SPEC Lab R Workshop Series: Session 6

Therese Anders

1 for Loops in R

Loops are a staple of programming. Loops allow us to automate our code, and are particularily useful when you find yourself doing a task over and over again. While in practice, we try to vectorize our operations as much as possible (see the solution above where we convert factors to characters), being comfortable with loops is crucial for many programming tasks.

1.1 Basic loop structure

General syntax of a for loop:

`for(iterator in iterations){function/output}`

Lets write a loop that prints out the numbers 1 through 10.

```
for(i in 1:10){
  print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
```

Suppose, the numbers we wanted to print out are part of a vector. We can use loops to iterate through this vector.

```
vec <- seq(11, 20, 1)
for(k in vec){
  print(k)
}
```

```
## [1] 11
## [1] 12
## [1] 13
## [1] 14
## [1] 15
## [1] 16
## [1] 17
## [1] 18
## [1] 19
## [1] 20
```

Exercise 1 What do you think does the following output do?

```
for(l in 5:length(vec)){
  print(l)
}
```

Of course, we can use loops to automate more complex tasks. For example, we could use it to change a batch of variables to `character()`. To try this, re-load the data from session 5.

```
data_new <- read.csv("hw5_data.csv")
str(data_new)
```

```
## 'data.frame':    56 obs. of  10 variables:
## $ State            : Factor w/ 56 levels "Alabama","Alaska",..: 10 33 43 44 24 13 49 8 3 23 ...
##  $ Pop..dens..Rank        : int  1 2 3 4 5 6 7 8 9 10 ...
## $ Pop..dens.Rank50.states: Factor w/ 51 levels "-","1","10","11",..: 1 2 1 13 24 1 1 35 1 46 ...
##  $ Density.Pop...mi2.     : int  11011 1218 1046 1021 871 808 799 741 721 618 ...
##  $ Density.Pop...km2.     : int  4251 470 404 394 336 314 308 286 279 238 ...
##  $ Pop..Rank              : int  50 11 29 44 15 53 54 30 55 19 ...
## $ X2015population      : Factor w/ 56 levels "1,032,949","1,056,298",..: 48 53 27 2 45 15 11 26 39 41 ...
##  $ Land.Rank              : int  56 46 49 51 45 52 54 48 55 42 ...
## $ Landarea.mi2.        : Factor w/ 56 levels "1,034","1,949",..: 38 43 14 1 44 11 7 20 47 54 ...
## $ Landarea.km2.        : Factor w/ 56 levels "1,477,953.4",..: 20 26 55 30 31 50 45 8 29 39 ...
```

```
test1 <- data_new
for(l in 1:ncol(test1)){
  test1[,l] <- as.character(test1[,l])
}
str(test1)
```

```
## 'data.frame':    56 obs. of  10 variables:
## $ State            : chr  "District of Columbia" "New Jersey" "Puerto Rico" "Rhode Island" ...
##  $ Pop..dens..Rank        : chr  "1" "2" "3" "4" ...
##  $ Pop..dens.Rank50.states: chr  "-" "1" "-" "2" ...
##  $ Density.Pop...mi2.     : chr  "11011" "1218" "1046" "1021" ...
##  $ Density.Pop...km2.     : chr  "4251" "470" "404" "394" ...
##  $ Pop..Rank              : chr  "50" "11" "29" "44" ...
##  $ X2015population        : chr  "672,228" "8,958,013" "3,680,058" "1,056,298" ...
##  $ Land.Rank              : chr  "56" "46" "49" "51" ...
##  $ Landarea.mi2.          : chr  "61" "7,354" "3,515" "1,034" ...
##  $ Landarea.km2.          : chr  "158.0" "19,046.8" "9,103.8" "2,678.0" ...
```

As a more sophisticated version, we could convert only those variables that are factors (not numerical variables like `Density.Pop....km2..`) to characters, using an `if` statement.

```
test2 <- data_new
for(k in 1:ncol(test2)){
  if(is.factor(test2[,k])){
    test2[,k] <- as.character(test2[,k])
  }
}
str(test2)
```

```
## 'data.frame':    56 obs. of  10 variables:
## $ State            : chr  "District of Columbia" "New Jersey" "Puerto Rico" "Rhode Island" ...
##  $ Pop..dens..Rank        : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Pop..dens.Rank50.states: chr  "-" "1" "-" "2" ...
##  $ Density.Pop...mi2.     : int  11011 1218 1046 1021 871 808 799 741 721 618 ...
##  $ Density.Pop...km2.     : int  4251 470 404 394 336 314 308 286 279 238 ...
```

```
##  $ Pop..Rank              : int  50 11 29 44 15 53 54 30 55 19 ...
##  $ X2015population        : chr  "672,228" "8,958,013" "3,680,058" "1,056,298" ...
##  $ Land.Rank              : int  56 46 49 51 45 52 54 48 55 42 ...
##  $ Landarea.mi2.          : chr  "61" "7,354" "3,515" "1,034" ...
##  $ Landarea.km2.          : chr  "158.0" "19,046.8" "9,103.8" "2,678.0" ...
```

