2 How to read many files into R

> **What will this tutorial cover?**
>
> In this tutorial you will learn how to read many files into R. You will learn how to create a character vector of file paths and then read them into R. You will also be able to work with some edge cases that you might stumble upon.

> **Who do I have to thank?**
>
> I have to thank Beatriz Milz who asked me to write a tutorial on this topic. I also want to thank Jim Hester for his excellent blog post about readr. He showed me that I don't need `map_dfr` to read many files into R, but can achieve the same results with the `read_*` functions.

You don't always read just one file into R. It is not uncommon for your data to be scattered in hundreds or thousands of files. Of course, you don't want to read these files into R manually. So you need an automatic method for reading in files.

To illustrate how this works in tidyverse, lets create 25 CSV files. How we create the CSV files is not important here. Essentially, I sample 20 rows from the mpg dataset 25 times and save these data frames to disk. Also, at the beginning of the code, I create a new directory named `many_files` where the CSV files will be stored.

```
library(tidyverse)
library(fs) # install.packages("fs")
# Create dir
dir_create(c("many_files"))

mpg_samples <- map(1:25, ~ slice_sample(mpg, n = 20))

iwalk(mpg_samples, ~ write_csv(., paste0("many_files/", .y, ".csv")))
```

Later in this course we will discuss the `iwalk` function in more detail. If you look in the `many_files` directory, you should now see 25 CSV files.

2.1 How to create a character vector of file paths?

Before we can read the files into R, we need to create a character vector of the file paths. You have several options to create such a vector. You can use the R base function `list.files`, which returns character vectors of the names of files in a directory or you use the function `dir_ls` from the fs package. Let's start with `list.files`:

```
(csv_files_list_files <- list.files(path = "many_files",
                                    pattern = "csv", full.names = TRUE))
```

```
 [1] "many_files/1.csv"  "many_files/10.csv" "many_files/11.csv"
 [4] "many_files/12.csv" "many_files/13.csv" "many_files/14.csv"
 [7] "many_files/15.csv" "many_files/16.csv" "many_files/17.csv"
[10] "many_files/18.csv" "many_files/19.csv" "many_files/2.csv"
[13] "many_files/20.csv" "many_files/21.csv" "many_files/22.csv"
[16] "many_files/23.csv" "many_files/24.csv" "many_files/25.csv"
[19] "many_files/3.csv"  "many_files/4.csv"  "many_files/5.csv"
[22] "many_files/6.csv"  "many_files/7.csv"  "many_files/8.csv"
[25] "many_files/9.csv"
```

The function has a number of arguments. With path you specify where to find your files. Since the path is relative, make sure you are either working in an R-Studio project or have defined your working directory. `pattern` receives a regular expression. In this case, we said that the file should contain the string "csv". Finally, the `full.names` argument indicates that we want to store the full paths of the files, not just the file names. If you do not set this argument to TRUE, you will have trouble reading in your files later.

The other option is to use the `dir_ls` function from the fs package. fs provides a cross-platform interface for accessing files on your hard disk. It supports all file operations (deleting, creating files, moving files, etc.).

```
# library(fs) # install.packages("fs")
(csv_files_dir_ls <- dir_ls(path = "many_files/",
                            glob = "*.csv", type = "file"))
```

```
many_files/1.csv  many_files/10.csv many_files/11.csv many_files/12.csv
many_files/13.csv many_files/14.csv many_files/15.csv many_files/16.csv
many_files/17.csv many_files/18.csv many_files/19.csv many_files/2.csv
many_files/20.csv many_files/21.csv many_files/22.csv many_files/23.csv
many_files/24.csv many_files/25.csv many_files/3.csv  many_files/4.csv
```

```
many_files/5.csv  many_files/6.csv  many_files/7.csv  many_files/8.csv
many_files/9.csv
```

The results are the same as above. Again, you specify the path where your files are stored. The glob argument is used to specify the file type of your files. With `type` you indicate that you are looking for a file and not a folder or something else.

