Data manipulation in R (online workshop)

Sigrid Weber

Contents

Chapter 1

Introduction

Data manipulation - or the process of cleaning, organising and preparing your data for further analysis - is required for most projects involving real-world datasets. This workshop teaches you how to use R to manipulate raw data and prepare it for analysis. We will cover the following topics:

- The grammar of data manipulation
- Merging multiple datasets
- Creating subsets of data using filters
- Reshaping data between long and wide formats
- Summarising data with group-wise operation
- Setting up data pipelines in dplyr for efficient data manipulation

The workshop is designed for individuals who are already familiar with R but wish to learn efficient techniques for data manipulation. The workshop will be a combination of coding demonstrations by me and exercises for you to try on your own computer. Please make sure that R and RStudio are installed for the workshop. You may also want to install the package *tidyverse* in advance. Instructions how to do so can be found in the section *3 Getting started*.

Towards the end of the workshop, we will hopefully have time to discuss your dissertation projects and data challenges. Please feel free to reach out to me anytime after the workshop if you face specific problems manipulating your data for your research projects: s.weber.17@ucl.ac.uk.

Chapter 2

Resources

- Google (in particular Stack Overflow)
- Tidy Data
- Tidyverse
- Data Wrangling with dplyr and tidyr Cheat Sheet
- R for Data Science
- Advanced R
- Data manipulation in R with dplyr
- Hands-on dplyr tutorial for faster data manipulation in R

Chapter 3

Getting Started

3.1 Prerequisites

Basic knowledge of working with datasets in R is essential. This course assumes that you're comfortable with reading datasets, working with script files, and navigating in RStudio.

3.2 Software Requirements

3.2.1 R and RStudio

Recent versions of R (version 3.2 or newer) and RStudio (version 1.0 above) are required.

You can download the latest versions from the links below:

- Download R
- Download RStudio

You can find out the version of R installed by typing `version` at the console:

```
version
```

```
##                _
## platform       x86_64-w64-mingw32
## arch           x86_64
## os             mingw32
## system         x86_64, mingw32
## status
## major          4
## minor          1.0
## year           2021
```

```
## month          05
## day            18
## svn rev        80317
## language       R
## version.string R version 4.1.0 (2021-05-18)
## nickname       Camp Pontanezen
```

3.3 Required Packages

This workshop relies on three packages: `dplyr`, `tidyr`, and `readr`. There are two ways to install these packages:

3.3.1 Option 1: Use `tidyverse`

You can either install these two packages individually or use `tidyverse`. The `tidyverse` package is a collection of packages used for data manipulation and visualization. In addition to `dplyr`, `tidyr`, and `readr`, it also includes the following:

```
##  [1] "broom"         "cli"           "crayon"        "dbplyr"
##  [5] "dplyr"         "dtplyr"        "forcats"       "googledrive"
##  [9] "googlesheets4" "ggplot2"       "haven"         "hms"
## [13] "httr"          "jsonlite"      "lubridate"     "magrittr"
## [17] "modelr"        "pillar"        "purrr"         "readr"
## [21] "readxl"        "reprex"        "rlang"         "rstudioapi"
## [25] "rvest"         "stringr"       "tibble"        "tidyr"
## [29] "xml2"          "tidyverse"
```

You can install `tidyverse` using the `install.packages()` function:

```
install.packages("tidyverse")
```

You can find out the version of `tidyverse` installed using the `packageVersion()` function:

```
packageVersion("tidyverse")
```

```
## [1] '1.3.1'
```

To update `tidyverse` packages, you can use the `tidyverse_update()` function:

```
tidyverse::tidyverse_update()
```

3.3.2 Option 2: Install Individual Packages

If you encounter any problems installing `tidyverse`, then the other option is to install `dplyr`, `tidyr`, and `readr` individually.

```
install.packages("dplyr")
install.packages("tidyr")
install.packages("readr")
```

Chapter 4

Basic Operations

Let's start off by creating a new R script and loading `tidyverse`:

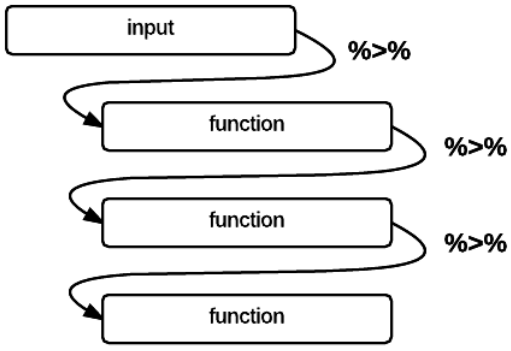
```
library(tidyverse)
```

Clear everything to make sure there's nothing leftover in our environment

```
rm(list = ls())
```

4.1 Data pipelines

Dplyr makes it easy to "chain" functions together using the *pipe* operator `%>%`. The following diagram illustrates the general concept of pipes where data flows from one pipe to another until all the processing is completed.