2 Data cleaning in R

In this example, we will use our new data management skills to clean a data set for inclusion in SPEC's IPE data resource. The data are figures on US Foreign Direct Investment (FDI) from the Bureau of Economic Analysis. Here are the data cleaning tasks that we are going to do:

1. Recode missing values.
2. Delete the header and footer.
3. Add COW country codes.
4. Drop observations from combined countries. Drop duplicated observations from Poland, Bahamas, Hungary, Czech Republic, Russia, Jamaica, Trinidad and Tobago, Guatemala, and conflicted observations from Serbia/Yugoslavia, Russia/USSR, Zaire/Congo, Timor-Leste, Micronesia, and Samoa.
5. Reshape data to long format.

To start, lets read the data and take a look at it using the `View()` in RStudio.

```
library(foreign)
library(tidyverse)
fdi <- read.csv("fdidata.csv",
                stringsAsFactors = F)
```

2.1 Recoding missing values

The original data specifies a number of different ways of how missing values are coded, specifically `n.s.`, `(*)`, `--`, `---`, `----`, and `(D)`. In principle, we could recode the missing values with the following command.

```
fdi[fdi == "(D)" | fdi == "n.s." | fdi == "(*)" | fdi == "----" | fdi == "---" | fdi ==
"--"] <- NA
```

However, we could also make our life easier by specifying all possible values for `NA` when reading the data.

```
fdi_nona <- read.csv("fdidata.csv",
                     stringsAsFactors = F,
                     na.strings = c("(D)", "n.s.", "(*)", "----", "---", "--"))
```

2.2 Dropping header and footer

There are a number of ways we could drop the header and footer. In principle, we could just inspect the data frame and manually delete all the rows that belong to the header or footer. However, there is a pattern here: The header and footer rows do not have entries in the second column. Note that when the value of the second column is `NA`, `R` would also drop that row. Therefore, we have to account for both, rows that have a non-empty value or `NA` in the second column, when selecting the rows to eliminate.

```
empty <- fdi_nona[,2] == ""
head(empty, 10)
```

```
##  [1]  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE    NA
```

```
fdi_new <- fdi_nona[empty %in% c(NA, F),]
```

Subsequently, we want to turn the second row into the column names and drop the first and second rows.

```
names(fdi_new) <- unname(fdi_new[2,])
fdi_new <- fdi_new[-c(1,2),]
```

2.3 Stripping white space and dropping irrelevant observations

After renaming the first column of our new dataframe to `country`, lets look at the observations (i.e. countries that we have in this data set). Use the `View()` function or the Environment menu to inspect the values of the variable `country`.

```
names(fdi_new)[1] <- "country"
head(fdi_new$country)
```

```
## [1] "Canada"           "Europe"           "  Austria"
## [4] "  Belgium"        "  Czech Republic" "  Denmark"
```

The first thing to notice is that many elements of the `country` variable have leading whitespace (i.e. spaces or tabs). This is a problem when trying to attach countrycodes later on. We will use operations from the `stringr` package to get rid of this leading whitespace. Note that we do **not** want to remove all white space, because country names such as `Czech Republic` would not be recognized by the `countrycode` package any longer if we stripped the character value from all whitespace. We will use `stringr`'s `str_trim()` function to remove leading (and trailing) whitespace from all elements of the variable `country`.

```
fdi_new$country[5]
```

```
## [1] "  Czech Republic"
```

Second, there are a number of observations in the `country` variable that do not contain proper country names, such as `Other`, `Other Western Hemisphere`, or `Latin America and Other Western Hemisphere`. We can use regular expressions to drop these observations.

```
nrow(fdi_new)
```

```
## [1] 242
```

```
fdi_new_sub <- fdi_new %>%
  mutate(country = str_trim(country)) %>%
  filter(!str_detect(country, "[Oo]ther"))
```

```
## Warning: package 'bindrcpp' was built under R version 3.4.4
```

```
nrow(fdi_new_sub)
```

```
## [1] 233
```