2.2 How to read the files into R from a character vector of paths

Now that we know the file paths, we can load the files into R. The tidyverse way to do this is to use the `map_dfr` function from the purrr package. `map_dfr` loops through all the file paths and binds the data frames into a single data frame. The `.x` in the following code stands for the file name. To output the actual csv files and not the filenames, we need to put `.x` (the path) in a `read_*` function. In this example we are working with CSV files. The trick works the same for all rectangular file formats.

```
data_frames <- map_dfr(csv_files_dir_ls,
                       ~ read_csv(.x, show_col_types = FALSE))
glimpse(data_frames)
```

```
Rows: 500
Columns: 11
$ manufacturer <chr> "toyota", "chevrolet", "lincoln", "volkswagen", "toyota",~
$ model        <chr> "camry solara", "malibu", "navigator 2wd", "jetta", "camr~
$ displ        <dbl> 3.3, 3.1, 5.4, 2.0, 2.2, 2.4, 4.4, 3.3, 1.9, 5.3, 2.4, 1.~
$ year         <dbl> 2008, 1999, 1999, 2008, 1999, 1999, 2008, 2008, 1999, 200~
$ cyl          <dbl> 6, 6, 8, 4, 4, 4, 8, 6, 4, 8, 4, 4, 4, 6, 4, 8, 6, 4, 4, ~
$ trans        <chr> "auto(s5)", "auto(l4)", "auto(l4)", "auto(s6)", "manual(m~
$ drv          <chr> "f", "f", "r", "f", "f", "f", "4", "f", "f", "4", "f", "f~
$ cty          <dbl> 18, 18, 11, 22, 21, 19, 12, 17, 33, 11, 21, 24, 18, 17, 2~
$ hwy          <dbl> 27, 26, 17, 29, 29, 27, 18, 24, 44, 14, 31, 33, 26, 22, 3~
$ fl           <chr> "r", "r", "r", "p", "r", "r", "r", "r", "d", "e", "r", "r~
$ class        <chr> "compact", "midsize", "suv", "compact", "midsize", "compa~
```

Another approach is to use the `read_csv` function directly by putting the character vector of the file names directly into `read_csv`. I found this trick in the blog post by Jim Hester on tidyverse.org. A neat trick is to specify an id argument that adds a new column to the data frame indicating which files the data came from:

```
  read_csv(csv_files_dir_ls, id = "filename",
           show_col_types = FALSE) %>%
    glimpse
```

```
Rows: 500
Columns: 12
$ filename     <chr> "many_files/1.csv", "many_files/1.csv", "many_files/1.csv~
$ manufacturer <chr> "toyota", "chevrolet", "lincoln", "volkswagen", "toyota",~
$ model        <chr> "camry solara", "malibu", "navigator 2wd", "jetta", "camr~
$ displ        <dbl> 3.3, 3.1, 5.4, 2.0, 2.2, 2.4, 4.4, 3.3, 1.9, 5.3, 2.4, 1.~
$ year         <dbl> 2008, 1999, 1999, 2008, 1999, 1999, 2008, 2008, 1999, 200~
$ cyl          <dbl> 6, 6, 8, 4, 4, 4, 8, 6, 4, 8, 4, 4, 4, 6, 4, 8, 6, 4, 4, ~
$ trans        <chr> "auto(s5)", "auto(l4)", "auto(l4)", "auto(s6)", "manual(m~
$ drv          <chr> "f", "f", "r", "f", "f", "f", "4", "f", "f", "4", "f", "f~
$ cty          <dbl> 18, 18, 11, 22, 21, 19, 12, 17, 33, 11, 21, 24, 18, 17, 2~
$ hwy          <dbl> 27, 26, 17, 29, 29, 27, 18, 24, 44, 14, 31, 33, 26, 22, 3~
$ fl           <chr> "r", "r", "r", "p", "r", "r", "r", "r", "d", "e", "r", "r~
$ class        <chr> "compact", "midsize", "suv", "compact", "midsize", "compa~
```