The syntax of the pipe operator `%>%` might appear unusual at first, but once you get used to it you'll start to appreciate its power and flexibility.

4.2 Dataset

We're using a dataset of flight departures from Houston in 2011.

Filename	Description
flights.csv	Flight departures from Houston in 2011
weather.csv	Hourly weather
planes.csv	Metadata for planes
airports.csv	Metadata for airports

We're going to use the `readr` package which provides improved functions for reading datasets from files. Instead of the usual `read.csv()` function, we'll use the `read_csv()` function from `readr`.

```
flights <- read_csv("https://raw.githubusercontent.com/SigWeber/data-manipulation-works
```

Now let's examine the dataset

```
flights
```

```
## # A tibble: 227,496 x 15
##    date        hour minute   dep   arr dep_delay arr_delay carrier flight origin
##    <chr>      <dbl>  <dbl> <dbl> <dbl>     <dbl>     <dbl> <chr>    <dbl> <chr>
##  1 01/01/201~    14      0  1400  1500         0       -10 AA         428 IAH
##  2 02/01/201~    14      1  1401  1501         1        -9 AA         428 IAH
##  3 03/01/201~    13     52  1352  1502        -8        -8 AA         428 IAH
##  4 04/01/201~    14      3  1403  1513         3         3 AA         428 IAH
##  5 05/01/201~    14      5  1405  1507         5        -3 AA         428 IAH
##  6 06/01/201~    13     59  1359  1503        -1        -7 AA         428 IAH
##  7 07/01/201~    13     59  1359  1509        -1        -1 AA         428 IAH
##  8 08/01/201~    13     55  1355  1454        -5       -16 AA         428 IAH
##  9 09/01/201~    14     43  1443  1554        43        44 AA         428 IAH
## 10 10/01/201~    14     43  1443  1553        43        43 AA         428 IAH
## # ... with 227,486 more rows, and 5 more variables: dest <chr>, plane <chr>,
## #   cancelled <dbl>, time <dbl>, dist <dbl>
```

Notice that because we used `read_csv()`, the data frame we received now prints nicely without having to use the `head()` function and does not clutter your screen.

4.3 Select

The `select` function is used to select columns.

- Select the destination, duration and distance columns (`dest`, `time` and `dist`)

```
flights %>%
  select(dest, time, dist)
```

```
## # A tibble: 227,496 x 3
##    dest   time  dist
##    <chr> <dbl> <dbl>
##  1 DFW      40   224
##  2 DFW      45   224
##  3 DFW      48   224
##  4 DFW      39   224
##  5 DFW      44   224
##  6 DFW      45   224
##  7 DFW      43   224
##  8 DFW      40   224
##  9 DFW      41   224
## 10 DFW      45   224
## # ... with 227,486 more rows
```

Add the arrival delay (`arr_delay`) and departure delay (`dep_delay`) columns as well.

```
flights %>%
  select(dest, time, dist, arr_delay, dep_delay)
```

```
## # A tibble: 227,496 x 5
##    dest   time  dist arr_delay dep_delay
##    <chr> <dbl> <dbl>     <dbl>     <dbl>
##  1 DFW      40   224       -10         0
##  2 DFW      45   224        -9         1
##  3 DFW      48   224        -8        -8
##  4 DFW      39   224         3         3
##  5 DFW      44   224        -3         5
##  6 DFW      45   224        -7        -1
##  7 DFW      43   224        -1        -1
##  8 DFW      40   224       -16        -5
##  9 DFW      41   224        44        43
## 10 DFW      45   224        43        43
## # ... with 227,486 more rows
```

Other ways to do the same

```
flights %>%
  select(dest, time, dist, ends_with("delay"))
```

```
## # A tibble: 227,496 x 5
##    dest   time  dist dep_delay arr_delay
##    <chr> <dbl> <dbl>     <dbl>     <dbl>
##  1 DFW      40   224         0       -10
```

```
##  2 DFW      45   224         1        -9
##  3 DFW      48   224        -8        -8
##  4 DFW      39   224         3         3
##  5 DFW      44   224         5        -3
##  6 DFW      45   224        -1        -7
##  7 DFW      43   224        -1        -1
##  8 DFW      40   224        -5       -16
##  9 DFW      41   224        43        44
## 10 DFW      45   224        43        43
## # ... with 227,486 more rows
```

