Data Manipulation in R

Contents

1 Introduction

Data manipulation is the process of cleaning, organising and preparing data in a way that makes it suitable for analysis. Most real-world datasets require some form of manipulation to facilitate the downstream analysis and this process is often repeated a number of times during the data analysis cycle. In this workshop you will learn how to apply a consistent grammar of data manipulation to raw data and prepare it for analysis. The following topics are covered in the workshop:

- Learning to use the grammar of data manipulation
- Merging multiple datasets and creating subsets using filters
- Reshaping data between long and wide formats
- Summarising data with group-wise operation
- Setting up data pipelines for efficient data manipulation

This workshop is designed for individuals who are already familiar with R but wish to learn efficient techniques for data manipulation. It is recommended that you bring your own laptop with the latest version of R and RStudio installed.

Last Updated: Nov 23, 2017 12:36 AM

2 Acknowledgments

Content of this workshop is based on the following:

- Introduction to dplyr
- Data manipulation with dplyr, 2014
- Hands-on dplyr tutorial for faster data manipulation in R

3 Resources

- Google
- Tidy Data
- Tidyverse
- Data Wrangling with dplyr and tidyr Cheat Sheet
- R for Data Science
- Advanced R
- Data manipulation with dplyr, 2014
- Introduction to dplyr
- Hands-on dplyr tutorial for faster data manipulation in R

4 Getting Started

4.1 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.

4.2 Software Requirements

4.2.1 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-pc-linux-gnu
arch           x86_64
os             linux-gnu
system         x86_64, linux-gnu
status
major          3
minor          4.2
year           2017
month          01
day            27
svn rev        73369
language       R
version.string R version 3.4.2 (2017-01-27)
nickname       Short Summer
```

4.3 Required Packages

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

4.3.1 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"      "dplyr"       "dbplyr"
 [6] "forcats"     "ggplot2"     "haven"       "hms"         "httr"
[11] "jsonlite"    "lubridate"   "magrittr"    "modelr"      "purrr"
[16] "readr"       "readxl\n(>=" "reprex"      "rlang"       "rstudioapi"
[21] "rvest"       "stringr"     "tibble"      "tidyr"       "xml2"
[26] "tidyverse"
```

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

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

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

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

```
[1] '1.2.1'
```

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

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

4.3.2 Option 2: Install Individual Packages

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

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

5 Basic Operations

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

```r
library(tidyverse)
```

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

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

5.1 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.

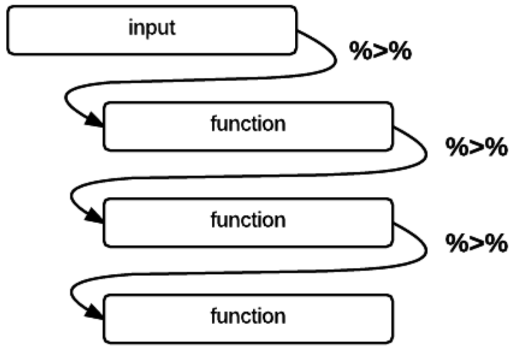

Figure 1:

5.2 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/altaf-ali/tidydata_tutorial/master/data/flights.c
weather <- read_csv("https://raw.githubusercontent.com/altaf-ali/tidydata_tutorial/master/data/weather.c
planes <- read_csv("https://raw.githubusercontent.com/altaf-ali/tidydata_tutorial/master/data/planes.cs
airports <- read_csv("https://raw.githubusercontent.com/altaf-ali/tidydata_tutorial/master/data/airport
```

Now let's examine the dataset

