Chapter_3_Cleaning_Data_With_R

Ravi Mummigatti

7/7/2021

Contents

Introduction

A significant part of data science involves acquiring raw data and getting it into a form ready for analysis. It is estimated that data scientists spend 80% of their time cleaning and manipulating data, and only 20% of their time actually analyzing it or building models from it.

When we receive raw data, we have to do a number of things before we're ready to analyze it, possibly including:

- diagnosing the "tidiness" of the data — how much data cleaning we will have to do
- reshaping the data — getting the right rows and columns for effective analysis
- combining multiple files
- changing the types of values — how we fix a column where numerical values are stored as strings, for example
- dropping or filling missing values - how we deal with data that is incomplete or missing
- manipulating strings to represent the data better

We will go through the techniques data scientists use to accomplish these goals by looking at some "unclean" datasets and trying to get them into a good, clean state. Along the way we will use the powerful tidyverse packages dplyr and tidyr to get our data squeaky clean!

We have been provided an example of data representing exam scores from `1000` students in an online math class.

These data frames, which you can view in the rendered notebook, are hard to work with. They're separated into multiple tables, and the values don't lend themselves well to analysis. We would like to plot the exam score average against the age of the students in the class , which is not an easy task with given data.

In the ensuing exercises, we will transform this data (given in 10 csv files) so that performing a visualization would be simple

```
# load libraries
library(tidyverse)
```

Let us first look at two data sets "exams_0" and "exams_1"

```
# load data frame
students_1 <- read_csv('exams_0.csv')
students_2 <- read_csv('exams_1.csv')
```

```
# inspect data frame
head(students_1)
```

```
## # A tibble: 6 x 6
##      id full_name         gender_age fractions probability grade
##   <dbl> <chr>             <chr>      <chr>     <chr>       <dbl>
## 1     0 Moses Kirckman    M14        69%       89%            11
## 2     1 Timofei Strowan   M18        63%       76%            11
## 3     2 Silvain Poll      M18        69%       77%             9
## 4     3 Lezley Pinxton    M18        <NA>      72%            11
## 5     4 Bernadene Saunper F17        72%       84%            11
## 6     5 Eldin Spitell     M16        <NA>      83%            11
```

```
head(students_2)
```

```
## # A tibble: 6 x 6
##      id full_name             gender_age fractions probability grade
##   <dbl> <chr>                 <chr>      <chr>     <chr>       <dbl>
## 1     0 Morissa Skade         F16        72%       70%            10
```

```
## 2      1 Jasper Comfort         M14         76%        73%              9
## 3      2 Siana Pallas           F18         58%        77%             10
## 4      3 Aldous Peele           M16         86%        83%              9
## 5      4 Ethelred Easun         M14         85%        <NA>            12
## 6      5 Goldarina Championnet  F14         83%        85%             10
```

Diagnose the Data

We often describe data that is easy to analyze and visualize as "tidy data". What does it mean to have tidy data?

For data to be tidy, it must have:

- Each variable as a separate column
- Each row as a separate observation

For example, we would want to reshape a table like:

Account	Checkings	Savings
"12456543"	8500	8900
"12283942"	6410	8020
"12839485"	78000	92000

Into a table that looks more like:

Account	Account Type	Amount
"12456543"	"Checking"	8500
"12456543"	"Savings"	8900
"12283942"	"Checking"	6410
"12283942"	"Savings"	8020
"12839485"	"Checking"	78000
"12839485"	"Savings"	920000

The first step of diagnosing whether or not a dataset is tidy is using base R and dplyr functions to explore/probe the dataset.

You've seen most of the functions we often use to diagnose a dataset for cleaning. Some of the most useful ones are:

- `head()` — display the first 6 rows of the table
- `summary()` — display the summary statistics of the table
- `colnames()` — display the column names of the table

We have been provided two data frames, `grocery_1` and `grocery_2`.

1. Begin by viewing the `head() , summary() and colnames()` of both `grocery_1` and `grocery_2`.
2. Which data frame is "clean", tidy, and ready for analysis?

```r
# load the data sets
grocery_1 <- read_csv("grocery_1.csv")
grocery_2 <- read_csv("grocery_2.csv")
```

```r
# top rows of grocery_1 and grocery_2
head(grocery_1)
```

```
## # A tibble: 3 x 4
##   `Grocery Item` `Cake Recipe` `Pancake Recipe` `Cookie Recipe`
##   <chr>                  <dbl>            <dbl>           <dbl>
## 1 Eggs                       2                3               1
## 2 Milk                       1                2               1
## 3 Flour                      2                1               2
```

```r
head(grocery_2)
```

```
## # A tibble: 6 x 3
##   `Grocery Item` Recipe         Number
##   <chr>          <chr>           <dbl>
## 1 Eggs           Cake Recipe         2
## 2 Milk           Cake Recipe         1
## 3 Flour          Cake Recipe         2
## 4 Eggs           Pancake Recipe      3
## 5 Milk           Pancake Recipe      2
## 6 Flour          Pancake Recipe      1
```

```r
# summary of grocery_1 and grocery_2
summary(grocery_1)
```

```
##  Grocery Item        Cake Recipe    Pancake Recipe Cookie Recipe
##  Length:3           Min.   :1.000   Min.   :1.0    Min.   :1.000
##  Class :character   1st Qu.:1.500   1st Qu.:1.5    1st Qu.:1.000
##  Mode  :character   Median :2.000   Median :2.0    Median :1.000
##                     Mean   :1.667   Mean   :2.0    Mean   :1.333
##                     3rd Qu.:2.000   3rd Qu.:2.5    3rd Qu.:1.500
##                     Max.   :2.000   Max.   :3.0    Max.   :2.000
```

```r
summary(grocery_2)
```

```
##  Grocery Item          Recipe              Number
##  Length:9           Length:9           Min.   :1.000
##  Class :character   Class :character   1st Qu.:1.000
##  Mode  :character   Mode  :character   Median :2.000
##                                        Mean   :1.667
##                                        3rd Qu.:2.000
##                                        Max.   :3.000
```

```r
# column names of grocery_1 and grocery_2
print(colnames(grocery_1))
```

```
## [1] "Grocery Item"   "Cake Recipe"    "Pancake Recipe" "Cookie Recipe"
```

```
print(colnames(grocery_2))
```

```
## [1] "Grocery Item" "Recipe"       "Number"
```

Looking at the two data sets it is clear that "grocery_2" follows the tidy format wherein each variable has a separate column and each row is a separate observation

Dealing with Multiple Files

Often, we have the same data separated out into multiple files.Let's say that you have a ton of files following the filename structure: `'file_1.csv'`, `'file_2.csv'`, `'file_3.csv'`, and so on. The power of dplyr and tidyr is mainly in being able to manipulate large amounts of structured data, so you want to be able to get all of the relevant information into one table so that you can analyze the aggregate data.

