Quantitative social science with R

Get and manipulate data

Edu Gonzalo Almorox

18/10/2017

Quantitative social science with R

Get and manipulate data

Edu Gonzalo Almorox

18/10/2017

Outline

1. Import data
    - .csv
    - .excel
2. Types of data
    - factors
    - strings
    - dates
3. Manipulate data
    - dplyr
    - tidyr

Import data into R

Getting data

- There are two possible ways to input information:
  - **Manually**
  - **Import from somewhere**
- The majority of the analyses consist of data created externally
  - Data are delivered in different **formats**
  - Important to understand how information is **structured**

Step 1: Get started

The first commands consist of the following instructions intended to locate the project in the computer and load the set of functions needed for the analysis

- Working directory

```
# working directory
setwd("/Users/my-file/my_document")
```

- Packages

```
install.packages() # for installing packages
```

- Library

```
library() # for loading libraries
```

Step 2: Load your data

- Various packages and libraries that allow for this
- Use depends on the type of data to be loaded

Function	Package	Format
`read.csv()`	`base`	Comma separated values
`read.dta()`	`foreign`	Stata files
`import()`	`rio`	All types
`read_csv()`	`readr`	Comma separated values
`read_xlsx()`	`readr`	Excel type
`read_excel()`	`readxl`	Excel files
`read_xml()`	`xml2`	XML files

Load data: csv files

- Comma separated files (`csv`) are quite common
- Values are separated by commas
- Base solution

```
# load data including working directory
my_df = read.csv("data/data_intro_r.csv",
                 sep = ",")

head(my_df)
```

```
##   location.id location.region       date jsa.fem
## 1 1-125058834 East of England 2010-10-01      55
## 2 1-125058834 East of England 2010-12-21      55
## 3 1-125058834 East of England       <NA>      55
## 4 1-125058834 East of England       <NA>      55
## 5 1-118532985 East of England 2010-10-01      40
## 6 1-118532985 East of England 2011-01-06      40
```

Load data: csv files

- There are other functions: `import()`...

```r
# load data including working directory
library(rio)
my_df_import = import("data/data_intro_r.csv")

head(my_df_import, 3)
```

```
##   location.id location.region       date jsa.fem
## 1 1-125058834 East of England 2010-10-01      55
## 2 1-125058834 East of England 2010-12-21      55
## 3 1-125058834 East of England       <NA>      55
```

Load data: csv files

- ... and `read_csv()`

```r
# load data including working directory

library(readr)
my_df_readr = read_csv("data/data_intro_r.csv")

head(my_df_readr)
```

```
## # A tibble: 6 x 4
##   location.id location.region       date jsa.fem
##         <chr>           <chr>     <date>   <int>
## 1 1-125058834 East of England 2010-10-01      55
## 2 1-125058834 East of England 2010-12-21      55
## 3 1-125058834 East of England         NA      55
## 4 1-125058834 East of England         NA      55
## 5 1-118532985 East of England 2010-10-01      40
## 6 1-118532985 East of England 2011-01-06      40
```

- Faster

- More flexible to read different types of variable (e.g. dates, times, currencies...)

Load data: excel files

- Excel files have been tedious to parse in R
- `readxl` works very well for this

```
# load data including working directory

library(readxl)
my_excel = read_excel("data/r_intro.xlsx")


head(my_excel, 5)
```

```
## # A tibble: 5 x 13
##                              `benefit payments - pensio
##
## 1 ONS Crown Copyright Reserved [from Nomis on 14 Septe
## 2
## 3
## 4
## 5
## # ... with 12 more variables: X__1 <chr>, X__2 <chr>, 
## #   X__4 <chr>, X__5 <chr>, X__6 <chr>, X__7 <chr>, X_
## #   X__9 <chr>, X__10 <chr>, X__11 <chr>, X__12 <chr>
```

Load data: excel files

- Refine the information that can be loaded controlling for sheets, rows and columns
    - **sheet names**

```
# name of the sheet
library(readxl)
my_excel = read_excel("data/r_intro.xlsx", sheet

head(my_excel, 3)
```

```
## # A tibble: 3 x 13
##                              `benefit payments - pensio
##
## 1 ONS Crown Copyright Reserved [from Nomis on 14 Septe
## 2
## 3
## # ... with 11 more variables: X__2 <chr>, X__3 <chr>, 
## #   X__5 <chr>, X__6 <chr>, X__7 <chr>, X__8 <chr>, X_
## #   X__10 <chr>, X__11 <chr>, X__12 <chr>
```

