Data Management*

Chapter 5

Anna Ziff

R Workflow for Economists

Contents

---

*Please contact anna.ziff@duke.edu if there are errors.

In this chapter, we will go through the R functions needed for data management. The built-in R functions are useful tools and it is important to know their syntax. There are several packages that are widely used that are helpful to work with larger data, produce cleaner code, and be more efficient in data management. The suite of packages called `tidyverse` is especially common.

Here are all the libraries you should install for this chapter. Most of these are packages in `tidyverse`.

```
library(dplyr)
library(ggplot2)
library(magrittr)
library(readr)
library(readxl)
library(stringr)
library(tidyr)
```

We will practice importing data. Go to the Dropbox folder with the example data. Download the entire folder to a convenient file on your computer and save the file path for use in the below notes.

Built-in Functions

Import and Export

Importing text files, including those files with extensions `.txt` and `.csv` can be done with the function `read.table()`. This function reads a file and creates a data frame. The function `read.csv()` is a wrapper meaning it implements the same command but sets some defaults optimized for `.csv` files.

```
df1 <- read.csv("gapminder.csv")
str(df1)
```

```
## 'data.frame':    197 obs. of  4 variables:
##  $ country: chr  "Afghanistan" "Albania" "Algeria" "Andorra" ...
##  $ gdp    : int  574 4520 4780 42100 3750 13300 10600 3920 55100 47800 ...
##  $ gini   : num  36.8 29 27.6 40 42.6 40 41.8 31.9 32.3 30.6 ...
##  $ region : chr  "Asia & Pacific" "Europe" "Arab States" "Europe" ...
```

```
head(df1) # Display the first 5 rows
```

```
##                 country   gdp gini              region
## 1           Afghanistan   574 36.8      Asia & Pacific
## 2               Albania  4520 29.0              Europe
## 3               Algeria  4780 27.6         Arab States
## 4               Andorra 42100 40.0              Europe
## 5                Angola  3750 42.6              Africa
## 6 Antigua and Barbuda 13300 40.0 South/Latin America
```

This command does the exact same thing.

```
df2 <- read.table("gapminder.csv", header = TRUE, sep = ",")
str(df2)
```

```
## 'data.frame':    197 obs. of  4 variables:
##  $ country: chr  "Afghanistan" "Albania" "Algeria" "Andorra" ...
##  $ gdp    : int  574 4520 4780 42100 3750 13300 10600 3920 55100 47800 ...
##  $ gini   : num  36.8 29 27.6 40 42.6 40 41.8 31.9 32.3 30.6 ...
##  $ region : chr  "Asia & Pacific" "Europe" "Arab States" "Europe" ...
```

Here is an example of reading a `.txt` file with `read.delim()`. Note that we need to specify the delimiter, in this case a space. You will need to inspect your file to determine the delimiter.

```r
df3 <- read.delim("gapminder.txt", sep = " ")
str(df3)
```

```
## 'data.frame':    197 obs. of  4 variables:
##  $ country: chr  "Afghanistan" "Albania" "Algeria" "Andorra" ...
##  $ gdp    : int  574 4520 4780 42100 3750 13300 10600 3920 55100 47800 ...
##  $ gini   : num  36.8 29 27.6 40 42.6 40 41.8 31.9 32.3 30.6 ...
##  $ region : chr  "Asia & Pacific" "Europe" "Arab States" "Europe" ...
```

These three functions have many arguments available to adjust how the data files are read. The argument `stringsAsFactors` is automatically set to `FALSE`. If it is set to `TRUE`, then variables with character strings are read in as factors.

```r
df4 <- read.csv("gapminder.csv", stringsAsFactors = TRUE)
str(df4)
```

```
## 'data.frame':    197 obs. of  4 variables:
##  $ country: Factor w/ 195 levels "Afghanistan",..: 1 2 3 4 5 6 7 8 9 10 ...
##  $ gdp    : int  574 4520 4780 42100 3750 13300 10600 3920 55100 47800 ...
##  $ gini   : num  36.8 29 27.6 40 42.6 40 41.8 31.9 32.3 30.6 ...
##  $ region : Factor w/ 7 levels "Africa","Arab States",..: 3 4 2 4 1 7 7 4 3 4 ...
```

You can specify the classes of all the columns using the argument `colClasses`. This is especially usefull if the dataset is larger as it means that R does not need to determine the classes itself.

```r
df5 <- read.csv("gapminder.csv",
                colClasses = c("character", "integer", "double", "factor"))
str(df5)
```

```
## 'data.frame':    197 obs. of  4 variables:
##  $ country: chr  "Afghanistan" "Albania" "Algeria" "Andorra" ...
##  $ gdp    : int  574 4520 4780 42100 3750 13300 10600 3920 55100 47800 ...
##  $ gini   : num  36.8 29 27.6 40 42.6 40 41.8 31.9 32.3 30.6 ...
##  $ region : Factor w/ 7 levels "Africa","Arab States",..: 3 4 2 4 1 7 7 4 3 4 ...
```

Column names (or variable names) and row names can be set while reading the file as well.

```r
df6 <- read.csv("gapminder.csv",
                col.names = c("Country", "GDP", "GiniIndex", "Region"))
# row.names for rows
str(df6)
```

```
## 'data.frame':    197 obs. of  4 variables:
##  $ Country  : chr  "Afghanistan" "Albania" "Algeria" "Andorra" ...
##  $ GDP      : int  574 4520 4780 42100 3750 13300 10600 3920 55100 47800 ...
##  $ GiniIndex: num  36.8 29 27.6 40 42.6 40 41.8 31.9 32.3 30.6 ...
##  $ Region   : chr  "Asia & Pacific" "Europe" "Arab States" "Europe" ...
```

If you just want to get a sense of what types of variables a dataset contains, you can use the `nrows` argument to read in very few rows. This is especially helpful with larger datasets.

```r
checkcols <- read.csv("gapminder.csv",
                      nrows = 3)
checkcols
```

```
##       country  gdp gini         region
## 1 Afghanistan  574 36.8 Asia & Pacific
## 2     Albania 4520 29.0         Europe
## 3     Algeria 4780 27.6    Arab States
```

The built-in functions to export data are very similar to those to import data. Again, `write.table()` is the general function with `write.csv()` being a wrapper for different file types. Here is an example data frame that we will export.

```
df <- data.frame(id = seq(1:50),
                 v1 = rnorm(50, mean = 10, sd = 2),
                 v2 = rbinom(50, size = 1, prob = 0.5),
                 v3 = c(TRUE, FALSE),
                 v4 = c("Group 1", "Group 2", "Group 3", "Group 4", "Group 5"))
head(df)
```

```
##   id        v1 v2    v3      v4
## 1  1  9.718255  0  TRUE Group 1
## 2  2  9.689573  0 FALSE Group 2
## 3  3 10.072202  1  TRUE Group 3
## 4  4 13.597885  1 FALSE Group 4
## 5  5 10.897250  1  TRUE Group 5
## 6  6  9.775413  0 FALSE Group 1
```

Before exporting, make sure the correct directory is set. Remember you can use `getwd()` to check and `setwd()` to change the directory.

The function `write.csv()` exports a comma-delimited text file. You need to specify the object to be saved and the name of the file. The argument `row.names` determines whether the row names are exported as well. Unless you have custom row names, it is useful to set this argument to `FALSE`.

```
write.csv(df, file = "df_csv.csv", row.names = FALSE)
```

For greater generality, `write.table()` is available.

```
write.table(df, file = "df_table.txt", sep = "\t")
```

If you want to read Excel files, you will need an external package. A good option is the package `readxl` to access the function `read_excel()`. This package relies on tibbles just like `readr` (discussed below).

```
tib4 <- read_excel("gapminder.xlsx")
head(tib4)
```

```
## # A tibble: 6 x 4
##   country             gdp    gini region
##   <chr>               <chr> <dbl> <chr>
## 1 Afghanistan         574    36.8 Asia & Pacific
## 2 Albania             4520   29   Europe
## 3 Algeria             4780   27.6 Arab States
## 4 Andorra             42100  40   Europe
## 5 Angola              3750   42.6 Africa
## 6 Antigua and Barbuda 13300  40   South/Latin America
```

Other packages that allow you to read and write Excel files include `xlsx` and `r2excel`.

There are other packages that allow you to import and export datasets in other formats. For example, the `foreign` package allows for data files from SPSS, SAS, and STATA.