```
flights
```

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

```
weather
```

```
# A tibble: 8,723 x 14
         date  hour  temp dew_point humidity pressure visibility wind_dir
       <date> <int> <dbl>     <dbl>    <int>    <dbl>      <dbl>    <chr>
 1 2011-01-01     0  59.0      28.9       32    29.86         10      NNE
 2 2011-01-01     1  57.2      28.4       33    29.88         10      NNE
 3 2011-01-01     2  55.4      28.4       36    29.93         10      NNW
 4 2011-01-01     3  53.6      28.4       38    29.94         10    North
 5 2011-01-01     4    NA        NA       NA    29.99         10      NNW
 6 2011-01-01     5    NA        NA       NA    30.02         10    North
 7 2011-01-01     6  53.1      17.1       24    30.05         10    North
 8 2011-01-01     7  53.1      16.0       23    30.07         10    North
 9 2011-01-01     8  54.0      18.0       24    30.09         10    North
10 2011-01-01     9  55.4      17.6       23    30.09         10      NNE
# ... with 8,713 more rows, and 6 more variables: wind_dir2 <int>,
#   wind_speed <dbl>, gust_speed <dbl>, precip <dbl>, conditions <chr>,
#   events <chr>
```

```r
planes
```

```
# A tibble: 2,853 x 9
    plane  year               mfr           model no.eng no.seats speed
    <chr> <int>             <chr>           <chr>  <int>    <int> <int>
 1 N576AA  1991 MCDONNELL DOUGLAS DC-9-82(MD-82)      2      172    NA
 2 N557AA  1993        MARZ BARRY      KITFOX IV      1        2    NA
 3 N403AA  1974             RAVEN            S55A     NA        1    60
 4 N492AA  1989 MCDONNELL DOUGLAS DC-9-82(MD-82)      2      172    NA
 5 N262AA  1985 MCDONNELL DOUGLAS DC-9-82(MD-82)      2      172    NA
 6 N493AA  1989 MCDONNELL DOUGLAS DC-9-82(MD-82)      2      172    NA
 7 N477AA  1988 MCDONNELL DOUGLAS DC-9-82(MD-82)      2      172    NA
 8 N476AA  1988 MCDONNELL DOUGLAS DC-9-82(MD-82)      2      172    NA
 9 N504AA    NA AUTHIER ANTHONY P      TIERRA II      1        2    NA
10 N565AA  1987 MCDONNELL DOUGLAS DC-9-83(MD-83)      2      172    NA
# ... with 2,843 more rows, and 2 more variables: engine <chr>, type <chr>
```

```r
airports
```

```
# A tibble: 3,376 x 7
    iata              airport             city state country      lat
   <chr>                <chr>            <chr> <chr>   <chr>    <dbl>
 1   00M              Thigpen      Bay Springs    MS     USA 31.95376
 2   00R Livingston Municipal       Livingston    TX     USA 30.68586
 3   00V          Meadow Lake Colorado Springs    CO     USA 38.94575
 4   01G         Perry-Warsaw            Perry    NY     USA 42.74135
 5   01J     Hilliard Airpark         Hilliard    FL     USA 30.68801
 6   01M    Tishomingo County          Belmont    MS     USA 34.49167
 7   02A           Gragg-Wade          Clanton    AL     USA 32.85049
 8   02C              Capitol       Brookfield    WI     USA 43.08751
 9   02G    Columbiana County   East Liverpool    OH     USA 40.67331
10   03D     Memphis Memorial          Memphis    MO     USA 40.44726
# ... with 3,366 more rows, and 1 more variables: long <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.

5.3 Select

The `select` function is used to select columns.

