Data_Manipulation_R

Ravi Mummigatti

9/27/2021

Contents

Introduction

Data manipulation involves modifying data to make it easier to read and to be more organized. We manipulate data for analysis and visualization. It is also used with the term 'data exploration' which involves organizing data using available sets of variables.
At times, the data collection process done by machines involves a lot of errors and inaccuracies in reading. Data manipulation is also used to remove these inaccuracies and make data more accurate and precise.

We will apply various functions from the **"Tidyverse"** package to manipulate / transform our data to derive meaningful insights

Load tidyverse

We will use the dplyr package from the tidyverse meta package to accomplish various tasks

```
library(tidyverse)
```

Pipe (%>%) Operator

Modern filtering is done with **dplyr** package. It has a consistent and clean way of defining filters.It makes use of piping, which is why this technique is shown here.To understand why it is useful, you need to understand what the drawbacks without piping are. Assume you have a vector of numbers. Can you understand what the second line of this expression does?

```
x <- c(0.109, 0.359, 0.63, 0.996, 0.515, 0.142, 0.017, 0.829, 0.907)
round(exp(diff(log(x))), 1)
```

```
## [1]  3.3  1.8  1.6  0.5  0.3  0.1 48.8  1.1
```

You have to read this from "Inside to Outside"

Step 1 : Take the log of (x)

Step 2 : Take the difference

Step 3 : Take the Exponent

Step 4 : Round the result to "1" decimal

Each function is nested within another function. It is hard to read and understand

The same task can be performed by "piping" using the "%>%" operator. The %>% "PIPE" takes the variable on the left side of the piping operator as first parameter of the function. There is no nesting of functions any more.

```
x <- c(0.109, 0.359, 0.63, 0.996, 0.515, 0.142, 0.017, 0.829, 0.907)

x %>%  # take the vector (x)
  log() %>% # take the logarithm
  diff() %>% # take the difference
  exp() %>%  # take the exponent
  round(1) # finally round to 1 decimal
```

```
## [1]  3.3  1.8  1.6  0.5  0.3  0.1 48.8  1.1
```

Transforming Data

In this section we will learn to use four basic dplyr verbs to explore and transform a dataset.

Dataset Reference

We will reference the United States Census data for the year 2015.A state is one of 50 regions within the United States, such as New York, California, or Texas. A county is a sub-region of one of those states, e.g. Los-Angeles is a county in the state of California.This dataset includes information about people living in each county, such as the population, the unemployment rate, their income, and their racial and gender breakdown, so there are a lot of questions we can ask of our data.There are 40 variables in this data

Load Data

We will read the .rds file from the specified url and store it in a dataframe mydata

```r
# mydata <- readRDS(url("https://assets.datacamp.com/production/repositories/4984/datasets/a924bf7063f0
```

```r
# store the read data as a .csv file
# write.csv(mydata, file = "mydata.csv")
```

```r
# load the data
mydata <- read_csv("mydata.csv")
```

Explore Values ; glimpse()

If you want to see a few values from all the columns, you can use glimpse()

```r
glimpse(mydata)
```

```
## Rows: 3,138
## Columns: 41
## $ X1                 <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, ~
## $ census_id          <dbl> 1001, 1003, 1005, 1007, 1009, 1011, 1013, 1015, 101~
## $ state              <chr> "Alabama", "Alabama", "Alabama", "Alabama", "Alabam~
## $ county             <chr> "Autauga", "Baldwin", "Barbour", "Bibb", "Blount", ~
## $ region             <chr> "South", "South", "South", "South", "South", "South~
## $ metro              <chr> "Metro", "Metro", "Nonmetro", "Metro", "Metro", "No~
## $ population         <dbl> 55221, 195121, 26932, 22604, 57710, 10678, 20354, 1~
## $ men                <dbl> 26745, 95314, 14497, 12073, 28512, 5660, 9502, 5627~
## $ women              <dbl> 28476, 99807, 12435, 10531, 29198, 5018, 10852, 603~
## $ hispanic           <dbl> 2.6, 4.5, 4.6, 2.2, 8.6, 4.4, 1.2, 3.5, 0.4, 1.5, 7~
## $ white              <dbl> 75.8, 83.1, 46.2, 74.5, 87.9, 22.2, 53.3, 73.0, 57.~
## $ black              <dbl> 18.5, 9.5, 46.7, 21.4, 1.5, 70.7, 43.8, 20.3, 40.3,~
## $ native             <dbl> 0.4, 0.6, 0.2, 0.4, 0.3, 1.2, 0.1, 0.2, 0.2, 0.6, 0~
## $ asian              <dbl> 1.0, 0.7, 0.4, 0.1, 0.1, 0.2, 0.4, 0.9, 0.8, 0.3, 0~
## $ pacific            <dbl> 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0~
## $ citizens           <dbl> 40725, 147695, 20714, 17495, 42345, 8057, 15581, 88~
## $ income             <dbl> 51281, 50254, 32964, 38678, 45813, 31938, 32229, 41~
## $ income_err         <dbl> 2391, 1263, 2973, 3995, 3141, 5884, 1793, 925, 2949~
## $ income_per_cap     <dbl> 24974, 27317, 16824, 18431, 20532, 17580, 18390, 21~
## $ income_per_cap_err <dbl> 1080, 711, 798, 1618, 708, 2055, 714, 489, 1366, 15~
## $ poverty            <dbl> 12.9, 13.4, 26.7, 16.8, 16.7, 24.6, 25.4, 20.5, 21.~
## $ child_poverty      <dbl> 18.6, 19.2, 45.3, 27.9, 27.2, 38.4, 39.2, 31.6, 37.~
## $ professional       <dbl> 33.2, 33.1, 26.8, 21.5, 28.5, 18.8, 27.5, 27.3, 23.~
## $ service            <dbl> 17.0, 17.7, 16.1, 17.9, 14.1, 15.0, 16.6, 17.7, 14.~
## $ office             <dbl> 24.2, 27.1, 23.1, 17.8, 23.9, 19.7, 21.9, 24.2, 26.~
## $ construction       <dbl> 8.6, 10.8, 10.8, 19.0, 13.5, 20.1, 10.3, 10.5, 11.5~
## $ production         <dbl> 17.1, 11.2, 23.1, 23.7, 19.9, 26.4, 23.7, 20.4, 24.~
## $ drive              <dbl> 87.5, 84.7, 83.8, 83.2, 84.9, 74.9, 84.5, 85.3, 85.~
## $ carpool            <dbl> 8.8, 8.8, 10.9, 13.5, 11.2, 14.9, 12.4, 9.4, 11.9, ~
## $ transit            <dbl> 0.1, 0.1, 0.4, 0.5, 0.4, 0.7, 0.0, 0.2, 0.2, 0.2, 0~
## $ walk               <dbl> 0.5, 1.0, 1.8, 0.6, 0.9, 5.0, 0.8, 1.2, 0.3, 0.6, 1~
## $ other_transp       <dbl> 1.3, 1.4, 1.5, 1.5, 0.4, 1.7, 0.6, 1.2, 0.4, 0.7, 1~
## $ work_at_home       <dbl> 1.8, 3.9, 1.6, 0.7, 2.3, 2.8, 1.7, 2.7, 2.1, 2.5, 1~
## $ mean_commute       <dbl> 26.5, 26.4, 24.1, 28.8, 34.9, 27.5, 24.6, 24.1, 25.~
## $ employed           <dbl> 23986, 85953, 8597, 8294, 22189, 3865, 7813, 47401,~
```

```
## $ private_work       <dbl> 73.6, 81.5, 71.8, 76.8, 82.0, 79.5, 77.4, 74.1, 85.~
## $ public_work        <dbl> 20.9, 12.3, 20.8, 16.1, 13.5, 15.1, 16.2, 20.8, 12.~
## $ self_employed      <dbl> 5.5, 5.8, 7.3, 6.7, 4.2, 5.4, 6.2, 5.0, 2.8, 7.9, 4~
## $ family_work        <dbl> 0.0, 0.4, 0.1, 0.4, 0.4, 0.0, 0.2, 0.1, 0.0, 0.5, 0~
## $ unemployment       <dbl> 7.6, 7.5, 17.6, 8.3, 7.7, 18.0, 10.9, 12.3, 8.9, 7.~
## $ land_area          <dbl> 594.44, 1589.78, 884.88, 622.58, 644.78, 622.81, 77~
```