You can combine the base R functions `list.files()` and `lapply()` with readr and dplyr to organize this data better:

```
files <- list.files(pattern = "file_.*csv")
df_list <- lapply(files,read_csv)
df <- bind_rows(df_list)
```

- The first line uses `list.files()` and a regular expression, a sequence of characters describing a pattern of text that should be matched, to find any file in the current directory that starts with `'file_'` and has an extension of `csv`, storing the name of each file in a vector `files`
- The second line uses `lapply()` to read each file in `files` into a data frame with `read_csv()`, storing the data frames in `df_list`
- The third line then concatenates all of those data frames together with dplyr's `bind_rows()` function

You have 10 different files containing 100 students each. These files follow the naming structure:

- `exams_0.csv ; exams_1.csv ;`... up to `exams_9.csv`

You are going to read each file into an individual data frame and then combine all of the entries into one data frame.

1. First, create a variable called `student_files` and set it equal to the `list.files()` of all of the CSV files we want to import.

```
# create a list of files with a pattern
student_files = list.files(pattern = "exams_.*csv")

# print the list of files to download
print(student_files)
```

```
##  [1] "exams_0.csv" "exams_1.csv" "exams_2.csv" "exams_3.csv" "exams_4.csv"
##  [6] "exams_5.csv" "exams_6.csv" "exams_7.csv" "exams_8.csv" "exams_9.csv"
```

2. Read each file in `student_files` into a data frame using `lapply()` and save the result to `df_list`.

```
# read each file into a data frame using lappy method
df_list <- lapply(student_files,read_csv)
```

3. Concatenate all of the data frames in `df_list` into one data frame called `students`.
4. Inspect `students`. Save the number of rows in `students` to `nrow_students`

```
# combine each data frame by rows i.e. append one after the other "bind_rows"
df <- bind_rows(df_list)

# inspect
df
```

```
## # A tibble: 1,000 x 6
##       id full_name         gender_age fractions probability grade
##    <dbl> <chr>             <chr>      <chr>     <chr>       <dbl>
##  1     0 Moses Kirckman    M14        69%       89%            11
##  2     1 Timofei Strowan   M18        63%       76%            11
##  3     2 Silvain Poll      M18        69%       77%             9
##  4     3 Lezley Pinxton    M18        <NA>      72%            11
##  5     4 Bernadene Saunper F17        72%       84%            11
##  6     5 Eldin Spitell     M16        <NA>      83%            11
##  7     6 Christi Lesser    F17        86%       84%             9
##  8     7 Papageno Rummin   M17        81%       77%            11
##  9     8 Nissa Wrotchford  F18        68%       75%            12
## 10     9 Vincent Blumer    M14        59%       <NA>           11
## # ... with 990 more rows
```

```
# print number of rows and number of columns
print(nrow(df))
```

```
## [1] 1000
```

```
print(ncol(df))
```

```
## [1] 6
```

Reshaping your Data

Since we want

- Each variable as a separate column
- Each row as a separate observation

We would want to reshape a table like:

Account	*Checking*	*Savings*
"12456543"	8500	8900
"12283942"	6410	8020
"12839485"	78000	92000

Into a table that looks more like:

Account	*Account Type*	*Amount*
"12456543"	"Checking"	8500
"12456543"	"Savings"	8900
"12283942"	"Checking"	6410
"12283942"	"Savings"	8020
"12839485"	"Checking"	78000
"12839485"	"Savings"	920000

We can use tidyr's `gather()` function to do this transformation. `gather()` takes a data frame and the columns to unpack:

```
df %>%
  gather('Checking','Savings',key='Account Type',value='Amount')
```

The arguments you provide are:

- `df`: the data frame you want to gather, which can be piped into `gather()`
- `Checking` and `Savings`: the columns of the old data frame that you want to turn into variables
- `key`: what to call the column of the new data frame that stores the variables
- `value`: what to call the column of the new data frame that stores the values

We will now re-shape our student marks data frame.

There is a column for the scores on the `fractions` exam, and a column for the scores on the `probability` exam.

We want to make each row an observation, so we want to transform this table to look like:

full_name	*exam*	*score*	*gender_age*	*grade*
"First Student"	"fractions"	score%	...	...
"First Student"	"probability"	score%	...	...
"Second Student"	"fractions"	score%	...	...
"Second Student"	"probability"	score%	...	...
...	...	...	...	...

- Use `gather` to create a new table (still called `students`) that follows this structure. Then view the `head()` of students.

```
# print the original column names before reshaping
original_col_names <- colnames(df)
print(original_col_names)
```

```
## [1] "id"          "full_name"   "gender_age"  "fractions"   "probability"
## [6] "grade"
```

```
# gather the "fractions" and "probability" columns
# new column name for the gathered column will be stored in "exam"
# new values of the gathered columns will be stored in "score"
df <- df %>%
    gather("fractions" , "probability" ,
           key = "exam" ,
           value = "score")
# inspect
head(df)
```

```
## # A tibble: 6 x 6
##      id full_name         gender_age grade exam      score
##   <dbl> <chr>             <chr>      <dbl> <chr>     <chr>
## 1     0 Moses Kirckman    M14           11 fractions 69%
## 2     1 Timofei Strowan   M18           11 fractions 63%
## 3     2 Silvain Poll      M18            9 fractions 69%
## 4     3 Lezley Pinxton    M18           11 fractions <NA>
## 5     4 Bernadene Saunper F17           11 fractions 72%
## 6     5 Eldin Spitell     M16           11 fractions <NA>
```

- Save the columns names of the updated `students` data frame to `gathered_col_names` and print it.

```
gathered_col_names <- colnames(df)
print(gathered_col_names)
```

```
## [1] "id"         "full_name"  "gender_age" "grade"      "exam"
## [6] "score"
```

- The dplyr function `count()` takes a data frame and a column as arguments and returns a table with counts of the unique values in the named column.Find the count of each unique value in the `exam` column. Save the result to `exam_counts` and view `exam_counts`.

```
# how manu students took each exam
exam_count <- df %>%
    count(exam)

# print the result
exam_count
```

```
## # A tibble: 2 x 2
##   exam            n
##   <chr>       <int>
## 1 fractions    1000
## 2 probability  1000
```

Dealing with Duplicates

Often we see duplicated rows of data in the data frames we are working with. This could happen due to errors in data collection or in saving and loading the data.

