chapter 4(4.1 - 4.6)

Chapter 4 The tidyverse

Up to now we have been manipulating vectors by reordering and subsetting them through indexing. However, once we start more advanced analyses, the preferred unit for data storage is not the vector but the data frame. In this chapter we learn to work directly with data frames, which greatly facilitate the organization of information. We will be using data frames for the majority of this book. We will focus on a specific data format referred to as *tidy* and on specific collection of packages that are particularly helpful for working with *tidy data* referred to as the *tidyverse*.

We can load all the tidyverse packages at once by installing and loading the **tidyverse** package:

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.4     v dplyr   1.0.7
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   2.0.1     v forcats 0.5.1

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

We will learn how to implement the tidyverse approach throughout the book, but before delving into the details, in this chapter we introduce some of the most widely used tidyverse functionality, starting with the **dplyr** package for manipulating data frames and the **purrr** package for working with functions. Note that the tidyverse also includes a graphing package, **ggplot2**, which we introduce later in Chapter 7 in the Data Visualization part of the book; the **readr** package discussed in Chapter 5; and many others. In this chapter, we first introduce the concept of *tidy data* and then demonstrate how we use the tidyverse to work with data frames in this format.

4.1 Tidy data

We say that a data table is in *tidy* format if each row represents one observation and columns represent the different variables available for each of these observations. The `murders` dataset is an example of a tidy data frame.

```
#>        state abb region population total
#> 1    Alabama  AL  South    4779736   135
#> 2     Alaska  AK   West     710231    19
#> 3    Arizona  AZ   West    6392017   232
#> 4   Arkansas  AR  South    2915918    93
#> 5 California  CA   West   37253956  1257
#> 6   Colorado  CO   West    5029196    65
```

Each row represent a state with each of the five columns providing a different variable related to these states: name, abbreviation, region, population, and total murders.

To see how the same information can be provided in different formats, consider the following example:

```
#>       country year fertility
#> 1     Germany 1960      2.41
#> 2 South Korea 1960      6.16
#> 3     Germany 1961      2.44
#> 4 South Korea 1961      5.99
#> 5     Germany 1962      2.47
#> 6 South Korea 1962      5.79
```

This tidy dataset provides fertility rates for two countries across the years. This is a tidy dataset because each row presents one observation with the three variables being country, year, and fertility rate. However, this dataset originally came in another format and was reshaped for the **dslabs** package. Originally, the data was in the following format:

```
#>       country 1960 1961 1962
#> 1     Germany 2.41 2.44 2.47
#> 2 South Korea 6.16 5.99 5.79
```

The same information is provided, but there are two important differences in the format: 1) each row includes several observations and 2) one of the variables, year, is stored in the header. For the tidyverse packages to be optimally used, data need to be reshaped into tidy format, which you will learn to do in the Data Wrangling part of the book. Until then, we will use example datasets that are already in tidy format.

Although not immediately obvious, as you go through the book you will start to appreciate the advantages of working in a framework in which functions use tidy formats for both inputs and outputs. You will see how this permits the data analyst to focus on more important aspects of the analysis rather than the format of the data.

4.2 Exercises

1. Examine the built-in dataset `co2`. Which of the following is true:

a. `co2` is tidy data: it has one year for each row.
b. `co2` is not tidy: we need at least one column with a character vector.
c. `co2` is not tidy: it is a matrix instead of a data frame.
d. `co2` is not tidy: to be tidy we would have to wrangle it to have three columns (year, month and value), then each `co2` observation would have a row.

```
# co2
```

d

2. Examine the built-in dataset `ChickWeight`. Which of the following is true:

a. `ChickWeight` is not tidy: each chick has more than one row.
b. `ChickWeight` is tidy: each observation (a weight) is represented by one row. The chick from which this measurement came is one of the variables.
c. `ChickWeight` is not tidy: we are missing the year column.
d. `ChickWeight` is tidy: it is stored in a data frame.

```
# ChickWeight
```

b

3. Examine the built-in dataset `BOD`. Which of the following is true:

a. `BOD` is not tidy: it only has six rows.
b. `BOD` is not tidy: the first column is just an index.
c. `BOD` is tidy: each row is an observation with two values (time and demand)
d. `BOD` is tidy: all small datasets are tidy by definition.

```
# BOD
```

b

4. Which of the following built-in datasets is tidy (you can pick more than one):

a.`BJsales` b.`EuStockMarkets` c `DNase` d `Formaldehyde` e.`Orange` f `UCBAdmissions`

```
# BJsales

# EuStockMarkets

# DNase

# Formaldehyde

# Orange

# UCBAdmissions
```

c,d,e

4.3 Manipulating data frames

The **dplyr** package from the **tidyverse** introduces functions that perform some of the most common operations when working with data frames and uses names for these functions that are relatively easy to remember. For instance, to change the data table by adding a new column, we use `mutate`. To filter the data table to a subset of rows, we use `filter`. Finally, to subset the data by selecting specific columns, we use `select`.

