Lab 2 Solutions

Steven Boyd

10/8/2021

Getting started

Today, we are going to work on loading and tidying data. Step 1 is to download the following zip file from the course git repository "gdp_per_cap_1995_2015.zip." Save it in the same folder as this markdown.

First, we will make sure everyone can extract data from a zip. One option is to open the zipped file and use your OS' built in extraction capabilities (assuming it has those). However, it is also possible to access the data within R Studio. In fact, readr can grab data inside a zip without extracting at all!

```
gdp_per_cap <- read_csv('data/gdp_per_cap.zip')
```

```
## Multiple files in zip: reading 'b9988620-72da-4463-b88c-93fa96b7075f_Data.csv'
```

```
## Rows: 269 Columns: 25
```

```
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (25): Series Name, Series Code, Country Name, Country Code, 1995 [YR1995...
```

```
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
            # load data directly from zip
```

Notice the warning text. It is telling us that there are multiple csv files within the zip file. This data was downloaded from the World Bank earlier this year in this format. Read the name of the file carefully. What does it say at the end of the file name?

If you look inside the zip file, you'll notice another csv with *almost* the same name. The only difference is in the suffix. The one marked "metadata" contains information *about* the data, but no data we can actually analyze. It is common for csv files like this to be generated automatically when you download data from a huge database like the World Bank's DataBank.

We don't really need to worry that R didn't read the metadata, because it is only useful as documentation. If you do have multiple csv files in a zip file, you can sepcify which one you want to read, but you need to unzip first.

```
gdp_per_cap_2 <- read_csv(unz(description =  'data/gdp_per_cap.zip',
                  filename =   'b9988620-72da-4463-b88c-93fa96b7075f_Data.csv'))
```

```
## Rows: 269 Columns: 25

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (25): Series Name, Series Code, Country Name, Country Code, 1995 [YR1995...


##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
                 # load data from specific csv
```

