes, or ISO country abbreviations. In these cases, we need to add country names before running `append_ids`.

As an example, let's look at the COW National Military Capabilities Data.

```
mc <- read.csv(paste(rawdata,"example_RAWDATA_MC_2017.csv",sep=""))
names(mc)
```

```
##  [1] "stateabb" "ccode"    "year"     "milex"    "milper"   "irst"
##  [7] "pec"      "tpop"     "upop"     "cinc"     "version"
```

```
str(mc$ccode)
```

```
##  int [1:15171] 2 2 2 2 2 2 2 2 2 2 ...
```

```
str(mc$stateabb)
```

```
##  chr [1:15171] "USA" "USA" "USA" "USA" "USA" "USA" "USA" "USA" "USA" "USA" ...
```

We can add country names using the `countrycode` package.

```
library(countrycode)
mc <- mutate(mc, country = countrycode(ccode,            # variable containing country codes
                                       'cown',           # coding scheme of origin (COW numeric)
                                       'country.name'))  # coding scheme of destination (full names)
```

```
## Warning in countrycode(ccode, "cown", "country.name"): Some values were not matched unambiguously: 2
```

We get a warning message that some of the codes weren't matched, so we need to add names for those codes manually. We can find the correct names by looking them up in the COW list of country codes:\ https://correlatesofwar.org/data-sets/downloadable-files/cow-country-codes/cow-country-codes/view.

```
mc$country[mc$ccode == 730] <- "Korea"
mc$country[mc$ccode == 260] <- "German Federal Republic"
```

COW codes are only one of many types of country identifiers that `countrycode()` can work with. Let's look at the others:

```
?codelist
```

What if there's no year variable?

Some data is cross-sectional (it just contains country-level variables that don't change over time). In this case, `append_ids()` will add country codes based on 2015 country names, and will give you a warning message.

One example is the NationsOnLine Official Languages data.

```
library(readxl)
lang <-read_excel(paste(rawdata,"example_RAWDATA_LANG.xlsx", sep=""),sheet = 6, na = "NA")
names(lang)
```

```
## [1] "country"    "language 1" "language2"
```

```r
lang <-  append_ids(lang, breaks = F)
```

```
## Warning in append_ids(lang, breaks = F): NO YEAR VARIABLE PROVIDED. If this is not intended to be a c
##              GWNOs for cross-sectional datasets will be assigned based on 2015 country names.
```

```
## [1] "The following countries were not given gwno codes: "
## [1] "American Samoa"
## [1] "Anguilla"
## [1] "Aruba"
## [1] "Cayman Islands"
## [1] "Cook Islands"
## [1] "French Guiana"
## [1] "French Polynesia"
## [1] "Guadeloupe"
## [1] "Guam"
## [1] "Martinique"
## [1] "New Caledonia"
## [1] "Niue"
## [1] "Northern Mariana Islands"
## [1] "Palestine"
## [1] "Pitcairn"
## [1] "Puerto Rico"
## [1] "Réunion"
## [1] "Saint Helena"
## [1] "Virgin Islands"
## [1] "Western Sahara"
```

Usually it makes sense to just store these datasets as cross-sectional data, rather than repeating observations for each year. But what if we wanted to crate a country-year panel of the official language data?

We can do this by merging the data with a panel of Gleditsch-Ward country years, from the `countrycode` package.

```r
names(codelist_panel)
```

```
##  [1] "country.name.en"             "year"
##  [3] "ar5"                         "cctld"
##  [5] "continent"                   "country.name.de"
##  [7] "country.name.de.regex"       "country.name.en.regex"
##  [9] "cowc"                        "cown"
## [11] "currency"                    "dhs"
## [13] "ecb"                         "eu28"
## [15] "eurocontrol_pru"             "eurocontrol_statfor"
## [17] "eurostat"                    "fao"
## [19] "fips"                        "gaul"
## [21] "genc2c"                      "genc3c"
## [23] "genc3n"                      "gwc"
## [25] "gwn"                         "icao.region"
## [27] "imf"                         "ioc"
## [29] "iso2c"                       "iso3c"
## [31] "iso3n"                       "iso4217c"
## [33] "iso4217n"                    "p4c"
## [35] "p4n"                         "region"
## [37] "region23"                    "un"
## [39] "un.region.code"              "un.regionintermediate.code"
## [41] "un.regionsub.code"           "unicode.symbol"
## [43] "unpd"                          "vdem"
## [45] "wb"                            "wb_api2c"
## [47] "wb_api3c"                      "wvs"
```

```
lang_panel <- codelist_panel %>%
  dplyr::select(country.name.en, year, gwno = gwn) %>%
  left_join(lang, by = "gwno")
```

If we want to have *all* the country IDs added by `append_ids()` for every observation, we need to remove those columns and re-run `append_ids()`.

```
lang_panel <- lang_panel %>%
  dplyr::select(country = country.name.en, year, `language 1`, language2) %>%
  append_ids(breaks = F)
```

```
## [1] "The following countries were not given gwno codes: "
## [1] "Brunswick"
## [1] "Eswatini"
## [1] "Hamburg"
## [1] "Nassau"
## [1] "North Macedonia"
## [1] "Oldenburg"
## [1] "Palestinian Territories"
## [1] "Piedmont-Sardinia"
## [1] "Saxe-Weimar-Eisenach"
## [1] "Somaliland"
## [1] "Vatican City"
## [1] "Würtemberg"
```

Note: This approach (merging with a country-year panel) is also useful when turning data that is not in country-year format int country-year data. For instance, if we had a dataset of wars that contained only country-years where a war *did* occur, we could merge that data with a panel (after appropriate cleaning/aggregation of the original data), and then add zeros for the country-years where no war occured.

Practice: write a full prep script for the National Military Capabilities Data.

You can use the code from above to add country names. Don't forget to delete unnecessary variables, such as the COW country abbreviations and the dataset version. Save the prepped data in the prepped folder with the appropriate file name (and the prefix "practice_").