4.3.1 Adding a column with `mutate`

We want all the necessary information for our analysis to be included in the data table. So the first task is to add the murder rates to our murders data frame. The function `mutate` takes the data frame as a first argument and the name and values of the variable as a second argument using the convention `name = values`. So, to add murder rates, we use:

```
library(dslabs)
data("murders")
murders <- mutate(murders, rate = total / population * 100000)
```

Notice that here we used `total` and `population` inside the function, which are objects that are **not** defined in our workspace. But why don't we get an error?

This is one of **dplyr**'s main features. Functions in this package, such as `mutate`, know to look for variables in the data frame provided in the first argument. In the call to mutate above, `total` will have the values in `murders$total`. This approach makes the code much more readable.

We can see that the new column is added:

```
head(murders)
```

```
##        state abb region population total     rate
## 1    Alabama  AL  South    4779736   135 2.824424
## 2     Alaska  AK   West     710231    19 2.675186
## 3    Arizona  AZ   West    6392017   232 3.629527
## 4   Arkansas  AR  South    2915918    93 3.189390
## 5 California  CA   West   37253956  1257 3.374138
## 6   Colorado  CO   West    5029196    65 1.292453
```

```
#>        state abb region population total rate
#> 1    Alabama  AL  South    4779736   135 2.82
#> 2     Alaska  AK   West     710231    19 2.68
#> 3    Arizona  AZ   West    6392017   232 3.63
#> 4   Arkansas  AR  South    2915918    93 3.19
#> 5 California  CA   West   37253956  1257 3.37
#> 6   Colorado  CO   West    5029196    65 1.29
```

Although we have overwritten the original `murders` object, this does not change the object that loaded with `data(murders)`. If we load the `murders` data again, the original will overwrite our mutated version.

4.3.2 Subsetting with `filter`

Now suppose that we want to filter the data table to only show the entries for which the murder rate is lower than 0.71. To do this we use the `filter` function, which takes the data table as the first argument and then the conditional statement as the second. Like `mutate`, we can use the unquoted variable names from `murders` inside the function and it will know we mean the columns and not objects in the workspace.

```
filter(murders, rate <= 0.71)
```

```
##           state abb        region population total      rate
## 1        Hawaii  HI          West    1360301     7 0.5145920
## 2          Iowa  IA North Central    3046355    21 0.6893484
## 3 New Hampshire  NH     Northeast    1316470     5 0.3798036
## 4  North Dakota  ND North Central     672591     4 0.5947151
## 5       Vermont  VT     Northeast     625741     2 0.3196211
```

```
#>           state abb        region population total  rate
#> 1        Hawaii  HI          West    1360301     7 0.515
#> 2          Iowa  IA North Central    3046355    21 0.689
#> 3 New Hampshire  NH     Northeast    1316470     5 0.380
#> 4  North Dakota  ND North Central     672591     4 0.595
#> 5       Vermont  VT     Northeast     625741     2 0.320
```

4.3.3 Selecting columns with `select`

Although our data table only has six columns, some data tables include hundreds. If we want to view just a few, we can use the **dplyr** `select` function. In the code below we select three columns, assign this to a new object and then filter the new object:

```
new_table <- select(murders, state, region, rate)
filter(new_table, rate <= 0.71)
```

```
##           state        region      rate
## 1        Hawaii          West 0.5145920
## 2          Iowa North Central 0.6893484
## 3 New Hampshire     Northeast 0.3798036
## 4  North Dakota North Central 0.5947151
## 5       Vermont     Northeast 0.3196211
```

```
#>           state        region  rate
#> 1        Hawaii          West 0.515
#> 2          Iowa North Central 0.689
#> 3 New Hampshire     Northeast 0.380
#> 4  North Dakota North Central 0.595
#> 5       Vermont     Northeast 0.320
```

In the call to select, the first argument murders is an object, but state, region, and rate are variable names.

4.4 Exercises

1. Load the **dplyr** package and the murders dataset.

```
library(dplyr)
library(dslabs)
data(murders)
```

You can add columns using the **dplyr** function `mutate`. This function is aware of the column names and inside the function you can call them unquoted:

```
murders <- mutate(murders, population_in_millions = population / 10^6)
```

We can write `population` rather than `murders$population`. The function `mutate` knows we are grabbing columns from `murders`.