and ...

```
flights %>%
  select(dest, time, dist, contains("delay"))
```

```
## # A tibble: 227,496 x 5
##    dest   time  dist dep_delay arr_delay
##    <chr> <dbl> <dbl>     <dbl>     <dbl>
##  1 DFW      40   224         0       -10
##  2 DFW      45   224         1        -9
##  3 DFW      48   224        -8        -8
##  4 DFW      39   224         3         3
##  5 DFW      44   224         5        -3
##  6 DFW      45   224        -1        -7
##  7 DFW      43   224        -1        -1
##  8 DFW      40   224        -5       -16
##  9 DFW      41   224        43        44
## 10 DFW      45   224        43        43
## # ... with 227,486 more rows
```

Select all columns from `date` to `arr`

```
flights %>%
  select(date:arr)
```

```
## # A tibble: 227,496 x 5
##    date              hour minute   dep   arr
##    <chr>            <dbl>  <dbl> <dbl> <dbl>
##  1 01/01/2011 12:00    14      0  1400  1500
##  2 02/01/2011 12:00    14      1  1401  1501
##  3 03/01/2011 12:00    13     52  1352  1502
##  4 04/01/2011 12:00    14      3  1403  1513
##  5 05/01/2011 12:00    14      5  1405  1507
##  6 06/01/2011 12:00    13     59  1359  1503
##  7 07/01/2011 12:00    13     59  1359  1509
##  8 08/01/2011 12:00    13     55  1355  1454
##  9 09/01/2011 12:00    14     43  1443  1554
```

```
## 10 10/01/2011 12:00    14     43  1443  1553
## # ... with 227,486 more rows
```

Select all *except* `plane` column using the *minus* sign

```
flights %>%
  select(-plane)
```

```
## # A tibble: 227,496 x 14
##    date        hour minute   dep   arr dep_delay arr_delay carrier flight origin
##    <chr>      <dbl>  <dbl> <dbl> <dbl>     <dbl>     <dbl> <chr>    <dbl> <chr>
##  1 01/01/201~    14      0  1400  1500         0       -10 AA         428 IAH
##  2 02/01/201~    14      1  1401  1501         1        -9 AA         428 IAH
##  3 03/01/201~    13     52  1352  1502        -8        -8 AA         428 IAH
##  4 04/01/201~    14      3  1403  1513         3         3 AA         428 IAH
##  5 05/01/201~    14      5  1405  1507         5        -3 AA         428 IAH
##  6 06/01/201~    13     59  1359  1503        -1        -7 AA         428 IAH
##  7 07/01/201~    13     59  1359  1509        -1        -1 AA         428 IAH
##  8 08/01/201~    13     55  1355  1454        -5       -16 AA         428 IAH
##  9 09/01/201~    14     43  1443  1554        43        44 AA         428 IAH
## 10 10/01/201~    14     43  1443  1553        43        43 AA         428 IAH
## # ... with 227,486 more rows, and 4 more variables: dest <chr>,
## #   cancelled <dbl>, time <dbl>, dist <dbl>
```

4.4 Filter

The `filter()` function returns rows with matching conditions. We can find all flights to Boston (BOS) like this:

```
flights %>%
  filter(dest == "BOS")
```

```
## # A tibble: 1,752 x 15
##    date        hour minute   dep   arr dep_delay arr_delay carrier flight origin
##    <chr>      <dbl>  <dbl> <dbl> <dbl>     <dbl>     <dbl> <chr>    <dbl> <chr>
##  1 31/01/201~     7     35   735  1220         0         4 CO         282 IAH
##  2 31/01/201~    10     47  1047  1526        -3        -5 CO         382 IAH
##  3 31/01/201~    13      5  1305  1746         0        -3 CO         482 IAH
##  4 31/01/201~    19      1  1901  2332         6        -1 CO         582 IAH
##  5 31/01/201~    15     50  1550  2012         0       -25 CO         682 IAH
##  6 30/01/201~    10     46  1046  1518        -4        -8 CO         382 IAH
##  7 30/01/201~    13     19  1319  1811        14        22 CO         482 IAH
##  8 30/01/201~    19      9  1909    23        14        50 CO         582 IAH
##  9 30/01/201~    15     53  1553  2030         3        -7 CO         682 IAH
## 10 29/01/201~     7     40   740  1227         5        16 CO         282 IAH
## # ... with 1,742 more rows, and 5 more variables: dest <chr>, plane <chr>,
## #   cancelled <dbl>, time <dbl>, dist <dbl>
```