The dataset has 3138 observations (rows) across 40 variables(columns)

Selecting Columns : select()

Datasets often come with more variables than you need.. We will collect only a few variables: the state, the county, the total population, and the unemployment rate. We can do this using the select() verb. ***select()*** extracts only particular variables from a dataset. In this case, you can type counties, then the pipe operator, then select, then the variables of interest.

```
mydata %>% # call the data set
    select(state, county, population, unemployment) # select desired columns
```

```
## # A tibble: 3,138 x 4
##    state   county   population unemployment
##    <chr>   <chr>         <dbl>        <dbl>
##  1 Alabama Autauga       55221          7.6
##  2 Alabama Baldwin      195121          7.5
##  3 Alabama Barbour       26932         17.6
##  4 Alabama Bibb          22604          8.3
##  5 Alabama Blount        57710          7.7
##  6 Alabama Bullock       10678         18
##  7 Alabama Butler        20354         10.9
##  8 Alabama Calhoun      116648         12.3
##  9 Alabama Chambers      34079          8.9
## 10 Alabama Cherokee      26008          7.9
## # ... with 3,128 more rows
```

Sometimes you want to keep the data you've selected. You can use assignment to create a new table. Recall that you use the arrow operator, written as "less than dash", for this

```
# creating a new table from existing table
counties_selected <- mydata %>%
    select(state, county, population, unemployment)

# glimpse the new table
glimpse(counties_selected)
```

```
## Rows: 3,138
## Columns: 4
## $ state        <chr> "Alabama", "Alabama", "Alabama", "Alabama", "Alabama", "A~
## $ county       <chr> "Autauga", "Baldwin", "Barbour", "Bibb", "Blount", "Bullo~
## $ population   <dbl> 55221, 195121, 26932, 22604, 57710, 10678, 20354, 116648,~
## $ unemployment <dbl> 7.6, 7.5, 17.6, 8.3, 7.7, 18.0, 10.9, 12.3, 8.9, 7.9, 9.1~
```

Let us select only the state ; county ; population and poverty

```
mydata %>%
    select(state , county , population , poverty)
```

```
## # A tibble: 3,138 x 4
##    state   county   population poverty
##    <chr>   <chr>         <dbl>   <dbl>
##  1 Alabama Autauga       55221    12.9
##  2 Alabama Baldwin      195121    13.4
##  3 Alabama Barbour       26932    26.7
##  4 Alabama Bibb          22604    16.8
##  5 Alabama Blount        57710    16.7
##  6 Alabama Bullock       10678    24.6
##  7 Alabama Butler        20354    25.4
##  8 Alabama Calhoun      116648    20.5
##  9 Alabama Chambers      34079    21.6
## 10 Alabama Cherokee      26008    19.2
## # ... with 3,128 more rows
```

Arranging Rows : arrange()

Sometimes all the data you want is in your data frame, but it's all unorganized! The dplyr function arrange() will sort the rows of a data frame in ascending order by the column provided as an argument.

- ***For numeric columns, ascending order means from lower to higher numbers.***
- ***For character columns, ascending order means alphabetical order from A to Z.***

Syntax : dataframe %>% arrange(variable(s)_to_sort)

Note : By default the arrange() function arranges data by ascending order

Let us arrange our "counties_selected" table by population

```
# arrange ascending order of population
counties_selected %>%
    arrange(population)
```

```
## # A tibble: 3,138 x 4
##    state      county    population unemployment
##    <chr>      <chr>          <dbl>        <dbl>
##  1 Hawaii     Kalawao           85          0
##  2 Texas      King             267          5.1
##  3 Nebraska   McPherson        433          0.9
##  4 Montana    Petroleum        443          6.6
##  5 Nebraska   Arthur           448          4
##  6 Nebraska   Loup             548          0.7
##  7 Nebraska   Blaine           551          0.7
##  8 New Mexico Harding          565          6
##  9 Texas      Kenedy           565          0
## 10 Colorado   San Juan         606         13.8
## # ... with 3,128 more rows
```

There is one county in Hawaii which has a population of 85 people

Let us examine which county has the highest population.To do this we ass the desc() argument

```
# arrange descending order of population
counties_selected %>%
    arrange(desc(population))
```

```
## # A tibble: 3,138 x 4
##    state      county      population unemployment
##    <chr>      <chr>            <dbl>        <dbl>
##  1 California Los Angeles   10038388         10
##  2 Illinois   Cook           5236393         10.7
##  3 Texas      Harris         4356362          7.5
##  4 Arizona    Maricopa       4018143          7.7
##  5 California San Diego      3223096          8.7
##  6 California Orange         3116069          7.6
##  7 Florida    Miami-Dade     2639042         10
##  8 New York   Kings          2595259         10
##  9 Texas      Dallas         2485003          7.6
## 10 New York   Queens         2301139          8.6
## # ... with 3,128 more rows
```

The highest population is Los Angeles, California, which is one of the biggest cities in the United States

Filtering Rows : filter()

Filter by Condition

The filter() function can subset rows of a data frame based on logical operations of certain columns. The condition of the filter should be explicitly passed as a parameter

Syntax: name of the column, operator(<,==,>,!=) and value.