Use the function mutate to add a murders column named rate with the per 100,000 murder rate as in the example code above. Make sure you redefine murders as done in the example code above (murders <- [your code]) so we can keep using this variable.

```
murders <- mutate(murders, rate = total / population * 100000)
murders
```

```
##                  state abb        region population total
## 1              Alabama  AL         South    4779736   135
## 2               Alaska  AK          West     710231    19
```

```
## 3               Arizona  AZ          West    6392017   232
## 4              Arkansas  AR         South    2915918    93
## 5            California  CA          West   37253956  1257
## 6              Colorado  CO          West    5029196    65
## 7           Connecticut  CT     Northeast    3574097    97
## 8              Delaware  DE         South     897934    38
## 9  District of Columbia  DC         South     601723    99
## 10              Florida  FL         South   19687653   669
## 11              Georgia  GA         South    9920000   376
## 12               Hawaii  HI          West    1360301     7
## 13                Idaho  ID          West    1567582    12
## 14             Illinois  IL North Central   12830632   364
## 15              Indiana  IN North Central    6483802   142
## 16                 Iowa  IA North Central    3046355    21
## 17               Kansas  KS North Central    2853118    63
## 18             Kentucky  KY         South    4339367   116
## 19            Louisiana  LA         South    4533372   351
## 20                Maine  ME     Northeast    1328361    11
## 21             Maryland  MD         South    5773552   293
## 22        Massachusetts  MA     Northeast    6547629   118
## 23             Michigan  MI North Central    9883640   413
## 24            Minnesota  MN North Central    5303925    53
## 25          Mississippi  MS         South    2967297   120
## 26             Missouri  MO North Central    5988927   321
## 27              Montana  MT          West     989415    12
## 28             Nebraska  NE North Central    1826341    32
## 29               Nevada  NV          West    2700551    84
## 30        New Hampshire  NH     Northeast    1316470     5
## 31           New Jersey  NJ     Northeast    8791894   246
## 32           New Mexico  NM          West    2059179    67
## 33             New York  NY     Northeast   19378102   517
## 34       North Carolina  NC         South    9535483   286
## 35         North Dakota  ND North Central     672591     4
## 36                 Ohio  OH North Central   11536504   310
## 37             Oklahoma  OK         South    3751351   111
## 38               Oregon  OR          West    3831074    36
## 39         Pennsylvania  PA     Northeast   12702379   457
## 40         Rhode Island  RI     Northeast    1052567    16
## 41       South Carolina  SC         South    4625364   207
## 42         South Dakota  SD North Central     814180     8
## 43            Tennessee  TN         South    6346105   219
## 44                Texas  TX         South   25145561   805
## 45                 Utah  UT          West    2763885    22
## 46              Vermont  VT     Northeast     625741     2
## 47             Virginia  VA         South    8001024   250
## 48           Washington  WA          West    6724540    93
## 49        West Virginia  WV         South    1852994    27
## 50            Wisconsin  WI North Central    5686986    97
## 51              Wyoming  WY          West     563626     5
##    population_in_millions       rate
## 1                4.779736  2.8244238
## 2                0.710231  2.6751860
## 3                6.392017  3.6295273
## 4                2.915918  3.1893901
```

```
## 5              37.253956  3.3741383
## 6               5.029196  1.2924531
## 7               3.574097  2.7139722
## 8               0.897934  4.2319369
## 9               0.601723 16.4527532
## 10             19.687653  3.3980688
## 11              9.920000  3.7903226
## 12              1.360301  0.5145920
## 13              1.567582  0.7655102
## 14             12.830632  2.8369608
## 15              6.483802  2.1900730
## 16              3.046355  0.6893484
## 17              2.853118  2.2081106
## 18              4.339367  2.6732010
## 19              4.533372  7.7425810
## 20              1.328361  0.8280881
## 21              5.773552  5.0748655
## 22              6.547629  1.8021791
## 23              9.883640  4.1786225
## 24              5.303925  0.9992600
## 25              2.967297  4.0440846
## 26              5.988927  5.3598917
## 27              0.989415  1.2128379
## 28              1.826341  1.7521372
## 29              2.700551  3.1104763
## 30              1.316470  0.3798036
## 31              8.791894  2.7980319
## 32              2.059179  3.2537239
## 33             19.378102  2.6679599
## 34              9.535483  2.9993237
## 35              0.672591  0.5947151
## 36             11.536504  2.6871225
## 37              3.751351  2.9589340
## 38              3.831074  0.9396843
## 39             12.702379  3.5977513
## 40              1.052567  1.5200933
## 41              4.625364  4.4753235
## 42              0.814180  0.9825837
## 43              6.346105  3.4509357
## 44             25.145561  3.2013603
## 45              2.763885  0.7959810
## 46              0.625741  0.3196211
## 47              8.001024  3.1246001
## 48              6.724540  1.3829942
## 49              1.852994  1.4571013
## 50              5.686986  1.7056487
## 51              0.563626  0.8871131
```