To check for duplicates, we can use the base R function `duplicated()`, which will return a logical vector telling us which rows are duplicate rows.Let's say we have a data frame `fruits` that represents this table:

item	*price*	*calories*
"banana"	"$1"	105
"apple"	"$0.75"	95
"apple"	"$0.75"	95
"peach"	"$3"	55
"peach"	"$4"	55
"clementine"	"$2.5"	35

If we call `fruits %>% duplicated()`, we would get the following vector:

```
>> [1] FALSE FALSE TRUE FALSE FALSE FALSE
```

We can see that the third row, which represents an `"apple"` with price `"$0.75"` and `95` calories, is a duplicate row. Every value in this row is the same as in another row (the previous row).

We can use the dplyr `distinct()` function to remove all rows of a data frame that are duplicates of another row.

If we call `fruits %>% distinct()`, we would get the table:

item	*price*	*calories*
"banana"	"$1"	105
"apple"	"$0.75"	95
"peach"	"$3"	55
"peach"	"$4"	55
"clementine"	"$2.5"	35

The `"apple"` row was deleted because it was exactly the same as another row. But the two `"peach"` rows remain because there is a difference in the *price* column.

If we wanted to remove every row with a duplicate value in the *item* column, we could specify a `subset`:

```
fruits %>%
  distinct(item,.keep_all=TRUE)
```

- The `students` data frame has a column `id` that is neither unique nor required for our analysis. Drop the `id` column from the data frame and save the result to `students`. View the `head()` of `students`

```r
# drop the id column
df_new <- df %>%
    select(-id)

# inspeact new data frame
head(df_new)
```

```
## # A tibble: 6 x 5
##   full_name          gender_age grade exam      score
##   <chr>              <chr>      <dbl> <chr>     <chr>
```

```
## 1 Moses Kirckman    M14           11 fractions 69%
## 2 Timofei Strowan   M18           11 fractions 63%
## 3 Silvain Poll      M18            9 fractions 69%
## 4 Lezley Pinxton    M18           11 fractions <NA>
## 5 Bernadene Saunper F17           11 fractions 72%
## 6 Eldin Spitell     M16           11 fractions <NA>
```

- It seems like in the data collection process, some rows may have been recorded twice. Use the `duplicated()` function on the `students` data frame to make a vector object called `duplicates`.
- `table()` is a base R function that takes any R object as an argument and returns a table with the counts of each unique value in the object.Pipe the result from the previous checkpoint into `table()` to see how many rows are exact duplicates. Make sure to save the result to `duplicates`, and view `duplicates`.

```
# find and count duplicated rows
duplicate_df <- df_new %>%
  duplicated() %>%
    table()
duplicate_df
```

```
## .
## FALSE  TRUE
##  1976    24
```

There are 24 duplicate values.Update the value of `students` to be the `students` data frame with only unique/distinct rows

```
# use the distinct() method to remove duplicate values
df_final <- df_new %>%
  distinct()
```

Use the `duplicated()` function again to make an object called `df_analysis` after dropping the duplicates. Pipe the result into `table()` to see if any duplicates remain, and view `df_analysis`. Are there any `TRUE`s left?

```
# final dataframe putting it all together
students <- df %>%
    # remove id column
    select(-id) %>%
    # use the distinct() method to remove duplicate values
    distinct()

# check for duplicate values
students %>%
  duplicated() %>%
    table()
```

```
## .
## FALSE
##  1976
```

There are no duplicate values. We can now use the “df_analysis” which is in tidy form.

Splitting By Index

In trying to get clean data, we want to make sure each column represents one type of measurement. Often, multiple measurements are recorded in the same column, and we want to separate these out so that we can do individual analysis on each variable.

Let's say we have a column "birthday" with data formatted in MMDDYYYY format. In other words, "11011993" represents a birthday of November 1, 1993. We want to split this data into day, month, and year so that we can use these columns as separate features.

In this case, we know the exact structure of these strings. The first two characters will always correspond to the month, the second two to the day, and the rest of the string will always correspond to year. We can easily break the data into three separate columns by splitting the strings into substrings using `str_sub()`, a helpful function from the stringr package:

```
# Create the 'month' column
df %>%
  mutate(month = str_sub(birthday,1,2))

# Create the 'day' column
df %>%
  mutate(day = str_sub(birthday,3,4))

# Create the 'year' column
df %>%
  mutate(year = str_sub(birthday,5))
```

- The first command takes the characters starting at index `1` and ending at index `2` of each value in the `birthday` column and puts it into a `month` column.
- The second command takes the characters starting at index `3` and ending at index `4` of each value in the `birthday` column and puts it into a `day` column.
- The third command takes the characters starting at index `5` and ending at the end of the value in the `birthday` column and puts it into a `year` column.

This would transform a table like:

Table 8: into a table like:

id	*birthday*
1011	"12241989"
1112	"10311966"
1113	"01052011"

id	*birthday*	*month*	*day*	*year*
1011	"12241989"	"12"	"24"	"1989"
1112	"10311966"	"10"	"31"	"1966"
1113	"01052011"	"01"	"05"	"2011"

- Print out the columns of the `students` data frame.

```
colnames(students)
```

```
## [1] "full_name"  "gender_age" "grade"      "exam"       "score"
```

- The column `gender_age` sounds like it contains both `gender` and `age`! View the `head()` of `students` to see what kind of data `gender_age` contains.

```
head(students)
```

```
## # A tibble: 6 x 5
##   full_name         gender_age grade exam      score
##   <chr>             <chr>      <dbl> <chr>     <chr>
## 1 Moses Kirckman    M14           11 fractions 69%
## 2 Timofei Strowan   M18           11 fractions 63%
## 3 Silvain Poll      M18            9 fractions 69%
## 4 Lezley Pinxton    M18           11 fractions <NA>
## 5 Bernadene Saunper F17           11 fractions 72%
## 6 Eldin Spitell     M16           11 fractions <NA>
```

- It looks like the first character of the values in `gender_age` contains the gender, while the rest of the string contains the age. Let's separate out the gender data into a new column called `gender`. Save the result to `students`, and view the `head()`.
- We don't need that `gender_age` column anymore. Drop `gender_age` from `students`, and save the result to `students`. View the `head()` of `students`

```
students <- students %>%
    # separate the gender which starts at 1st index and ends at 1st index
    mutate(gender = str_sub(gender_age , 1,1)) %>%
    # separate the age which starts at 2nd index
    mutate(age = str_sub(gender_age , 2)) %>%
    # drop gender_age column
    select(-gender_age)

head(students)
```

```
## # A tibble: 6 x 6
##   full_name         grade exam      score gender age
##   <chr>             <dbl> <chr>     <chr> <chr>  <chr>
## 1 Moses Kirckman       11 fractions 69%   M      14
## 2 Timofei Strowan      11 fractions 63%   M      18
## 3 Silvain Poll          9 fractions 69%   M      18
## 4 Lezley Pinxton       11 fractions <NA>  M      18
## 5 Bernadene Saunper    11 fractions 72%   F      17
## 6 Eldin Spitell        11 fractions <NA>  M      16
```

Splitting By Character

Let's say we have a column called `"type"` with data entries in the format `"admin_US"` or `"user_Kenya"`, as shown in the table below.

id	*type*
1011	"user_Kenya"
1112	"admin_US"
1113	"moderator_UK"

Just like we saw before, this column actually contains two types of data. One seems to be the user type (with values like "admin" or "user") and one seems to be the country this user is in (with values like "US" or "Kenya").