You can add a pipe operator, then add another verb. You can pipe any number of verbs together to transform your dataset.For example, after the arrange(), you could add filter state equals equals quote New York to get only counties in the state of New York

```
# arrange descending order of population
counties_selected %>%
    arrange(desc(population)) %>%
    # filer only those rows where state is New York
    filter(state == "New York")
```

```
## # A tibble: 62 x 4
##    state    county      population unemployment
##    <chr>    <chr>            <dbl>        <dbl>
##  1 New York Kings          2595259         10
##  2 New York Queens         2301139          8.6
##  3 New York New York       1629507          7.5
##  4 New York Suffolk        1501373          6.4
##  5 New York Bronx          1428357         14
##  6 New York Nassau         1354612          6.4
```

```
##   7 New York Westchester     967315          7.6
##   8 New York Erie            921584          7
##   9 New York Monroe          749356          7.7
##  10 New York Richmond        472481          6.9
## # ... with 52 more rows
```

Notice that the observations are filtered, but they're still sorted by population thanks to our arrange()

Besides "==", you can filter based on logical operators like less than or greater than. For example, you could filter for counties that have an unemployment rate of less than 6 percent. The condition in the filter would be unemployment less than 6.

```
# arrange descending order of population
counties_selected %>%
    arrange(desc(population)) %>%
    # filter only those rows where unemployment is < 6
    filter(unemployment < 6)
```

```
## # A tibble: 949 x 4
##    state    county        population unemployment
##    <chr>    <chr>              <dbl>        <dbl>
##  1 Virginia Fairfax          1128722          4.9
##  2 Utah     Salt Lake        1078958          5.8
##  3 Hawaii   Honolulu          984178          5.6
##  4 Texas    Collin            862215          4.9
##  5 Texas    Denton            731851          5.7
##  6 Texas    Fort Bend         658331          5.1
##  7 Kansas   Johnson           566814          4.5
##  8 Maryland Anne Arundel      555280          5.9
##  9 Colorado Jefferson         552344          5.9
## 10 Utah     Utah              551957          5.5
## # ... with 939 more rows
```

The largest counties with an unemployment rate below 6 percent are Fairfax, Virginia and Salt Late, Utah

The filter() function also allows for more complex filtering with the help of logical operators!

We filtered for the state of New York and for unemployment below 6 percent. We can do both at the same time if we separate them with a comma

```
# arrange descending order of population
counties_selected %>%
    arrange(desc(population)) %>%
    # filter for New York and unployment < 6
    filter(state == "New York" ,
           unemployment < 6)
```

```
## # A tibble: 5 x 4
##   state    county     population unemployment
##   <chr>    <chr>           <dbl>        <dbl>
## 1 New York Tompkins       103855          5.9
## 2 New York Chemung         88267          5.4
## 3 New York Madison         72427          5.1
## 4 New York Livingston      64801          5.4
## 5 New York Seneca          35144          5.5
```

It looks like only a few counties in New York have an unemployment rate that is < 6%.

Let us see the counties_selected dataset with a few interesting variables selected. These variables: private_work, public_work, self_employed describe whether people work for the government, for private companies, or for themselves.

```
# selecting key columns
counties_selected_2 <- mydata %>%
  select(state, county, population, private_work, public_work, self_employed)

counties_selected_2 %>%
  # Add a verb to sort in descending order of public_work
  arrange(desc(public_work))
```

```
## # A tibble: 3,138 x 6
##    state     county             population private_work public_work self_employed
##    <chr>     <chr>                   <dbl>        <dbl>       <dbl>         <dbl>
##  1 Hawaii    Kalawao                    85         25          64.1          10.9
##  2 Alaska    Yukon-Koyukuk Ce~        5644         33.3        61.7           5.1
##  3 Wisconsin Menominee                4451         36.8        59.1           3.7
##  4 North Da~ Sioux                    4380         32.9        56.8          10.2
##  5 South Da~ Todd                     9942         34.4        55             9.8
##  6 Alaska    Lake and Peninsu~        1474         42.2        51.6           6.1
##  7 Californ~ Lassen                  32645         42.6        50.5           6.8
##  8 South Da~ Buffalo                  2038         48.4        49.5           1.8
##  9 South Da~ Dewey                    5579         34.9        49.2          14.7
## 10 Texas     Kenedy                    565         51.9        48.1           0
## # ... with 3,128 more rows
```

This looks odd.. 64% of the population of Kalawao county are employed in public office.

Let us look at observations in counties that have a large population (< 1M)

Let us look at only state, county and population and then filter rows with population > 1M

```
# create anew table with desired columns only
counties_selected_3 <- mydata %>%
    select(state , county , population)

glimpse(counties_selected_3)
```

```
## Rows: 3,138
## Columns: 3
## $ state      <chr> "Alabama", "Alabama", "Alabama", "Alabama", "Alabama", "Ala~
## $ county     <chr> "Autauga", "Baldwin", "Barbour", "Bibb", "Blount", "Bullock~
## $ population <dbl> 55221, 195121, 26932, 22604, 57710, 10678, 20354, 116648, 3~
```

```
# filter counties with pupulation > 1M
counties_selected_3 %>%
    filter(population > 1000000)
```

```
## # A tibble: 41 x 3
##    state      county          population
```

```
##    <chr>      <chr>                 <dbl>
##  1 Arizona    Maricopa            4018143
##  2 California Alameda             1584983
##  3 California Contra Costa        1096068
##  4 California Los Angeles        10038388
##  5 California Orange              3116069
##  6 California Riverside           2298032
##  7 California Sacramento          1465832
##  8 California San Bernardino      2094769
##  9 California San Diego           3223096
## 10 California Santa Clara         1868149
## # ... with 31 more rows
```

Let us find only the counties in the state of California that also have a population above one million

```
counties_selected_3 %>%
    filter(population > 1000000 ,
           state == "California")
```

```
## # A tibble: 9 x 3
##   state      county         population
##   <chr>      <chr>               <dbl>
## 1 California Alameda           1584983
## 2 California Contra Costa      1096068
## 3 California Los Angeles      10038388
## 4 California Orange            3116069
## 5 California Riverside         2298032
## 6 California Sacramento        1465832
## 7 California San Bernardino    2094769
## 8 California San Diego         3223096
## 9 California Santa Clara       1868149
```

There are 9 counties in the state of California with a population greater than one million

Filtering and Arranging

We're often interested in both filtering and sorting a dataset, to focus on observations of particular interest.

Let us filter for counties in the state of Texas that have more than ten thousand people (10000), and sort them in descending order of the percentage of people employed in private work.

```
# Filter for Texas and more than 10000 people; sort in descending order of private_work
counties_selected_2 %>%
  # Filter for Texas and more than 10000 people
  filter(state == "Texas" ,
         population > 10000) %>%
  # Sort in descending order of private_work
  arrange(desc(private_work))
```

```
## # A tibble: 169 x 6
##    state county  population private_work public_work self_employed
##    <chr> <chr>        <dbl>        <dbl>       <dbl>         <dbl>
```

```
##   1 Texas Gregg       123178         84.7         9.8           5.4
##   2 Texas Collin      862215         84.1        10             5.8
##   3 Texas Dallas     2485003         83.9         9.5           6.4
##   4 Texas Harris     4356362         83.4        10.1           6.3
##   5 Texas Andrews      16775         83.1         9.6           6.8
##   6 Texas Tarrant    1914526         83.1        11.4           5.4
##   7 Texas Titus        32553         82.5        10             7.4
##   8 Texas Denton      731851         82.2        11.9           5.7
##   9 Texas Ector       149557         82          11.2           6.7
##  10 Texas Moore        22281         82          11.7           5.9
## # ... with 159 more rows
```

We find that there are extreme examples of what fraction of the population works in the private sector

Adding a Column : mutate()

When working with data frames, we often need to modify the columns for our analysis at hand.Th new column(s) could be a calculation based on the data that you already have. You can add a new column to the data frame using the mutate function.mutate() takes a name-value pair as an argument. The name will be the name of the new column you are adding, and the value is an expression defining the values of the new column in terms of the existing columns. mutate() returns a new data frame with the added column.