We can confirm that these contain the same data by examining each:

```
str(gdp_per_cap)  # check structure of first load
```

```
## spec_tbl_df [269 x 25] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ Series Name  : chr [1:269] "GDP per capita, PPP (current international $)" "GDP per capita, PPP (
##  $ Series Code  : chr [1:269] "NY.GDP.PCAP.PP.CD" "NY.GDP.PCAP.PP.CD" "NY.GDP.PCAP.PP.CD" "NY.GDP.PCA
##  $ Country Name : chr [1:269] "Afghanistan" "Albania" "Algeria" "American Samoa" ...
##  $ Country Code : chr [1:269] "AFG" "ALB" "DZA" "ASM" ...
##  $ 1995 [YR1995]: chr [1:269] ".." "2665.10903649849" "7082.78475736955" ".." ...
##  $ 1996 [YR1996]: chr [1:269] ".." "2979.3330899284" "7377.69949168115" ".." ...
##  $ 1997 [YR1997]: chr [1:269] ".." "2716.69355834805" "7465.89932260811" ".." ...
##  $ 1998 [YR1998]: chr [1:269] ".." "3020.93057263444" "7816.77057806543" ".." ...
##  $ 1999 [YR1999]: chr [1:269] ".." "3471.65635908955" "8068.2961352061" ".." ...
##  $ 2000 [YR2000]: chr [1:269] ".." "3862.48437763265" "8446.58797859344" ".." ...
##  $ 2001 [YR2001]: chr [1:269] ".." "4301.38282973192" "8775.11702646461" ".." ...
##  $ 2002 [YR2002]: chr [1:269] "877.014423503263" "4661.37914319822" "9293.83679843142" ".." ...
##  $ 2003 [YR2003]: chr [1:269] "927.857547917432" "4994.92469274671" "10019.3578261386" ".." ...
##  $ 2004 [YR2004]: chr [1:269] "925.441616196776" "5423.20168382235" "10591.034224285" ".." ...
##  $ 2005 [YR2005]: chr [1:269] "1023.05162974192" "5865.31759085604" "11405.6403648217" ".." ...
##  $ 2006 [YR2006]: chr [1:269] "1077.76190658441" "6557.80133674236" "11776.0413453018" ".." ...
##  $ 2007 [YR2007]: chr [1:269] "1228.70413531195" "7274.52468523841" "12311.0390253609" ".." ...
##  $ 2008 [YR2008]: chr [1:269] "1272.57320417194" "8228.32755984622" "12643.1491551178" ".." ...
##  $ 2009 [YR2009]: chr [1:269] "1519.69254818311" "8819.51009173416" "12722.3781829231" ".." ...
##  $ 2010 [YR2010]: chr [1:269] "1710.57564538432" "9636.10870469977" "13095.4468249877" ".." ...
##  $ 2011 [YR2011]: chr [1:269] "1699.48799733991" "10207.7335023376" "13500.0416554875" ".." ...
##  $ 2012 [YR2012]: chr [1:269] "1914.77435127371" "10526.3189737173" "13303.3317888752" ".." ...
##  $ 2013 [YR2013]: chr [1:269] "2015.51496204541" "10570.9681040326" "13056.8036185828" ".." ...
##  $ 2014 [YR2014]: chr [1:269] "2069.42464180385" "11259.2967004521" "13003.267118909" ".." ...
##  $ 2015 [YR2015]: chr [1:269] "2087.30532306683" "11658.8660596051" "12015.6313951015" ".." ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   `Series Name` = col_character(),
##   ..   `Series Code` = col_character(),
##   ..   `Country Name` = col_character(),
##   ..   `Country Code` = col_character(),
##   ..   `1995 [YR1995]` = col_character(),
##   ..   `1996 [YR1996]` = col_character(),
##   ..   `1997 [YR1997]` = col_character(),
##   ..   `1998 [YR1998]` = col_character(),
##   ..   `1999 [YR1999]` = col_character(),
##   ..   `2000 [YR2000]` = col_character(),
##   ..   `2001 [YR2001]` = col_character(),
##   ..   `2002 [YR2002]` = col_character(),
##   ..   `2003 [YR2003]` = col_character(),
##   ..   `2004 [YR2004]` = col_character(),
##   ..   `2005 [YR2005]` = col_character(),
##   ..   `2006 [YR2006]` = col_character(),
##   ..   `2007 [YR2007]` = col_character(),
##   ..   `2008 [YR2008]` = col_character(),
##   ..   `2009 [YR2009]` = col_character(),
##   ..   `2010 [YR2010]` = col_character(),
##   ..   `2011 [YR2011]` = col_character(),
##   ..   `2012 [YR2012]` = col_character(),
##   ..   `2013 [YR2013]` = col_character(),
##   ..   `2014 [YR2014]` = col_character(),
##   ..   `2015 [YR2015]` = col_character()
##   .. )
##  - attr(*, "problems")=<externalptr>
```