R Saved Objects

There are R-specific data formats to save the environment or components of it. To save the entire environment, use the `.RData` format.

```
ids <- 1:100
verbose_sqrt <- function(num) {
```

```r
  if (num >= 0) {
    return(sqrt(num))
  } else {
    return("Negative number input.")
  }
}
save(ids, verbose_sqrt, file = "workspace.RData")
```

This file includes both the objects and the names of the objects. You can directly load `.RData` and the workspace is populated. If you only want to save one object, you can use `.rds` files instead. These do not save the object's name. They are very memory-efficient (similar to saving a zipped file).

```r
head(df)
```

```
##   id        v1 v2    v3      v4
## 1  1  9.718255  0  TRUE Group 1
## 2  2  9.689573  0 FALSE Group 2
## 3  3 10.072202  1  TRUE Group 3
## 4  4 13.597885  1 FALSE Group 4
## 5  5 10.897250  1  TRUE Group 5
## 6  6  9.775413  0 FALSE Group 1
```

```r
saveRDS(df, "dataframe.rds")
```

Importing these objects is done as follows.

```r
load("workspace.RData") # Imports objects and names
mydf <- readRDS("dataframe.rds") # Imports one object assigned to mydf
```

Select Variables

```r
df <- read.csv("gapminder_large.csv")
str(df)
```

```
## 'data.frame':    195 obs. of  21 variables:
##  $ country     : chr  "Afghanistan" "Albania" "Algeria" "Andorra" ...
##  $ gdp_2015    : int  574 4520 4780 42100 3750 13300 10600 3920 55100 47800 ...
##  $ gini_2015   : num  36.8 29 27.6 40 42.6 40 41.8 31.9 32.3 30.6 ...
##  $ region      : chr  "Asia & Pacific" "Europe" "Arab States" "Europe" ...
##  $ co2_2015    : num  0.262 1.6 3.8 5.97 1.22 5.84 4.64 1.65 16.8 7.7 ...
##  $ co2_2016    : num  0.245 1.57 3.64 6.07 1.18 5.9 4.6 1.76 17 7.7 ...
##  $ co2_2017    : num  0.247 1.61 3.56 6.27 1.14 5.89 4.55 1.7 17 7.94 ...
##  $ co2_2018    : num  0.254 1.59 3.69 6.12 1.12 5.88 4.41 1.89 16.9 7.75 ...
##  $ cpi_2012    : int  8 33 34 NA 22 NA 35 34 85 69 ...
##  $ cpi_2013    : int  8 31 36 NA 23 NA 34 36 81 69 ...
##  $ cpi_2014    : int  12 33 36 NA 19 NA 34 37 80 72 ...
##  $ cpi_2015    : int  11 36 36 NA 15 NA 32 35 79 76 ...
##  $ cpi_2016    : int  15 39 34 NA 18 NA 36 33 79 75 ...
##  $ cpi_2017    : int  15 38 33 NA 19 NA 39 35 77 75 ...
##  $ lifeexp_2012: num  60.8 77.8 76.8 82.4 61.3 76.7 76 74.7 82.5 81 ...
##  $ lifeexp_2013: num  61.3 77.9 76.9 82.5 61.9 76.8 76.1 75.2 82.6 81.2 ...
##  $ lifeexp_2014: num  61.2 77.9 77 82.5 62.8 76.8 76.4 75.3 82.5 81.4 ...
##  $ lifeexp_2015: num  61.2 78 77.1 82.6 63.3 76.9 76.5 75.3 82.5 81.5 ...
##  $ lifeexp_2016: num  61.2 78.1 77.4 82.7 63.8 77 76.5 75.4 82.5 81.7 ...
##  $ lifeexp_2017: num  63.4 78.2 77.7 82.7 64.2 77 76.7 75.6 82.4 81.8 ...
##  $ lifeexp_2018: num  63.7 78.3 77.9 NA 64.6 77.2 76.8 75.8 82.5 81.9 ...
```

The built-in functions import data as data frames. Chapter 1 discusses how to select variables (columns). Here is a small review.

```
df[, 1:3]
df[, c(2, 4)]
df[, "cpi_2017"]
df[, c("lifeexp_2012", "cpi_2016")]
df[c("country", "region")]
df[1:3]
df$gini_2015
```

Rename and Create Variables

The names of a data frame can be access with `names()`. This is an attribute of the data frame and can be used to rename all the variables this way.

```
names(df)
```

```
##  [1] "country"      "gdp_2015"     "gini_2015"    "region"       "co2_2015"
##  [6] "co2_2016"     "co2_2017"     "co2_2018"     "cpi_2012"     "cpi_2013"
## [11] "cpi_2014"     "cpi_2015"     "cpi_2016"     "cpi_2017"     "lifeexp_2012"
## [16] "lifeexp_2013" "lifeexp_2014" "lifeexp_2015" "lifeexp_2016" "lifeexp_2017"
## [21] "lifeexp_2018"
```

```
names(df) <- paste0("var", 1:length(names(df)))
names(df)
```

```
##  [1] "var1"  "var2"  "var3"  "var4"  "var5"  "var6"  "var7"  "var8"  "var9"
## [10] "var10" "var11" "var12" "var13" "var14" "var15" "var16" "var17" "var18"
## [19] "var19" "var20" "var21"
```

An alternative is to use the function `setNames()`. This function can also be used for other data structures besides data frames, such as vectors.

```
vnames <- c("country", "gdp_2015", "gini_2015", "region",
            "co2_2015", "co2_2016", "co2_2017", "co2_2018",
            "cpi_2012", "cpi_2013", "cpi_2014", "cpi_2015",
            "cpi_2016", "cpi_2017", "lifeexp_2012", "lifeexp_2013",
            "lifeexp_2014", "lifeexp_2015", "lifeexp_2016", "lifeexp_2017",
            "lifeexp_2018")
df <- setNames(df, vnames)
names(df)
```

```
##  [1] "country"      "gdp_2015"     "gini_2015"    "region"       "co2_2015"
##  [6] "co2_2016"     "co2_2017"     "co2_2018"     "cpi_2012"     "cpi_2013"
## [11] "cpi_2014"     "cpi_2015"     "cpi_2016"     "cpi_2017"     "lifeexp_2012"
## [16] "lifeexp_2013" "lifeexp_2014" "lifeexp_2015" "lifeexp_2016" "lifeexp_2017"
## [21] "lifeexp_2018"
```

It is also possible to rename a subset of the variables.

```
names(df)[1] <- "COUNTRY"
names(df)
```

```
##  [1] "COUNTRY"      "gdp_2015"     "gini_2015"    "region"       "co2_2015"
##  [6] "co2_2016"     "co2_2017"     "co2_2018"     "cpi_2012"     "cpi_2013"
## [11] "cpi_2014"     "cpi_2015"     "cpi_2016"     "cpi_2017"     "lifeexp_2012"
## [16] "lifeexp_2013" "lifeexp_2014" "lifeexp_2015" "lifeexp_2016" "lifeexp_2017"
## [21] "lifeexp_2018"
```

```r
names(df)[2:3] <- c("GDP", "GINI")
names(df)
```

```
##  [1] "COUNTRY"      "GDP"          "GINI"         "region"       "co2_2015"
##  [6] "co2_2016"     "co2_2017"     "co2_2018"     "cpi_2012"     "cpi_2013"
## [11] "cpi_2014"     "cpi_2015"     "cpi_2016"     "cpi_2017"     "lifeexp_2012"
## [16] "lifeexp_2013" "lifeexp_2014" "lifeexp_2015" "lifeexp_2016" "lifeexp_2017"
## [21] "lifeexp_2018"
```

Creating new variables can be done with `cbind()` as discussed in chapter 1.

```r
random1 <- rnorm(dim(df)[1])
head(random1)
```

```
## [1] -0.9034616 -0.9315785  0.7420078 -0.5748372  0.5144001 -0.5338265
```

```r
df <- cbind(df, random1)
df[1:5, c("COUNTRY", "random1")]
```

```
##       COUNTRY    random1
## 1 Afghanistan -0.9034616
## 2     Albania -0.9315785
## 3     Algeria  0.7420078
## 4     Andorra -0.5748372
## 5      Angola  0.5144001
```

This method has the advantage that it can be used to add more than one variable at a time.

```r
random2 <- runif(dim(df)[1])
random3 <- rexp(dim(df)[1])
df <- cbind(df, random2, random3)
df[1:5, c("COUNTRY", "random2", "random3")]
```

```
##       COUNTRY   random2   random3
## 1 Afghanistan 0.2628448 0.7706266
## 2     Albania 0.8769630 0.8532965
## 3     Algeria 0.9611621 0.2807853
## 4     Andorra 0.6933212 1.9286870
## 5      Angola 0.9711144 0.2093739
```

The following shortcut is helpful to create one variable at a time.

```r
df$random4 <- df$random3^2
df[1:5, c("COUNTRY", "random4")]
```

```
##       COUNTRY    random4
## 1 Afghanistan 0.59386536
## 2     Albania 0.72811496
## 3     Algeria 0.07884039
## 4     Andorra 3.71983369
## 5      Angola 0.04383745
```

Filter Observations

Filtering observations can be done by row name or number, as shown in chapter 1.

```r
df[1:3, ]
df[c(3, 40), ]
df[c("4", "17"), ]
```

```
df[!c(1:190), ]
df[-c(1:190), ]
```