Note : It is a best practice to give the added column a name.

What if we are interested in the total number of unemployed people in a county, rather than as a percentage of the population? We could use the formula population times unemployment divided by 100.We can also name this new column as unemployed_population

```
# unemployed population
counties_selected %>%
    mutate(unemployed_population = population * unemployment / 100)
```

```
## # A tibble: 3,138 x 5
##    state   county   population unemployment unemployed_population
##    <chr>   <chr>         <dbl>        <dbl>                 <dbl>
##  1 Alabama Autauga       55221          7.6                 4197.
##  2 Alabama Baldwin      195121          7.5                14634.
##  3 Alabama Barbour       26932         17.6                 4740.
##  4 Alabama Bibb          22604          8.3                 1876.
##  5 Alabama Blount        57710          7.7                 4444.
##  6 Alabama Bullock       10678         18                   1922.
##  7 Alabama Butler        20354         10.9                 2219.
##  8 Alabama Calhoun      116648         12.3                14348.
##  9 Alabama Chambers      34079          8.9                 3033.
## 10 Alabama Cherokee      26008          7.9                 2055.
## # ... with 3,128 more rows
```

We can combine this new variable with other verbs to ask more questions of your data. For example, what counties have the highest number of unemployed people? We add arrange desc unemployed-underscore-population to our mutate.

```r
# unemployed population
counties_selected %>%
    # create a new column of unemployed population
    mutate(unemployed_population = population * unemployment / 100) %>%
    # arrange by descending order of unemployed population
    arrange(desc(unemployed_population)) %>%
    # look at top 6
    head()
```

```
## # A tibble: 6 x 5
##   state      county      population unemployment unemployed_population
##   <chr>      <chr>            <dbl>        <dbl>                 <dbl>
## 1 California Los Angeles   10038388         10                1003839.
## 2 Illinois   Cook           5236393         10.7               560294.
## 3 Texas      Harris         4356362          7.5               326727.
## 4 Arizona    Maricopa       4018143          7.7               309397.
## 5 California Riverside      2298032         12.9               296446.
## 6 California San Diego      3223096          8.7               280409.
```

Los Angeles has the highest unemployed population close to 1M which is 10% of the total population of LA

Let us shift our focus to "Government Employees". Let us first calculate the number of public workers. Notice that % of public workers is recorded in the public_work column. We will use this to derive the total number of public workers

```r
# create the data frame of selected columns only
counties_selected_4 <- mydata %>%
  select(state, county, population, public_work)

glimpse(counties_selected_4)
```

```
## Rows: 3,138
## Columns: 4
## $ state       <chr> "Alabama", "Alabama", "Alabama", "Alabama", "Alabama", "Al~
## $ county      <chr> "Autauga", "Baldwin", "Barbour", "Bibb", "Blount", "Bulloc~
## $ population  <dbl> 55221, 195121, 26932, 22604, 57710, 10678, 20354, 116648, ~
## $ public_work <dbl> 20.9, 12.3, 20.8, 16.1, 13.5, 15.1, 16.2, 20.8, 12.1, 18.5~
```

Create the new column with number of public workers and then sort in descending order

```r
# Create a new column for # of public workers
counties_selected_4 %>%
  # Add public_workers with the number of people employed in public work
  mutate(public_workers = public_work * population / 100) %>%
  # Arrange in descending order
  arrange(desc(public_workers)) %>%
  # Top 6 Rows
  head()
```

```
## # A tibble: 6 x 5
##   state      county      population public_work public_workers
##   <chr>      <chr>            <dbl>       <dbl>          <dbl>
```

```
## 1 California Los Angeles   10038388        11.5       1154415.
## 2 Illinois   Cook            5236393        11.5        602185.
## 3 California San Diego      3223096        14.8        477018.
## 4 Arizona    Maricopa        4018143        11.7        470123.
## 5 Texas      Harris          4356362        10.1        439993.
## 6 New York   Kings           2595259        14.4        373717.
```

It looks like Los Angeles is the county with the most government employees

Let us look at Gender Diversity i.e. how many women employees do we have.The dataset includes columns for the total number (not percentage) of men and women in each county. We could use this, along with the `population` variable, to compute the fraction of men (or women) within each county.

```
# Select only the desired columns we want
mydata %>%
    # Select the columns state, county, population, men, and women
  select(state, county, population, men , women) %>%
    # Calculate proportion_women as the fraction of the total population
  mutate(proportion_women = women / population)
```

```
## # A tibble: 3,138 x 6
##    state   county   population   men women proportion_women
##    <chr>   <chr>         <dbl> <dbl> <dbl>            <dbl>
##  1 Alabama Autauga       55221 26745 28476            0.516
##  2 Alabama Baldwin      195121 95314 99807            0.512
##  3 Alabama Barbour       26932 14497 12435            0.462
##  4 Alabama Bibb          22604 12073 10531            0.466
##  5 Alabama Blount        57710 28512 29198            0.506
##  6 Alabama Bullock       10678  5660  5018            0.470
##  7 Alabama Butler        20354  9502 10852            0.533
##  8 Alabama Calhoun      116648 56274 60374            0.518
##  9 Alabama Chambers      34079 16258 17821            0.523
## 10 Alabama Cherokee      26008 12975 13033            0.501
## # ... with 3,128 more rows
```

Notice that the proportion_women variable was added as a column to the dataset, and the data now has 6 columns instead of 5

Select+Mutate + Filter + Arrange

Putting it all together

In this exercise, we will put together everything we have learned so far (select(), mutate(), filter() and arrange()), to find the counties with the highest proportion of men.

- Select only the columns state, county, population, men, and women.
- Add a variable proportion_men with the fraction of the county's population made up of men.
- Filter for counties with a population of at least ten thousand (10000).
- Arrange counties in descending order of their proportion of men.

```r
mydata %>%
    # Select the five columns
    select(state , county , population , men , women) %>%
    # Add the proportion_men variable
    mutate(proportion_men = men / population) %>%
    # Filter for population of at least 10,000
    filter(population >= 10000) %>%
    # Arrange proportion of men in descending order
    arrange(desc(proportion_men)) %>%
    # View Top 6 Rows
    head()
```

```
## # A tibble: 6 x 6
##   state      county          population   men women proportion_men
##   <chr>      <chr>                <dbl> <dbl> <dbl>          <dbl>
## 1 Virginia   Sussex               11864  8130  3734          0.685
## 2 California Lassen               32645 21818 10827          0.668
## 3 Georgia    Chattahoochee        11914  7940  3974          0.666
## 4 Louisiana  West Feliciana       15415 10228  5187          0.664
## 5 Florida    Union                15191  9830  5361          0.647
## 6 Texas      Jones                19978 12652  7326          0.633
```

Notice Sussex County in Virginia is more than two thirds male: this is because of two men's prisons in the county

Aggregating Data

Now that we know how to transform your data, we want to know more about how to aggregate your data to make it more interpretable. There are a number of functions use can use to take many observations in a dataset and summarize them, Common data aggregation include count , minimum , maximum , mean, median, and standard deviation to name a few.