```
str(gdp_per_cap_2)  # check structure of second load
```

```
## spec_tbl_df [269 x 25] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ Series Name  : chr [1:269] "GDP per capita, PPP (current international $)" "GDP per capita, PPP (
##  $ Series Code  : chr [1:269] "NY.GDP.PCAP.PP.CD" "NY.GDP.PCAP.PP.CD" "NY.GDP.PCAP.PP.CD" "NY.GDP.PCA
##  $ Country Name : chr [1:269] "Afghanistan" "Albania" "Algeria" "American Samoa" ...
##  $ Country Code : chr [1:269] "AFG" "ALB" "DZA" "ASM" ...
##  $ 1995 [YR1995]: chr [1:269] ".." "2665.10903649849" "7082.78475736955" ".." ...
##  $ 1996 [YR1996]: chr [1:269] ".." "2979.3330899284" "7377.69949168115" ".." ...
##  $ 1997 [YR1997]: chr [1:269] ".." "2716.69355834805" "7465.89932260811" ".." ...
##  $ 1998 [YR1998]: chr [1:269] ".." "3020.93057263444" "7816.77057806543" ".." ...
##  $ 1999 [YR1999]: chr [1:269] ".." "3471.65635908955" "8068.2961352061" ".." ...
##  $ 2000 [YR2000]: chr [1:269] ".." "3862.48437763265" "8446.58797859344" ".." ...
##  $ 2001 [YR2001]: chr [1:269] ".." "4301.38282973192" "8775.11702646461" ".." ...
##  $ 2002 [YR2002]: chr [1:269] "877.014423503263" "4661.37914319822" "9293.83679843142" ".." ...
##  $ 2003 [YR2003]: chr [1:269] "927.857547917432" "4994.92469274671" "10019.3578261386" ".." ...
##  $ 2004 [YR2004]: chr [1:269] "925.441616196776" "5423.20168382235" "10591.034224285" ".." ...
##  $ 2005 [YR2005]: chr [1:269] "1023.05162974192" "5865.31759085604" "11405.6403648217" ".." ...
##  $ 2006 [YR2006]: chr [1:269] "1077.76190658441" "6557.80133674236" "11776.0413453018" ".." ...
##  $ 2007 [YR2007]: chr [1:269] "1228.70413531195" "7274.52468523841" "12311.0390253609" ".." ...
##  $ 2008 [YR2008]: chr [1:269] "1272.57320417194" "8228.32755984622" "12643.1491551178" ".." ...
##  $ 2009 [YR2009]: chr [1:269] "1519.69254818311" "8819.51009173416" "12722.3781829231" ".." ...
##  $ 2010 [YR2010]: chr [1:269] "1710.57564538432" "9636.10870469977" "13095.4468249877" ".." ...
##  $ 2011 [YR2011]: chr [1:269] "1699.48799733991" "10207.7335023376" "13500.0416554875" ".." ...
##  $ 2012 [YR2012]: chr [1:269] "1914.77435127371" "10526.3189737173" "13303.3317888752" ".." ...
##  $ 2013 [YR2013]: chr [1:269] "2015.51496204541" "10570.9681040326" "13056.8036185828" ".." ...
##  $ 2014 [YR2014]: chr [1:269] "2069.42464180385" "11259.2967004521" "13003.267118909" ".." ...
##  $ 2015 [YR2015]: chr [1:269] "2087.30532306683" "11658.8660596051" "12015.6313951015" ".." ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   `Series Name` = col_character(),
##   ..   `Series Code` = col_character(),
##   ..   `Country Name` = col_character(),
##   ..   `Country Code` = col_character(),
##   ..   `1995 [YR1995]` = col_character(),
##   ..   `1996 [YR1996]` = col_character(),
##   ..   `1997 [YR1997]` = col_character(),
##   ..   `1998 [YR1998]` = col_character(),
##   ..   `1999 [YR1999]` = col_character(),
##   ..   `2000 [YR2000]` = col_character(),
##   ..   `2001 [YR2001]` = col_character(),
##   ..   `2002 [YR2002]` = col_character(),
##   ..   `2003 [YR2003]` = col_character(),
##   ..   `2004 [YR2004]` = col_character(),
##   ..   `2005 [YR2005]` = col_character(),
##   ..   `2006 [YR2006]` = col_character(),
##   ..   `2007 [YR2007]` = col_character(),
##   ..   `2008 [YR2008]` = col_character(),
##   ..   `2009 [YR2009]` = col_character(),
##   ..   `2010 [YR2010]` = col_character(),
##   ..   `2011 [YR2011]` = col_character(),
##   ..   `2012 [YR2012]` = col_character(),
##   ..   `2013 [YR2013]` = col_character(),
##   ..   `2014 [YR2014]` = col_character(),
##   ..   `2015 [YR2015]` = col_character()
##   .. )
##  - attr(*, "problems")=<externalptr>
```