Let's build on the previous exercise and find all flights to Boston (BOS) and select only the `dest`, `time`, `dist` columns:

```
flights %>%
  select(dest, time, dist) %>%
  filter(dest == "BOS")
```

```
## # A tibble: 1,752 x 3
##    dest   time  dist
##    <chr> <dbl> <dbl>
##  1 BOS     195  1597
##  2 BOS     188  1597
##  3 BOS     190  1597
##  4 BOS     188  1597
##  5 BOS     180  1597
##  6 BOS     190  1597
##  7 BOS     185  1597
##  8 BOS     198  1597
##  9 BOS     194  1597
## 10 BOS     203  1597
## # ... with 1,742 more rows
```

Now let's do the filter first and then select the columns

```
flights %>%
  filter(dest == "BOS") %>%
  select(dest, time, dist)
```

```
## # A tibble: 1,752 x 3
##    dest   time  dist
##    <chr> <dbl> <dbl>
##  1 BOS     195  1597
##  2 BOS     188  1597
##  3 BOS     190  1597
##  4 BOS     188  1597
##  5 BOS     180  1597
##  6 BOS     190  1597
##  7 BOS     185  1597
##  8 BOS     198  1597
##  9 BOS     194  1597
## 10 BOS     203  1597
## # ... with 1,742 more rows
```

In this case the order doesn't matter, but when using pipes make sure you understand that each function is executed in sequence and the results are then fed to the next one.

4.4.1 Exercise

Find all flights that match the following conditions:

1. To SFO or OAK
2. In January
3. Delayed departure by more than an hour
4. Departure delay more than twice the arrival delay

Here's a brief summary of operators you can use:

Comparison Operators

Operator	Description	Example (assume x is 5)	Result
>	greater than	x > 5	FALSE
>=	greater than or equal to	x >= 5	TRUE
<	less than	x < 5	FALSE
<=	less than or equal to	x <= 5	TRUE
==	equal to	x == 5	TRUE
!=	not equal to	x != 5	FALSE

Logical Operators

Operator	Description
!	not
|	or
&	and

Other Operators

Operator	Description	Example (assume x is 5)	Result
%in%	check element in a vector	x %in% c(1, 3, 5, 7) x %in% c(2, 4, 6, 8)	TRUE FALSE

4.5 Arrange

The `arrange()` function is used to sort the rows based on one or more columns

```
flights %>%
  arrange(dest) %>% select(date,dest)
```

```
## # A tibble: 227,496 x 2
##    date             dest
##    <chr>            <chr>
##  1 31/01/2011 12:00 ABQ
##  2 30/01/2011 12:00 ABQ
##  3 29/01/2011 12:00 ABQ
##  4 28/01/2011 12:00 ABQ
##  5 27/01/2011 12:00 ABQ
##  6 26/01/2011 12:00 ABQ
##  7 25/01/2011 12:00 ABQ
```

```
##  8 24/01/2011 12:00 ABQ
##  9 23/01/2011 12:00 ABQ
## 10 22/01/2011 12:00 ABQ
## # ... with 227,486 more rows
```

4.5.1 Exercise

1. Order flights by departure date and time
2. Which flights were most delayed?
3. Which flights caught up the most time during flight?

4.6 Mutate

The `mutate()` function is used to create new variables.

Up until now we've only been examining the dataset but haven't made any changes to it. All our functions so far have simply displayed the results on screen but haven't created or modified existing variables. Let's see how we can create a new variable called `speed` based on the distance and duration in the flights dataframe.

In this exercise we're adding a new variable to an existing dataframe so we'll just overwrite the `flights` dataframe with the one that has a `speed` column

```
flights <- flights %>%
  mutate(speed = dist / (time / 60))

flights %>%
  select(speed)%>%
  arrange(speed)
```

```
## # A tibble: 227,496 x 1
##    speed
##    <dbl>
##  1  98.8
##  2 106.
##  3 109.
##  4 115.
##  5 125.
##  6 127.
##  7 127.
##  8 129.
##  9 140
## 10 141.
## # ... with 227,486 more rows
```

4.6.1 Exercise

1. Add a variable to show how much time was made up (or lost) during flight
2. Add a variable to identify all short flights that take less than one hour

4.7 Summarise

Let's count the number of flights departing each day.

```
flights %>%
  group_by(date) %>%
  summarise(count = n())
```

```
## # A tibble: 365 x 2
##    date             count
##    <chr>            <int>
##  1 01/01/2011 12:00   552
##  2 01/02/2011 12:00   577
##  3 01/03/2011 12:00   591
##  4 01/04/2011 12:00   684
##  5 01/05/2011 12:00   619
##  6 01/06/2011 12:00   591
##  7 01/07/2011 12:00   699
##  8 01/08/2011 12:00   699
##  9 01/09/2011 12:00   670
## 10 01/10/2011 12:00   490
## # ... with 355 more rows
```

Here's a nice little trick. You can use `View()` to look at the results of a pipe operation without creating new variables.

```
flights %>%
  group_by(date) %>%
  summarise(count = n()) %>%
  View()
```

Of course, often times we'd want to save the summary in a variable for further analysis.