Filtering can also be done using logical statements.

```
df[df$random2 >= 1, ]
```

```
##  [1] COUNTRY      GDP          GINI         region       co2_2015
##  [6] co2_2016     co2_2017     co2_2018     cpi_2012     cpi_2013
## [11] cpi_2014     cpi_2015     cpi_2016     cpi_2017     lifeexp_2012
## [16] lifeexp_2013 lifeexp_2014 lifeexp_2015 lifeexp_2016 lifeexp_2017
## [21] lifeexp_2018 random1      random2      random3      random4
## <0 rows> (or 0-length row.names)
```

```
df[df$random2 >= 1 & df$random3 <= 0.5, ]
```

```
##  [1] COUNTRY      GDP          GINI         region       co2_2015
##  [6] co2_2016     co2_2017     co2_2018     cpi_2012     cpi_2013
## [11] cpi_2014     cpi_2015     cpi_2016     cpi_2017     lifeexp_2012
## [16] lifeexp_2013 lifeexp_2014 lifeexp_2015 lifeexp_2016 lifeexp_2017
## [21] lifeexp_2018 random1      random2      random3      random4
## <0 rows> (or 0-length row.names)
```

```
subset(df, df$random3 <= 0.05)[, c("COUNTRY", "random3")]
```

```
##            COUNTRY     random3
## 17         Belgium 0.017757062
## 35           Chile 0.007570992
## 46  Czech Republic 0.021670612
## 64         Georgia 0.030343974
## 65         Germany 0.043363370
## 98   Liechtenstein 0.014375120
## 99       Lithuania 0.036783109
## 100     Luxembourg 0.040162183
## 119        Namibia 0.006381497
## 126        Nigeria 0.011836921
## 135       Paraguay 0.018873897
## 144          Samoa 0.028333847
## 147   Saudi Arabia 0.023012085
## 178         Tuvalu 0.029306085
```

The `which()` function returns the row numbers that are being filtered.

```
which(df$random3 <= 0.05)
```

```
##  [1]  17  35  46  64  65  98  99 100 119 126 135 144 147 178
```

Organize

Sorting can be done by one or more columns. Note that even though the rows are re-ordered, the original row names remain.

```
dforder1 <- order(df$GINI)
head(df[dforder1, c("COUNTRY", "GINI")])
```

```
##             COUNTRY GINI
## 180         Ukraine 24.8
## 154        Slovenia 25.6
```

```
## 46   Czech Republic 26.0
## 153 Slovak Republic 26.7
## 16          Belarus 26.9
## 87       Kazakhstan 26.9
```

```
dforder2 <- order(df$region, df$GINI)
head(df[dforder2, c("COUNTRY", "region", "GINI")])
```

```
##                     COUNTRY region GINI
## 146 Sao Tome and Principe Africa 30.8
## 105                  Mali Africa 33.0
## 96                Liberia Africa 33.3
## 70                 Guinea Africa 33.7
## 151          Sierra Leone Africa 34.0
## 125                 Niger Africa 34.1
```

Merge

As discussed in chapter 1, `rbind()` can be used to append additional observations. If using this approach, it is better to transform the new row(s) into a data frame. This will help avoid silently changing a variable type.

```
df1 <- df[1:98, ]
df2 <- df[99:195, ]
rbind(df1, df2)
```

An even more robust approach is to use the `merge()` function. This allows for the two data frames to have different variables and similar observations. As long as there is at least one variable common to both data frames, they can be merged. Here is a very simple example.

```
df1 <- df[1:5, c("COUNTRY", "region")]
df2 <- df[1:7, c("COUNTRY", "GDP", "GINI")]
merge(df1, df2, by = "COUNTRY")
```

```
##       COUNTRY         region   GDP GINI
## 1 Afghanistan Asia & Pacific   574 36.8
## 2     Albania         Europe  4520 29.0
## 3     Algeria    Arab States  4780 27.6
## 4     Andorra         Europe 42100 40.0
## 5      Angola         Africa  3750 42.6
```

Note that `df2` has 7 observations while `df1` only has 5. Yet, the output of the merge has 5 observations. This is because the arguments `all.x` and `all.y` are set to `FALSE` by default. This means that only rows that appear in both are present in the output. If we set `all.y = TRUE`, all the rows of `df2` are added with missing values for `region`.

```
merge(df1, df2, by = "COUNTRY", all.y = TRUE)
```

```
##               COUNTRY         region   GDP GINI
## 1         Afghanistan Asia & Pacific   574 36.8
## 2             Albania         Europe  4520 29.0
## 3             Algeria    Arab States  4780 27.6
## 4             Andorra         Europe 42100 40.0
## 5              Angola         Africa  3750 42.6
## 6 Antigua and Barbuda           <NA> 13300 40.0
## 7           Argentina           <NA> 10600 41.8
```

If you want to keep all the rows in both data frames, the argument `all = TRUE` sets both `all.x = TRUE` and `all.y = TRUE`.

```r
merge(df1, df2, by = "COUNTRY", all = TRUE)
```

```
##               COUNTRY         region   GDP GINI
## 1         Afghanistan Asia & Pacific   574 36.8
## 2             Albania         Europe  4520 29.0
## 3             Algeria    Arab States  4780 27.6
## 4             Andorra         Europe 42100 40.0
## 5              Angola         Africa  3750 42.6
## 6 Antigua and Barbuda           <NA> 13300 40.0
## 7           Argentina           <NA> 10600 41.8
```

Suppose the variable you are merging on has different names in the two data frames. The arguments `by.x` and `by.y` allow for you to specify both variables.

```r
names(df1)[1] <- "country"
merge(df1, df2, by.x = "country", by.y = "COUNTRY")
```

```
##       country         region   GDP GINI
## 1 Afghanistan Asia & Pacific   574 36.8
## 2     Albania         Europe  4520 29.0
## 3     Algeria    Arab States  4780 27.6
## 4     Andorra         Europe 42100 40.0
## 5      Angola         Africa  3750 42.6
```

If the two data frames have different variables with the same name, the merge will not combine these columns. This even applies if the columns are different types.

```r
df1 <- df[c("COUNTRY", "region", "GDP")]
df1$GDP <- as.character(df1$GDP) # GDP is now character in df1
merge(df1, df2, by = "COUNTRY")
```

```
##               COUNTRY              region GDP.x GDP.y GINI
## 1         Afghanistan      Asia & Pacific   574   574 36.8
## 2             Albania              Europe  4520  4520 29.0
## 3             Algeria         Arab States  4780  4780 27.6
## 4             Andorra              Europe 42100 42100 40.0
## 5              Angola              Africa  3750  3750 42.6
## 6 Antigua and Barbuda South/Latin America 13300 13300 40.0
## 7           Argentina South/Latin America 10600 10600 41.8
```

`tidyverse` Functions

Hadley Wickham developed the idea behind a suite of packages that streamline data work called `tidyverse`. There are many packages in this suite that relate to different types of datasets and parts of the data process. This chapter goes through `dplyr`, `tidyr`, and `readr`.

Import and Export: `readr`

The functions in the `readr` package to read and write data are faster than the built-in functions. Apart from efficiency, they have another advantage in that they help ensure consistency in the imported data. For example, if there are spaces in the variable name, `read.csv()`, the built-in function, will automatically remove these. The `readr` function `read_csv()` will not remove them.

```r
tib1 <- read_csv("gapminder.csv")
```

```
## Rows: 197 Columns: 4
```

```
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (2): country, region
## dbl (2): gdp, gini
##
## 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.
```

```
str(tib1)
```

```
## spec_tbl_df [197 x 4] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ country: chr [1:197] "Afghanistan" "Albania" "Algeria" "Andorra" ...
##  $ gdp    : num [1:197] 574 4520 4780 42100 3750 13300 10600 3920 55100 47800 ...
##  $ gini   : num [1:197] 36.8 29 27.6 40 42.6 40 41.8 31.9 32.3 30.6 ...
##  $ region : chr [1:197] "Asia & Pacific" "Europe" "Arab States" "Europe" ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   country = col_character(),
##   ..   gdp = col_double(),
##   ..   gini = col_double(),
##   ..   region = col_character()
##   .. )
##  - attr(*, "problems")=<externalptr>
```

Immediately, you can see that the data structure is different. The package `readr`, and all the packages in the `tidyverse` suite, rely on a data structure called tibbles instead of data frames. The two main differences between tibbles and data frames are the following. More information on the differences is available here.

- Unlike data frames, tibbles only show the first 10 rows and enough columns to fit on the screen. Each column is printed with its type.
- When subsetting, `[]` always returns another tibble and `[[]]` always returns a vector.