"Tidy"ing the data

Is this data in "tidy" format? Why or why not?

No, it is not tidy. Remember that in a tidy data set, each row is an observation, each column is a variable, and each cell is a value. As we discussed in lecture, tidy data can look different depending on what you want to do with it! Suppose that we want our observations to be country year (this is common when working with panel data). As you can see, each row has many country-years. Furthermore, year is a variable, but in its current format each year is in its own column.

What can we do to `gdp_per_cap` to make it tidy?

\textbf{We need to use a pivot. There are two sorts of pivots. We need what is now called pivot_longer(). It used to be known as gather(). gather() still works, but is no longer being developed, and the devleopers of the tidyverse recommend using pivot_longer() instead (but gather still works). If you have data that is too long (i.e. you need to add columns and eliminate rows), use pivot_wider(). I encourage you to read the documentation for both if you haven't already.}

```r
gdp_per_cap_tidy <- gdp_per_cap %>%  #pipe in data
  pivot_longer(c(`1995 [YR1995]`:`2015 [YR2015]`), #specify which cols to pivot
               names_to = "year",     #new col name for gathered cols
               values_to = "gdppc")   #new col name for gathered values
```

There are more efficent ways to specify your columns. Try to come up with one.

Cleaning up the data

Let's look at our new tidy data. Now we are getting closer to something we can work with! Is there anything you would want to change about it?

```
head(gdp_per_cap_tidy)   #check cols and values
```

```
## # A tibble: 6 x 6
##   `Series Name`        `Series Code`  `Country Name` `Country Code` year   gdppc
##   <chr>                <chr>          <chr>          <chr>          <chr>  <chr>
## 1 GDP per capita, PPP~ NY.GDP.PCAP.P~ Afghanistan    AFG            1995 ~ ..
## 2 GDP per capita, PPP~ NY.GDP.PCAP.P~ Afghanistan    AFG            1996 ~ ..
## 3 GDP per capita, PPP~ NY.GDP.PCAP.P~ Afghanistan    AFG            1997 ~ ..
## 4 GDP per capita, PPP~ NY.GDP.PCAP.P~ Afghanistan    AFG            1998 ~ ..
## 5 GDP per capita, PPP~ NY.GDP.PCAP.P~ Afghanistan    AFG            1999 ~ ..
## 6 GDP per capita, PPP~ NY.GDP.PCAP.P~ Afghanistan    AFG            2000 ~ ..
```

The first thing I would want to do is replace the unsightly values that are labeled like "1995 [YR1995]". One disadvantage of leaving it like this is that R will never be able to recognize them as numbers. This would make some things we might want to do later difficult or impossible (for example, we could never use "year" as a continuous variable).

There may be easier methods that make use of regular expressions. If you are familiar with regex and want to try, please feel free. If you don't know regex, you should be able manage this problem with dplyr functions like recode() or case_when():

```
gdp_per_cap_tidy <- gdp_per_cap_tidy %>%   #pipe in data (notice: overwriting)
  mutate(year = recode(year,               #overwrite year with years as ints
                       '1995 [YR1995]' = 1995,
                       '1996 [YR1996]' = 1996,
                       '1997 [YR1997]' = 1997,
                       '1998 [YR1998]' = 1998,
                       '1999 [YR1999]' = 1999,
                       '2000 [YR2000]' = 2000,
                       '2001 [YR2001]' = 2001,
                       '2002 [YR2002]' = 2002,
                       '2003 [YR2003]' = 2003,
                       '2004 [YR2004]' = 2004,
                       '2005 [YR2005]' = 2005,
                       '2006 [YR2006]' = 2006,
                       '2007 [YR2007]' = 2007,
                       '2008 [YR2008]' = 2008,
                       '2009 [YR2009]' = 2009,
                       '2010 [YR2010]' = 2010,
                       '2011 [YR2011]' = 2011,
                       '2012 [YR2012]' = 2012,
                       '2013 [YR2013]' = 2013,
                       '2014 [YR2014]' = 2014,
                       '2015 [YR2015]' = 2015))
```

Do we need all of the variables in the dataset? Which would you remove and why? Are there any you would rename? Write code to rename and remove variables you want to change.

```
gdp_per_cap_tidy <- gdp_per_cap_tidy %>%   #pipe in data (notice: overwriting)
  rename('country' = `Country Name`,       #rename columns
         'ccode' = `Country Code`) %>%
  select(country,                          #select only informative cols
         ccode,
         year,
         gdppc)
```

Again, there are other ways to do this. The most important thing is that your code is legible and safe.