Let's find the average departure delay for each destination

```
delays <- flights %>%
    group_by(dest) %>%
    summarise(average_delay = mean(dep_delay))

delays
```

```
## # A tibble: 116 x 2
```

```
##    dest  average_delay
##    <chr>         <dbl>
##  1 ABQ            NA
##  2 AEX            NA
##  3 AGS            10
##  4 AMA            NA
##  5 ANC            25.0
##  6 ASE            NA
##  7 ATL            NA
##  8 AUS            NA
##  9 AVL            NA
## 10 BFL            NA
## # ... with 106 more rows
```

4.7.1 Exercise

1. What's wrong with the results above, and how would you fix the problem?
2. Can you think of using filter to solve the problem?
3. How many different destinations can you fly to from Houston?
4. Which destinations have the highest average delays?

4.8 Unite

The `unite` function is useful for combining multiple columns together. In the example below, we join the `carrier` and `flight` to create a unique `flight_id` column.

```
flights %>%
  unite(flight_id, carrier, flight, sep = "-", remove = FALSE) %>%
  select(date, carrier, flight, flight_id)
```

```
## # A tibble: 227,496 x 4
##    date             carrier flight flight_id
##    <chr>            <chr>    <dbl> <chr>
##  1 01/01/2011 12:00 AA         428 AA-428
##  2 02/01/2011 12:00 AA         428 AA-428
##  3 03/01/2011 12:00 AA         428 AA-428
##  4 04/01/2011 12:00 AA         428 AA-428
##  5 05/01/2011 12:00 AA         428 AA-428
##  6 06/01/2011 12:00 AA         428 AA-428
##  7 07/01/2011 12:00 AA         428 AA-428
##  8 08/01/2011 12:00 AA         428 AA-428
##  9 09/01/2011 12:00 AA         428 AA-428
## 10 10/01/2011 12:00 AA         428 AA-428
```

```
## # ... with 227,486 more rows
```

4.9 Separate

The `separate` function works the other way around by splitting a single column into multiple columns. Let's split the `date` column into separate `date` and `time` columns.

```
flights %>%
  separate(date, c("date", "time"), sep = " ")
```

```
## # A tibble: 227,496 x 18
##    date       time   hour minute   dep   arr dep_delay arr_delay carrier flight
##    <chr>      <chr> <dbl>  <dbl> <dbl> <dbl>     <dbl>     <dbl> <chr>    <dbl>
##  1 01/01/2011 12:00    14      0  1400  1500         0       -10 AA         428
##  2 02/01/2011 12:00    14      1  1401  1501         1        -9 AA         428
##  3 03/01/2011 12:00    13     52  1352  1502        -8        -8 AA         428
##  4 04/01/2011 12:00    14      3  1403  1513         3         3 AA         428
##  5 05/01/2011 12:00    14      5  1405  1507         5        -3 AA         428
##  6 06/01/2011 12:00    13     59  1359  1503        -1        -7 AA         428
##  7 07/01/2011 12:00    13     59  1359  1509        -1        -1 AA         428
##  8 08/01/2011 12:00    13     55  1355  1454        -5       -16 AA         428
##  9 09/01/2011 12:00    14     43  1443  1554        43        44 AA         428
## 10 10/01/2011 12:00    14     43  1443  1553        43        43 AA         428
## # ... with 227,486 more rows, and 8 more variables: origin <chr>, dest <chr>,
## #   plane <chr>, cancelled <dbl>, dist <dbl>, speed <dbl>,
## #   delay_difference <dbl>, short_flights <lgl>
```

4.9.1 Exercise

1. Split the `date` column into `year`, `month`, and `day` columns
2. Ensure that the `year`, `month`, and `day` columns are of type *integer* (NOT *character*)
    - HINT: Use online help for `separate` for an easy way to do this

Chapter 5

Merging Datasets

Let's start by loading the `tidyverse` package

```
library(tidyverse)
```

Clear everything to make sure there's nothing leftover in our environment

```
rm(list = ls())
```