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

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

```
# A tibble: 227,496 x 3
    dest  time  dist
   <chr> <int> <int>
 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> <int> <int>     <int>     <int>
 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> <int> <int>     <int>     <int>
 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> <int> <int>     <int>     <int>
 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
                <dttm> <int>  <int> <int> <int>
 1 2011-01-01 12:00:00    14      0  1400  1500
 2 2011-01-02 12:00:00    14      1  1401  1501
 3 2011-01-03 12:00:00    13     52  1352  1502
 4 2011-01-04 12:00:00    14      3  1403  1513
 5 2011-01-05 12:00:00    14      5  1405  1507
 6 2011-01-06 12:00:00    13     59  1359  1503
 7 2011-01-07 12:00:00    13     59  1359  1509
 8 2011-01-08 12:00:00    13     55  1355  1454
 9 2011-01-09 12:00:00    14     43  1443  1554
10 2011-01-10 12:00:00    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 13
                  date  hour minute   dep   arr dep_delay arr_delay
                <dttm> <int>  <int> <int> <int>     <int>     <int>
 1 2011-01-01 12:00:00    14      0  1400  1500         0       -10
 2 2011-01-02 12:00:00    14      1  1401  1501         1        -9
 3 2011-01-03 12:00:00    13     52  1352  1502        -8        -8
 4 2011-01-04 12:00:00    14      3  1403  1513         3         3
 5 2011-01-05 12:00:00    14      5  1405  1507         5        -3
```

```
 6 2011-01-06 12:00:00    13     59  1359  1503        -1        -7
 7 2011-01-07 12:00:00    13     59  1359  1509        -1        -1
 8 2011-01-08 12:00:00    13     55  1355  1454        -5       -16
 9 2011-01-09 12:00:00    14     43  1443  1554        43        44
10 2011-01-10 12:00:00    14     43  1443  1553        43        43
# ... with 227,486 more rows, and 6 more variables: carrier <chr>,
#   flight <int>, dest <chr>, cancelled <int>, time <int>, dist <int>
```

5.4 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 14
                  date  hour minute   dep   arr dep_delay arr_delay
                <dttm> <int>  <int> <int> <int>     <int>     <int>
 1 2011-01-31 12:00:00     7     35   735  1220         0         4
 2 2011-01-31 12:00:00    10     47  1047  1526        -3        -5
 3 2011-01-31 12:00:00    13      5  1305  1746         0        -3
 4 2011-01-31 12:00:00    19      1  1901  2332         6        -1
 5 2011-01-31 12:00:00    15     50  1550  2012         0       -25
 6 2011-01-30 12:00:00    10     46  1046  1518        -4        -8
 7 2011-01-30 12:00:00    13     19  1319  1811        14        22
 8 2011-01-30 12:00:00    19      9  1909    23        14        50
 9 2011-01-30 12:00:00    15     53  1553  2030         3        -7
10 2011-01-29 12:00:00     7     40   740  1227         5        16
# ... with 1,742 more rows, and 7 more variables: carrier <chr>,
#   flight <int>, dest <chr>, plane <chr>, cancelled <int>, time <int>,
#   dist <int>
```

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> <int> <int>
 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

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

```
# A tibble: 1,752 x 3
    dest  time  dist
   <chr> <int> <int>
 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.

5.4.1 Exercise

Find all flights that match the following conditions:

1. To SFO or OAK
2. In January
3. Delayed by more than an hour
4. Departed between midnight and 5am
5. Arrival delay more than twice the departure 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

5.5 Arrange

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

```r
flights %>%
  arrange(dest)
```

```
# A tibble: 227,496 x 14
                  date  hour minute   dep   arr dep_delay arr_delay
                <dttm> <int>  <int> <int> <int>     <int>     <int>
 1 2011-01-31 12:00:00    17     33  1733  1901        -2        -4
 2 2011-01-30 12:00:00    17     50  1750  1913        15         8
 3 2011-01-29 12:00:00    17     32  1732  1837        -3       -23
 4 2011-01-28 12:00:00    17     33  1733  1848        -2       -17
 5 2011-01-27 12:00:00    17     41  1741  1854         6       -11
 6 2011-01-26 12:00:00    17     32  1732  1853        -3       -12
 7 2011-01-25 12:00:00    17     29  1729  1858        -6        -7
 8 2011-01-24 12:00:00    17     34  1734  1845        -1       -20
 9 2011-01-23 12:00:00    17     35  1735  1853         0       -12
10 2011-01-22 12:00:00    17     33  1733  1843        -2       -17
# ... with 227,486 more rows, and 7 more variables: carrier <chr>,
#   flight <int>, dest <chr>, plane <chr>, cancelled <int>, time <int>,
#   dist <int>
```

5.5.1 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?