Subsetting the Data

This dataset now looks much cleaner, is easier to work with, and does not contain much unnecessary information. Now suppose that we only want to keep countries for which we have at least some data (i.e. remove countries which are missing "gdppc" for all years in the dataset). How would you approach this problem? Fill in the code to do it for you.

```
gdp_per_cap_tidy_subset <- gdp_per_cap_tidy %>% #create new df, pipe in data
  group_by(country) %>%                         #generate country groups
  filter(!all(is.na(gdppc)))   %>%              #only keep non-empty groups
  ungroup()                                     #remove groups before return
```

This doesn't seem to work. Any guesses as to why? We can check if values we expect to be "NA" actually are by running this code:

It appears that missing data is not being recognized as NA. What values are in the cells where you expect to see NA? One thing I like to check when my code isn't running as expected is the class of the column. Run the following code to check what class "gdppc" is:

```
class(gdp_per_cap_tidy$gdppc)      #check class of gdppc column
```

```
## [1] "character"
```

It is not numeric, but we want it to be. How can we coerce these values to be numeric? Does anyone know why this might solve the code issue we just experienced?

```
gdp_per_cap_tidy <- gdp_per_cap_tidy %>%  #pipe data (notice: overwriting)
  mutate(gdppc = as.numeric(gdppc))       #coerce gdppc to num and overwrite
```

```
## Warning in mask$eval_all_mutate(quo): NAs introduced by coercion
```

As you can see, when we told R that the column should contain numbers, there were some values that it didn't know how to handle (because they weren't numbers). By default, it sets these values to "NA" which actually makes them easier for us to filter out.

Now, let's try removing countries with no data again:

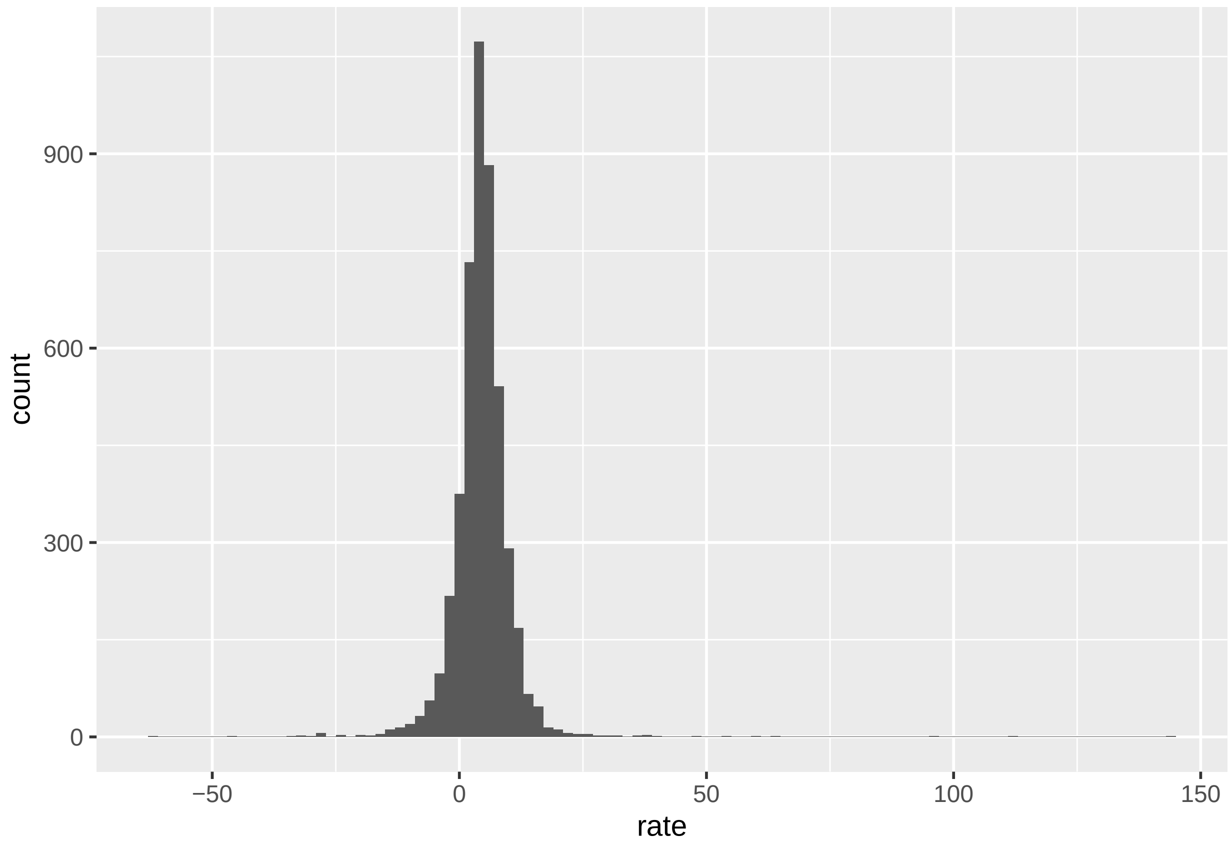
```
gdp_per_cap_tidy_subset <- gdp_per_cap_tidy %>% #create new df, pipe in data
  group_by(country) %>%                         #generate country groups
  filter(!all(is.na(gdppc)))   %>%              #only keep non-empty groups
  ungroup()                                     #remove groups before return
```