The first column now specifies the actual file name. We can do the same with our previous approach. We would just have to add a new column with mutate representing the file name:

```
  map_dfr(csv_files_dir_ls,
          ~ read_csv(.x, , show_col_types = FALSE) %>%
            mutate(filename = .x)) %>%
    glimpse()
```

```
Rows: 500
Columns: 12
$ manufacturer <chr> "toyota", "chevrolet", "lincoln", "volkswagen", "toyota",~
$ model        <chr> "camry solara", "malibu", "navigator 2wd", "jetta", "camr~
$ displ        <dbl> 3.3, 3.1, 5.4, 2.0, 2.2, 2.4, 4.4, 3.3, 1.9, 5.3, 2.4, 1.~
$ year         <dbl> 2008, 1999, 1999, 2008, 1999, 1999, 2008, 2008, 1999, 200~
$ cyl          <dbl> 6, 6, 8, 4, 4, 4, 8, 6, 4, 8, 4, 4, 4, 6, 4, 8, 6, 4, 4, ~
$ trans        <chr> "auto(s5)", "auto(l4)", "auto(l4)", "auto(s6)", "manual(m~
$ drv          <chr> "f", "f", "r", "f", "f", "f", "4", "f", "f", "4", "f", "f~
$ cty          <dbl> 18, 18, 11, 22, 21, 19, 12, 17, 33, 11, 21, 24, 18, 17, 2~
$ hwy          <dbl> 27, 26, 17, 29, 29, 27, 18, 24, 44, 14, 31, 33, 26, 22, 3~
$ fl           <chr> "r", "r", "r", "p", "r", "r", "r", "r", "d", "e", "r", "r~
$ class        <chr> "compact", "midsize", "suv", "compact", "midsize", "compa~
$ filename     <fs::path> "many_files/1.csv", "many_files/1.csv", "many_files/~
```

2.3 But what if the column names of the files are not consistent?

Unfortunately, we live in a messy world. Not all column names are the same. Let's create a messy dataset with inconsistent column names (we have covered this technique in `make_clean_names` in our last tutorial):

```
mpg_samples <- map(1:10, ~ slice_sample(mpg, n = 20))

inconsistent_dframes <- map(mpg_samples,
                            ~ janitor::clean_names(dat = .x, case = "random"))
```

The column names of these 10 data frames consist of the same names, but are randomly written in upper or lower case.

```
map(inconsistent_dframes, ~ colnames(.x)) %>%
  head
```

```
[[1]]
 [1] "mANufActurer" "mOdEL"        "disPL"        "yEAR"         "cyL"
 [6] "tRAns"        "drV"          "cty"          "Hwy"          "fl"
[11] "cLass"

[[2]]
 [1] "manUfACTuRer" "modEl"        "DisPl"        "yeaR"         "cYL"
 [6] "TraNs"        "drv"          "cTY"          "Hwy"          "fl"
[11] "ClASs"

[[3]]
 [1] "mAnuFACtUrEr" "mODEL"        "diSpl"        "YeAR"         "cyl"
 [6] "traNs"        "DrV"          "Cty"          "hwy"          "fl"
[11] "cLass"

[[4]]
 [1] "maNufacTuReR" "MOdeL"        "diSPL"        "yEAr"         "cyL"
 [6] "TraNS"        "DrV"          "ctY"          "Hwy"          "fL"
[11] "ClAss"

[[5]]
 [1] "manuFAcTUrER" "moDEL"        "displ"        "YeaR"         "cYl"
 [6] "tRANs"        "DRv"          "cty"          "hwY"          "fL"
[11] "Class"
```

```
[[6]]
 [1] "mANufaCtUrer" "MOdeL"        "diSpL"        "YeAR"         "CYl"
 [6] "tRaNS"        "dRv"          "CTY"          "hwy"          "fl"
[11] "ClASS"
```