Load data: excel files

- Refine the information that can be loaded controlling for sheets, rows and columns
  - **sheet position**

```
# name of the sheet
library(readxl)
my_excel = read_excel("data/r_intro.xlsx", sheet

head(my_excel, 2)
```

```
## # A tibble: 2 x 13
##                              `benefit payments - pensio
##
## 1 ONS Crown Copyright Reserved [from Nomis on 14 Septe
## 2
## # ... with 11 more variables: X__2 <chr>, X__3 <chr>, X
## #   X__5 <chr>, X__6 <chr>, X__7 <chr>, X__8 <chr>, X_
## #   X__10 <chr>, X__11 <chr>, X__12 <chr>
```

Load data: excel files

- Refine the information that can be loaded controlling for sheets, rows and columns
    - **rows**

```
# rows
library(readxl)
my_excel_clean = read_excel("data/r_intro.xlsx",

head(my_excel_clean, 3)
```

```
## # A tibble: 3 x 13
##
##
## 1 local authority: district / unitary (prior to April 
## 2                                                    Ba
## 3                                                   Bas
## # ... with 11 more variables: X__3 <chr>, X__4 <chr>, 
## #   X__6 <chr>, X__7 <chr>, X__8 <chr>, X__9 <chr>, X_
## #   X__11 <chr>, X__12 <chr>, X__13 <chr>
```

Load data: excel files

- Refine the information that can be loaded controlling for sheets, rows and columns
    - **columns**

```
library(readxl)

# columns

my_excel_clean_cols = read_excel("data/r_intro.xl

head(my_excel_clean_cols, 3)
```

```
## # A tibble: 3 x 10
##   `August 2014` `November 2014` `February 2015` `May 20
##           <dbl>           <dbl>           <dbl>      <d
## 1          2900            2860            2790
## 2          6210            6150            6010
## 3          4580            4530            4430
## # ... with 5 more variables: `November 2015` <dbl>, `Fe
## #   `May 2016` <dbl>, `August 2016` <dbl>, `November 20
```

Load data: excel files

- Refine the information that can be loaded controlling for sheets, rows and columns
    - **range**

```r
library(readxl)

# range
my_excel_clean_range = read_excel("data/r_intro.x

head(my_excel_clean_range, 3)
```

```
## # A tibble: 3 x 13
##   `local authority: district / unitary (prior to April
##
## 1                                                      
## 2                                                     B
## 3                                                      
## # ... with 11 more variables: `August 2014` <dbl>, `No
## #   `February 2015` <dbl>, `May 2015` <dbl>, `August 2
## #   `November 2015` <dbl>, `February 2016` <dbl>, `May
## #   `August 2016` <dbl>, `November 2016` <dbl>, `Febru
```

Exercise

Load information corresponding to 2015

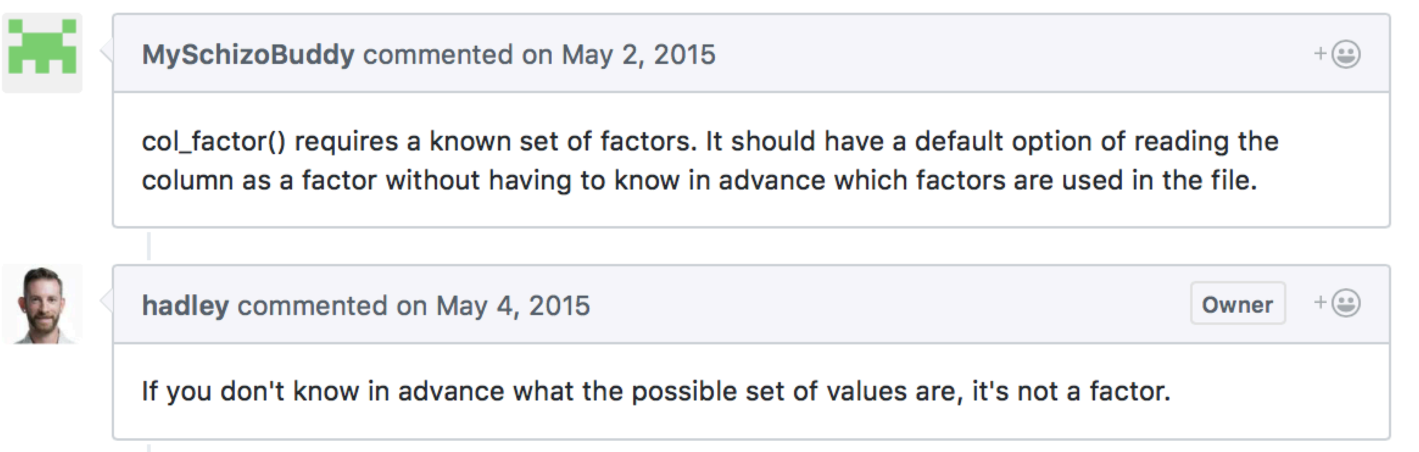
```
## # A tibble: 6 x 4
##   `February 2015` `May 2015` `August 2015` `November 20
##             <dbl>      <dbl>         <dbl>            <d
## 1            2790       2690          2640            1
## 2            6010       5760          5670            5
## 3            4430       4260          4210            4
## 4            4530       4330          4270            4
## 5            4950       4740          4670            4
## 6            1740       1660          1640            1
```