Just like in `read.csv()`, you can specify the columns.

```
tib2 <- read_csv("gapminder.csv",
                 col_types = list(col_character(),
                                  col_integer(),
                                  col_double(),
                                  col_factor()))
str(tib2)
```

```
## spec_tbl_df [197 x 4] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ country: chr [1:197] "Afghanistan" "Albania" "Algeria" "Andorra" ...
##  $ gdp    : int [1:197] 574 4520 4780 42100 3750 13300 10600 3920 55100 47800 ...
##  $ gini   : num [1:197] 36.8 29 27.6 40 42.6 40 41.8 31.9 32.3 30.6 ...
##  $ region : Factor w/ 7 levels "Asia & Pacific",..: 1 2 3 2 4 5 5 2 1 2 ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   country = col_character(),
##   ..   gdp = col_integer(),
##   ..   gini = col_double(),
##   ..   region = col_factor(levels = NULL, ordered = FALSE, include_na = FALSE)
##   .. )
##  - attr(*, "problems")=<externalptr>
```

If you want to completely rename the columns, you can do so with the option `col_names`. You will just need to tell R to skip reading in the first line of the file.

```r
tib3 <- read_csv("gapminder.csv", skip = 1,
                 col_names = c("V1", "V2", "V3", "V4"))
```

```
## Rows: 197 Columns: 4
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (2): V1, V4
## dbl (2): V2, V3
##
## 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.
```

```r
str(tib3)
```

```
## spec_tbl_df [197 x 4] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ V1: chr [1:197] "Afghanistan" "Albania" "Algeria" "Andorra" ...
##  $ V2: num [1:197] 574 4520 4780 42100 3750 13300 10600 3920 55100 47800 ...
##  $ V3: num [1:197] 36.8 29 27.6 40 42.6 40 41.8 31.9 32.3 30.6 ...
##  $ V4: chr [1:197] "Asia & Pacific" "Europe" "Arab States" "Europe" ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   V1 = col_character(),
##   ..   V2 = col_double(),
##   ..   V3 = col_double(),
##   ..   V4 = col_character()
##   .. )
##  - attr(*, "problems")=<externalptr>
```

The argument `n_max` determines the maximum number of lines that are read.

```r
read_csv("gapminder.csv", n_max = 3)
```

```
## Rows: 3 Columns: 4
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (2): country, region
## dbl (2): gdp, gini
##
## 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.
```

```
## # A tibble: 3 x 4
##   country       gdp  gini region
##   <chr>       <dbl> <dbl> <chr>
## 1 Afghanistan   574  36.8 Asia & Pacific
## 2 Albania      4520  29   Europe
## 3 Algeria      4780  27.6 Arab States
```

The `readr` analogues to `read.table()` and `read.delim()` are `read_table()` and `read_delim()`. They have similar arguments as `read_csv()`. Reading in data files usually presents unexpected difficulties and complications, and the myriad of arguments available can help address any formatting issues automatically.

The write functions in `readr` are faster than the built-in functions and automatically omit row names.

```r
write_csv(df, file = "df_csv_readr.csv")
```

Practice Exercises 5.1

1. Another way to list the column types is string shortcuts. For example `"d"` for double, `"c"` for character, etc. Check the documentation for `read_csv()`, and call in `"gapminder.csv"` with a character column, an integer column, a double column, and a factor column.
2. You can also easily skip columns with this shorthand. Why do you think this be useful? Call in `"gapminder.csv"` again skipping the `Region` column.

Transform: `dplyr`

The package `dplyr` includes functions that transform tibbles and data frames.

```
df <- read_csv("gapminder.csv")
```

```
## Rows: 197 Columns: 4
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (2): country, region
## dbl (2): gdp, gini
##
## 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.
```

```
head(df)
```

```
## # A tibble: 6 x 4
##   country               gdp  gini region
##   <chr>               <dbl> <dbl> <chr>
## 1 Afghanistan           574  36.8 Asia & Pacific
## 2 Albania              4520  29   Europe
## 3 Algeria              4780  27.6 Arab States
## 4 Andorra             42100  40   Europe
## 5 Angola               3750  42.6 Africa
## 6 Antigua and Barbuda 13300  40   South/Latin America
```

Select Variables

The general form of functions in `dplyr` involves identifying the data frame first and then specifying the options. To demonstrate, the function `select` chooses which variables.

```
select(tib1, country)
```

```
## # A tibble: 197 x 1
##    country
##    <chr>
##  1 Afghanistan
##  2 Albania
##  3 Algeria
##  4 Andorra
##  5 Angola
##  6 Antigua and Barbuda
##  7 Argentina
##  8 Armenia
##  9 Australia
## 10 Austria
## # ... with 187 more rows
```

Note that the original data frame is not changed. You will have to assign an object if you want to save this selection in an object.

```
head(tib1)
```

```
## # A tibble: 6 x 4
##   country              gdp  gini region
##   <chr>              <dbl> <dbl> <chr>
## 1 Afghanistan          574  36.8 Asia & Pacific
## 2 Albania             4520  29   Europe
## 3 Algeria             4780  27.6 Arab States
## 4 Andorra            42100  40   Europe
## 5 Angola              3750  42.6 Africa
## 6 Antigua and Barbuda 13300  40   South/Latin America
```

There are several ways to select more than one variable. The last method used a helper, `starts_with()`. See the documentation for `select` for other helpers.

```
select(tib1, country, gdp)
select(tib1, gdp:gini)
select(tib1, -gdp)
select(tib1, -c(country, gini))
select(tib1, starts_with("g"))
```

Sometimes it is desirable to rename variables when selecting them. This is very convenient in `select`!

```
select(tib1, country_name = country)
```

```
## # A tibble: 197 x 1
##    country_name
##    <chr>
##  1 Afghanistan
##  2 Albania
##  3 Algeria
##  4 Andorra
##  5 Angola
##  6 Antigua and Barbuda
##  7 Argentina
##  8 Armenia
##  9 Australia
## 10 Austria
## # ... with 187 more rows
```

```
select(tib1, var = starts_with("g"))
```

```
## # A tibble: 197 x 2
##     var1  var2
##    <dbl> <dbl>
##  1   574  36.8
##  2  4520  29
##  3  4780  27.6
##  4 42100  40
##  5  3750  42.6
##  6 13300  40
##  7 10600  41.8
##  8  3920  31.9
##  9 55100  32.3
```

```
## 10 47800  30.6
## # ... with 187 more rows
```

Rename and Create Variables

If you want to rename variables without dropping any, use the function `rename`.

```
rename(tib1, country_name = country)
```

```
## # A tibble: 197 x 4
##    country_name          gdp  gini region
##    <chr>               <dbl> <dbl> <chr>
##  1 Afghanistan           574  36.8 Asia & Pacific
##  2 Albania              4520  29   Europe
##  3 Algeria              4780  27.6 Arab States
##  4 Andorra             42100  40   Europe
##  5 Angola               3750  42.6 Africa
##  6 Antigua and Barbuda 13300  40   South/Latin America
##  7 Argentina           10600  41.8 South/Latin America
##  8 Armenia              3920  31.9 Europe
##  9 Australia           55100  32.3 Asia & Pacific
## 10 Austria             47800  30.6 Europe
## # ... with 187 more rows
```

```
rename(tib1, country_name = country, gdp_percapita = gdp)
```

```
## # A tibble: 197 x 4
##    country_name        gdp_percapita  gini region
##    <chr>                       <dbl> <dbl> <chr>
##  1 Afghanistan                   574  36.8 Asia & Pacific
##  2 Albania                      4520  29   Europe
##  3 Algeria                      4780  27.6 Arab States
##  4 Andorra                     42100  40   Europe
##  5 Angola                       3750  42.6 Africa
##  6 Antigua and Barbuda         13300  40   South/Latin America
##  7 Argentina                   10600  41.8 South/Latin America
##  8 Armenia                      3920  31.9 Europe
##  9 Australia                   55100  32.3 Asia & Pacific
## 10 Austria                     47800  30.6 Europe
## # ... with 187 more rows
```

The function `mutate()` allows for new variables to be added to the data frame or existing variables to be modified without changing the other variables.

```
mutate(tib1, gdp_sq = gdp^2)
```

```
## # A tibble: 197 x 5
##    country               gdp  gini region                  gdp_sq
##    <chr>               <dbl> <dbl> <chr>                    <dbl>
##  1 Afghanistan           574  36.8 Asia & Pacific          329476
##  2 Albania              4520  29   Europe                20430400
##  3 Algeria              4780  27.6 Arab States           22848400
##  4 Andorra             42100  40   Europe              1772410000
##  5 Angola               3750  42.6 Africa                14062500
##  6 Antigua and Barbuda 13300  40   South/Latin America  176890000
##  7 Argentina           10600  41.8 South/Latin America  112360000
##  8 Armenia              3920  31.9 Europe                15366400
```

```
## 9 Australia           55100  32.3 Asia & Pacific      3036010000
## 10 Austria            47800  30.6 Europe              2284840000
## # ... with 187 more rows
```

```r
mutate(tib1, row_id = 1:length(country))
```

```
## # A tibble: 197 x 5
##    country              gdp  gini region              row_id
##    <chr>              <dbl> <dbl> <chr>                <int>
##  1 Afghanistan          574  36.8 Asia & Pacific           1
##  2 Albania             4520  29   Europe                   2
##  3 Algeria             4780  27.6 Arab States              3
##  4 Andorra            42100  40   Europe                   4
##  5 Angola              3750  42.6 Africa                   5
##  6 Antigua and Barbuda 13300  40   South/Latin America      6
##  7 Argentina          10600  41.8 South/Latin America      7
##  8 Armenia             3920  31.9 Europe                   8
##  9 Australia          55100  32.3 Asia & Pacific           9
## 10 Austria            47800  30.6 Europe                  10
## # ... with 187 more rows
```

```r
mutate(tib1, gdp_large = ifelse(gdp >= 25000, TRUE, FALSE))
```

```
## # A tibble: 197 x 5
##    country              gdp  gini region              gdp_large
##    <chr>              <dbl> <dbl> <chr>               <lgl>
##  1 Afghanistan          574  36.8 Asia & Pacific      FALSE
##  2 Albania             4520  29   Europe              FALSE
##  3 Algeria             4780  27.6 Arab States         FALSE
##  4 Andorra            42100  40   Europe              TRUE
##  5 Angola              3750  42.6 Africa              FALSE
##  6 Antigua and Barbuda 13300  40   South/Latin America FALSE
##  7 Argentina          10600  41.8 South/Latin America FALSE
##  8 Armenia             3920  31.9 Europe              FALSE
##  9 Australia          55100  32.3 Asia & Pacific      TRUE
## 10 Austria            47800  30.6 Europe              TRUE
## # ... with 187 more rows
```