Count : count()

Simple Count

The simplest way we can aggregate data is to count it: to find out the number of observations. The dplyr verb for this is count() , which results is a one-row table, with one column called.

```r
# simple count of observations in the data
mydata %>%
    count()
```

```
## # A tibble: 1 x 1
##       n
##   <int>
## 1  3138
```

This tells us there are 3,138 observations in the table. Counting the total data is a little useful, but the real value of the verb is when you give it a specific variable to count. For example, we could count the number of counties in each state.

```r
# how many counties
mydata %>%
    count(state)
```

```
## # A tibble: 50 x 2
##    state           n
##    <chr>       <int>
##  1 Alabama        67
##  2 Alaska         28
##  3 Arizona        15
##  4 Arkansas       75
##  5 California     58
##  6 Colorado       64
##  7 Connecticut     8
##  8 Delaware        3
##  9 Florida        67
## 10 Georgia       159
## # ... with 40 more rows
```

Notice that the result has 50 observations: one for each of the 50 states. We've aggregated more than three thousand observations into a more manageable number. The second column, n, tells us there are 67 counties in Alabama, 28 in Alaska, and so on

Count and Sort

The count verb takes a second argument sort that's very useful for that. resulting in rows sorted from the most common observations to the least.Note that we need to specif "sort = TRUE"

```r
# counting the number of counties by state and sorting
mydata %>%
    count(state , sort = TRUE)
```

```
## # A tibble: 50 x 2
##    state              n
##    <chr>          <int>
##  1 Texas            253
##  2 Georgia          159
##  3 Virginia         133
##  4 Kentucky         120
##  5 Missouri         115
##  6 Kansas           105
##  7 Illinois         102
##  8 North Carolina   100
##  9 Iowa              99
## 10 Tennessee         95
## # ... with 40 more rows
```

This tells us that Texas is the state with the most counties, followed by Georgia and Virginia

Count + Weight + Sort

Let us aggregate the total population of each county

We can add the argument wt, which stands for "weight", equals population. This means that the n column will be weighted by the population. In the result, instead of seeing the number of counties in each state, we'd see the total population

```
# population of each county
mydata %>%
    count(state , wt = population , sort = TRUE)
```

```
## # A tibble: 50 x 2
##    state                 n
##    <chr>             <dbl>
##  1 California     38421464
##  2 Texas          26538497
##  3 New York       19673174
##  4 Florida        19645772
##  5 Illinois       12873761
##  6 Pennsylvania   12779559
##  7 Ohio           11575977
##  8 Georgia        10006693
##  9 Michigan        9900571
## 10 North Carolina  9845333
## # ... with 40 more rows
```

Here we can see that California is the US state with the highest population, followed by Texas and New York

The counties dataset contains columns for region, state, population, and the number of citizens, which we selected and saved as the counties_selected.Let us create a separate table with county , region , state , population and citizens

```
# create a new table with desired columns
counties_selected <- mydata %>%
    select(county , region , state , population , citizens)

counties_selected %>%
    head()
```

```
## # A tibble: 6 x 5
##   county  region state   population citizens
##   <chr>   <chr>  <chr>        <dbl>    <dbl>
## 1 Autauga South  Alabama      55221    40725
## 2 Baldwin South  Alabama     195121   147695
## 3 Barbour South  Alabama      26932    20714
## 4 Bibb    South  Alabama      22604    17495
## 5 Blount  South  Alabama      57710    42345
## 6 Bullock South  Alabama      10678     8057
```

1. **How many counties within each region?**

```
counties_selected %>%
    count(region , sort = TRUE)
```

```
## # A tibble: 4 x 2
##   region            n
##   <chr>         <int>
## 1 South          1420
## 2 North Central  1054
## 3 West            447
## 4 Northeast       217
```

Since the results have been arranged, you can see that the South has the greatest number of counties

2. **How many counties in each state, weighted based on the citizens**

```
counties_selected %>%
    count(state , wt = citizens , sort = TRUE)
```

```
## # A tibble: 50 x 2
##    state                  n
##    <chr>              <dbl>
##  1 California     24280349
##  2 Texas          16864864
##  3 Florida        13933052
##  4 New York       13531404
##  5 Pennsylvania    9710416
##  6 Illinois        8979999
##  7 Ohio            8709050
##  8 Michigan        7380136
##  9 North Carolina  7107998
## 10 Georgia         6978660
## # ... with 40 more rows
```

From our result, we can see that California is the state with the most citizens

Count + Mutate

We can combine multiple verbs together to answer increasingly complicated questions of our data.

3. **What are the US states where the most people walk to work?**

Let us use the walk column, which offers a **percentage of people** in each county that walk to work, to add a new column and count based on it.

```
# update the table to include walk
counties_selected <- mydata %>%
    select(county, region, state, population,citizens , walk )

counties_selected %>% head()
```

```
## # A tibble: 6 x 6
##   county  region state   population citizens  walk
##   <chr>   <chr>  <chr>        <dbl>    <dbl> <dbl>
## 1 Autauga South  Alabama      55221    40725   0.5
## 2 Baldwin South  Alabama     195121   147695   1
## 3 Barbour South  Alabama      26932    20714   1.8
## 4 Bibb    South  Alabama      22604    17495   0.6
## 5 Blount  South  Alabama      57710    42345   0.9
## 6 Bullock South  Alabama      10678     8057   5
```

Let us use the mutate statement to calculate and add a column called population_walk, containing the total number of people who walk to work in a county.Note the walk column gives % of population who walk.

population who walk = (population *(% walk) /100

```
counties_selected %>%
    # create a column population who walk
    mutate(population_walk = population * walk/100)
```

```
## # A tibble: 3,138 x 7
##    county   region state   population citizens  walk population_walk
##    <chr>    <chr>  <chr>        <dbl>    <dbl> <dbl>           <dbl>
##  1 Autauga  South  Alabama      55221    40725   0.5            276.
##  2 Baldwin  South  Alabama     195121   147695   1             1951.
##  3 Barbour  South  Alabama      26932    20714   1.8            485.
##  4 Bibb     South  Alabama      22604    17495   0.6            136.
##  5 Blount   South  Alabama      57710    42345   0.9            519.
##  6 Bullock  South  Alabama      10678     8057   5              534.
##  7 Butler   South  Alabama      20354    15581   0.8            163.
##  8 Calhoun  South  Alabama     116648    88612   1.2           1400.
##  9 Chambers South  Alabama      34079    26462   0.3            102.
## 10 Cherokee South  Alabama      26008    20600   0.6            156.
## # ... with 3,128 more rows
```