5.6 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` variable with the one that has a `speed` column

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

5.6.1 Exercise

1. Add a variable to show how much time was made up (or lost) during flight

5.7 Summarise

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

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

```
# A tibble: 365 x 2
                  date count
                <dttm> <int>
 1 2011-01-01 12:00:00   552
 2 2011-01-02 12:00:00   678
 3 2011-01-03 12:00:00   702
 4 2011-01-04 12:00:00   583
 5 2011-01-05 12:00:00   590
 6 2011-01-06 12:00:00   660
 7 2011-01-07 12:00:00   661
 8 2011-01-08 12:00:00   500
 9 2011-01-09 12:00:00   602
10 2011-01-10 12:00:00   659
# ... 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.

```r
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

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

delays
```

```
# A tibble: 116 x 2
    dest   mean
   <chr>  <dbl>
 1   ABQ     NA
 2   AEX     NA
 3   AGS 10.000
 4   AMA     NA
 5   ANC 24.952
 6   ASE     NA
 7   ATL     NA
 8   AUS     NA
 9   AVL     NA
10   BFL     NA
# ... with 106 more rows
```

5.7.1 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. Use help to find out two other ways to do summarize/n combination in dplyr.
4. How many different destinations can you fly to from Houston?
5. Which destinations have the highest average delays?
6. Which flights (carrier + flight number) happen everyday and where do they fly?
7. How do delays (of non-cancelled flights) vary over the course of a day?

5.8 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
 *              <dttm>   <chr>  <int>     <chr>
 1 2011-01-01 12:00:00      AA    428    AA-428
 2 2011-01-02 12:00:00      AA    428    AA-428
 3 2011-01-03 12:00:00      AA    428    AA-428
 4 2011-01-04 12:00:00      AA    428    AA-428
 5 2011-01-05 12:00:00      AA    428    AA-428
 6 2011-01-06 12:00:00      AA    428    AA-428
 7 2011-01-07 12:00:00      AA    428    AA-428
 8 2011-01-08 12:00:00      AA    428    AA-428
 9 2011-01-09 12:00:00      AA    428    AA-428
10 2011-01-10 12:00:00      AA    428    AA-428
# ... with 227,486 more rows
```

5.9 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 16
         date     time  hour minute   dep   arr dep_delay arr_delay
 *      <chr>    <chr> <int>  <int> <int> <int>     <int>     <int>
 1 2011-01-01 12:00:00    14      0  1400  1500         0       -10
 2 2011-01-02 12:00:00    14      1  1401  1501         1        -9
 3 2011-01-03 12:00:00    13     52  1352  1502        -8        -8
 4 2011-01-04 12:00:00    14      3  1403  1513         3         3
 5 2011-01-05 12:00:00    14      5  1405  1507         5        -3
 6 2011-01-06 12:00:00    13     59  1359  1503        -1        -7
```

```
 7 2011-01-07 12:00:00    13     59  1359  1509        -1        -1
 8 2011-01-08 12:00:00    13     55  1355  1454        -5       -16
 9 2011-01-09 12:00:00    14     43  1443  1554        43        44
10 2011-01-10 12:00:00    14     43  1443  1553        43        43
# ... with 227,486 more rows, and 8 more variables: carrier <chr>,
#   flight <int>, dest <chr>, plane <chr>, cancelled <int>, time <int>,
#   dist <int>, speed <dbl>
```

5.9.1 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

6 Merging Datasets

Let's start by loading the `tidyverse` package

```r
library(tidyverse)
```

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

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

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