If you want to create a new variable and drop the other variables, use the function `transmute()`.

```r
transmute(tib1, gini_small = ifelse(gini <= 40, TRUE, FALSE))
```

```
## # A tibble: 197 x 1
##    gini_small
##    <lgl>
##  1 TRUE
##  2 TRUE
##  3 TRUE
##  4 TRUE
##  5 FALSE
##  6 TRUE
##  7 FALSE
##  8 TRUE
##  9 TRUE
## 10 TRUE
## # ... with 187 more rows
```

Filter Observations

The function `select` allows you to choose which variables (columns) are included in your data. The function `filter` allows you choose which observations (rows) are included in your data.

```
filter(tib1, region == "North America")
```

```
## # A tibble: 2 x 4
##   country         gdp  gini region
##   <chr>         <dbl> <dbl> <chr>
## 1 Canada        50300  31.7 North America
## 2 United States 52100  41.3 North America
```

```
filter(tib1, is.na(gdp))
```

```
## # A tibble: 8 x 4
##   country         gdp  gini region
##   <chr>         <dbl> <dbl> <chr>
## 1 Djibouti         NA  44.1 Arab States
## 2 Eritrea          NA  40   Africa
## 3 Liechtenstein    NA  40   Europe
## 4 Venezuela        NA  46.9 South/Latin America
## 5 Holy See         NA  40   Europe
## 6 North Korea      NA  37   Asia & Pacific
## 7 Somalia          NA  48   Arab States
## 8 Syria            NA  35.2 Asia & Pacific
```

```
filter(tib1, gdp > 25000 & region != "Europe")
```

```
## # A tibble: 12 x 4
##    country                gdp  gini region
##    <chr>                <dbl> <dbl> <chr>
##  1 Australia            55100  32.3 Asia & Pacific
##  2 Bahamas              27500  43.7 South/Latin America
##  3 Brunei               32900  40   Asia & Pacific
##  4 Canada               50300  31.7 North America
##  5 Japan                47100  32.1 Asia & Pacific
##  6 Kuwait               36000  40   Middle east
##  7 New Zealand          36800  34.5 Asia & Pacific
##  8 Qatar                65100  40   Middle east
##  9 Singapore            54000  40.9 Asia & Pacific
## 10 South Korea          26100  31.6 Asia & Pacific
## 11 United Arab Emirates 40200  40   Middle east
## 12 United States        52100  41.3 North America
```

```
filter(tib1, region %in% c("North America", "Middle east"))
```

```
## # A tibble: 14 x 4
##    country                gdp  gini region
##    <chr>                <dbl> <dbl> <chr>
##  1 Canada               50300  31.7 North America
##  2 Egypt                 2700  31.2 Middle east
##  3 Iran                  6070  38.5 Middle east
##  4 Iraq                  5300  29.5 Middle east
##  5 Jordan                3310  33.7 Middle east
##  6 Kuwait               36000  40   Middle east
##  7 Lebanon               6490  31.8 Middle east
```

```
##   8 Libya                 5900  40   Middle east
##   9 Oman                 16200  40   Middle east
##  10 Qatar                65100  40   Middle east
##  11 Saudi Arabia         21400  40   Middle east
##  12 United Arab Emirates 40200  40   Middle east
##  13 United States        52100  41.3 North America
##  14 Yemen                  785  36.7 Middle east
```

To select rows based on the number index, use `slice`.

```
slice(tib1, 32:37)
```

```
## # A tibble: 6 x 4
##   country                    gdp  gini region
##   <chr>                    <dbl> <dbl> <chr>
## 1 Cape Verde                3410  47.2 Africa
## 2 Central African Republic   347  56.2 Africa
## 3 Chad                       957  43.3 Africa
## 4 Chile                    14700  47.5 South/Latin America
## 5 China                     6500  39.4 Asia & Pacific
## 6 Colombia                  7580  51.7 South/Latin America
```

The function `distinct()` filters out duplicated rows.

```
distinct(tib1)
```

```
## # A tibble: 195 x 4
##    country               gdp  gini region
##    <chr>               <dbl> <dbl> <chr>
##  1 Afghanistan           574  36.8 Asia & Pacific
##  2 Albania              4520  29   Europe
##  3 Algeria              4780  27.6 Arab States
##  4 Andorra             42100  40   Europe
##  5 Angola               3750  42.6 Africa
##  6 Antigua and Barbuda 13300  40   South/Latin America
##  7 Argentina           10600  41.8 South/Latin America
##  8 Armenia              3920  31.9 Europe
##  9 Australia           55100  32.3 Asia & Pacific
## 10 Austria             47800  30.6 Europe
## # ... with 185 more rows
```

```
filter(tib1, duplicated(tib1)) # Check which observations are duplicated
```

```
## # A tibble: 2 x 4
##   country    gdp  gini region
##   <chr>    <dbl> <dbl> <chr>
## 1 Norway   90000  27.1 Europe
## 2 Suriname  8460  61   South/Latin America
```

The function `slice_sample()` randomly selects rows.

```
slice_sample(tib1, n = 4)
```

```
## # A tibble: 4 x 4
##   country         gdp  gini region
##   <chr>         <dbl> <dbl> <chr>
## 1 Guinea-Bissau   574  50.7 Africa
## 2 Dominica       6890  40   South/Latin America
```

```
## 3 Portugal      22000  35.7 Europe
## 4 Uganda          900  41.9 Africa
```

```
slice_sample(tib1, prop = 0.03)
```

```
## # A tibble: 5 x 4
##   country          gdp  gini region
##   <chr>          <dbl> <dbl> <chr>
## 1 Suriname        8460  61   South/Latin America
## 2 Ethiopia         483  37.7 Africa
## 3 United Kingdom 42000  33.4 Europe
## 4 South Sudan      731  45   Africa
## 5 Tajikistan       936  33.7 Asia & Pacific
```

Organize

The functions so far produce data frames that explicitly differ from the inputted data frame. There are some silent functions that change the underlying structure without changing the outputted data frame. The function `group_by()` is an example of these silent functions. It groups the data based on the values of a set of variables. It makes most sense to group by categorical variables. The only difference is that now it says `Groups:  region [7]`.

```
group_tib1 <- group_by(tib1, region)
group_tib1
```

```
## # A tibble: 197 x 4
## # Groups:   region [7]
##    country               gdp  gini region
##    <chr>               <dbl> <dbl> <chr>
##  1 Afghanistan           574  36.8 Asia & Pacific
##  2 Albania              4520  29   Europe
##  3 Algeria              4780  27.6 Arab States
##  4 Andorra             42100  40   Europe
##  5 Angola               3750  42.6 Africa
##  6 Antigua and Barbuda 13300  40   South/Latin America
##  7 Argentina           10600  41.8 South/Latin America
##  8 Armenia              3920  31.9 Europe
##  9 Australia           55100  32.3 Asia & Pacific
## 10 Austria             47800  30.6 Europe
## # ... with 187 more rows
```

Ungrouping the data is another silent function and it removes this underlying grouping.

```
ungroup(group_tib1)
```

```
## # A tibble: 197 x 4
##    country               gdp  gini region
##    <chr>               <dbl> <dbl> <chr>
##  1 Afghanistan           574  36.8 Asia & Pacific
##  2 Albania              4520  29   Europe
##  3 Algeria              4780  27.6 Arab States
##  4 Andorra             42100  40   Europe
##  5 Angola               3750  42.6 Africa
##  6 Antigua and Barbuda 13300  40   South/Latin America
##  7 Argentina           10600  41.8 South/Latin America
##  8 Armenia              3920  31.9 Europe
##  9 Australia           55100  32.3 Asia & Pacific
```

```
## 10 Austria             47800  30.6 Europe
## # ... with 187 more rows
```