Let us now use the weight method to count the total number of people who walk in each state

```
counties_selected %>%
    # create a column population who walk
    mutate(population_walk = population * walk/100) %>%
    # count each state weighted by population
    count(state, wt = population_walk, sort = TRUE)
```

```
## # A tibble: 50 x 2
##    state                n
##    <chr>            <dbl>
##  1 New York      1237938.
##  2 California    1017964.
##  3 Pennsylvania   505397.
##  4 Texas          430783.
##  5 Illinois       400346.
##  6 Massachusetts  316765.
##  7 Florida        284723.
##  8 New Jersey     273047.
```

```
## 9 Ohio          266911.
## 10 Washington    239764.
## # ... with 40 more rows
```

Though California had the largest total population, New York state has the largest number of people who walk to work.

Grouping + Summarizing : group_by() ; summarize()

Summarize() : Summary The summarize verb takes many observations and turns them into one observation.To *combine* all of the values from a column for a single calculation , we take the help of the dplyr function summarize(), which returns a new data frame containing the desired calculation.

The general syntax for summarizing calculations is:

```
df %>%
summarize(var_name = command(column_name))
```

- df is the data frame you are working with
- summarize is a dplyr function that reduces multiple values to a single value
- var_name is the name you assign to the column that stores the results of the summary function in the returned data frame
- command is the summary function that is applied to the column by summarize()
- column_name is the name of the column of df that is being summarized

The following table includes common summary functions that can be given as an argument to summarize():

Command	Description
mean()	Average of all values in column
median()	Median value of column
sd()	Standard deviation of column
var()	Variance of column
min()	Minimum value in column
max()	Maximum value in column
IQR()	Interquartile range of column
n_distinct()	Number of unique values in column
sum()	Sum values of column

Let us find the total population of the United States

```
mydata %>%
    summarize(total_population = sum(population))
```

```
## # A tibble: 1 x 1
##   total_population
##              <dbl>
## 1        315845353
```

We can define multiple variables in a summarize, and you can aggregate each in different ways. For example, you could find the total population, but also the average unemployment rate

```
mydata %>%
    summarize(total_population = sum(population) ,
              average_unemployment = mean(unemployment))
```

```
## # A tibble: 1 x 2
##   total_population average_unemployment
##              <dbl>                <dbl>
## 1        315845353                 7.80
```

Group_By() : Aggregates When we have a bunch of data, we often want to calculate aggregate statistics (mean, standard deviation, median, percentiles, etc.) over certain subsets of the data.We accomplish this by applying the group_by() verb followed by the summarize verb.This is called "Aggregation".

General syntax to calculate aggregates:

```
df %>%
group_by(column_1) %>%
summarize(aggregate_name =command(column_2))
```

- column_1 (student in our example) is the column that we want to group_by()
- column_2 (grade in our example) is the column that we want to apply command(), a summary function, to using summarize()
- aggregate_name is the name assigned to the calculated aggregate

Let us find the total population within each state and the average unemployment

```
mydata %>%
    group_by(state) %>% # grouping column
    summarize(state_pop   = sum(population) ,  # summary stat 1
              state_unemp = mean(unemployment)) %>% # summary stat 2
    ungroup() # ungroup
```

```
## # A tibble: 50 x 3
##    state       state_pop state_unemp
##    <chr>           <dbl>       <dbl>
##  1 Alabama       4830620       11.3
##  2 Alaska         725461        9.19
##  3 Arizona       6641928       12.0
##  4 Arkansas      2958208        8.98
##  5 California   38421464       10.8
##  6 Colorado      5278906        7.46
##  7 Connecticut   3593222        8.16
##  8 Delaware       926454        7.93
##  9 Florida      19645772       10.4
## 10 Georgia      10006693        9.97
## # ... with 40 more rows
```

Let us clean this further by arranging in descending order to find states with highest unemployment

```
mydata %>%
    group_by(state) %>%
    summarize(state_pop  = sum(population) ,
              mean_unemp = mean(unemployment)) %>%
    ungroup() %>%
    arrange(desc(mean_unemp))
```

```
## # A tibble: 50 x 3
##    state          state_pop mean_unemp
##    <chr>              <dbl>      <dbl>
##  1 Mississippi      2988081      12.0
##  2 Arizona          6641928      12.0
##  3 South Carolina   4777576      11.3
##  4 Alabama          4830620      11.3
##  5 California      38421464      10.8
##  6 Nevada           2798636      10.5
##  7 North Carolina   9845333      10.5
##  8 Florida         19645772      10.4
##  9 Georgia         10006693       9.97
## 10 Michigan         9900571       9.96
## # ... with 40 more rows
```

Mississippi is the state with the highest unemployment

Sometimes, we want to group by more than one column. We can do this by passing multiple column names as arguments to the group_by function.

The dataset also includes a metro column, which describes whether the county is a metro area- that is, a city or non-metro

Let us view the population by state and by metro

```
mydata %>%
    select(state, metro, county, population) %>%
    group_by(state , metro) %>%
    summarize(tot_pop = sum(population)) %>%
    ungroup()
```

```
## # A tibble: 97 x 3
##    state      metro     tot_pop
##    <chr>      <chr>       <dbl>
##  1 Alabama    Metro     3671377
##  2 Alabama    Nonmetro  1159243
##  3 Alaska     Metro      494990
##  4 Alaska     Nonmetro   230471
##  5 Arizona    Metro     6295145
##  6 Arizona    Nonmetro   346783
##  7 Arkansas   Metro     1806867
##  8 Arkansas   Nonmetro  1151341
##  9 California Metro    37587429
## 10 California Nonmetro   834035
## # ... with 87 more rows
```

Instead of 50 observations in the output, we have 97, since a few states don't have any counties that aren't metro areas. For instance, here we see that the total population in Alabama metro areas is 3-point-6 million, and the population in non-metro areas is 1.2M.

- Summarize the counties dataset to find the following columns: min_population (with the smallest population), max_unemployment (with the maximum unemployment), and average_income (with the mean of the income variable).Select only county, population, income, unemployment)

```
mydata %>%
    select(county , population , income , unemployment)%>%
    summarize(min_population    = min(population) ,
              max_unemployment  = max(unemployment) ,
              average_income = mean(income))
```

```
## # A tibble: 1 x 3
##   min_population max_unemployment average_income
##            <dbl>            <dbl>          <dbl>
## 1             85             29.4         46832.
```

Another interesting column is land_area, which shows the land area in square miles. Here, you'll summarize both population and land area by state, with the purpose of finding the density (in people per square miles).