Types of data

factors

Data types: factors

- Factors refer to variables that represent different categories: gender, labour status, being treated or not, etc...
- What do we mean by categories?
    - fixed and known set of possible values
    - order in some cases



MySchizoBuddy commented on May 2, 2015

col_factor() requires a known set of factors. It should have a default option of reading the column as a factor without having to know in advance which factors are used in the file.

hadley commented on May 4, 2015 Owner

If you don't know in advance what the possible set of values are, it's not a factor.

Data types: factors

```
##   location.id location.region       date jsa.fem
## 1 1-125058834 East of England 2010-10-01      55
## 2 1-125058834 East of England 2010-12-21      55
## 3 1-125058834 East of England       <NA>      55
## 4 1-125058834 East of England       <NA>      55
## 5 1-118532985 East of England 2010-10-01      40
## 6 1-118532985 East of England 2011-01-06      40
## 7 1-118532985 East of England       <NA>      40
```

Data types: factors

```
glimpse(my_data)
```

```
## Observations: 97,416
## Variables: 4
## $ location.id     <chr> "1-125058834", "1-125058834",
## $ location.region <chr> "East of England", "East of En
## $ date            <date> 2010-10-01, 2010-12-21, NA, N
## $ jsa.fem         <int> 55, 55, 55, 55, 40, 40, 40, 40
```

Data types: factors

- How can we transform `characters` into `factors`?
- Base solution: combination of `$` and `as.factor()`

```
my_data$location.id = as.factor(my_data$location.

glimpse(my_data)
```

```
## Observations: 97,416
## Variables: 4
## $ location.id     <fctr> 1-125058834, 1-125058834, 1-1
## $ location.region <chr> "East of England", "East of En
## $ date            <date> 2010-10-01, 2010-12-21, NA, N
## $ jsa.fem         <int> 55, 55, 55, 55, 40, 40, 40, 40
```

- We will see other alternatives further on

Data types: factors

- The categories of the factor can be retrieve with `levels()`
- *What are the regions of the locations?*

```
# transform into a factor
my_data$location.region = as.factor(my_data$locat

# get level
levels(my_data$location.region)
```

```
##  [1] "East Midlands"            "East of England"
##  [3] "London"                   "North East"
##  [5] "North West"               "South East"
##  [7] "South West"               "Unspecified"
##  [9] "West Midlands"            "Yorkshire and The Huml
```

Data types: factors

- Levels can be renamed
- *Define "Unspecified" as "no_name"*

```
library(forcats)

# transform into a factor
my_data$location.region = as.factor(my_data$locat

# get region levels
regions = levels(my_data$location.region)

fct_recode(regions, no_name = "Unspecified")
```

```
##  [1] East Midlands            East of England
##  [3] London                   North East
##  [5] North West               South East
##  [7] South West               no_name
##  [9] West Midlands            Yorkshire and The Humber
## 10 Levels: East Midlands East of England London North 
```

Data types: factors

- Levels can be summarised
- *How many observations do we have in each region?*

```
table(my_data$location.region)
```

```
##
##            East Midlands          East of England
##                     9100                    10180
##               North East               North West
##                     5560                    11508
##               South West              Unspecified
##                    12928                       16
## Yorkshire and The Humber
##                     9188
```