We can no longer just split along the first 4 characters because `admin` and `user` are of different lengths. Instead, we know that we want to split along the `"_"`. We can thus use the tidyr function `separate()` to split this column into two, separate columns:

```
# Create the 'user_type' and 'country' columns
df %>%
  separate(type,c('user_type','country'),'_')
```

- `type` is the column to split
- `c('user_type','country')` is a vector with the names of the two new columns
- `'_'` is the character to split on

This would transform the table above into a table like:

id	*type*	*country*	*usertype*
1011	"user_Kenya"	"Kenya"	"user"
1112	"admin_US"	"US"	"admin"
1113	"moderator_UK"	"UK"	"moderator"

```
head(students)
```

```
## # A tibble: 6 x 6
##   full_name         grade exam      score gender age
##   <chr>             <dbl> <chr>     <chr> <chr>  <chr>
## 1 Moses Kirckman       11 fractions 69%   M      14
## 2 Timofei Strowan      11 fractions 63%   M      18
## 3 Silvain Poll          9 fractions 69%   M      18
## 4 Lezley Pinxton       11 fractions <NA>  M      18
## 5 Bernadene Saunper    11 fractions 72%   F      17
## 6 Eldin Spitell        11 fractions <NA>  M      16
```

Notice that the students' names are stored in a column called `full_name`.

- Separate the `full_name` column into two new columns, `first_name` and `last_name`, by splitting on the `' '` character .
- Provide as an extra argument to the `separate()` function `extra ='merge'`. This will ensure that middle names or two-word last names will all end up in the `last_name` column.
- Save the result to `students`, and view the `head()`.

```r
students <- students %>%
    # separate into first name and last name
    separate(full_name , c("first_name" , "last_name"),
    # separator is "space"
             sep = ' ' ,
             extra = 'merge')
head(students)
```

```
## # A tibble: 6 x 7
##   first_name last_name grade exam      score gender age
##   <chr>      <chr>     <dbl> <chr>     <chr> <chr>  <chr>
## 1 Moses      Kirckman     11 fractions 69%   M      14
## 2 Timofei    Strowan      11 fractions 63%   M      18
## 3 Silvain    Poll          9 fractions 69%   M      18
## 4 Lezley     Pinxton      11 fractions <NA>  M      18
## 5 Bernadene  Saunper      11 fractions 72%   F      17
## 6 Eldin      Spitell      11 fractions <NA>  M      16
```

Looking at Data Types

Each column of a data frame can hold items of the same *data type*. The data types that R uses are: character, numeric (real or decimal), integer, logical, or complex. Often, we want to convert between types so that we can do better analysis. If a numerical category like `"num_users"` is stored as a vector of `character`s instead of `numeric`s, for example, it makes it more difficult to do something like make a line graph of users over time.

To see the types of each column of a data frame, we can use:

```r
str(df)
```

`str()` displays the internal structure of an R object. Calling `str()` with a data frame as an argument will return a variety of information, including the data types. For a data frame like this:

item	*price*	*calories*
"banana"	"$1"	105
"apple"	"$0.75"	95
"peach"	"$3"	55
"clementine"	"$2.5"	35

the data types would be:

```
#> $ item:        chr
#> $ price:       chr
#> $ calories:    num
```

We can see that the `price` column is made up of `character`s, which will probably make our analysis of price more difficult

Let's inspect the data types in the `students` table. by printing out the structure of `students`.

```r
# data structure Base R
str(students)
```

```
## tibble[,7] [1,976 x 7] (S3: tbl_df/tbl/data.frame)
##  $ first_name: chr [1:1976] "Moses" "Timofei" "Silvain" "Lezley" ...
##  $ last_name : chr [1:1976] "Kirckman" "Strowan" "Poll" "Pinxton" ...
##  $ grade     : num [1:1976] 11 11 9 11 11 11 9 11 12 11 ...
##  $ exam      : chr [1:1976] "fractions" "fractions" "fractions" "fractions" ...
##  $ score     : chr [1:1976] "69%" "63%" "69%" NA ...
##  $ gender    : chr [1:1976] "M" "M" "M" "M" ...
##  $ age       : chr [1:1976] "14" "18" "18" "18" ...
```

If we wanted to make a scatterplot of `age` vs average exam score, would we be able to do it with this type of data?

Running the code below will give us an error since "age" is non-numeric data type

```r
students %>%
    summarise(mean(age))
```

```
## Warning in mean.default(age): argument is not numeric or logical: returning NA
```

```
## # A tibble: 1 x 1
##   `mean(age)`
##         <dbl>
## 1          NA
```

String Parsing

Sometimes we need to modify strings in our data frames to help us transform them into more meaningful metrics. For example, in our fruits table from before:

item	*price*	*calories*
"banana"	"$1"	105
"apple"	"$0.75"	95
"peach"	"$3"	55
"peach"	"$4"	55
"clementine"	"$2.5"	35

We can see that the `'price'` column is actually composed of character strings representing dollar amounts. This column could be much better represented as numeric, so that we could take the mean, calculate other aggregate statistics, or compare different fruits to one another in terms of price.