To make this data set even more messy, let's select a random set of columns per data frame:

```
inconsistent_dframes <- map(inconsistent_dframes,
                            ~ .x[sample(1:length(.x), sample(1:length(.x), 1))])

map(inconsistent_dframes, ~ colnames(.x)) %>%
  head
```

```
[[1]]
[1] "cLass" "fl"    "cty"   "cyL"

[[2]]
[1] "ClASs" "Hwy"   "cTY"   "TraNs" "cYL"   "fl"    "modEl" "yeaR"  "drv"

[[3]]
 [1] "Cty"          "cLass"        "YeAR"         "DrV"          "fl"
 [6] "mODEL"        "hwy"          "mAnuFACtUrEr" "traNs"        "cyl"

[[4]]
 [1] "maNufacTuReR" "diSPL"        "cyL"          "fL"           "yEAr"
 [6] "TraNS"        "ClAss"        "ctY"          "MOdeL"        "Hwy"

[[5]]
[1] "cty"          "fL"           "hwY"          "manuFAcTUrER" "DRv"

[[6]]
 [1] "fl"           "dRv"          "ClASS"        "tRaNS"        "CTY"
 [6] "MOdeL"        "mANufaCtUrer" "diSpL"        "CYl"          "YeAR"
```

Finally, we save the data to disk:

```
dir_create(c("unclean_files"))

iwalk(inconsistent_dframes,
      ~ write_csv(.x, paste0("unclean_files/", .y, ".csv")))
```

If we tried to load this data using our previous approach, it would work, but the inconsistent column names would result in a plethora of columns:

```
many_columns_data_frame <- dir_ls(path = "unclean_files/",
                                  glob = "*.csv", type = "file") %>%
  map_dfr(~ read_csv(.x, show_col_types = FALSE) %>%
            mutate(filename = .x))

colnames(many_columns_data_frame) %>% sort
```

```
 [1] "claSS"        "clASS"        "cLass"        "cLAss"        "ClAss"
 [6] "ClASs"        "ClASS"        "cty"          "ctY"          "cTy"
[11] "cTY"          "Cty"          "CTy"          "CTY"          "cyl"
[16] "cyL"          "cYL"          "CYl"          "diSpL"        "diSPL"
[21] "DisPL"        "drv"          "dRv"          "DrV"          "DRv"
[26] "filename"     "fl"           "fL"           "Fl"           "hwy"
[31] "hwY"          "Hwy"          "manuFaCturER" "manuFAcTUrER" "maNufacTuReR"
[36] "mAnuFACtUrEr" "mANufaCtUrer" "mANUfactuRer" "modEl"        "mODel"
[41] "mODEL"        "ModEl"        "MoDEl"        "MOdeL"        "traNs"
[46] "tRaNS"        "TraNs"        "TraNS"        "yeaR"         "yEAr"
[51] "Year"         "YeaR"         "YeAR"
```

Clearly, that's not what we want. Instead, we can use our last trick and clean up the data frames and convert the column names to a specific naming convention. We write them all in lowercase and bind them together:

```
many_columns_data_frame <- dir_ls(path = "unclean_files/",
                                  glob = "*.csv", type = "file") %>%
  map_dfr(~ read_csv(.x, name_repair = tolower, show_col_types = FALSE) %>%
            mutate(filename = .x))

many_columns_data_frame %>% glimpse()
```

```
Rows: 200
Columns: 12
$ class        <chr> "midsize", "midsize", "compact", "suv", "pickup", "subcom~
$ fl           <chr> "p", "p", "r", "r", "r", "r", "r", "p", "r", "r", "r", "d~
$ cty          <dbl> 16, 19, 20, 15, 14, 15, 14, 21, 13, 13, 13, 35, 9, 14, 15~
$ cyl          <dbl> 6, 4, 4, 6, 8, 8, 8, 4, 8, 8, 6, 4, 8, 8, 6, 8, 6, 4, 4, ~
$ filename     <fs::path> "unclean_files/1.csv", "unclean_files/1.csv", "uncle~
```

```
$ year         <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
$ trans        <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
$ model        <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
$ manufacturer <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
$ hwy          <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
$ drv          <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
$ displ        <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
```

As you can see, the new file has the same number of columns as the original mpg data frame. Only some values are `NAs` because we selected a random set of columns per file.