2. If `rank(x)` gives you the ranks of `x` from lowest to highest, `rank(-x)` gives you the ranks from highest to lowest. Use the function `mutate` to add a column `rank` containing the rank, from highest to lowest murder rate. Make sure you redefine murders so we can keep using this variable.

```
murders <- mutate(murders, rank = rank(murders$rate))
murders
```

```
##                   state abb        region population total
## 1               Alabama  AL         South    4779736   135
## 2                Alaska  AK          West     710231    19
## 3               Arizona  AZ          West    6392017   232
## 4              Arkansas  AR         South    2915918    93
## 5            California  CA          West   37253956  1257
## 6              Colorado  CO          West    5029196    65
## 7           Connecticut  CT     Northeast    3574097    97
## 8              Delaware  DE         South     897934    38
## 9  District of Columbia  DC         South     601723    99
## 10              Florida  FL         South   19687653   669
## 11              Georgia  GA         South    9920000   376
## 12               Hawaii  HI          West    1360301     7
## 13                Idaho  ID          West    1567582    12
## 14             Illinois  IL North Central   12830632   364
## 15              Indiana  IN North Central    6483802   142
## 16                 Iowa  IA North Central    3046355    21
## 17               Kansas  KS North Central    2853118    63
## 18             Kentucky  KY         South    4339367   116
## 19            Louisiana  LA         South    4533372   351
## 20                Maine  ME     Northeast    1328361    11
## 21             Maryland  MD         South    5773552   293
## 22        Massachusetts  MA     Northeast    6547629   118
## 23             Michigan  MI North Central    9883640   413
## 24            Minnesota  MN North Central    5303925    53
## 25          Mississippi  MS         South    2967297   120
## 26             Missouri  MO North Central    5988927   321
## 27              Montana  MT          West     989415    12
## 28             Nebraska  NE North Central    1826341    32
## 29               Nevada  NV          West    2700551    84
## 30        New Hampshire  NH     Northeast    1316470     5
## 31           New Jersey  NJ     Northeast    8791894   246
## 32           New Mexico  NM          West    2059179    67
## 33             New York  NY     Northeast   19378102   517
## 34       North Carolina  NC         South    9535483   286
## 35         North Dakota  ND North Central     672591     4
## 36                 Ohio  OH North Central   11536504   310
## 37             Oklahoma  OK         South    3751351   111
## 38               Oregon  OR          West    3831074    36
## 39         Pennsylvania  PA     Northeast   12702379   457
## 40         Rhode Island  RI     Northeast    1052567    16
## 41       South Carolina  SC         South    4625364   207
## 42         South Dakota  SD North Central     814180     8
## 43            Tennessee  TN         South    6346105   219
## 44                Texas  TX         South   25145561   805
## 45                 Utah  UT          West    2763885    22
## 46              Vermont  VT     Northeast     625741     2
## 47             Virginia  VA         South    8001024   250
## 48           Washington  WA          West    6724540    93
## 49        West Virginia  WV         South    1852994    27
```

```
## 50            Wisconsin  WI North Central    5686986    97
## 51              Wyoming  WY          West     563626     5
##    population_in_millions       rate rank
## 1                4.779736  2.8244238   29
## 2                0.710231  2.6751860   25
## 3                6.392017  3.6295273   42
## 4                2.915918  3.1893901   35
## 5               37.253956  3.3741383   38
## 6                5.029196  1.2924531   14
## 7                3.574097  2.7139722   27
## 8                0.897934  4.2319369   46
## 9                0.601723 16.4527532   51
## 10              19.687653  3.3980688   39
## 11               9.920000  3.7903226   43
## 12               1.360301  0.5145920    3
## 13               1.567582  0.7655102    6
## 14              12.830632  2.8369608   30
## 15               6.483802  2.1900730   21
## 16               3.046355  0.6893484    5
## 17               2.853118  2.2081106   22
## 18               4.339367  2.6732010   24
## 19               4.533372  7.7425810   50
## 20               1.328361  0.8280881    8
## 21               5.773552  5.0748655   48
## 22               6.547629  1.8021791   20
## 23               9.883640  4.1786225   45
## 24               5.303925  0.9992600   12
## 25               2.967297  4.0440846   44
## 26               5.988927  5.3598917   49
## 27               0.989415  1.2128379   13
## 28               1.826341  1.7521372   19
## 29               2.700551  3.1104763   33
## 30               1.316470  0.3798036    2
## 31               8.791894  2.7980319   28
## 32               2.059179  3.2537239   37
## 33              19.378102  2.6679599   23
## 34               9.535483  2.9993237   32
## 35               0.672591  0.5947151    4
## 36              11.536504  2.6871225   26
## 37               3.751351  2.9589340   31
## 38               3.831074  0.9396843   10
## 39              12.702379  3.5977513   41
## 40               1.052567  1.5200933   17
## 41               4.625364  4.4753235   47
## 42               0.814180  0.9825837   11
## 43               6.346105  3.4509357   40
## 44              25.145561  3.2013603   36
## 45               2.763885  0.7959810    7
## 46               0.625741  0.3196211    1
## 47               8.001024  3.1246001   34
## 48               6.724540  1.3829942   15
## 49               1.852994  1.4571013   16
## 50               5.686986  1.7056487   18
## 51               0.563626  0.8871131    9
```