First, we can use a regular expression, a sequence of characters that describe a pattern of text to be matched, to remove all of the dollar signs. The base R function `gsub()` will remove the `$` from the `price` column, replacing the symbol with an empty string `''`:

```r
fruit %>%
  mutate(price=gsub('\\$','',price))
```

Then, we can use the base R function `as.numeric()` to convert character strings containing numerical values to numeric:

```
fruit %>%
  mutate(price = as.numeric(price))
```

Now, we have a data frame that looks like:

item	*price*	*calories*
"banana"	1	105
"apple"	0.75	95
"peach"	3	55
"peach"	4	55
"clementine"	2.5	35

We saw in the last exercise that finding the mean of the `score` column is hard to do when the data is stored as `characters` and not numbers. Let us View the `head()` of `students` to take a look at the values in the `score` column.

```
# top 6 rows
head(students)
```

```
## # A tibble: 6 x 7
##   first_name last_name grade exam      score gender age
##   <chr>      <chr>     <dbl> <chr>     <chr> <chr>  <chr>
## 1 Moses      Kirckman     11 fractions 69%   M      14
## 2 Timofei    Strowan      11 fractions 63%   M      18
## 3 Silvain    Poll          9 fractions 69%   M      18
## 4 Lezley     Pinxton      11 fractions <NA>  M      18
## 5 Bernadene  Saunper      11 fractions 72%   F      17
## 6 Eldin      Spitell      11 fractions <NA>  M      16
```

Remove the `'%'` symbol from the `score` column, and save the resulting data frame to `students`. View `students`.

```
# remove % sign using gsub() from Base R
students <- students %>%
    mutate(score = gsub('\\%' , '' , score))

head(students)
```

```
## # A tibble: 6 x 7
##   first_name last_name grade exam      score gender age
##   <chr>      <chr>     <dbl> <chr>     <chr> <chr>  <chr>
## 1 Moses      Kirckman     11 fractions 69    M      14
## 2 Timofei    Strowan      11 fractions 63    M      18
## 3 Silvain    Poll          9 fractions 69    M      18
## 4 Lezley     Pinxton      11 fractions <NA>  M      18
## 5 Bernadene  Saunper      11 fractions 72    F      17
## 6 Eldin      Spitell      11 fractions <NA>  M      16
```

Convert the `score` column to a numerical type using the `as.numeric()` function. Save this new data frame to `students`, and view it

```
# convert score from character to numeric
students <- students %>%
    mutate(score = as.numeric(score))

head(students)
```

```
## # A tibble: 6 x 7
##   first_name last_name grade exam      score gender age
##   <chr>      <chr>     <dbl> <chr>     <dbl> <chr>  <chr>
## 1 Moses      Kirckman     11 fractions    69 M      14
## 2 Timofei    Strowan      11 fractions    63 M      18
## 3 Silvain    Poll          9 fractions    69 M      18
## 4 Lezley     Pinxton      11 fractions    NA M      18
## 5 Bernadene  Saunper      11 fractions    72 F      17
## 6 Eldin      Spitell      11 fractions    NA M      16
```

Convert the `age` column to a numerical type using the `as.numeric()` function into `students`, and view it

```
# convert age to numeric
students <- students%>%
    mutate(age = as.numeric(age))

# view
str(students)
```

```
## tibble[,7] [1,976 x 7] (S3: tbl_df/tbl/data.frame)
##  $ first_name: chr [1:1976] "Moses" "Timofei" "Silvain" "Lezley" ...
##  $ last_name : chr [1:1976] "Kirckman" "Strowan" "Poll" "Pinxton" ...
##  $ grade     : num [1:1976] 11 11 9 11 11 11 9 11 12 11 ...
##  $ exam      : chr [1:1976] "fractions" "fractions" "fractions" "fractions" ...
##  $ score     : num [1:1976] 69 63 69 NA 72 NA 86 81 68 59 ...
##  $ gender    : chr [1:1976] "M" "M" "M" "M" ...
##  $ age       : num [1:1976] 14 18 18 18 17 16 17 17 18 14 ...
```

Cleaning US Census Data

You just got hired as a Data Analyst at the Census Bureau, which collects census data and finds interesting insights from it.

The person who previously had your job left you all the data they had for the most recent census. The data is spread across multiple `csv` files. They didn't use R, and they would manually look through these `csv` files whenever they wanted to find something. Sometimes they would copy and paste certain numbers into Excel for analysis.This is not scalable or repeatable for you to dig into the data and find some insights by the end of the day.

We have been provided 9 csv files containing census data across states of the US.We will review these files and clean the data applying various data cleaning methods we have learnt so far.

Load and Inspect the data

- Load the desired libraries for Data Cleaning and review a few files

```
# load required libraries for data cleaning
library(readr)
library(tidyr)
library(dplyr)
library(stringr)
```

```
# load 2 files into a data frame
states_0 <- read_csv("states_0.csv")
states_1 <- read_csv("states_1.csv")

# inspect head
head(states_0)
```

```
## # A tibble: 6 x 11
##      X1 State     TotalPop Hispanic White  Black  Native Asian Pacific Income
##   <dbl> <chr>        <dbl> <chr>    <chr>  <chr>  <chr>  <chr> <chr>   <chr>
## 1     0 Alabama    4830620 3.75%    61.88% 31.25% 0.45%  1.05% 0.03%   $43,296.~
## 2     1 Alaska      733375 5.91%    60.91% 2.85%  16.39% 5.45% 1.06%   $70,354.~
## 3     2 Arizona    6641928 29.57%   57.12% 3.85%  4.36%  2.88% 0.17%   $54,207.~
## 4     3 Arkansas   2958208 6.22%    71.14% 18.97% 0.52%  1.14% 0.15%   $41,935.~
## 5     4 Californ~ 38421464 37.29%   40.22% 5.68%  0.41%  13.0~ 0.35%   $67,264.~
## 6     5 Colorado   5278906 20.78%   69.90% 3.55%  0.57%  2.66% 0.12%   $64,657.~
## # ... with 1 more variable: GenderPop <chr>
```

```
head(states_1)
```

```
## # A tibble: 6 x 11
##      X1 State        TotalPop Hispanic White  Black Native Asian Pacific Income
##   <dbl> <chr>           <dbl> <chr>    <chr>  <chr> <chr>  <chr> <chr>   <chr>
## 1     0 Colorado      5278906 20.78%   69.90% 3.55% 0.57%  2.66% 0.12%   $64,65~
## 2     1 Connecticut   3593222 15.60%   67.68% 10.3~ 0.13%  4.02% 0.02%   $76,14~
## 3     2 Delaware       926454 8.82%    64.63% 20.7~ 0.26%  3.27% 0.02%   $61,82~
## 4     3 District of~   647484 9.17%    33.10% 51.7~ 0.20%  3.38% 0.03%   $75,46~
## 5     4 Florida      19645772 21.34%   59.08% 15.1~ 0.21%  2.28% 0.05%   $50,69~
## 6     5 Georgia      10006693 8.42%    54.29% 32.0~ 0.19%  3.10% 0.05%   $50,81~
## # ... with 1 more variable: GenderPop <chr>
```

It will be easier to inspect the data stored in these files once you have it in a data frame.