```r
universities <- read_csv("https://raw.githubusercontent.com/altaf-ali/tidydata_tutorial/master/data/uni
cities <- read_csv("https://raw.githubusercontent.com/altaf-ali/tidydata_tutorial/master/data/cities.cs
states <- read_csv("https://raw.githubusercontent.com/altaf-ali/tidydata_tutorial/master/data/states.cs
```

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

universities

university	city
Cornell	Ithaca
Harvard	Cambridge
MIT	Cambridge
Yale	New Haven

cities

city	state
Cambridge	Massachusetts
Ithaca	New York
Seattle	Washington

6.1 Left Join

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

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

6.2 Right Join

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

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

6.3 Inner Join

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

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

6.4 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

6.5 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

6.6 Exercise

1. Load the following datasets:

```
presidents <- read_csv("https://raw.githubusercontent.com/altaf-ali/tidydata_tutorial/master/data/p
presidents_home <- read_csv("https://raw.githubusercontent.com/altaf-ali/tidydata_tutorial/master/d
```

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
Franklin	D.	Roosevelt	4/03/1933	12/4/1945	New York
John	Quincy	Adams	NA	NA	Massachusetts
William	Howard	Taft	NA	NA	Ohio
George	W.	Bush	20/01/2001	20/01/2009	Texas

7 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/altaf-ali/tidydata_tutorial/master/data/wdi.csv", na

wdi
```

```
# A tibble: 5 x 7
      `¬Series.Name`    Series.Code Country.Name Country.Code X1995.YR1995
               <chr>          <chr>        <chr>        <chr>        <dbl>
1 Maternal mortality    SH.STA.MMRT       France          FRA    15.000000
2 Maternal mortality    SH.STA.MMRT        Spain          ESP     6.000000
3 Maternal mortality    SH.STA.MMRT                                     NA
4 Health expenditure SH.XPD.TOTL.ZS       France          FRA    10.355906
5 Health expenditure SH.XPD.TOTL.ZS        Spain          ESP     7.444592
# ... with 2 more variables: X2000.YR2000 <dbl>, X2005.YR2005 <dbl>
```

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

```
# A tibble: 6 x 5
  CountryCode CountryName  Year MaternalMortality HealthExpenditure
        <chr>       <chr> <dbl>             <dbl>             <dbl>
1         ESP       Spain  1995                 6          7.444592
2         ESP       Spain  2000                 5          7.214756
```

```
3          ESP      Spain  2005                  5          8.288271
4          FRA     France  1995                 15         10.355906
5          FRA     France  2000                 12         10.084833
6          FRA     France  2005                 10         10.932626
```

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
               <chr>          <chr>        <chr>        <chr>        <dbl>
1 Maternal mortality    SH.STA.MMRT       France          FRA    15.000000
2 Maternal mortality    SH.STA.MMRT        Spain          ESP     6.000000
3 Health expenditure SH.XPD.TOTL.ZS       France          FRA    10.355906
4 Health expenditure SH.XPD.TOTL.ZS        Spain          ESP     7.444592
# ... with 2 more variables: X2000.YR2000 <dbl>, 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.

We need to gather all columns that start with "X" that contain per-year values for each series (for example X1960..YR1960)

```
wdi %>%
  filter(Country.Code != "") %>%
  gather(Year, Value, starts_with("X"))
```

```
# A tibble: 12 x 6
       `¬Series.Name`    Series.Code Country.Name Country.Code
                <chr>          <chr>        <chr>        <chr>
 1 Maternal mortality    SH.STA.MMRT       France          FRA
 2 Maternal mortality    SH.STA.MMRT        Spain          ESP
 3 Health expenditure SH.XPD.TOTL.ZS       France          FRA
 4 Health expenditure SH.XPD.TOTL.ZS        Spain          ESP
 5 Maternal mortality    SH.STA.MMRT       France          FRA
 6 Maternal mortality    SH.STA.MMRT        Spain          ESP
 7 Health expenditure SH.XPD.TOTL.ZS       France          FRA
 8 Health expenditure SH.XPD.TOTL.ZS        Spain          ESP
 9 Maternal mortality    SH.STA.MMRT       France          FRA
10 Maternal mortality    SH.STA.MMRT        Spain          ESP
11 Health expenditure SH.XPD.TOTL.ZS       France          FRA
12 Health expenditure SH.XPD.TOTL.ZS        Spain          ESP
# ... with 2 more variables: Year <chr>, Value <dbl>
```