The function `arrange` sorts the data based on the rank order of a set of variables. Adding `desc()` changes the rank-order to descending.

```
arrange(tib1, gini)
```

```
## # A tibble: 197 x 4
##    country           gdp  gini region
##    <chr>           <dbl> <dbl> <chr>
##  1 Ukraine          2830  24.8 Europe
##  2 Slovenia        23800  25.6 Europe
##  3 Czech Republic  21400  26   Europe
##  4 Slovak Republic 18900  26.7 Europe
##  5 Belarus          6380  26.9 Europe
##  6 Kazakhstan      10600  26.9 Asia & Pacific
##  7 Moldova          2950  27   Europe
##  8 Finland         45600  27.1 Europe
##  9 Norway          90000  27.1 Europe
## 10 Norway          90000  27.1 Europe
## # ... with 187 more rows
```

```
arrange(tib1, desc(region), gini)
```

```
## # A tibble: 197 x 4
##    country                          gdp  gini region
##    <chr>                          <dbl> <dbl> <chr>
##  1 Antigua and Barbuda            13300  40   South/Latin America
##  2 Dominica                        6890  40   South/Latin America
##  3 Grenada                         8190  40   South/Latin America
##  4 St. Kitts and Nevis            16700  40   South/Latin America
##  5 St. Vincent and the Grenadines  6580  40   South/Latin America
##  6 Uruguay                        13900  40.1 South/Latin America
##  7 El Salvador                     3310  41.1 South/Latin America
##  8 Trinidad and Tobago            16800  41.3 South/Latin America
##  9 Argentina                      10600  41.8 South/Latin America
## 10 St. Lucia                       8490  42.6 South/Latin America
## # ... with 187 more rows
```

Merge

Merging data frames is useful when there are several data frames with similar observations but different variables. To demonstrate the join functions in `dplyr`, we have two datasets. One is the population of all countries and the other is the population of all countries that begin with "A." Neither of these datasets have duplicates.

```
pop <- read_csv("population.csv")
```

```
## Rows: 195 Columns: 2
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): country
## dbl (1): population
##
## 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.
```

```r
popA <- read_csv("population_A.csv")
```

```
## Rows: 11 Columns: 2
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): country
## dbl (1): population
##
## 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.
```

The different join functions relate to which observations are kept. In `full_join()`, all observations in the two data frames are kept, even if there are unmatched observations. The argument `by` indicates which variable on which to match.

```r
full_join(tib1, pop, by = "country")
```

```
## # A tibble: 197 x 5
##    country               gdp  gini region              population
##    <chr>               <dbl> <dbl> <chr>                    <dbl>
##  1 Afghanistan           574  36.8 Asia & Pacific        34400000
##  2 Albania              4520  29   Europe                 2890000
##  3 Algeria              4780  27.6 Arab States           39700000
##  4 Andorra             42100  40   Europe                   78000
##  5 Angola               3750  42.6 Africa                27900000
##  6 Antigua and Barbuda 13300  40   South/Latin America      93600
##  7 Argentina           10600  41.8 South/Latin America   43100000
##  8 Armenia              3920  31.9 Europe                 2930000
##  9 Australia           55100  32.3 Asia & Pacific        23900000
## 10 Austria             47800  30.6 Europe                 8680000
## # ... with 187 more rows
```

The function `inner_join()` only keeps observations that are present in both data frames. In this case, that is only countries that begin with "A."

```r
inner_join(tib1, popA, by = "country")
```

```
## # A tibble: 11 x 5
##    country               gdp  gini region              population
##    <chr>               <dbl> <dbl> <chr>                    <dbl>
##  1 Afghanistan           574  36.8 Asia & Pacific        34400000
##  2 Albania              4520  29   Europe                 2890000
##  3 Algeria              4780  27.6 Arab States           39700000
##  4 Andorra             42100  40   Europe                   78000
##  5 Angola               3750  42.6 Africa                27900000
##  6 Antigua and Barbuda 13300  40   South/Latin America      93600
##  7 Argentina           10600  41.8 South/Latin America   43100000
##  8 Armenia              3920  31.9 Europe                 2930000
##  9 Australia           55100  32.3 Asia & Pacific        23900000
## 10 Austria             47800  30.6 Europe                 8680000
## 11 Azerbaijan           6060  32.4 Asia & Pacific         9620000
```

The function `left_join()` only keeps from the data frame in the left argument (`tib1` in this case).

```r
left_join(tib1, popA, by = "country")
```

```
## # A tibble: 197 x 5
```

```
##    country                gdp  gini region                population
##    <chr>                <dbl> <dbl> <chr>                    <dbl>
##  1 Afghanistan            574  36.8 Asia & Pacific        34400000
##  2 Albania               4520  29   Europe                 2890000
##  3 Algeria               4780  27.6 Arab States           39700000
##  4 Andorra              42100  40   Europe                   78000
##  5 Angola                3750  42.6 Africa                27900000
##  6 Antigua and Barbuda 13300  40   South/Latin America      93600
##  7 Argentina            10600  41.8 South/Latin America   43100000
##  8 Armenia               3920  31.9 Europe                 2930000
##  9 Australia            55100  32.3 Asia & Pacific        23900000
## 10 Austria              47800  30.6 Europe                 8680000
## # ... with 187 more rows
```

The function `right_join()` is the same except it only keeps the observations from the data frame in the right argument.

```
right_join(tib1, popA, by = "country")
```

```
## # A tibble: 11 x 5
##    country                gdp  gini region                population
##    <chr>                <dbl> <dbl> <chr>                    <dbl>
##  1 Afghanistan            574  36.8 Asia & Pacific        34400000
##  2 Albania               4520  29   Europe                 2890000
##  3 Algeria               4780  27.6 Arab States           39700000
##  4 Andorra              42100  40   Europe                   78000
##  5 Angola                3750  42.6 Africa                27900000
##  6 Antigua and Barbuda 13300  40   South/Latin America      93600
##  7 Argentina            10600  41.8 South/Latin America   43100000
##  8 Armenia               3920  31.9 Europe                 2930000
##  9 Australia            55100  32.3 Asia & Pacific        23900000
## 10 Austria              47800  30.6 Europe                 8680000
## 11 Azerbaijan            6060  32.4 Asia & Pacific         9620000
```

The function `semi_join()` keeps all rows in `tib1` that have a match in `popA`.

```
semi_join(tib1, popA, by = "country")
```

```
## # A tibble: 11 x 4
##    country                gdp  gini region
##    <chr>                <dbl> <dbl> <chr>
##  1 Afghanistan            574  36.8 Asia & Pacific
##  2 Albania               4520  29   Europe
##  3 Algeria               4780  27.6 Arab States
##  4 Andorra              42100  40   Europe
##  5 Angola                3750  42.6 Africa
##  6 Antigua and Barbuda 13300  40   South/Latin America
##  7 Argentina            10600  41.8 South/Latin America
##  8 Armenia               3920  31.9 Europe
##  9 Australia            55100  32.3 Asia & Pacific
## 10 Austria              47800  30.6 Europe
## 11 Azerbaijan            6060  32.4 Asia & Pacific
```

The function `anti_join()` keeps all rows in `tib1` that do not have a match in `popA`.

```
anti_join(tib1, popA, by = "country")
```

```
## # A tibble: 186 x 4
```

```
##    country      gdp  gini region
##    <chr>      <dbl> <dbl> <chr>
##  1 Bahamas    27500  43.7 South/Latin America
##  2 Bahrain    22400  40   Arab States
##  3 Bangladesh  1000  32.3 Asia & Pacific
##  4 Barbados   15800  43.8 South/Latin America
##  5 Belarus     6380  26.9 Europe
##  6 Belgium    45500  27.8 Europe
##  7 Belize      4300  53.3 South/Latin America
##  8 Benin       1130  46.9 Africa
##  9 Bhutan      2780  38   Asia & Pacific
## 10 Bolivia     2360  46.3 South/Latin America
## # ... with 176 more rows
```

Practice Exercises 5.2

1. Before running this code, what do you think the output will be? Check to see if you were right!

```
anti_join(popA, tib1, by = "country")
```

Reshape: tidyr

The `tidyr` package provides an efficient way to reshape and reformat data.

```
tib1 <- read_csv("gapminder_large.csv")
```

```
## Rows: 195 Columns: 21
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (2): country, region
## dbl (19): gdp_2015, gini_2015, co2_2015, co2_2016, co2_2017, co2_2018, cpi_2...
##
## 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.
```

```
head(tib1)
```

```
## # A tibble: 6 x 21
##   country gdp_2015 gini_2015 region co2_2015 co2_2016 co2_2017 co2_2018 cpi_2012
##   <chr>      <dbl>     <dbl> <chr>     <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
## 1 Afghan~      574      36.8 Asia ~    0.262    0.245    0.247    0.254        8
## 2 Albania     4520      29   Europe    1.6      1.57     1.61     1.59        33
## 3 Algeria     4780      27.6 Arab ~    3.8      3.64     3.56     3.69        34
## 4 Andorra    42100      40   Europe    5.97     6.07     6.27     6.12        NA
## 5 Angola      3750      42.6 Africa    1.22     1.18     1.14     1.12        22
## 6 Antigu~    13300      40   South~    5.84     5.9      5.89     5.88        NA
## # ... with 12 more variables: cpi_2013 <dbl>, cpi_2014 <dbl>, cpi_2015 <dbl>,
## #   cpi_2016 <dbl>, cpi_2017 <dbl>, lifeexp_2012 <dbl>, lifeexp_2013 <dbl>,
## #   lifeexp_2014 <dbl>, lifeexp_2015 <dbl>, lifeexp_2016 <dbl>,
## #   lifeexp_2017 <dbl>, lifeexp_2018 <dbl>
```