Next, we load three datasets of universities, cities, and states.

```
universities <- read_csv("https://raw.githubusercontent.com/SigWeber/data-manipulation-workshop/m
cities <- read_csv("https://raw.githubusercontent.com/SigWeber/data-manipulation-workshop/master/
states <- read_csv("https://raw.githubusercontent.com/SigWeber/data-manipulation-workshop/master/
```

Let's see how we can merge the `universities` dataset with the `cities` dataset.

```
## Warning: `data_frame()` was deprecated in tibble 1.1.0.
## Please use `tibble()` instead.
```

universities

university	city
Cornell	Ithaca
Harvard	Cambridge
MIT	Cambridge
Yale	New Haven

cities

city	state
Cambridge	Massachusetts
Ithaca	New York
Seattle	Washington

5.1 Left Join

```
universities %>%
  left_join(cities, by = "city")
```

```
left_join(universities,cities,by = "city")
```

university	city	state
Cornell	Ithaca	New York
Harvard	Cambridge	Massachusetts
MIT	Cambridge	Massachusetts
Yale	New Haven	NA

5.2 Right Join

```
universities %>%
  right_join(cities, by = "city")
```

university	city	state
Cornell	Ithaca	New York
Harvard	Cambridge	Massachusetts
MIT	Cambridge	Massachusetts
NA	Seattle	Washington

5.3 Inner Join

```
universities %>%
  inner_join(cities, by = "city")
```

university	city	state
Cornell	Ithaca	New York
Harvard	Cambridge	Massachusetts
MIT	Cambridge	Massachusetts

5.4 Full Join

```
universities %>%
  full_join(cities, by = "city")
```

university	city	state
Cornell	Ithaca	New York
Harvard	Cambridge	Massachusetts
MIT	Cambridge	Massachusetts
Yale	New Haven	NA
NA	Seattle	Washington

5.5 Different Column Names

In the previous example both our datasets included a column named `city`. But what if the names of the columns in the two datasets were not the same? For example, let's take a look at the `states` table:

states

code	statename
CT	Connecticut
MA	Massachusetts
NY	New York
WA	Washington

What if we were to merge the `cities` dataset with `states`?

cities

city	state
Cambridge	Massachusetts
Ithaca	New York
Seattle	Washington

states

code	statename
CT	Connecticut
MA	Massachusetts
NY	New York
WA	Washington

One option would be to rename the columns so their names would match, but you don't really need to do that. You can simply tell the join functions the mapping between the different names.

```
cities %>%
  left_join(states, by = c("state" = "statename"))
```

In the above example, we're telling `left_join()` to merge using the `state` column from the `cities` data frame and `statename` column from the `states` data frame.

city	state	code
Cambridge	Massachusetts	MA
Ithaca	New York	NY
Seattle	Washington	WA

5.6 Exercise

1. Load the following datasets:

```
presidents <- read_csv("https://raw.githubusercontent.com/SigWeber/data-manipulati
presidents_home <- read_csv("https://raw.githubusercontent.com/SigWeber/data-manip
```

The datasets include names of U.S. presidents:

presidents

First	Middle	Last	TookOffice	LeftOffice
George	H. W.	Bush	20/01/1989	20/01/1993
George	W.	Bush	20/01/2001	20/01/2009
Dwight	D.	Eisenhower	20/01/1953	20/01/1961
John	F.	Kennedy	20/01/1961	22/11/1963
Franklin	D.	Roosevelt	4/03/1933	12/4/1945

presidents_home

GivenName	Middle	Surname	HomeState
George	H. W.	Bush	Texas
Franklin	D.	Roosevelt	New York
John	Quincy	Adams	Massachusetts
William	Howard	Taft	Ohio
George	W.	Bush	Texas

2. Merge the two datasets so that it ONLY includes observations that exist in BOTH the datasets. There should be no missing values or `NA` in the merged table. The results should match the following:

First	Middle	Last	TookOffice	LeftOffice	HomeState
George	H. W.	Bush	20/01/1989	20/01/1993	Texas
George	W.	Bush	20/01/2001	20/01/2009	Texas
Franklin	D.	Roosevelt	4/03/1933	12/4/1945	New York

3. Merge the two datasets so that it includes ALL the observations from both the datasets. Some `TookOffice, LeftOffice` and `HomeState` values will be `NA` and that's ok. The results should match the following:

First	Middle	Last	TookOffice	LeftOffice	HomeState
George	H. W.	Bush	20/01/1989	20/01/1993	Texas
George	W.	Bush	20/01/2001	20/01/2009	Texas
Dwight	D.	Eisenhower	20/01/1953	20/01/1961	NA
John	F.	Kennedy	20/01/1961	22/11/1963	NA
Franklin	D.	Roosevelt	4/03/1933	12/4/1945	New York
John	Quincy	Adams	NA	NA	Massachusetts
William	Howard	Taft	NA	NA	Ohio