- Let us begin by creating a variable called `files` and set it equal to the `list.files()` of all of the `csv` files to import.
- Read each file in `files` into a data frame using `lapply()` and save the result to `df_list`
- Concatenate all of the data frames in `df_list` into one data frame called `us_census`

```r
# create a list of files with a pattern
list_files = list.files(pattern = "states_.*csv")

# print the list of files to download
print(list_files)
```

```
##  [1] "states_0.csv" "states_1.csv" "states_2.csv" "states_3.csv" "states_4.csv"
##  [6] "states_5.csv" "states_6.csv" "states_7.csv" "states_8.csv" "states_9.csv"
```

```r
# read each file into a data frame using lappy method
df_list <- lapply(list_files,read_csv)

# combine each data frame by rows i.e. append one after the other "bind_rows"
us_census <- bind_rows(df_list)
```

Inspect the `us_census` data frame by printing the column names, looking at the data types with `str()`, and viewing the `head()`.

- What columns have symbols that will prevent calculations?

{Answer : Hispanic , White , Black , Native , Asian , Pacific , Income and GenderPop}

- What are the data types of the columns?

{Answer : Except X1 and Total Pop which are "Numeric" , all other columns have "Non-Numeric" Data Types

- Do any columns contain multiple kinds of information?

{Answer : Column Gender_prop contains Numeric and Text Mixed Data}

```r
str(us_census)
```

```
## spec_tbl_df[,11] [61 x 11] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ X1       : num [1:61] 0 1 2 3 4 5 0 1 2 3 ...
##  $ State    : chr [1:61] "Alabama" "Alaska" "Arizona" "Arkansas" ...
##  $ TotalPop : num [1:61] 4830620 733375 6641928 2958208 38421464 ...
##  $ Hispanic : chr [1:61] "3.75%" "5.91%" "29.57%" "6.22%" ...
##  $ White    : chr [1:61] "61.88%" "60.91%" "57.12%" "71.14%" ...
##  $ Black    : chr [1:61] "31.25%" "2.85%" "3.85%" "18.97%" ...
##  $ Native   : chr [1:61] "0.45%" "16.39%" "4.36%" "0.52%" ...
##  $ Asian    : chr [1:61] "1.05%" "5.45%" "2.88%" "1.14%" ...
##  $ Pacific  : chr [1:61] "0.03%" "1.06%" "0.17%" "0.15%" ...
##  $ Income   : chr [1:61] "$43,296.36" "$70,354.74" "$54,207.82" "$41,935.63" ...
##  $ GenderPop: chr [1:61] "2341093M_2489527F" "384160M_349215F" "3299088M_3342840F" "1451913M_1506295
##  - attr(*, "spec")=
##   .. cols(
##   ..   X1 = col_double(),
##   ..   State = col_character(),
##   ..   TotalPop = col_double(),
```

```
##    ..   Hispanic = col_character(),
##    ..   White = col_character(),
##    ..   Black = col_character(),
##    ..   Native = col_character(),
##    ..   Asian = col_character(),
##    ..   Pacific = col_character(),
##    ..   Income = col_character(),
##    ..   GenderPop = col_character()
##    .. )
```

Remove and Reformat Columns

- When inspecting `us_census` you notice a column `X1` that stores meaningless information.
- Drop the `X1` column from `us_census`, and save the resulting data frame to `us_census`. View the head of `us_census`.

```
# drop column X1
us_census <- us_census %>%
    select(-X1)

# inspect
head(us_census)
```

```
## # A tibble: 6 x 10
##   State   TotalPop Hispanic White  Black Native Asian Pacific Income GenderPop
##   <chr>      <dbl> <chr>    <chr>  <chr> <chr>  <chr> <chr>   <chr>  <chr>
## 1 Alabama  4830620 3.75%    61.88% 31.2~ 0.45%  1.05% 0.03%   $43,2~ 2341093M_2~
## 2 Alaska    733375 5.91%    60.91% 2.85% 16.39% 5.45% 1.06%   $70,3~ 384160M_34~
## 3 Arizona  6641928 29.57%   57.12% 3.85% 4.36%  2.88% 0.17%   $54,2~ 3299088M_3~
## 4 Arkans~  2958208 6.22%    71.14% 18.9~ 0.52%  1.14% 0.15%   $41,9~ 1451913M_1~
## 5 Califo~ 38421464 37.29%   40.22% 5.68% 0.41%  13.0~ 0.35%   $67,2~ 19087135M_~
## 6 Colora~  5278906 20.78%   69.90% 3.55% 0.57%  2.66% 0.12%   $64,6~ 2648667M_2~
```

- You notice that there are 6 columns representing the population percentage for different races. The columns include the percent symbol `%`.
- Remove the percent symbol `%` from each of the race columns (`Hispanic`,`White`,`Black`,`Native`,`Asian`,`Pacific`). Save the resulting data frame to `us_census`, and view the head.

```
# use gsub() to remove "%" symbol from each column
us_census <- us_census %>%
    mutate(Hispanic    = gsub('\\%' , '' , Hispanic) ,
           White       = gsub('\\%' , '' , White) ,
           Black       = gsub('\\%' , '' , Black) ,
           Native      = gsub('\\%' , '' , Native) ,
           Asian       = gsub('\\%' , '' , Asian) ,
           Pacific     = gsub('\\%' , '' , Pacific))

# inspect
head(us_census)
```

```
## # A tibble: 6 x 10
##   State   TotalPop Hispanic White Black Native Asian Pacific Income  GenderPop
##   <chr>      <dbl> <chr>    <chr> <chr> <chr>  <chr> <chr>   <chr>   <chr>
## 1 Alabama  4830620 3.75     61.88 31.25 0.45   1.05  0.03    $43,29~ 2341093M_2~
## 2 Alaska    733375 5.91     60.91 2.85  16.39  5.45  1.06    $70,35~ 384160M_34~
## 3 Arizona  6641928 29.57    57.12 3.85  4.36   2.88  0.17    $54,20~ 3299088M_3~
## 4 Arkans~  2958208 6.22     71.14 18.97 0.52   1.14  0.15    $41,93~ 1451913M_1~
## 5 Califo~ 38421464 37.29    40.22 5.68  0.41   13.05 0.35    $67,26~ 19087135M_~
## 6 Colora~  5278906 20.78    69.90 3.55  0.57   2.66  0.12    $64,65~ 2648667M_2~
```