2.4 What if the files are not in the same folder?

So far, we have assumed that all files are in the same folder. This is of course not always the case. Sometimes your files are deeply nested. In that case, we need to search through each folder recursively. Recursively means that we search through each folder until we can't find another folder to crawl. Before we see how this works, let's store our data in two folders (this trick works with as many folders as you want):

```
mpg_samples <- map(1:40, ~ slice_sample(mpg, n = 20))

# Create directories
dir_create(c("nested_folders",
             "nested_folders/first_nested_folder",
             "nested_folders/second_nested_folder"))

# First folder
iwalk(mpg_samples[1:20],
      ~ write_csv(.x,
                  paste0("nested_folders/first_nested_folder/", .y, "_first.csv")))

# Second folder
iwalk(mpg_samples[21:40],
      ~ write_csv(.x,
                  paste0("nested_folders/second_nested_folder/", .y, "_second.csv")))
```

If you now try to load all csv files from the `nested_folders` folder, you would get an empty vector:

```
(csv_files_nested <- dir_ls("nested_folders/", glob = "*.csv", type = "file"))
```

```
character(0)
```

This is because `dir_ls` does not look in the nested folders, but only in the parent folder. To make `dir_ls` search through the folders recursively, you need to set the `recurse` argument to `TRUE`:

```
(csv_files_nested <- dir_ls("nested_folders/", glob = "*.csv", type = "file",
                            recurse = TRUE))
```

```
nested_folders/first_nested_folder/10_first.csv
nested_folders/first_nested_folder/11_first.csv
nested_folders/first_nested_folder/12_first.csv
nested_folders/first_nested_folder/13_first.csv
nested_folders/first_nested_folder/14_first.csv
nested_folders/first_nested_folder/15_first.csv
nested_folders/first_nested_folder/16_first.csv
nested_folders/first_nested_folder/17_first.csv
nested_folders/first_nested_folder/18_first.csv
nested_folders/first_nested_folder/19_first.csv
nested_folders/first_nested_folder/1_first.csv
nested_folders/first_nested_folder/20_first.csv
nested_folders/first_nested_folder/2_first.csv
nested_folders/first_nested_folder/3_first.csv
nested_folders/first_nested_folder/4_first.csv
nested_folders/first_nested_folder/5_first.csv
nested_folders/first_nested_folder/6_first.csv
nested_folders/first_nested_folder/7_first.csv
nested_folders/first_nested_folder/8_first.csv
nested_folders/first_nested_folder/9_first.csv
nested_folders/second_nested_folder/10_second.csv
nested_folders/second_nested_folder/11_second.csv
nested_folders/second_nested_folder/12_second.csv
nested_folders/second_nested_folder/13_second.csv
nested_folders/second_nested_folder/14_second.csv
nested_folders/second_nested_folder/15_second.csv
nested_folders/second_nested_folder/16_second.csv
nested_folders/second_nested_folder/17_second.csv
nested_folders/second_nested_folder/18_second.csv
nested_folders/second_nested_folder/19_second.csv
nested_folders/second_nested_folder/1_second.csv
nested_folders/second_nested_folder/20_second.csv
nested_folders/second_nested_folder/2_second.csv
```

```
nested_folders/second_nested_folder/3_second.csv
nested_folders/second_nested_folder/4_second.csv
nested_folders/second_nested_folder/5_second.csv
nested_folders/second_nested_folder/6_second.csv
nested_folders/second_nested_folder/7_second.csv
nested_folders/second_nested_folder/8_second.csv
nested_folders/second_nested_folder/9_second.csv
```