3. With dplyr, we can use `select` to show only certain columns. For example, with this code we would only show the states and population sizes:

```
select(murders, state, population) %>% head()
```

```
##        state population
## 1    Alabama    4779736
## 2     Alaska     710231
## 3    Arizona    6392017
## 4   Arkansas    2915918
## 5 California   37253956
## 6   Colorado    5029196
```

Use `select` to show the state names and abbreviations in `murders`. Do not redefine `murders`, just show the results.

```
select(murders, state, abb) %>% head()
```

```
##        state abb
## 1    Alabama  AL
## 2     Alaska  AK
## 3    Arizona  AZ
## 4   Arkansas  AR
## 5 California  CA
## 6   Colorado  CO
```

4. The **dplyr** function `filter` is used to choose specific rows of the data frame to keep. Unlike `select` which is for columns, `filter` is for rows. For example, you can show just the New York row like this:

```
filter(murders, state == "New York")
```

```
##      state abb    region population total population_in_millions    rate rank
## 1 New York  NY Northeast   19378102   517                19.3781 2.66796   23
```

You can use other logical vectors to filter rows.

Use `filter` to show the top 5 states with the highest murder rates. After we add murder rate and rank, do not change the murders dataset, just show the result. Remember that you can filter based on the `rank` column.

```
filter(murders, rank < 6)
```

```
##           state abb        region population total population_in_millions
## 1        Hawaii  HI          West    1360301     7                1.360301
## 2          Iowa  IA North Central    3046355    21                3.046355
## 3 New Hampshire  NH     Northeast    1316470     5                1.316470
## 4  North Dakota  ND North Central     672591     4                0.672591
## 5       Vermont  VT     Northeast     625741     2                0.625741
##        rate rank
## 1 0.5145920    3
## 2 0.6893484    5
## 3 0.3798036    2
## 4 0.5947151    4
## 5 0.3196211    1
```

5. We can remove rows using the `!=` operator. For example, to remove Florida, we would do this:

```
no_florida <- filter(murders, state != "Florida")
```