- The `Income` column also incudes a `$` symbol along with the number representing median income for a state.
- Remove the `$` from the `Income` column. Save the resulting data frame to `us_census`.
- View the head of `us_census`.

```
# remove $ symbol from Income columns
us_census <- us_census %>%
    mutate(Income = gsub("\\$" , "" , Income))

# inspect
head(us_census)
```

```
## # A tibble: 6 x 10
##   State   TotalPop Hispanic White Black Native Asian Pacific Income  GenderPop
##   <chr>      <dbl> <chr>    <chr> <chr> <chr>  <chr> <chr>   <chr>   <chr>
## 1 Alabama  4830620 3.75     61.88 31.25 0.45   1.05  0.03    43,296~ 2341093M_2~
## 2 Alaska    733375 5.91     60.91 2.85  16.39  5.45  1.06    70,354~ 384160M_34~
## 3 Arizona  6641928 29.57    57.12 3.85  4.36   2.88  0.17    54,207~ 3299088M_3~
## 4 Arkans~  2958208 6.22     71.14 18.97 0.52   1.14  0.15    41,935~ 1451913M_1~
## 5 Califo~ 38421464 37.29    40.22 5.68  0.41   13.05 0.35    67,264~ 19087135M_~
## 6 Colora~  5278906 20.78    69.90 3.55  0.57   2.66  0.12    64,657~ 2648667M_2~
```

The `GenderPop` column appears to hold the male and female population counts.

Separate this column at the `_` character to create two new columns: `male_pop` and `female_pop`.

Save the resulting data frame to `us_census`, and view the head.

```
# use separate() function to seperate the Gender_Prop column
# separator is "_"
us_census <- us_census %>%
    separate(GenderPop,c('male_prop','female_prop'),'_')

# inspect
head(us_census)
```

```
## # A tibble: 6 x 11
##   State     TotalPop Hispanic White Black Native Asian Pacific Income  male_prop
##   <chr>        <dbl> <chr>    <chr> <chr> <chr>  <chr> <chr>   <chr>   <chr>
## 1 Alabama    4830620 3.75     61.88 31.25 0.45   1.05  0.03    43,296~ 2341093M
## 2 Alaska      733375 5.91     60.91 2.85  16.39  5.45  1.06    70,354~ 384160M
## 3 Arizona    6641928 29.57    57.12 3.85  4.36   2.88  0.17    54,207~ 3299088M
```

```
## 4 Arkansas   2958208 6.22     71.14 18.97 0.52   1.14  0.15    41,935~ 1451913M
## 5 Californ~ 38421464 37.29    40.22 5.68  0.41   13.05 0.35    67,264~ 19087135M
## 6 Colorado   5278906 20.78    69.90 3.55  0.57   2.66  0.12    64,657~ 2648667M
## # ... with 1 more variable: female_prop <chr>
```

- You notice the new `male_pop` and `female_pop` columns contain extra characters `M` and `F`, respectively.
- Remove these extra characters from the columns.
- Save the resulting data frame to `us_census`, and view the head.

```
# use gsub() to remove "M" and "F" and replace with "nothing"
us_census <- us_census %>%
    # replace "M" from male_prop with ""
    mutate(male_prop = gsub("M" , "" , male_prop)) %>%
    # replace "F" from female_prop with ""
    mutate(female_prop = gsub("F" , "" , female_prop))

# inspect
head(us_census)
```

```
## # A tibble: 6 x 11
##   State     TotalPop Hispanic White Black Native Asian Pacific Income  male_prop
##   <chr>        <dbl> <chr>    <chr> <chr> <chr>  <chr> <chr>   <chr>   <chr>
## 1 Alabama    4830620 3.75     61.88 31.25 0.45   1.05  0.03    43,296~ 2341093
## 2 Alaska      733375 5.91     60.91 2.85  16.39  5.45  1.06    70,354~ 384160
## 3 Arizona    6641928 29.57    57.12 3.85  4.36   2.88  0.17    54,207~ 3299088
## 4 Arkansas   2958208 6.22     71.14 18.97 0.52   1.14  0.15    41,935~ 1451913
## 5 Californ~ 38421464 37.29    40.22 5.68  0.41   13.05 0.35    67,264~ 19087135
## 6 Colorado   5278906 20.78    69.90 3.55  0.57   2.66  0.12    64,657~ 2648667
## # ... with 1 more variable: female_prop <chr>
```

Update the Data Types

Now that you have removed extra symbols from many of the columns that contain numerical data, you notice that the data type for these columns is still `chr`, or character.

Convert all of these columns (`Hispanic`,`White`,`Black`,`Native`,`Asian`,`Pacific`,`Income`,`male_pop`,`female_pop`) to have a data type of numeric. Save the resulting data frame to `us_census`, and view the head.

```
# convert columns to numeric
us_census <- us_census %>%
    mutate(Hispanic = as.numeric(Hispanic) ,
           White    = as.numeric(White) ,
           Black    = as.numeric(Black) ,
           Native   = as.numeric(Native) ,
           Asian    = as.numeric(Asian) ,
           Pacific  = as.numeric(Pacific) ,
           male_prop = as.numeric(male_prop) ,
           female_prop = as.numeric(female_prop),
           )

# inspect
str(us_census)
```

```
## tibble[,11] [61 x 11] (S3: tbl_df/tbl/data.frame)
##  $ State      : chr [1:61] "Alabama" "Alaska" "Arizona" "Arkansas" ...
##  $ TotalPop   : num [1:61] 4830620 733375 6641928 2958208 38421464 ...
##  $ Hispanic   : num [1:61] 3.75 5.91 29.57 6.22 37.29 ...
##  $ White      : num [1:61] 61.9 60.9 57.1 71.1 40.2 ...
##  $ Black      : num [1:61] 31.25 2.85 3.85 18.97 5.68 ...
##  $ Native     : num [1:61] 0.45 16.39 4.36 0.52 0.41 ...
##  $ Asian      : num [1:61] 1.05 5.45 2.88 1.14 13.05 ...
##  $ Pacific    : num [1:61] 0.03 1.06 0.17 0.15 0.35 0.12 0.12 0.02 0.02 0.03 ...
##  $ Income     : chr [1:61] "43,296.36" "70,354.74" "54,207.82" "41,935.63" ...
##  $ male_prop  : num [1:61] 2341093 384160 3299088 1451913 19087135 ...
##  $ female_prop: num [1:61] 2489527 349215 3342840 1506295 19334329 ...
```