Wide data have one row per unit while long data have more than one row per unit. To convert wide data to long, use `pivot_longer()`. There are several different ways to get the same output.

```
# Select columns using tidy-select
# Dictate pattern with names_sep
long_tib1 <- pivot_longer(tib1,
```

```
                        contains("_"),
                        names_to = c("var", "year"),
                        names_sep = "_")

# Select columns using column indices
pivot_longer(tib1,
             5:21,
             names_to = c("var", "year"),
             names_sep = "_")
```

```
## # A tibble: 3,315 x 7
##    country     gdp_2015 gini_2015 region         var   year   value
##    <chr>          <dbl>     <dbl> <chr>          <chr> <chr>  <dbl>
##  1 Afghanistan      574      36.8 Asia & Pacific co2   2015   0.262
##  2 Afghanistan      574      36.8 Asia & Pacific co2   2016   0.245
##  3 Afghanistan      574      36.8 Asia & Pacific co2   2017   0.247
##  4 Afghanistan      574      36.8 Asia & Pacific co2   2018   0.254
##  5 Afghanistan      574      36.8 Asia & Pacific cpi   2012   8
##  6 Afghanistan      574      36.8 Asia & Pacific cpi   2013   8
##  7 Afghanistan      574      36.8 Asia & Pacific cpi   2014  12
##  8 Afghanistan      574      36.8 Asia & Pacific cpi   2015  11
##  9 Afghanistan      574      36.8 Asia & Pacific cpi   2016  15
## 10 Afghanistan      574      36.8 Asia & Pacific cpi   2017  15
## # ... with 3,305 more rows
```

```
# Dictate pattern with names_pattern
pivot_longer(tib1,
             5:21,
             names_to = c("var", "year"),
             names_pattern = "(.*)_(.*)")
```

```
## # A tibble: 3,315 x 7
##    country     gdp_2015 gini_2015 region         var   year   value
##    <chr>          <dbl>     <dbl> <chr>          <chr> <chr>  <dbl>
##  1 Afghanistan      574      36.8 Asia & Pacific co2   2015   0.262
##  2 Afghanistan      574      36.8 Asia & Pacific co2   2016   0.245
##  3 Afghanistan      574      36.8 Asia & Pacific co2   2017   0.247
##  4 Afghanistan      574      36.8 Asia & Pacific co2   2018   0.254
##  5 Afghanistan      574      36.8 Asia & Pacific cpi   2012   8
##  6 Afghanistan      574      36.8 Asia & Pacific cpi   2013   8
##  7 Afghanistan      574      36.8 Asia & Pacific cpi   2014  12
##  8 Afghanistan      574      36.8 Asia & Pacific cpi   2015  11
##  9 Afghanistan      574      36.8 Asia & Pacific cpi   2016  15
## 10 Afghanistan      574      36.8 Asia & Pacific cpi   2017  15
## # ... with 3,305 more rows
```

To go from long data to wide, use `pivot_wider`. There are several options to dictate the names of the newly created variables.

```
wide_tib1 <- pivot_wider(long_tib1,
                         names_from = c("var", "year"),
                         values_from = "value")
head(wide_tib1)
```

```
## # A tibble: 6 x 21
##   country region gdp_2015 gini_2015 co2_2015 co2_2016 co2_2017 co2_2018 cpi_2012
```

```
##   <chr>   <chr>     <dbl>     <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
## 1 Afghan~ Asia ~      574      36.8     0.262    0.245    0.247    0.254        8
## 2 Albania Europe     4520      29       1.6      1.57     1.61     1.59        33
## 3 Algeria Arab ~     4780      27.6     3.8      3.64     3.56     3.69        34
## 4 Andorra Europe    42100      40       5.97     6.07     6.27     6.12        NA
## 5 Angola  Africa     3750      42.6     1.22     1.18     1.14     1.12        22
## 6 Antigu~ South~    13300      40       5.84     5.9      5.89     5.88        NA
## # ... with 12 more variables: cpi_2013 <dbl>, cpi_2014 <dbl>, cpi_2015 <dbl>,
## #   cpi_2016 <dbl>, cpi_2017 <dbl>, lifeexp_2012 <dbl>, lifeexp_2013 <dbl>,
## #   lifeexp_2014 <dbl>, lifeexp_2015 <dbl>, lifeexp_2016 <dbl>,
## #   lifeexp_2017 <dbl>, lifeexp_2018 <dbl>
```

```
pivot_wider(long_tib1,
            names_from = c("var", "year"),
            values_from = "value",
            names_sep = ".")
```

```
## # A tibble: 195 x 21
##    country     region      gdp.2015 gini.2015 co2.2015 co2.2016 co2.2017 co2.2018
##    <chr>       <chr>          <dbl>     <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
##  1 Afghanistan Asia & Pa~       574      36.8    0.262    0.245    0.247    0.254
##  2 Albania     Europe          4520      29      1.6      1.57     1.61     1.59
##  3 Algeria     Arab Stat~      4780      27.6    3.8      3.64     3.56     3.69
##  4 Andorra     Europe         42100      40      5.97     6.07     6.27     6.12
##  5 Angola      Africa          3750      42.6    1.22     1.18     1.14     1.12
##  6 Antigua an~ South/Lat~     13300      40      5.84     5.9      5.89     5.88
##  7 Argentina   South/Lat~     10600      41.8    4.64     4.6      4.55     4.41
##  8 Armenia     Europe          3920      31.9    1.65     1.76     1.7      1.89
##  9 Australia   Asia & Pa~     55100      32.3   16.8     17       17       16.9
## 10 Austria     Europe         47800      30.6    7.7      7.7      7.94     7.75
## # ... with 185 more rows, and 13 more variables: cpi.2012 <dbl>,
## #   cpi.2013 <dbl>, cpi.2014 <dbl>, cpi.2015 <dbl>, cpi.2016 <dbl>,
## #   cpi.2017 <dbl>, lifeexp.2012 <dbl>, lifeexp.2013 <dbl>, lifeexp.2014 <dbl>,
## #   lifeexp.2015 <dbl>, lifeexp.2016 <dbl>, lifeexp.2017 <dbl>,
## #   lifeexp.2018 <dbl>
```

It might be useful to reference this chapter on strings and regular expressions. There are many ways to represent different patterns in character strings, and a standardized approach exists to minimize the need to type out everything explicitly.

Pipes

The `magrittr` package contains the pipe operator, `%>%`. The purpose of this operator is to make code clearer and more efficient. The idea is to minimize unnecessary saved objects. For example, if you are cleaning a dataset, it would be cumbersome to save a new data frame for each step in the cleaning process. Pipe operators, or pipes, help with this.

The idea is that the pipe forwards a value to the next function. The two lines result in the same output. The first argument of `filter()` is forwarded by the pipe operator.

```
filter(tib1, region == "North America")
```

```
## # A tibble: 2 x 21
##   country gdp_2015 gini_2015 region co2_2015 co2_2016 co2_2017 co2_2018 cpi_2012
##   <chr>      <dbl>     <dbl> <chr>     <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
## 1 Canada     50300      31.7 North~     16       15.5     15.6     15.3       84
```

```
## 2 United~    52100      41.3 North~     16.9     16.4     16.2     16.6       73
## # ... with 12 more variables: cpi_2013 <dbl>, cpi_2014 <dbl>, cpi_2015 <dbl>,
## #   cpi_2016 <dbl>, cpi_2017 <dbl>, lifeexp_2012 <dbl>, lifeexp_2013 <dbl>,
## #   lifeexp_2014 <dbl>, lifeexp_2015 <dbl>, lifeexp_2016 <dbl>,
## #   lifeexp_2017 <dbl>, lifeexp_2018 <dbl>
```

```
tib1 %>% filter(region == "North America")
```

```
## # A tibble: 2 x 21
##   country gdp_2015 gini_2015 region co2_2015 co2_2016 co2_2017 co2_2018 cpi_2012
##   <chr>      <dbl>     <dbl> <chr>     <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
## 1 Canada     50300      31.7 North~     16       15.5     15.6     15.3       84
## 2 United~    52100      41.3 North~     16.9     16.4     16.2     16.6       73
## # ... with 12 more variables: cpi_2013 <dbl>, cpi_2014 <dbl>, cpi_2015 <dbl>,
## #   cpi_2016 <dbl>, cpi_2017 <dbl>, lifeexp_2012 <dbl>, lifeexp_2013 <dbl>,
## #   lifeexp_2014 <dbl>, lifeexp_2015 <dbl>, lifeexp_2016 <dbl>,
## #   lifeexp_2017 <dbl>, lifeexp_2018 <dbl>
```