Now you can access all files inside the nested_folders directory:

```
map_dfr(csv_files_nested, ~ read_csv(.x, show_col_types = FALSE) %>%
          mutate(filename = .x)) %>%
  glimpse()
```

```
Rows: 800
Columns: 12
$ manufacturer <chr> "jeep", "volkswagen", "hyundai", "nissan", "toyota", "hyu~
$ model        <chr> "grand cherokee 4wd", "jetta", "sonata", "maxima", "land ~
$ displ        <dbl> 5.7, 2.0, 2.5, 3.5, 4.7, 3.3, 3.1, 2.5, 2.5, 4.0, 2.8, 3.~
$ year         <dbl> 2008, 2008, 1999, 2008, 1999, 2008, 2008, 1999, 2008, 199~
$ cyl          <dbl> 8, 4, 6, 6, 8, 6, 6, 4, 4, 6, 6, 6, 8, 4, 4, 4, 8, 6, 4, ~
$ trans        <chr> "auto(l5)", "manual(m6)", "manual(m5)", "auto(av)", "auto~
$ drv          <chr> "4", "f", "f", "f", "4", "f", "4", "4", "4", "4", "4", "f~
$ cty          <dbl> 13, 21, 18, 19, 11, 19, 17, 19, 19, 14, 15, 18, 16, 22, 2~
$ hwy          <dbl> 18, 29, 26, 25, 15, 28, 25, 26, 25, 17, 25, 26, 23, 29, 3~
$ fl           <chr> "r", "p", "r", "p", "r", "r", "p", "r", "p", "r", "p", "r~
$ class        <chr> "suv", "compact", "midsize", "midsize", "suv", "midsize",~
$ filename     <fs::path> "nested_folders/first_nested_folder/10_first.csv", "~
```

2.5 What if I don't need some of these files?

You don't always need all the files in your directory and need to remove some files from the list of file paths. A good way to do this is to use the `str_detect` function from the stringr package. Let's look at an example. In the following example, I created a character vector and kept the files that contain the string beach:

```
str_detect(c("house", "beach"), pattern = "beach")
```

```
[1] FALSE  TRUE
```

The function returns logical values. To change the actual character vector, we need to add these logical values to the character vector itself:

```
c("my house", "my beach")[str_detect(c("house", "beach"), pattern = "beach")]
```

```
[1] "my beach"
```

But what if you want to remove these files? With the `negate` argument you can find only the files that do not match the pattern:

```
c("my house", "my beach")[str_detect(c("house", "beach"), pattern = "beach",
                                     negate = TRUE)]
```

```
[1] "my house"
```

The hard part is finding the right pattern for your files. Suppose you don't want to keep CSV files that contain the numbers 2, 3 or 4:

```
csv_files_nested[str_detect(csv_files_nested, pattern = "[2-4]",
                            negate = TRUE)]
```

```
nested_folders/first_nested_folder/10_first.csv
nested_folders/first_nested_folder/11_first.csv
nested_folders/first_nested_folder/15_first.csv
nested_folders/first_nested_folder/16_first.csv
nested_folders/first_nested_folder/17_first.csv
nested_folders/first_nested_folder/18_first.csv
nested_folders/first_nested_folder/19_first.csv
nested_folders/first_nested_folder/1_first.csv
nested_folders/first_nested_folder/5_first.csv
nested_folders/first_nested_folder/6_first.csv
nested_folders/first_nested_folder/7_first.csv
nested_folders/first_nested_folder/8_first.csv
nested_folders/first_nested_folder/9_first.csv
nested_folders/second_nested_folder/10_second.csv
nested_folders/second_nested_folder/11_second.csv
nested_folders/second_nested_folder/15_second.csv
nested_folders/second_nested_folder/16_second.csv
nested_folders/second_nested_folder/17_second.csv
```

```
nested_folders/second_nested_folder/18_second.csv
nested_folders/second_nested_folder/19_second.csv
nested_folders/second_nested_folder/1_second.csv
nested_folders/second_nested_folder/5_second.csv
nested_folders/second_nested_folder/6_second.csv
nested_folders/second_nested_folder/7_second.csv
nested_folders/second_nested_folder/8_second.csv
nested_folders/second_nested_folder/9_second.csv
```