Income column has "Currency with",". We will use gsub() to remove the"," and then convert it to numeric

```
# remove "," from Income
us_census <- us_census %>%
    mutate(Income = gsub('\\,', '', Income)) %>%
# convert Income to numeric
    mutate(Income = as.numeric(Income))

# inspect
str(us_census)
```

```
## tibble[,11] [61 x 11] (S3: tbl_df/tbl/data.frame)
##  $ State      : chr [1:61] "Alabama" "Alaska" "Arizona" "Arkansas" ...
##  $ TotalPop   : num [1:61] 4830620 733375 6641928 2958208 38421464 ...
##  $ Hispanic   : num [1:61] 3.75 5.91 29.57 6.22 37.29 ...
##  $ White      : num [1:61] 61.9 60.9 57.1 71.1 40.2 ...
##  $ Black      : num [1:61] 31.25 2.85 3.85 18.97 5.68 ...
##  $ Native     : num [1:61] 0.45 16.39 4.36 0.52 0.41 ...
##  $ Asian      : num [1:61] 1.05 5.45 2.88 1.14 13.05 ...
##  $ Pacific    : num [1:61] 0.03 1.06 0.17 0.15 0.35 0.12 0.12 0.02 0.02 0.03 ...
##  $ Income     : num [1:61] 43296 70355 54208 41936 67265 ...
##  $ male_prop  : num [1:61] 2341093 384160 3299088 1451913 19087135 ...
##  $ female_prop: num [1:61] 2489527 349215 3342840 1506295 19334329 ...
```

Remove Duplicate Rows

It's always a good idea to check if there are duplicate rows of data in a data set. Pipe `us_census` into the `duplicated()` function to see which rows are duplicated. Then pipe the result into `table()` to get a count of the duplicated rows.

```
# check for duplicate rows
us_census %>%
    duplicated() %>%
    table()
```

```
## .
## FALSE  TRUE
##    52     9
```

We have 9 duplicate rows so now update the value of `us_census` to be the `us_census` data frame with only unique rows

Confirm that there are no more duplicated rows in `us_census`. You should expect to see no `TRUE`s!

```
# remove duplicate rows with distinct()
us_census <- us_census %>%
    distinct()

# inspect
str(us_census)
```

```
## tibble[,11] [52 x 11] (S3: tbl_df/tbl/data.frame)
##  $ State      : chr [1:52] "Alabama" "Alaska" "Arizona" "Arkansas" ...
##  $ TotalPop   : num [1:52] 4830620 733375 6641928 2958208 38421464 ...
##  $ Hispanic   : num [1:52] 3.75 5.91 29.57 6.22 37.29 ...
##  $ White      : num [1:52] 61.9 60.9 57.1 71.1 40.2 ...
##  $ Black      : num [1:52] 31.25 2.85 3.85 18.97 5.68 ...
##  $ Native     : num [1:52] 0.45 16.39 4.36 0.52 0.41 ...
##  $ Asian      : num [1:52] 1.05 5.45 2.88 1.14 13.05 ...
##  $ Pacific    : num [1:52] 0.03 1.06 0.17 0.15 0.35 0.12 0.02 0.02 0.03 0.05 ...
##  $ Income     : num [1:52] 43296 70355 54208 41936 67265 ...
##  $ male_prop  : num [1:52] 2341093 384160 3299088 1451913 19087135 ...
##  $ female_prop: num [1:52] 2489527 349215 3342840 1506295 19334329 ...
```

```
# validate
us_census %>%
    duplicated() %>%
    table()
```

```
## .
## FALSE
##    52
```

```
# final inspection top 6 rows
head(us_census)
```

```
## # A tibble: 6 x 11
##   State      TotalPop Hispanic White Black Native Asian Pacific Income male_prop
##   <chr>         <dbl>    <dbl> <dbl> <dbl>  <dbl> <dbl>   <dbl>  <dbl>     <dbl>
## 1 Alabama     4830620     3.75  61.9 31.2    0.45  1.05    0.03 43296.   2341093
## 2 Alaska       733375     5.91  60.9  2.85  16.4   5.45    1.06 70355.    384160
## 3 Arizona     6641928    29.6   57.1  3.85   4.36  2.88    0.17 54208.   3299088
## 4 Arkansas    2958208     6.22  71.1 19.0    0.52  1.14    0.15 41936.   1451913
## 5 California 38421464    37.3   40.2  5.68   0.41 13.0     0.35 67265.  19087135
## 6 Colorado    5278906    20.8   69.9  3.55   0.57  2.66    0.12 64658.   2648667
## # ... with 1 more variable: female_prop <dbl>
```

```
# final inspection bottom 6 rows
tail(us_census)
```

```
## # A tibble: 6 x 11
```

```
##   State      TotalPop Hispanic White Black Native Asian Pacific Income male_prop
##   <chr>         <dbl>    <dbl> <dbl> <dbl>  <dbl> <dbl>   <dbl>  <dbl>     <dbl>
## 1 Vermont      626604     1.61  94.0  0.98   0.3   1.24    0.03 55603.    308573
## 2 Virginia    8256630     8.01  63.3 20.2    0.21  5.46    0.06 72866.   4060948
## 3 Washington  6985464    11.1   72.0  3.38   1.41  7.02    0.61 64494.   3487725
## 4 West Virg~  1851420     1.29  92.2  3.66   0.15  0.68    0.03 41437.    913631
## 5 Wisconsin   5742117     6.68  79.9  8.2    0.95  2.4     0.02 53899.   2851385
## 6 Wyoming      579679     9.67  84.3  1.05   1.95  0.89    0.07 58758.    295561
## # ... with 1 more variable: female_prop <dbl>
```

```
# final inspection structure
str(us_census)
```

```
## tibble[,11] [52 x 11] (S3: tbl_df/tbl/data.frame)
##  $ State      : chr [1:52] "Alabama" "Alaska" "Arizona" "Arkansas" ...
##  $ TotalPop   : num [1:52] 4830620 733375 6641928 2958208 38421464 ...
##  $ Hispanic   : num [1:52] 3.75 5.91 29.57 6.22 37.29 ...
##  $ White      : num [1:52] 61.9 60.9 57.1 71.1 40.2 ...
##  $ Black      : num [1:52] 31.25 2.85 3.85 18.97 5.68 ...
##  $ Native     : num [1:52] 0.45 16.39 4.36 0.52 0.41 ...
##  $ Asian      : num [1:52] 1.05 5.45 2.88 1.14 13.05 ...
##  $ Pacific    : num [1:52] 0.03 1.06 0.17 0.15 0.35 0.12 0.02 0.02 0.03 0.05 ...
##  $ Income     : num [1:52] 43296 70355 54208 41936 67265 ...
##  $ male_prop  : num [1:52] 2341093 384160 3299088 1451913 19087135 ...
##  $ female_prop: num [1:52] 2489527 349215 3342840 1506295 19334329 ...
```