2.4 Adding COW country codes

Next, we will use the `countrycode` package to add COW country codes to the data. Note, that this does not work perfect here–some countrynames are not matched. For demonstration purposes we will simply drop observations that were not matched. In reality, we might have to add some country codes manually, or use a more sophisticated algorithm to attach country codes. Note that the countrycode package allows you to specify custom country dictionaries, for example if you needed to attach Gleditsch-Ward rather than COW countrycodes.

```
# install.packages("countrycode")
library(countrycode)
fdi_new_sub <- fdi_new_sub %>%
  mutate(ccode = countrycode(country, "country.name", "cown"))
```

We can inspect the values that did not receive a COW code. The algorithm works as expected (Serbia does not have its own countrycode in the COW system). All the un-matched countries are either not fully sovereign regions (based on the COW system), continents, or smaller island nations that are not considered part of the international system, according to COW. We therefore drop all observations that did not receive a `ccode` coding.

```
fdi_new_sub$country[is.na(fdi_new_sub$ccode)]
```

```
##  [1] "Europe"                          "Gibraltar"
##  [3] "Greenland"                       "Serbia"
##  [5] "South America"                   "French Guiana"
##  [7] "Central America"                 "Bermuda"
##  [9] "Netherlands Antilles"            "Anguilla"
## [11] "Aruba"                           "Curaao"
## [13] "French Islands, Caribbean"       "Netherlands Antilles"
## [15] "Netherlands Islands, Caribbean" "Sint Maarten"
## [17] "Africa"                          "Middle East"
## [19] "Iraq-Saudi Arabia Neutral Zone" "Asia and Pacific"
## [21] "Hong Kong"                       "French Islands, Indian Ocean"
## [23] "French Islands, Pacific"         "Macau"
## [25] "Micronesia"
```

```
fdi_new_sub <- filter(fdi_new_sub, !is.na(ccode))
```

2.5 Handling duplicates

Before re-shaping our data, we need to check for duplicates. `R` provides a number of built-in functions to execute this task. Here, we will write a snipped of custom code using `dplyr` to show which observations (based on the `ccode` variable) have duplicates and how many.

```
dupes <- fdi_new_sub %>%
  group_by(ccode) %>%
  summarise(count = n()) %>%
  filter(count > 1)
```

We have quite a few duplicate observations. In reality, we would want to go through each of these observations and inspect whether they are "true" duplicates, or whether they are produced in the process of attaching COW codes. Here, we will only go through two examples.

First, let us look at all the duplicates with COW code 200 (United Kingdom). If we inspect all the observations, we see that we have one "true" UK observations, and a number of other observations that contain the word "United Kingdom," but do not refere to the mainland. What to do with these duplicates is a substantive question that needs to be decided by the researcher. Here, for the purpose of demonstation, we will assume that all the observation belong to the UK and sum over them (taking into account the NA values).

```
ccode200 <- fdi_new_sub %>%
  filter(ccode == 200)
```

Second, let us look at the cuplicates with COW code 365 (Russia). We can see that these are not "true" duplicates, that is no value is observed in two rows in the same year. We can therefore simply "melt" these three variables together to create one row for ccode 365.

```
ccode365 <- fdi_new_sub %>%
  filter(ccode == 365)
```