- Group the data by state, and summarize to create the columns total_area (with total area in square miles) and total_population (with total population).Select only state, county, population, land_area
- Next add a density column with the people per square mile, then arrange in descending order

```
mydata %>%
    select(state, county, population, land_area) %>% # select desired columns
    group_by(state) %>% # group by state
    summarize(total_area       = sum(land_area) , # stat 1
              total_population = sum(population) , # stat 2
              density          = total_population / total_area) %>%
    ungroup() %>%
    arrange(desc(density)) # arrange descending order
```

```
## # A tibble: 50 x 4
##    state         total_area total_population density
##    <chr>              <dbl>            <dbl>   <dbl>
##  1 New Jersey         7354.          8904413   1211.
##  2 Rhode Island       1034.          1053661   1019.
##  3 Massachusetts      7800.          6705586    860.
##  4 Connecticut        4842.          3593222    742.
##  5 Maryland           9707.          5930538    611.
##  6 Delaware           1949.           926454    475.
##  7 New York          47126.         19673174    417.
##  8 Florida           53625.         19645772    366.
##  9 Pennsylvania      44743.         12779559    286.
## 10 Ohio              40861.         11575977    283.
## # ... with 40 more rows
```

New Jersey and Rhode Island are the "most crowded" of the US states, with more than a thousand people per square mile

- Summarize to find the total population, as a column called total_pop, in each combination of region and state.
- Calculate two new columns: the average state population in each region (average_pop) and the median state population in each region (median_pop)

```
mydata %>%
    # Group and summarize to find the total population
  group_by(region, state) %>%
  summarize(total_pop = sum(population)) %>%
  # Calculate the average_pop and median_pop columns
  summarize(average_pop = mean(total_pop),
            median_pop = median(total_pop)) %>%
    ungroup()
```

```
## # A tibble: 4 x 3
##   region        average_pop median_pop
##   <chr>               <dbl>      <dbl>
## 1 North Central    5627687.    5580644
## 2 Northeast        6221058.    3593222
## 3 South            7370486     4804098
## 4 West             5722755.    2798636
```

The South Region has the highest average_pop of 7.3M, while North Central region has the highest median_pop of .5M.

Top(top_n) : Ranking

Let's say , instead of aggregating , we want to find only the largest or smallest value in a group. dplyr's top_n is very useful for keeping the most extreme observations from each group

Like summarize(), top_n operates on a grouped table. The function takes two arguments: the number of observations you want from each group, and the column you want to weight by.

General Syntax : top_n(x, n, wt)

x : A data frame.

n : # of rows to return for top_n(). If n is positive, selects the top rows. If negative, selects the bottom rows. If x is grouped, this is the number of rows per group.

wt : (Optional). The variable to use for ordering. If not specified, defaults to the last variable in the tbl.

Let us find out the county with highest population in each state. Select only state, county, population, unemployment, income

```
mydata %>%
    # select the desired columns
    select(state, county, population, unemployment, income) %>%
    # group by state
    group_by(state) %>%
    # get the "Top County" with highest population
    top_n(1 , population)
```

```
## # A tibble: 50 x 5
## # Groups:   state [50]
##    state       county                 population unemployment income
##    <chr>       <chr>                       <dbl>        <dbl>  <dbl>
##  1 Alabama     Jefferson                  659026          9.1  45610
##  2 Alaska      Anchorage Municipality     299107          6.7  78326
##  3 Arizona     Maricopa                  4018143          7.7  54229
##  4 Arkansas    Pulaski                    390463          7.5  46140
##  5 California  Los Angeles              10038388         10    56196
##  6 Colorado    El Paso                    655024          8.4  58206
##  7 Connecticut Fairfield                  939983          9    84233
##  8 Delaware    New Castle                 549643          7.4  65476
##  9 Florida     Miami-Dade                2639042         10    43129
## 10 Georgia     Fulton                     983903          9.9  57207
## # ... with 40 more rows
```

This tells us, for example, that Jefferson is the highest population county in Alabama with a population of 659K.

Let us find out which are the Top 3 counties within each state with highest unemployment

```
mydata %>%
    select(state, county, population, unemployment, income) %>%
    group_by(state) %>%
    top_n(3 , unemployment)
```

```
## # A tibble: 153 x 5
## # Groups:   state [50]
##    state    county                    population unemployment income
##    <chr>    <chr>                          <dbl>        <dbl>  <dbl>
##  1 Alabama  Conecuh                        12865         22.6  24900
##  2 Alabama  Monroe                         22217         20.7  27257
##  3 Alabama  Wilcox                         11235         20.8  23750
##  4 Alaska   Bethel Census Area             17776         17.6  51012
##  5 Alaska   Northwest Arctic Borough        7732         21.9  63648
##  6 Alaska   Yukon-Koyukuk Census Area       5644         18.2  38491
##  7 Arizona  Apache                         72124         18.2  31757
##  8 Arizona  Graham                         37407         14.1  45964
##  9 Arizona  Navajo                        107656         19.8  35921
## 10 Arkansas Desha                          12379         17.7  27197
## # ... with 143 more rows
```

For Alabama these turn out to be named Conecuh, Monroe, and Wilcox.

Top n is often used when creating graphs, where we're interested in pulling the extreme examples to include in the visualization

- Find the county in each region with the highest percentage of citizens who walk to work
- Select only region, state, county, metro, population, walk

```
mydata %>%
    # select the desired columns
    select(region, state, county, metro, population, walk) %>%
```

```r
    # group by region
    group_by(region) %>%
    # top county by % citizens who walk
    top_n(1 , walk)
```

```
## # A tibble: 4 x 6
## # Groups:   region [4]
##   region        state        county                 metro    population  walk
##   <chr>         <chr>        <chr>                  <chr>         <dbl> <dbl>
## 1 West          Alaska       Aleutians East Borough Nonmetro       3304  71.2
## 2 Northeast     New York     New York               Metro       1629507  20.7
## 3 North Central North Dakota McIntosh               Nonmetro       2759  17.5
## 4 South         Virginia     Lexington city         Nonmetro       7071  31.7
```

Notice that three of the places lots of people walk to work are low-population non-metro counties, New York City also pops up

- **Finding the highest-income state in each region.**

We will combine group_by(), summarize(), and top_n() to find the state in each region with the highest income.

When you group by multiple columns and then summarize, it's important to remember that the summarize "peels off" one of the groups, but leaves the rest on. For example, if you `group_by(X, Y)` then summarize, the result will still be grouped by `X`.

Select only region, state, county, population, income

```r
mydata %>%
    # select the desired columns
    select(region, state, county, population, income) %>%
    # group by region and state
    group_by(region , state) %>%
    # calculate the average income
    summarize(average_income = mean(income)) %>%
    # find the highest income state in each region
    top_n(1 , average_income) %>%
    ungroup()
```

```
## # A tibble: 4 x 3
##   region        state        average_income
##   <chr>         <chr>                 <dbl>
## 1 North Central North Dakota         55575.
## 2 Northeast     New Jersey           73014.
## 3 South         Maryland             69200.
## 4 West          Alaska               65125.
```

From our results, we can see that the New Jersey in the Northeast is the state with the highest average_income of 73014.