So, we've managed to remove all the countries (the dataset actually includes regions and other aggregate categories of states as well) that had no data. For our final task, let's compute the year-over-year change in GDP per capita for each observation as a percentage. After adding it to the dataset, create a histogram so you can see the distribution of annual growth or decline in GDP per capita. What other visualization would you use to understand something else about this data?

```
gdp_per_cap_change <- gdp_per_cap_tidy_subset %>% #new df, pipe subsetted data
  group_by(country) %>%                            #create country groups
  arrange(country, year, by_group = TRUE) %>%      #arrange by year w/in group
  mutate(rate = 100 * (gdppc - lag(gdppc))/lag(gdppc)) %>%  #add percent change
  ungroup()                                        #ungroup before returning
```

```
ggplot(gdp_per_cap_change,          #create plot
       mapping = aes(x = rate)) +   #specify global mapping (only 1 for hist)
  geom_histogram(binwidth = 2)      #add hist layer, make bins smaller
```

```
## Warning: Removed 398 rows containing non-finite values (stat_bin).
```



What would you do to make this plot more readable and informative?

Merging datasets

First, let's load the GNI data.

```
gni_per_cap <- read_csv('data/gni_per_cap.zip')
```

```
## Multiple files in zip: reading '4a503748-46be-48c5-9973-f8cd7e8ab4cd_Data.csv'
```

```
## Rows: 269 Columns: 27
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (27): Country Name, Country Code, Series Name, Series Code, 1993 [YR1993...

##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
            # load data directly from zip
```