The regular expression `[2-4]` looks for the numbers 2 to 4. But what if you want to exclude files that end with a 2, 3, or 4? Then, of course, this regular expression won't work. In this case we need another pattern:

```
csv_files_nested[str_detect(csv_files_nested,
                            pattern = "[2-4]_first|second\\.csv$",
                            negate = TRUE)]
```

```
nested_folders/first_nested_folder/10_first.csv
nested_folders/first_nested_folder/11_first.csv
nested_folders/first_nested_folder/15_first.csv
nested_folders/first_nested_folder/16_first.csv
nested_folders/first_nested_folder/17_first.csv
nested_folders/first_nested_folder/18_first.csv
nested_folders/first_nested_folder/19_first.csv
nested_folders/first_nested_folder/1_first.csv
nested_folders/first_nested_folder/20_first.csv
nested_folders/first_nested_folder/5_first.csv
nested_folders/first_nested_folder/6_first.csv
nested_folders/first_nested_folder/7_first.csv
nested_folders/first_nested_folder/8_first.csv
nested_folders/first_nested_folder/9_first.csv
```

This pattern is a bit more complicated. Again, we look for the numbers 2 to 4 followed by an underscore and the words first or second (indicated by a vertical bar |). Since a period represents any arbitrary character in regular expressions, we need to terminate it with two backslashes. Finally, our file should end with the characters csv, which is indicated by the dollar sign $.

The rest is similar to what we did before:

```
csv_files_nested[str_detect(csv_files_nested,
                            pattern = "[2-4]_first|second\\.csv$",
                            negate = TRUE)] %>%
map_dfr(~ read_csv(.x, show_col_types = FALSE) %>%
          mutate(filename = .x)) %>%
  glimpse()
```

```
Rows: 280
Columns: 12
$ manufacturer <chr> "jeep", "volkswagen", "hyundai", "nissan", "toyota", "hyu~
$ model        <chr> "grand cherokee 4wd", "jetta", "sonata", "maxima", "land ~
$ displ        <dbl> 5.7, 2.0, 2.5, 3.5, 4.7, 3.3, 3.1, 2.5, 2.5, 4.0, 2.8, 3.~
$ year         <dbl> 2008, 2008, 1999, 2008, 1999, 2008, 2008, 1999, 2008, 199~
$ cyl          <dbl> 8, 4, 6, 6, 8, 6, 6, 4, 4, 6, 6, 6, 8, 4, 4, 4, 8, 6, 4, ~
$ trans        <chr> "auto(l5)", "manual(m6)", "manual(m5)", "auto(av)", "auto~
$ drv          <chr> "4", "f", "f", "f", "4", "f", "4", "4", "4", "4", "4", "f~
$ cty          <dbl> 13, 21, 18, 19, 11, 19, 17, 19, 19, 14, 15, 18, 16, 22, 2~
$ hwy          <dbl> 18, 29, 26, 25, 15, 28, 25, 26, 25, 17, 25, 26, 23, 29, 3~
$ fl           <chr> "r", "p", "r", "p", "r", "r", "p", "r", "p", "r", "p", "r~
$ class        <chr> "suv", "compact", "midsize", "midsize", "suv", "midsize",~
$ filename     <fs::path> "nested_folders/first_nested_folder/10_first.csv", "~
```

Summary

Here is what you can take from this tutorial.

- To read many files into R, you need to create a character vector of file paths. Once you have this vector you can read the files with `map_dfr` or the `read_*` functions.
- If you only need a subset of data frames, you can filter the character vector of the file paths with regular expressions
- If your files are not in the same folder search them recursively
- If the column names of your files are not consistent, use the `name_repair` argument of your `read_*` functions