4. Merge the two datasets so that ALL observations from the `presidents` datasets are included. Some `HomeState` values will be `NA` and that's ok. The results should match the following:

First	Middle	Last	TookOffice	LeftOffice	HomeState
George	H. W.	Bush	20/01/1989	20/01/1993	Texas
George	W.	Bush	20/01/2001	20/01/2009	Texas
Dwight	D.	Eisenhower	20/01/1953	20/01/1961	NA
John	F.	Kennedy	20/01/1961	22/11/1963	NA
Franklin	D.	Roosevelt	4/03/1933	12/4/1945	New York

5. Merge the two datasets so that ALL observations from the `presidents_home` datasets are included. Some `TookOffice` and `LeftOffice` values will be `NA` and that's ok. The results should match the following:

First	Middle	Last	TookOffice	LeftOffice	HomeState
George	H. W.	Bush	20/01/1989	20/01/1993	Texas
George	W.	Bush	20/01/2001	20/01/2009	Texas
Franklin	D.	Roosevelt	4/03/1933	12/4/1945	New York
John	Quincy	Adams	NA	NA	Massachusetts
William	Howard	Taft	NA	NA	Ohio

Chapter 6

Reshaping

It's fairly common for datasets from public sources to come in formats that need to be reshaped. The World Development Indicators (WDI) is one such dataset that requires reshaping before we can analyse it. Let's go over the steps to see how we can reshape the WDI dataset.

Let's start by loading the `tidyverse` package first.

```
library(tidyverse)
```

Clear everything to make sure there's nothing leftover in our environment

```
rm(list = ls())
```

We're using a small sample of the WDI dataset here to simplify the tasks. Let's load the dataset and see what it looks like.

```
wdi <- read_csv("https://raw.githubusercontent.com/SigWeber/data-manipulation-workshop/master/dat

wdi
```

```
## # A tibble: 5 x 7
##   `¬Series.Name` Series.Code Country.Name Country.Code X1995.YR1995 X2000.YR2000
##   <chr>          <chr>       <chr>        <chr>                <dbl>        <dbl>
## 1 Maternal mort~ SH.STA.MMRT "France"     "FRA"                15           12
## 2 Maternal mort~ SH.STA.MMRT "Spain"      "ESP"                 6            5
## 3 Maternal mort~ SH.STA.MMRT ""           ""                   NA           NA
## 4 Health expend~ SH.XPD.TOT~ "France"     "FRA"                10.4         10.1
## 5 Health expend~ SH.XPD.TOT~ "Spain"      "ESP"                 7.44         7.21
## # ... with 1 more variable: X2005.YR2005 <dbl>
```

But ideally, we'd like our data to look something like this:

```
## # A tibble: 12 x 5
```

```
##    Indicator          Country.Name Country.Code  Year Development.Index
##    <chr>              <chr>        <chr>        <dbl>             <dbl>
##  1 Maternal mortality France       FRA           1995              15
##  2 Maternal mortality France       FRA           2000              12
##  3 Maternal mortality France       FRA           2005              10
##  4 Maternal mortality Spain        ESP           1995               6
##  5 Maternal mortality Spain        ESP           2000               5
##  6 Maternal mortality Spain        ESP           2005               5
##  7 Health expenditure France       FRA           1995              10.4
##  8 Health expenditure France       FRA           2000              10.1
##  9 Health expenditure France       FRA           2005              10.9
## 10 Health expenditure Spain        ESP           1995               7.44
## 11 Health expenditure Spain        ESP           2000               7.21
## 12 Health expenditure Spain        ESP           2005               8.29
```

So, what do we do to achieve this new data format? Note: We want to move away from a **wide** data frame to a **long** data frame!

Step 1: We can see that some country names and codes are blank, so let's get rid of them first

```
wdi %>%
  filter(Country.Code !=  "")
```

```
## # A tibble: 4 x 7
##   `¬Series.Name` Series.Code Country.Name Country.Code X1995.YR1995 X2000.YR2000
##   <chr>          <chr>       <chr>        <chr>                <dbl>        <dbl>
## 1 Maternal mort~ SH.STA.MMRT France       FRA                  15           12
## 2 Maternal mort~ SH.STA.MMRT Spain        ESP                   6            5
## 3 Health expend~ SH.XPD.TOT~ France       FRA                  10.4         10.1
## 4 Health expend~ SH.XPD.TOT~ Spain        ESP                   7.44         7.21
## # ... with 1 more variable: X2005.YR2005 <dbl>
```

So far so good. Note that we're not making any changes yet so we can just add one function at a time to the pipeline and check the results. Once we're satisfied with the results we save them to a variable.