Upon examination, we can see that it is in the same format as the GDP data.

```
head(gni_per_cap)
```

```
## # A tibble: 6 x 27
##   `Country Name` `Country Code` `Series Name`      `Series Code` `1993 [YR1993]`
##   <chr>          <chr>          <chr>              <chr>         <chr>
## 1 Afghanistan    AFG            GNI per capita, A~ NY.GNP.PCAP.~ ..
## 2 Albania        ALB            GNI per capita, A~ NY.GNP.PCAP.~ 320
## 3 Algeria        DZA            GNI per capita, A~ NY.GNP.PCAP.~ 1730
## 4 American Samoa ASM            GNI per capita, A~ NY.GNP.PCAP.~ ..
## 5 Andorra        AND            GNI per capita, A~ NY.GNP.PCAP.~ ..
## 6 Angola         AGO            GNI per capita, A~ NY.GNP.PCAP.~ 290
## # ... with 22 more variables: 1994 [YR1994] <chr>, 1995 [YR1995] <chr>,
## #   1996 [YR1996] <chr>, 1997 [YR1997] <chr>, 1998 [YR1998] <chr>,
## #   1999 [YR1999] <chr>, 2000 [YR2000] <chr>, 2001 [YR2001] <chr>,
## #   2002 [YR2002] <chr>, 2003 [YR2003] <chr>, 2004 [YR2004] <chr>,
## #   2005 [YR2005] <chr>, 2006 [YR2006] <chr>, 2007 [YR2007] <chr>,
## #   2008 [YR2008] <chr>, 2009 [YR2009] <chr>, 2010 [YR2010] <chr>,
## #   2011 [YR2011] <chr>, 2012 [YR2012] <chr>, 2013 [YR2013] <chr>, ...
```

Since we want to merge it with the gdp data, we need it to be in country-year format as well. Again, we need to use pivot_longer(). Since it looks like we have the same issue that missing data is represented by .. instead of NA and the class of gnipc is character instead of numeric. We can coerce the values in the new "gnipc" column to numeric within the call of pivot_longer() to save ourselves some time.

```
gni_per_cap_tidy <- gni_per_cap %>%  #pipe in data
  pivot_longer(c(`1993 [YR1993]`:`2015 [YR2015]`), #specify which cols to pivot
               names_to = "year",     #new col name for gathered cols
               values_to = "gnipc",   #new col name for gathered values
               values_transform = list(gnipc = as.numeric))
```

```
## Warning in lapply(val_cols, values_transform[[value]]): NAs introduced by
## coercion

## Warning in lapply(val_cols, values_transform[[value]]): NAs introduced by
## coercion

## Warning in lapply(val_cols, values_transform[[value]]): NAs introduced by
## coercion

## Warning in lapply(val_cols, values_transform[[value]]): NAs introduced by
## coercion

## Warning in lapply(val_cols, values_transform[[value]]): NAs introduced by
## coercion

## Warning in lapply(val_cols, values_transform[[value]]): NAs introduced by
## coercion

## Warning in lapply(val_cols, values_transform[[value]]): NAs introduced by
## coercion

## Warning in lapply(val_cols, values_transform[[value]]): NAs introduced by
## coercion

## Warning in lapply(val_cols, values_transform[[value]]): NAs introduced by
## coercion

## Warning in lapply(val_cols, values_transform[[value]]): NAs introduced by
## coercion

## Warning in lapply(val_cols, values_transform[[value]]): NAs introduced by
## coercion

## Warning in lapply(val_cols, values_transform[[value]]): NAs introduced by
## coercion

## Warning in lapply(val_cols, values_transform[[value]]): NAs introduced by
## coercion

## Warning in lapply(val_cols, values_transform[[value]]): NAs introduced by
## coercion

## Warning in lapply(val_cols, values_transform[[value]]): NAs introduced by
## coercion

## Warning in lapply(val_cols, values_transform[[value]]): NAs introduced by
## coercion

## Warning in lapply(val_cols, values_transform[[value]]): NAs introduced by
## coercion

## Warning in lapply(val_cols, values_transform[[value]]): NAs introduced by
## coercion

## Warning in lapply(val_cols, values_transform[[value]]): NAs introduced by
## coercion

## Warning in lapply(val_cols, values_transform[[value]]): NAs introduced by
## coercion

## Warning in lapply(val_cols, values_transform[[value]]): NAs introduced by
## coercion

## Warning in lapply(val_cols, values_transform[[value]]): NAs introduced by
## coercion

## Warning in lapply(val_cols, values_transform[[value]]): NAs introduced by
## coercion
```

```
                              #coerce gnipc to numeric
```