Create a new data frame called no_south that removes states from the South region. How many states are in this category? You can use the function `nrow` for this.

```
no_south <- filter(murders, region != 'South')
no_south
```

```
##            state abb        region population total population_in_millions
## 1         Alaska  AK          West     710231    19                0.710231
## 2        Arizona  AZ          West    6392017   232                6.392017
## 3     California  CA          West   37253956  1257               37.253956
## 4       Colorado  CO          West    5029196    65                5.029196
## 5    Connecticut  CT     Northeast    3574097    97                3.574097
## 6         Hawaii  HI          West    1360301     7                1.360301
## 7          Idaho  ID          West    1567582    12                1.567582
## 8       Illinois  IL North Central   12830632   364               12.830632
## 9        Indiana  IN North Central    6483802   142                6.483802
## 10          Iowa  IA North Central    3046355    21                3.046355
## 11        Kansas  KS North Central    2853118    63                2.853118
## 12         Maine  ME     Northeast    1328361    11                1.328361
## 13 Massachusetts  MA     Northeast    6547629   118                6.547629
## 14      Michigan  MI North Central    9883640   413                9.883640
## 15     Minnesota  MN North Central    5303925    53                5.303925
## 16      Missouri  MO North Central    5988927   321                5.988927
## 17       Montana  MT          West     989415    12                0.989415
## 18      Nebraska  NE North Central    1826341    32                1.826341
## 19        Nevada  NV          West    2700551    84                2.700551
## 20 New Hampshire  NH     Northeast    1316470     5                1.316470
## 21    New Jersey  NJ     Northeast    8791894   246                8.791894
## 22    New Mexico  NM          West    2059179    67                2.059179
## 23      New York  NY     Northeast   19378102   517               19.378102
## 24  North Dakota  ND North Central     672591     4                0.672591
## 25          Ohio  OH North Central   11536504   310               11.536504
## 26        Oregon  OR          West    3831074    36                3.831074
## 27  Pennsylvania  PA     Northeast   12702379   457               12.702379
## 28  Rhode Island  RI     Northeast    1052567    16                1.052567
## 29  South Dakota  SD North Central     814180     8                0.814180
## 30          Utah  UT          West    2763885    22                2.763885
## 31       Vermont  VT     Northeast     625741     2                0.625741
## 32    Washington  WA          West    6724540    93                6.724540
## 33     Wisconsin  WI North Central    5686986    97                5.686986
## 34       Wyoming  WY          West     563626     5                0.563626
##         rate rank
## 1  2.6751860   25
## 2  3.6295273   42
## 3  3.3741383   38
## 4  1.2924531   14
## 5  2.7139722   27
## 6  0.5145920    3
## 7  0.7655102    6
```

```
## 8  2.8369608   30
## 9  2.1900730   21
## 10 0.6893484    5
## 11 2.2081106   22
## 12 0.8280881    8
## 13 1.8021791   20
## 14 4.1786225   45
## 15 0.9992600   12
## 16 5.3598917   49
## 17 1.2128379   13
## 18 1.7521372   19
## 19 3.1104763   33
## 20 0.3798036    2
## 21 2.7980319   28
## 22 3.2537239   37
## 23 2.6679599   23
## 24 0.5947151    4
## 25 2.6871225   26
## 26 0.9396843   10
## 27 3.5977513   41
## 28 1.5200933   17
## 29 0.9825837   11
## 30 0.7959810    7
## 31 0.3196211    1
## 32 1.3829942   15
## 33 1.7056487   18
## 34 0.8871131    9
```

```
nrow(no_south)
```

```
## [1] 34
```

6. We can also use `%in%` to filter with **dplyr**. You can therefore see the data from New York and Texas like this:

```
filter(murders, state %in% c("New York", "Texas"))
```

```
##      state abb    region population total population_in_millions    rate rank
## 1 New York  NY Northeast   19378102   517               19.37810 2.66796   23
## 2    Texas  TX     South   25145561   805               25.14556 3.20136   36
```

Create a new data frame called `murders_nw` with only the states from the Northeast and the West. How many states are in this category?

```
murders_nw <- filter(murders, region %in% c("Northest", "West"))
murders_nw
```

```
##         state abb region population total population_in_millions      rate rank
## 1      Alaska  AK   West     710231    19               0.710231 2.6751860   25
## 2     Arizona  AZ   West    6392017   232               6.392017 3.6295273   42
## 3  California  CA   West   37253956  1257              37.253956 3.3741383   38
```

```
## 4    Colorado  CO   West    5029196    65              5.029196 1.2924531   14
## 5      Hawaii  HI   West    1360301     7              1.360301 0.5145920    3
## 6       Idaho  ID   West    1567582    12              1.567582 0.7655102    6
## 7     Montana  MT   West     989415    12              0.989415 1.2128379   13
## 8      Nevada  NV   West    2700551    84              2.700551 3.1104763   33
## 9  New Mexico  NM   West    2059179    67              2.059179 3.2537239   37
## 10     Oregon  OR   West    3831074    36              3.831074 0.9396843   10
## 11       Utah  UT   West    2763885    22              2.763885 0.7959810    7
## 12 Washington  WA   West    6724540    93              6.724540 1.3829942   15
## 13    Wyoming  WY   West     563626     5              0.563626 0.8871131    9
```

7. Suppose you want to live in the Northeast or West **and** want the murder rate to be less than 1. We want to see the data for the states satisfying these options. Note that you can use logical operators with `filter`. Here is an example in which we filter to keep only small states in the Northeast region.

```
filter(murders, population < 5000000 & region == "Northeast")
```

```
##           state abb    region population total population_in_millions      rate
## 1   Connecticut  CT Northeast    3574097    97               3.574097 2.7139722
## 2         Maine  ME Northeast    1328361    11               1.328361 0.8280881
## 3 New Hampshire  NH Northeast    1316470     5               1.316470 0.3798036
## 4  Rhode Island  RI Northeast    1052567    16               1.052567 1.5200933
## 5       Vermont  VT Northeast     625741     2               0.625741 0.3196211
##   rank
## 1   27
## 2    8
## 3    2
## 4   17
## 5    1
```