Exercise

Create a variable called `regions.recoded` where "Unspecified" regions are recoded as "No Name"

```
##   location.id location.region       date jsa.fem regio
## 1 1-125058834 East of England 2010-10-01      55 East o
## 2 1-125058834 East of England 2010-12-21      55 East o
## 3 1-125058834 East of England       <NA>      55 East o
## 4 1-125058834 East of England       <NA>      55 East o
## 5 1-118532985 East of England 2010-10-01      40 East o
## 6 1-118532985 East of England 2011-01-06      40 East o
```

```
## 
##          East Midlands          East of England 
##                   9100                    10180 
##             North East               North West 
##                   5560                    11508 
##             South West                  No name 
##                  12928                       16 
## Yorkshire and The Humber 
##                   9188 
```

Types of data

strings

Data types: strings

- Text can be analysed as data
- Text data are becoming more frequent: tweets, reviews, news…
- Text may appear in your data
  - remove a given character in the names of your variables
  - replace a given character in your data
  - extact a given character in your data

Data types: strings

- Strings can be manipulated in multiple ways
- stringr package is a complete tool

Function	Description
str_c()	string concatenation
str_length()	number of characters
str_sub()	extracts substrings
str_dup()	duplicates characters
str_trim()	removes leading and trailing whitespace pads a string

Source: *"Handling and Processing Strings in R"* (Sánchez, 2014)

Data types: strings

- Create a new variable that includes a particular string to a variable - for example `"region_"`
- Make it lower case

```
library(stringr)

# transform and add the string

my_data$new_region = str_c("region", my_data$loca

# make it lower

my_data$new_region = tolower(my_data$new_region)

head(my_data$new_region)
```

```
## [1] "region_east of england" "region_east of england"
## [3] "region_east of england" "region_east of england"
## [5] "region_east of england" "region_east of england"
```

Data types: strings

```
glimpse(my_data)
```

```
## Observations: 97,416
## Variables: 6
## $ location.id     <fctr> 1-125058834, 1-125058834, 1-12
## $ location.region <fctr> East of England, East of Engla
## $ date            <date> 2010-10-01, 2010-12-21, NA, NA
## $ jsa.fem         <int> 55, 55, 55, 55, 40, 40, 40, 40
## $ regions.recoded <fctr> East of England, East of Engla
## $ new_region      <chr> "region_east of england", "reg
```

Data types: strings

- Extract `region_` from the variable `new_region`

```
library(stringr)

my_data$new_region2 = str_sub(my_data$new_region,

head(my_data$new_region2, 10)
```

```
##  [1] "region_" "region_" "region_" "region_" "region_"
##  [8] "region_" "region_" "region_"
```

- Drop `"_"` from `new_region` variable

```
library(stringr)

my_data$new_region2 = gsub("_", " ", my_data$new_

head(my_data$new_region2)
```

```
## [1] "region east of england" "region east of england"
## [3] "region east of england" "region east of england"
## [5] "region east of england" "region east of england"
```

Data types: strings

- Identify those observations whose region contains the word `East`

```
my_data$region.east = str_detect(my_data$location

table(my_data$region.east)
```

```
##
## FALSE  TRUE
## 54164 43252
```

- Extract `East` from the names of the regions

```
my_data$region.east2 = as.factor(str_extract(my_d

summary(my_data$region.east2)
```

```
##  East  NA's
## 43252 54164
```

Types of data

dates and times

Data types: dates and times

- Dates can be considered differently depending on various issues
  - is there information on time? *"2002-06-09 12:45:40"*
  - does it have time zones? (POSIXct and POSIXlt classes)

```
dates <- c("02/27/92", "02/27/92", "01/14/92", "0
times <- c("23:03:20", "22:29:56", "01:03:30", "1
x <- data.frame(dates, times, dates_times = paste
```

- `lubridate` allows for great flexibility when dealing with both types of data.