Selecting and Transforming Data

This section focuses on advanced methods of selecting and transforming columns. We will cover select helpers, which are functions that specify criteria for columns you want to choose, as well as the rename and transmute verbs

Select Range of Columns

We have seen that we can select the columns that we're interested in, using the select verb. We can also select a range of columns.Our dataset has a set of columns about the breakdown of jobs across industries, and we want all of the columns from professional to production.

General Syntax : df %>% select(col1:col2)

df : data frame from which columns are needed

col1 : starting column name

col2 : ending column name

```
# column names of dataset
names(mydata)
```

```
##  [1] "X1"                 "census_id"          "state"
##  [4] "county"             "region"             "metro"
##  [7] "population"         "men"                "women"
## [10] "hispanic"           "white"              "black"
## [13] "native"             "asian"              "pacific"
## [16] "citizens"           "income"             "income_err"
## [19] "income_per_cap"     "income_per_cap_err" "poverty"
## [22] "child_poverty"      "professional"       "service"
## [25] "office"             "construction"       "production"
## [28] "drive"              "carpool"            "transit"
## [31] "walk"               "other_transp"       "work_at_home"
## [34] "mean_commute"       "employed"           "private_work"
## [37] "public_work"        "self_employed"      "family_work"
## [40] "unemployment"       "land_area"
```

```
mydata %>%
    # select range of columns using ":" notation
    select(state , county , professional:production) %>%
    head()
```

```
## # A tibble: 6 x 7
##   state   county  professional service office construction production
##   <chr>   <chr>          <dbl>   <dbl>  <dbl>        <dbl>      <dbl>
## 1 Alabama Autauga         33.2    17     24.2          8.6       17.1
## 2 Alabama Baldwin         33.1    17.7   27.1         10.8       11.2
## 3 Alabama Barbour         26.8    16.1   23.1         10.8       23.1
## 4 Alabama Bibb            21.5    17.9   17.8         19         23.7
## 5 Alabama Blount          28.5    14.1   23.9         13.5       19.9
## 6 Alabama Bullock         18.8    15     19.7         20.1       26.4
```

If we wanted to know just the columns about how people get to work, you could do drive : work_at_home

```
mydata %>%
    select(state , county , drive : work_at_home) %>%
    head()
```

```
## # A tibble: 6 x 8
##   state   county  drive carpool transit  walk other_transp work_at_home
##   <chr>   <chr>   <dbl>   <dbl>   <dbl> <dbl>        <dbl>        <dbl>
## 1 Alabama Autauga  87.5     8.8     0.1   0.5          1.3          1.8
## 2 Alabama Baldwin  84.7     8.8     0.1   1            1.4          3.9
## 3 Alabama Barbour  83.8    10.9     0.4   1.8          1.5          1.6
## 4 Alabama Bibb     83.2    13.5     0.5   0.6          1.5          0.7
## 5 Alabama Blount   84.9    11.2     0.4   0.9          0.4          2.3
## 6 Alabama Bullock  74.9    14.9     0.7   5            1.7          2.8
```

Let us arrange these columns by "drive" which is driving distance

```
mydata %>%
    select(state , county , drive : work_at_home) %>%
    arrange(drive)%>%
    head()
```

```
## # A tibble: 6 x 8
##   state   county           drive carpool transit  walk other_transp work_at_home
##   <chr>   <chr>            <dbl>   <dbl>   <dbl> <dbl>        <dbl>        <dbl>
## 1 New Yo~ New York           6.1     1.9    59.2  20.7          5.4          6.8
## 2 Alaska  Northwest Arcti~  16.5    10.4     0.4  46.9         21.2          4.6
## 3 Alaska  Aleutians East ~  18.4     4.9     0.5  71.2          2.2          2.8
## 4 New Yo~ Kings             18.6     4.4    61.7   8.8          2.5          3.9
## 5 Alaska  North Slope Bor~  20.1    17       2.8  37.9          7.9         14.3
## 6 Alaska  Lake and Penins~  21.2     6.8     1.1  36.2         32.4          2.4
```

Interesting insights : Alaska – Drive to Work maximum while New York – Transit to Work (Sub-way)

Select Helpers :

Select has other ways to get only the columns you want using "select helpers": functions that specify criteria for choosing columns

The following functions helps you to select variables based on their names

Helpers	Description
starts_with()	Starts with a prefix
ends_with()	Ends with a prefix
contains()	Contains a literal string
matches()	Matches a regular expression
num_range()	Numerical range like x01, x02, x03.
one_of()	Variables in character vector.
everything()	All variables.

Let us select all the columns that contain"work"

```r
mydata %>%
    select(state , county , contains("work")) %>%
    head()
```

```
## # A tibble: 6 x 6
##   state   county  work_at_home private_work public_work family_work
##   <chr>   <chr>          <dbl>        <dbl>       <dbl>       <dbl>
## 1 Alabama Autauga          1.8         73.6        20.9         0
## 2 Alabama Baldwin          3.9         81.5        12.3         0.4
## 3 Alabama Barbour          1.6         71.8        20.8         0.1
## 4 Alabama Bibb             0.7         76.8        16.1         0.4
## 5 Alabama Blount           2.3         82          13.5         0.4
## 6 Alabama Bullock          2.8         79.5        15.1         0
```

Notice that we put work into quotes, unlike state and county. Select helpers take **strings,** which means they're in quotes. The result has all the columns that contain the word "work".

Let us get all the columns that begin with the word "income", which are generally related to each other

```r
mydata %>%
    select(state , county , starts_with("income")) %>%
    head()
```

```
## # A tibble: 6 x 6
##   state   county  income income_err income_per_cap income_per_cap_err
##   <chr>   <chr>    <dbl>      <dbl>          <dbl>              <dbl>
## 1 Alabama Autauga  51281       2391          24974               1080
## 2 Alabama Baldwin  50254       1263          27317                711
## 3 Alabama Barbour  32964       2973          16824                798
## 4 Alabama Bibb     38678       3995          18431               1618
## 5 Alabama Blount   45813       3141          20532                708
## 6 Alabama Bullock  31938       5884          17580               2055
```

De-Selecting / Removing Columns

We can use select to remove variables from a table by adding a "minus" in front of the column name

Let us exclude the column census_id

```r
mydata %>%
    select(-census_id) %>%
    names()
```

```
##  [1] "X1"                 "state"              "county"
##  [4] "region"             "metro"              "population"
##  [7] "men"                "women"              "hispanic"
## [10] "white"              "black"              "native"
## [13] "asian"              "pacific"            "citizens"
## [16] "income"             "income_err"         "income_per_cap"
## [19] "income_per_cap_err" "poverty"            "child_poverty"
## [22] "professional"       "service"            "office"
## [25] "construction"       "production"         "drive"
```

```
## [28] "carpool"          "transit"          "walk"
## [31] "other_transp"     "work_at_home"     "mean_commute"
## [34] "employed"         "private_work"     "public_work"
## [37] "self_employed"    "family_work"      "unemployment"
## [40] "land_area"
```