Step 2: The dataset contains maternal mortality and health expenditure rates but I really don't like the variable names and what use do I have for the series code? I want to rename and exclude some variables!

```
wdi %>%
  filter(Country.Code != "") %>%
    select(-Series.Code) %>%
  rename(Indicator = `¬Series.Name`)
```

```
## # A tibble: 4 x 6
##   Indicator     Country.Name Country.Code X1995.YR1995 X2000.YR2000 X2005.YR2005
##   <chr>         <chr>        <chr>                <dbl>        <dbl>        <dbl>
```

```
## 1 Maternal mor~ France       FRA                 15           12           10
## 2 Maternal mor~ Spain        ESP                  6            5            5
## 3 Health expen~ France       FRA                 10.4         10.1         10.9
## 4 Health expen~ Spain        ESP                  7.44         7.21         8.29
```

That looks already but we still have to tackle our main task: how do we get all data entries for different years into one row instead of 3 columns? We want to put all etnries in the columns starting with an X to be below each other. The function to achieve this is *pivot_longer()*. All we have to do is to specify the data and to select the columns we want to reformat. However, we still want to know which value of the world development index was achieved in which year. So, we also tell this to the function:

```
wdi %>%
  filter(Country.Code != "") %>%
    select(-Series.Code) %>%
  rename(Indicator = `¬Series.Name`) %>%
  pivot_longer(cols = starts_with("X"),
               names_to = "Year",
               values_to = "Development.Index")
```

```
## # A tibble: 12 x 5
##    Indicator          Country.Name Country.Code Year         Development.Index
##    <chr>              <chr>        <chr>        <chr>                    <dbl>
##  1 Maternal mortality France       FRA          X1995.YR1995              15
##  2 Maternal mortality France       FRA          X2000.YR2000              12
##  3 Maternal mortality France       FRA          X2005.YR2005              10
##  4 Maternal mortality Spain        ESP          X1995.YR1995               6
##  5 Maternal mortality Spain        ESP          X2000.YR2000               5
##  6 Maternal mortality Spain        ESP          X2005.YR2005               5
##  7 Health expenditure France       FRA          X1995.YR1995              10.4
##  8 Health expenditure France       FRA          X2000.YR2000              10.1
##  9 Health expenditure France       FRA          X2005.YR2005              10.9
## 10 Health expenditure Spain        ESP          X1995.YR1995               7.44
## 11 Health expenditure Spain        ESP          X2000.YR2000               7.21
## 12 Health expenditure Spain        ESP          X2005.YR2005               8.29
```

Thats already pretty close. The Year column looks ugly but that's an easy fix. We can use the `substring()` function to take all the characters from position `2` to `5` and assign it back to the `Year` column.

Since this is the last step we might as well assign the results to a new dataset.

```
wdi_long <- wdi %>%
  filter(Country.Code != "") %>%
    select(-Series.Code) %>%
  rename(Indicator = `¬Series.Name`) %>%
  pivot_longer(cols = starts_with("X"),
```

```
               names_to = "Year",
               values_to = "Development.Index") %>%
  mutate(Year = as.numeric(substring(Year, 2, 5)))

wdi_long
```

```
## # A tibble: 12 x 5
##    Indicator          Country.Name Country.Code  Year Development.Index
##    <chr>              <chr>        <chr>        <dbl>             <dbl>
##  1 Maternal mortality France       FRA           1995              15
##  2 Maternal mortality France       FRA           2000              12
##  3 Maternal mortality France       FRA           2005              10
##  4 Maternal mortality Spain        ESP           1995               6
##  5 Maternal mortality Spain        ESP           2000               5
##  6 Maternal mortality Spain        ESP           2005               5
##  7 Health expenditure France       FRA           1995              10.4
##  8 Health expenditure France       FRA           2000              10.1
##  9 Health expenditure France       FRA           2005              10.9
## 10 Health expenditure Spain        ESP           1995               7.44
## 11 Health expenditure Spain        ESP           2000               7.21
## 12 Health expenditure Spain        ESP           2005               8.29
```

Sometimes, but less often, you will also reshape data to a wide format. That can be achieved with the sibling function *pivot_wider()* in the tidyverse. Check out the online help for this function if you want to use it.

Chapter 7

Acknowledgments

Content of this workshop is based on the following:

- Altaf Ali's tutorial last year
- Introduction to dplyr
- Data manipulation with dplyr, 2014
- Hands-on dplyr tutorial for faster data manipulation in R