Make sure `murders` has been defined with `rate` and `rank` and still has all states. Create a table called `my_states` that contains rows for states satisfying both the conditions: it is in the Northeast or West and the murder rate is less than 1. Use select to show only the state name, the rate, and the rank.

```
my_states <- filter(murders, region %in% c("Northeast", "West") & rate <1)
my_states
```

```
##           state abb    region population total population_in_millions      rate
## 1        Hawaii  HI      West    1360301     7               1.360301 0.5145920
## 2         Idaho  ID      West    1567582    12               1.567582 0.7655102
## 3         Maine  ME Northeast    1328361    11               1.328361 0.8280881
## 4 New Hampshire  NH Northeast    1316470     5               1.316470 0.3798036
## 5        Oregon  OR      West    3831074    36               3.831074 0.9396843
## 6          Utah  UT      West    2763885    22               2.763885 0.7959810
## 7       Vermont  VT Northeast     625741     2               0.625741 0.3196211
## 8       Wyoming  WY      West     563626     5               0.563626 0.8871131
##   rank
## 1    3
## 2    6
## 3    8
## 4    2
## 5   10
```

```
## 6    7
## 7    1
## 8    9
```

```
select(my_states, state, rate, rank)
```

```
##           state      rate rank
## 1        Hawaii 0.5145920    3
## 2         Idaho 0.7655102    6
## 3         Maine 0.8280881    8
## 4 New Hampshire 0.3798036    2
## 5        Oregon 0.9396843   10
## 6          Utah 0.7959810    7
## 7       Vermont 0.3196211    1
## 8       Wyoming 0.8871131    9
```

4.5 The pipe: %>%

With **dplyr** we can perform a series of operations, for example `select` and then `filter`, by sending the results of one function to another using what is called the *pipe* operator: `%>%`. Some details are included below.

We wrote code above to show three variables (state, region, rate) for states that have murder rates below 0.71. To do this, we defined the intermediate object `new_table`. In **dplyr** we can write code that looks more like a description of what we want to do without intermediate objects:

$$original\ data \text{ß} select \text{ß} filter$$

For such an operation, we can use the pipe `%>%`. The code looks like this:

```
murders %>% select(state, region, rate) %>% filter(rate <= 0.71)
```

```
##           state        region      rate
## 1        Hawaii          West 0.5145920
## 2          Iowa North Central 0.6893484
## 3 New Hampshire     Northeast 0.3798036
## 4  North Dakota North Central 0.5947151
## 5       Vermont     Northeast 0.3196211
```

```
#>           state        region  rate
#> 1        Hawaii          West 0.515
#> 2          Iowa North Central 0.689
#> 3 New Hampshire     Northeast 0.380
#> 4  North Dakota North Central 0.595
#> 5       Vermont     Northeast 0.320
```

This line of code is equivalent to the two lines of code above. What is going on here?

In general, the pipe sends the result of the left side of the pipe to be the first argument of the function on the right side of the pipe. Here is a very simple example:

```
16 %>% sqrt()
```

```
## [1] 4
```

We can continue to pipe values along:

```
16 %>% sqrt() %>% log2()
```

```
## [1] 2
```

The above statement is equivalent to `log2(sqrt(16))`.

Remember that the pipe sends values to the first argument, so we can define other arguments as if the first argument is already defined:

```
16 %>% sqrt() %>% log(base = 2)
```

```
## [1] 2
```

Therefore, when using the pipe with data frames and **dplyr**, we no longer need to specify the required first argument since the **dplyr** functions we have described all take the data as the first argument. In the code we wrote:

```
murders %>% select(state, region, rate) %>% filter(rate <= 0.71)
```

```
##           state        region      rate
## 1        Hawaii          West 0.5145920
## 2          Iowa North Central 0.6893484
## 3 New Hampshire     Northeast 0.3798036
## 4  North Dakota North Central 0.5947151
## 5       Vermont     Northeast 0.3196211
```

`murders` is the first argument of the `select` function, and the new data frame (formerly `new_table`) is the first argument of the `filter` function.

Note that the pipe works well with functions where the first argument is the input data. Functions in **tidyverse** packages like **dplyr** have this format and can be used easily with the pipe.