Now all values are in the `Value` column, so we need to spread them out to individual columns based on the `Series.Code`. We have to make sure that we only keep the columns that make the country-year observations unique. We use `select()` to keep `Country.Code`, `Country.Name`, `Year`, plus the two columns (`Series.Code` and `Value`) that will make up the key-value pair for the `spread()` function.

```r
wdi %>%
  filter(Country.Code != "") %>%
  gather(Year, Value, starts_with("X")) %>%
  select(Country.Code, Country.Name, Year, Series.Code, Value) %>%
  spread(Series.Code, Value)
```

```
# A tibble: 6 x 5
  Country.Code Country.Name         Year SH.STA.MMRT SH.XPD.TOTL.ZS
*        <chr>        <chr>        <chr>       <dbl>          <dbl>
1          ESP        Spain X1995.YR1995           6       7.444592
2          ESP        Spain X2000.YR2000           5       7.214756
3          ESP        Spain X2005.YR2005           5       8.288271
4          FRA       France X1995.YR1995          15      10.355906
5          FRA       France X2000.YR2000          12      10.084833
6          FRA       France X2005.YR2005          10      10.932626
```

It looks good, so we can rename the variables to something meaningful.

```r
wdi %>%
  filter(Country.Code != "") %>%
  gather(Year, Value, starts_with("X")) %>%
  select(Country.Code, Country.Name, Year, Series.Code, Value) %>%
  spread(Series.Code, Value) %>%
  rename(CountryName = Country.Name,
         CountryCode = Country.Code,
         MaternalMortality = SH.STA.MMRT,
         HealthExpenditure = SH.XPD.TOTL.ZS)
```

```
# A tibble: 6 x 5
  CountryCode CountryName         Year MaternalMortality HealthExpenditure
*       <chr>       <chr>        <chr>             <dbl>             <dbl>
1         ESP       Spain X1995.YR1995                 6          7.444592
2         ESP       Spain X2000.YR2000                 5          7.214756
3         ESP       Spain X2005.YR2005                 5          8.288271
4         FRA      France X1995.YR1995                15         10.355906
5         FRA      France X2000.YR2000                12         10.084833
6         FRA      France X2005.YR2005                10         10.932626
```

Now we just need to extract the 4-digit year from the `Year` column. The `Year` column is formatted as `X1995.YR1995` which means that the 4-digits for the year are in position 2,3,4, and 5. 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 variable.

```r
wdi_long <- wdi %>%
  filter(Country.Code != "") %>%
  gather(Year, Value, starts_with("X")) %>%
  select(Country.Code, Country.Name, Year, Series.Code, Value) %>%
  spread(Series.Code, Value) %>%
  rename(CountryName = Country.Name,
         CountryCode = Country.Code,
         MaternalMortality = SH.STA.MMRT,
```

```r
         HealthExpenditure = SH.XPD.TOTL.ZS) %>%
  mutate(Year = as.numeric(substring(Year, 2, 5)))

wdi_long
```

```
# A tibble: 6 x 5
  CountryCode CountryName  Year MaternalMortality HealthExpenditure
        <chr>       <chr> <dbl>             <dbl>             <dbl>
1         ESP       Spain  1995                 6          7.444592
2         ESP       Spain  2000                 5          7.214756
3         ESP       Spain  2005                 5          8.288271
4         FRA      France  1995                15         10.355906
5         FRA      France  2000                12         10.084833
6         FRA      France  2005                10         10.932626
```

You can assign it back to `wdi` if you want, but we're using a different name in case we make a mistake and have to start again. This way we would've have to reload the file all over again.