Depending on the type of duplicate, we might want to apply different functions to each instance of duplication. For simplicity, here we simply sum over all duplicates, excluding missing values. This will achieve the desired transformation for both the UK and the Russia duplicates. Note that before summing, we will have to convert our values to class `numeric`. Note also that upon applying the `summarise_if()` command, we loose the information on the `country` variable that contains the name of the country. There are ways to retain this information (for example by subsequently attaching the countryname with the `countrycode` package, see below).

```
for(i in 2:ncol(fdi_new_sub)){
  fdi_new_sub[,i] <- as.numeric(fdi_new_sub[,i])
}
str(fdi_new_sub)
```

```
## 'data.frame':    208 obs. of  35 variables:
##  $ country: chr  "Canada" "Austria" "Belgium" "Czech Republic" ...
##  $ 1982   : num  43511 562 5549 NA 1155 ...
##  $ 1983   : num  44779 548 5087 NA 1275 ...
##  $ 1984   : num  47498 534 5202 NA 1263 ...
##  $ 1985   : num  47934 509 5619 NA 1383 ...
##  $ 1986   : num  52006 736 5568 NA 1164 ...
##  $ 1987   : num  59145 711 7719 NA 1120 ...
##  $ 1988   : num  63900 697 7830 NA 1182 ...
##  $ 1989   : num  63948 962 7710 NA NA ...
##  $ 1990   : num  69508 1113 9464 NA 1726 ...
##  $ 1991   : num  70711 1268 10611 NA 1940 ...
##  $ 1992   : num  68690 1371 11381 NA 1676 ...
##  $ 1993   : num  69922 1312 11697 NA 1735 ...
##  $ 1994   : num  74221 2197 14714 NA 2030 ...
##  $ 1995   : num  83498 2829 18706 NA 2161 ...
##  $ 1996   : num  89592 2854 18740 NA 2554 ...
##  $ 1997   : num  96626 2646 17337 NA 2385 ...
##  $ 1998   : num  98200 3856 17899 NA 2764 ...
##  $ 1999   : num  119590 3848 21756 1038 3846 ...
##  $ 2000   : num  132472 2872 17973 1228 5270 ...
##  $ 2001   : num  152601 3964 22589 1179 5160 ...
##  $ 2002   : num  166473 4011 25727 1264 6184 ...
##  $ 2003   : num  187953 6366 27415 1668 5597 ...
##  $ 2004   : num  214931 9264 41840 2444 6815 ...
##  $ 2005   : num  231836 11236 49306 2729 6914 ...
##  $ 2006   : num  205134 14897 51862 3615 5849 ...
##  $ 2007   : num  250642 14646 62491 4066 8950 ...
##  $ 2008   : num  246483 13546 65279 5053 10481 ...
##  $ 2009   : num  274807 10954 46610 5372 13053 ...
##  $ 2010   : num  295206 11485 43975 5268 11802 ...
##  $ 2011   : num  330041 12556 50984 5840 14942 ...
##  $ 2012   : num  366709 14327 49144 6016 14306 ...
##  $ 2013   : num  390172 15641 51702 6990 13605 ...
##  $ 2014   : num  386121 15787 48128 7247 14108 ...
##  $ ccode  : num  20 305 211 316 390 375 220 255 350 310 ...
```

```
fdi_new_sub_nodupes_alt <- fdi_new_sub %>%
  group_by(ccode) %>%
```

```
  summarise_if(is.numeric, funs(sum(., na.rm = T)))
```

Unfortunately, this last version returns 0 for columns that have all `NA` values. We will therefore write a custom `function` to pass to the `summarise_if()` wrapper. The general syntax of a function is the following.

`functionname <- function(operand){operation, return value}`.

Within the function we use an `if...else` statement. The `if...else` statement specifies a conditional operation with the following general syntax:

`if(condition is true){output} else {output}`.

```
func_sum <- function(x){
  if(all(is.na(x))){
    return(NA)
  } else {
    return(sum(x, na.rm = T))
  }
}

fdi_new_sub_nodupes <- fdi_new_sub %>%
  group_by(ccode) %>%
  summarise_if(is.numeric, funs(func_sum))
```

2.6 Reshaping

The original data is in wide format. To make it compatible with the IPE data resource (and most other panel data sets) we need to reshape it into the long format, with one column indicating the country, another indicator accounting for the year, and lastly a column capturing the values of the respective indicator. In this case, the entire dataframe of the original data captures only one variable, namely "Outward FDI stocks from the US in USD (millions), all industries, BEA [OFS]." For simplicity, we will call this variable `outward_fdi_all`.

```
fdi_long <- fdi_new_sub_nodupes %>%
  # re-shape to long format
  gather(year, outward_fdi_all, 2:ncol(fdi_new_sub_nodupes)) %>%
  # Re-attach country names using countrycode package
  mutate(countryname = countrycode(ccode, "cown", "country.name"))
```

The result is a clean data frame that can now be merged into the IPE data resource.

```
head(fdi_long)
```

```
## # A tibble: 6 x 4
##   ccode year  outward_fdi_all countryname
##   <dbl> <chr>           <dbl> <chr>
## 1    20 1982            43511 Canada
## 2    31 1982             3121 Bahamas
## 3    40 1982               NA Cuba
## 4    41 1982               19 Haiti
## 5    42 1982              188 Dominican Republic
## 6    51 1982              386 Jamaica
```

Sources

U.S. Bureau of Economic Analysis, "U.S. Direct Investment Abroad, U.S. Direct Investment Position Abroad on a Historical-Cost Basis," http://www.bea.gov/international/di1usdbal.htm (accessed Jun 21 2016).