Let's recode the years like we did with the gdp data. Note that it has 2 more years (1993 and 1994) than the gdp data, so we need to account for those as well.

```
gni_per_cap_tidy <- gni_per_cap_tidy %>%   #pipe in data (notice: overwriting)
  mutate(year = recode(year,               #overwrite year with years as ints
                       '1993 [YR1993]' = 1993,
                       '1994 [YR1994]' = 1994,
                       '1995 [YR1995]' = 1995,
                       '1996 [YR1996]' = 1996,
                       '1997 [YR1997]' = 1997,
                       '1998 [YR1998]' = 1998,
                       '1999 [YR1999]' = 1999,
                       '2000 [YR2000]' = 2000,
                       '2001 [YR2001]' = 2001,
                       '2002 [YR2002]' = 2002,
                       '2003 [YR2003]' = 2003,
                       '2004 [YR2004]' = 2004,
                       '2005 [YR2005]' = 2005,
                       '2006 [YR2006]' = 2006,
                       '2007 [YR2007]' = 2007,
                       '2008 [YR2008]' = 2008,
                       '2009 [YR2009]' = 2009,
                       '2010 [YR2010]' = 2010,
                       '2011 [YR2011]' = 2011,
                       '2012 [YR2012]' = 2012,
                       '2013 [YR2013]' = 2013,
                       '2014 [YR2014]' = 2014,
                       '2015 [YR2015]' = 2015))
```

As with the GDP data, we don't need the series name or series code. Instead of selecting the columns we want to keep, we can also specify the columns we want to exclude. Let's also replace the column names containing spaces again (this way we can call the column names without backticks).

```
gni_per_cap_tidy <- gni_per_cap_tidy %>%   #pipe in data (notice: overwriting)
  rename('country' = `Country Name`,       #rename columns
         'ccode' = `Country Code`) %>%
  select(-c('Series Name', 'Series Code'))    #exclude uninformative columns
```

Now, let's remove countries for which we have no data in the sample.

```
gni_per_cap_tidy_subset <- gni_per_cap_tidy %>% #create new df, pipe in data
  group_by(country) %>%                         #generate country groups
  filter(!all(is.na(gnipc)))  %>%               #only keep non-empty groups
  ungroup()                                     #ungroup before return
```

Since these datasets came from the same database, there's a good chance that they use the same naming conventions for country names. However, I prefer to match on country code because there are less likely to have minor differences in spelling that can prevent successful matching. If you have datasets from different sources, I recommend reading the documentation to figure out which country code system they use (there are many standard ones). If you have data in different country code formats, there are helpful packages like `ccode` that can help you swap between different coding standards.

Fortunately, we don't have to do that here. Since we have fewer years of observations in the gdp data, let's use a left join and specify GDP data as the `x` argument. This will result in a merged dataset that keeps every observation in the GDP data and adds corresponding values from the GNI data. Let's try it and see what it looks like.

```
gdp_gni_merged <- left_join(x = gdp_per_cap_tidy_subset, #specify first df
                            y = gni_per_cap_tidy_subset, #specify second df
                            by = c("ccode", "year"))     #specify cols to match
```

Since we had a column named "country" in both datasets, but we did not use it to match, it was added from the GNI data to the GDP data in the merge. Notice that they have been renamed `country.x` and `country.y`, reflecting which of the datasets each column came from. They contain the same value for every observation, so it is redundant to keep both. There are several ways we could handle this, but one option is to exclude it from the join in the first place. We can use a pipe and select function on the second data frame within the join function like this:

```
gdp_gni_merged_2 <- left_join(x = gdp_per_cap_tidy_subset, #specify first df
                            y = gni_per_cap_tidy_subset %>% #specify second df
                              select(-country),             #exclude country
                            by = c("ccode", "year"))     #specify cols to match
```

Now that you have this merged data, what can you do with it? Try to come up with a visualization to explore the relationship between GNI and GDP!