4.6 Exercises

1. The pipe `%>%` can be used to perform operations sequentially without having to define intermediate objects. Start by redefining murder to include rate and rank.

```
murders <- mutate(murders, rate =  total / population * 100000,
                  rank = rank(-rate))
```

In the solution to the previous exercise, we did the following:

```
my_states <- filter(murders, region %in% c("Northeast", "West") &
                      rate < 1)

select(my_states, state, rate, rank)
```

```
##           state      rate rank
## 1        Hawaii 0.5145920   49
## 2         Idaho 0.7655102   46
## 3         Maine 0.8280881   44
## 4 New Hampshire 0.3798036   50
## 5        Oregon 0.9396843   42
## 6          Utah 0.7959810   45
## 7       Vermont 0.3196211   51
## 8       Wyoming 0.8871131   43
```

The pipe %>% permits us to perform both operations sequentially without having to define an intermediate variable my_states. We therefore could have mutated and selected in the same line like this:

```
mutate(murders, rate =  total / population * 100000,
       rank = rank(-rate)) %>% select(state, rate, rank)
```

```
##                  state       rate rank
## 1              Alabama  2.8244238   23
## 2               Alaska  2.6751860   27
## 3              Arizona  3.6295273   10
## 4             Arkansas  3.1893901   17
## 5           California  3.3741383   14
## 6             Colorado  1.2924531   38
## 7          Connecticut  2.7139722   25
## 8             Delaware  4.2319369    6
## 9  District of Columbia 16.4527532    1
## 10             Florida  3.3980688   13
## 11             Georgia  3.7903226    9
## 12              Hawaii  0.5145920   49
## 13               Idaho  0.7655102   46
## 14            Illinois  2.8369608   22
## 15             Indiana  2.1900730   31
## 16                Iowa  0.6893484   47
## 17              Kansas  2.2081106   30
## 18            Kentucky  2.6732010   28
## 19           Louisiana  7.7425810    2
## 20               Maine  0.8280881   44
## 21            Maryland  5.0748655    4
## 22       Massachusetts  1.8021791   32
## 23            Michigan  4.1786225    7
## 24           Minnesota  0.9992600   40
## 25         Mississippi  4.0440846    8
## 26            Missouri  5.3598917    3
## 27             Montana  1.2128379   39
## 28            Nebraska  1.7521372   33
## 29              Nevada  3.1104763   19
## 30       New Hampshire  0.3798036   50
```

```
## 31           New Jersey  2.7980319   24
## 32           New Mexico  3.2537239   15
## 33             New York  2.6679599   29
## 34       North Carolina  2.9993237   20
## 35         North Dakota  0.5947151   48
## 36                 Ohio  2.6871225   26
## 37             Oklahoma  2.9589340   21
## 38               Oregon  0.9396843   42
## 39         Pennsylvania  3.5977513   11
## 40         Rhode Island  1.5200933   35
## 41       South Carolina  4.4753235    5
## 42         South Dakota  0.9825837   41
## 43            Tennessee  3.4509357   12
## 44                Texas  3.2013603   16
## 45                 Utah  0.7959810   45
## 46              Vermont  0.3196211   51
## 47             Virginia  3.1246001   18
## 48           Washington  1.3829942   37
## 49        West Virginia  1.4571013   36
## 50            Wisconsin  1.7056487   34
## 51              Wyoming  0.8871131   43
```

Notice that `select` no longer has a data frame as the first argument. The first argument is assumed to be the result of the operation conducted right before the %>%.

Repeat the previous exercise, but now instead of creating a new object, show the result and only include the state, rate, and rank columns. Use a pipe %>% to do this in just one line.

```
mutate(murders, rate =  total / population * 100000, rank = rank(rate)) %>%
  filter(region %in% c("Northeast", "West") & rate < 1) %>%
  select(state, rate, rank, region)
```

```
##           state      rate rank    region
## 1        Hawaii 0.5145920    3      West
## 2         Idaho 0.7655102    6      West
## 3         Maine 0.8280881    8 Northeast
## 4 New Hampshire 0.3798036    2 Northeast
## 5        Oregon 0.9396843   10      West
## 6          Utah 0.7959810    7      West
## 7       Vermont 0.3196211    1 Northeast
## 8       Wyoming 0.8871131    9      West
```

2. Reset `murders` to the original table by using `data(murders)`. Use a pipe to create a new data frame called `my_states` that considers only states in the Northeast or West which have a murder rate lower than 1, and contains only the state, rate and rank columns. The pipe should also have four components separated by three %>%. The code should look something like this:

```
#my_states <- murders %>%
 # mutate SOMETHING %>%
 # filter SOMETHING %>%
 # select SOMETHING
```

```r
data(murders)
my_state <- murders %>%
  mutate (rate =  total / population * 100000, rank = rank(-rate)) %>%
  filter (region %in% c("Northeast", "West") & rate < 1) %>%
  select (state, rate, rank)

my_state
```

```
##           state      rate rank
## 1        Hawaii 0.5145920   49
## 2         Idaho 0.7655102   46
## 3         Maine 0.8280881   44
## 4 New Hampshire 0.3798036   50
## 5        Oregon 0.9396843   42
## 6          Utah 0.7959810   45
## 7       Vermont 0.3196211   51
## 8       Wyoming 0.8871131   43
```