Pipe operators are especially useful when there are several operations being applied to the same object.

```
tib1 %>%
  distinct() %>%
  full_join(pop, by = "country") %>%
  arrange(desc(region), desc(population)) %>%
  head()
```

```
## # A tibble: 6 x 22
##   country gdp_2015 gini_2015 region co2_2015 co2_2016 co2_2017 co2_2018 cpi_2012
##   <chr>      <dbl>     <dbl> <chr>     <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
## 1 Brazil     11400      51.6 South~     2.42     2.2      2.23     2.18       43
## 2 Mexico     10000      46.5 South~     3.96     3.94     3.95     3.79       34
## 3 Colomb~     7580      51.7 South~     1.96     2.01     1.91     1.96       36
## 4 Argent~    10600      41.8 South~     4.64     4.6      4.55     4.41       35
## 5 Peru        6110      43.7 South~     1.69     1.82     1.68     1.74       38
## 6 Venezu~       NA      46.9 South~     5.69     5.47     5.23     4.81       19
## # ... with 13 more variables: cpi_2013 <dbl>, cpi_2014 <dbl>, cpi_2015 <dbl>,
## #   cpi_2016 <dbl>, cpi_2017 <dbl>, lifeexp_2012 <dbl>, lifeexp_2013 <dbl>,
## #   lifeexp_2014 <dbl>, lifeexp_2015 <dbl>, lifeexp_2016 <dbl>,
## #   lifeexp_2017 <dbl>, lifeexp_2018 <dbl>, population <dbl>
```

To highlight the utility of pipe operators, consider these alternatives. They produce the same results. The first approach results in two objects that are not necessary for the final analysis, `tmp1` and `tmp2`. These objects are created with the sole purpose of being used in other functions. If the dataset is large, saving different versions of it can be burdensome. Additionally, the workspace becomes messy with so many temporary objects. While the second approach avoids temporary versions, it is difficult to read and understand.

```
tmp1 <- distinct(tib1)
tmp2 <- full_join(tmp1, pop, by = "country")
df <- arrange(tmp2, desc(region), desc(population))
head(df)
```

```
## # A tibble: 6 x 22
##   country gdp_2015 gini_2015 region co2_2015 co2_2016 co2_2017 co2_2018 cpi_2012
##   <chr>      <dbl>     <dbl> <chr>     <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
## 1 Brazil     11400      51.6 South~     2.42     2.2      2.23     2.18       43
## 2 Mexico     10000      46.5 South~     3.96     3.94     3.95     3.79       34
## 3 Colomb~     7580      51.7 South~     1.96     2.01     1.91     1.96       36
```

```
## 4 Argent~    10600      41.8 South~     4.64     4.6      4.55     4.41        35
## 5 Peru        6110      43.7 South~     1.69     1.82     1.68     1.74        38
## 6 Venezu~       NA      46.9 South~     5.69     5.47     5.23     4.81        19
## # ... with 13 more variables: cpi_2013 <dbl>, cpi_2014 <dbl>, cpi_2015 <dbl>,
## #   cpi_2016 <dbl>, cpi_2017 <dbl>, lifeexp_2012 <dbl>, lifeexp_2013 <dbl>,
## #   lifeexp_2014 <dbl>, lifeexp_2015 <dbl>, lifeexp_2016 <dbl>,
## #   lifeexp_2017 <dbl>, lifeexp_2018 <dbl>, population <dbl>
```

```
head(arrange(distinct(full_join(tib1, pop, by = "country")), desc(region), desc(population)))
```

```
## # A tibble: 6 x 22
##   country gdp_2015 gini_2015 region co2_2015 co2_2016 co2_2017 co2_2018 cpi_2012
##   <chr>      <dbl>     <dbl> <chr>     <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
## 1 Brazil     11400      51.6 South~     2.42     2.2      2.23     2.18        43
## 2 Mexico     10000      46.5 South~     3.96     3.94     3.95     3.79        34
## 3 Colomb~     7580      51.7 South~     1.96     2.01     1.91     1.96        36
## 4 Argent~    10600      41.8 South~     4.64     4.6      4.55     4.41        35
## 5 Peru        6110      43.7 South~     1.69     1.82     1.68     1.74        38
## 6 Venezu~       NA      46.9 South~     5.69     5.47     5.23     4.81        19
## # ... with 13 more variables: cpi_2013 <dbl>, cpi_2014 <dbl>, cpi_2015 <dbl>,
## #   cpi_2016 <dbl>, cpi_2017 <dbl>, lifeexp_2012 <dbl>, lifeexp_2013 <dbl>,
## #   lifeexp_2014 <dbl>, lifeexp_2015 <dbl>, lifeexp_2016 <dbl>,
## #   lifeexp_2017 <dbl>, lifeexp_2018 <dbl>, population <dbl>
```

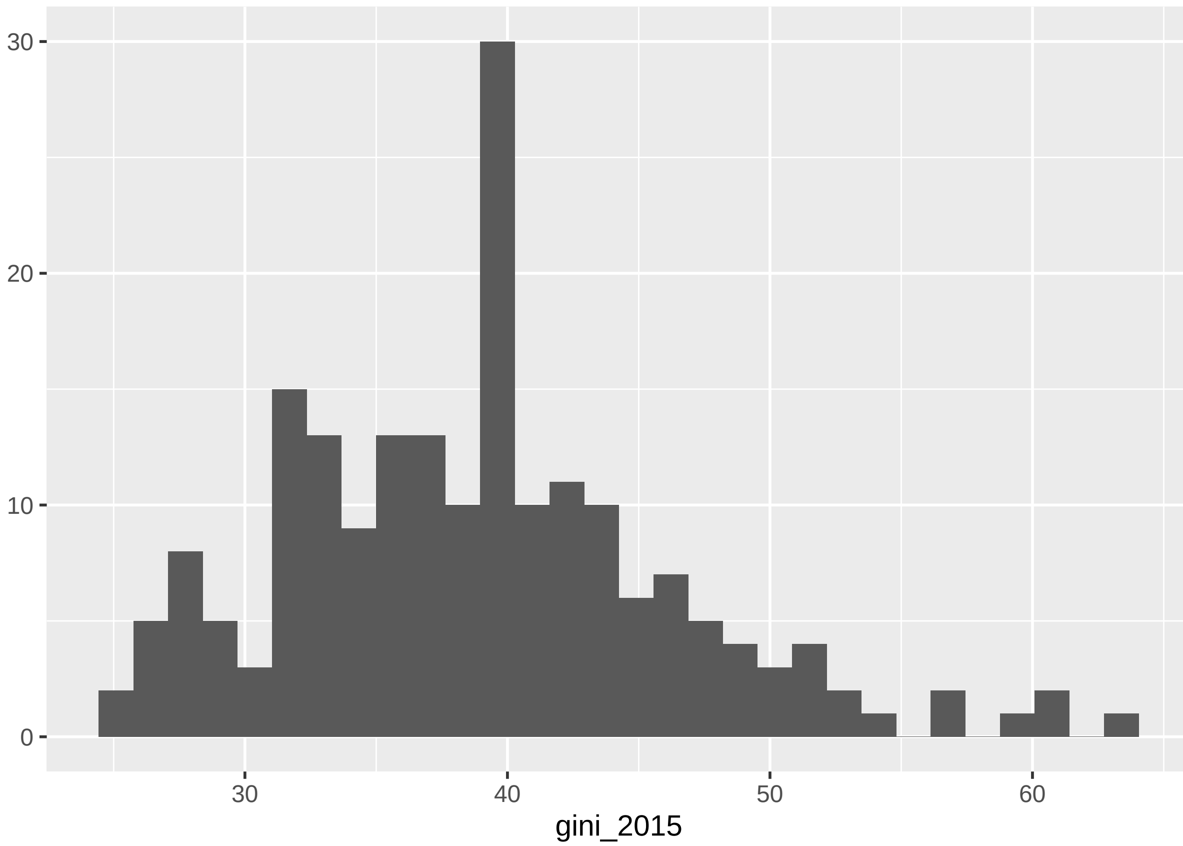
Note that the pipe operator can be used with functions outside of the `tidyverse` functions.

```
full_join(df, pop, by = "country") %>%
  write.csv("country_info.csv")
```

Pipe operators can forward objects to other arguments besides the first one. A period (`.`) indicates this. Here is an example with plotting (see chapter 5).

```
tib1 %>%
  distinct() %>%
  full_join(pop, by = "country") %>%
  qplot(x = gini_2015, data = .)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Getting comfortable with `%>%` can vastly smooth your workflow in R.

Practice Exercises 5.3

1. Use pipes to accomplish the following tasks on `tib1`: select `country`, `region`, `co2_2015`, and `co2_2016`, remove rows with missing values for either CO2 variables, create a variable that is `TRUE` when CO2 emissions in 2016 are smaller than those in 2015, and only keep the rows where this variable is `TRUE`.
2. Look at the documentation for `?magrittr`. There are four types of pipes. Take a moment to familiarize yourself with their differences.

Further Reading

There are many great resources online, including cheat sheets. Here is one for `dplyr`. Save this cheat sheet if you find it useful! More cheat sheets can be found here.

The above information comes from chapters 5.1-5.3, 6, and 21 of Boehmke (2016), chapters 2.2.5 and 3 of Zamora Saiz et al. (2020). See Zamora Saiz et al. (2020) chapter 3 for information on `data.table`.

References

Boehmke, Bradley C. 2016. *Data Wrangling with R*. Use R! Springer. https://link-springer-com.proxy.lib.duke.edu/content/pdf/10.1007%2F978-3-319-45599-0.pdf.

Zamora Saiz, Alfonso, Carlos Quesada González, Lluís Hurtado Gil, and Diego Mondéjar Ruiz. 2020. *An Introduction to Data Analysis in R: Hands-on Coding, Data Mining, Visualization and Statistics from Scratch*.