Data types: dates

- Transform to date format: `mdy_hms()`

```
library(lubridate)
str(x)
```

```
## 'data.frame':    5 obs. of  3 variables:
##  $ dates      : Factor w/ 4 levels "01/14/92","02/01/9
##  $ times      : Factor w/ 5 levels "01:03:30","16:56:2
##  $ dates_times: Factor w/ 5 levels "01/14/92 01:03:30"
```

```
# transform to "date format"

x$new_datetime = mdy_hms(x$dates_times)
str(x)
```

```
## 'data.frame':    5 obs. of  4 variables:
##  $ dates       : Factor w/ 4 levels "01/14/92","02/01/
##  $ times       : Factor w/ 5 levels "01:03:30","16:56:
##  $ dates_times : Factor w/ 5 levels "01/14/92 01:03:30
##  $ new_datetime: POSIXct, format: "1992-02-27 23:03:20
```

Data types: dates

- Extract relevant information - e.g: *hours*, *day of the week*

```
library(lubridate)

# hours
x$hour = hour(x$new_datetime)


# week-day to "date format"
x$weekday = wday(x$new_datetime, label = TRUE)

x
```

```
##      dates    times       dates_times        new_datet
## 1 02/27/92 23:03:20 02/27/92 23:03:20 1992-02-27 23:03
## 2 02/27/92 22:29:56 02/27/92 22:29:56 1992-02-27 22:29
## 3 01/14/92 01:03:30 01/14/92 01:03:30 1992-01-14 01:03
## 4 02/28/92 18:21:03 02/28/92 18:21:03 1992-02-28 18:21
## 5 02/01/92 16:56:26 02/01/92 16:56:26 1992-02-01 16:56
```

Data types: dates

- Arithmetic operations
    - what´s the average date and time?
    - what are the max and min date and time?

```
library(lubridate)

summary(x$new_datetime)
```

```
##                   Min.                 1st Qu.
## "1992-01-14 01:03:30" "1992-02-01 16:56:26" "1992-02-2
##                   Mean                 3rd Qu.
## "1992-02-13 21:10:51" "1992-02-27 23:03:20" "1992-02-2
```

Manipulation of data

dplyr and tidyr

Data manipulation: dplyr

- `dplyr` is the backbone of the grammar for data manipulation
- Compatibility with the pipes: `%>%`
- Main functions associated with different tasks

Function	Description
mutate()	adds new variables that are functions of existing variables
select()	picks variables based on their names.
filter()	picks cases based on their values.
summarise()	reduces multiple values down to a single summary.
arrange()	changes the ordering of the rows.

Source: tidyverse

Data manipulation: dplyr

- Working example
    - create a variable that reflects the level of female unemployment in the region of the location. Less than 30 claimants is below the average, 30-35 is in the average and more than 35 is above the average.

```
library(rio)
library(dplyr)

my_data = import("data/data_intro_r.csv")

my_data = my_data %>% mutate(unemp_level = ifelse
                                  ifelse(jsa.fem
                                  ifelse(jsa.fem

head(my_data,4)
```

```
##   location.id location.region       date jsa.fem unemp_
## 1 1-125058834 East of England 2010-10-01      55
## 2 1-125058834 East of England 2010-12-21      55
## 3 1-125058834 East of England       <NA>      55
## 4 1-125058834 East of England       <NA>      55
```

Data manipulation: dplyr

- Select locations in the South East and South West and order by date.

```
library(dplyr)

regions_south = c("South East", "South West")

my_data_south = my_data %>%
  filter(location.region %in% regions_south) %>%
  arrange(date)

head(my_data_south,7)
```

```
##   location.id location.region       date jsa.fem unemp
## 1       RX229      South East 2010-04-01      10
## 2       RX2Y5      South East 2010-04-01      25
## 3       RXXDL      South East 2010-04-01      10
## 4       RXXDM      South East 2010-04-01      10
## 5       RXXY3      South East 2010-04-01      10
## 6       RXXEC      South East 2010-04-01      30
## 7       RXXY2      South East 2010-04-01       5
```

Data manipulation: dplyr

- What´s the average, max and min number of claimants in each region?

```
sum_jsa = my_data %>%
  group_by(location.region) %>%
  summarise(mean.jsa = mean(jsa.fem),
            min.jsa = min(jsa.fem),
            max.jsa = max(jsa.fem))

sum_jsa
```

```
## # A tibble: 10 x 4
##             location.region mean.jsa min.jsa max.jsa
##                       <chr>    <dbl>   <dbl>   <dbl>
##  1            East Midlands 38.54505       0     200
##  2          East of England 33.36542       0     250
##  3                   London 42.06765       0     160
##  4               North East 51.70144       0     265
##  5               North West 39.80883       0     320
##  6               South East 26.16772       0     375
##  7               South West 27.69028       0     210
##  8              Unspecified 13.75000       5      25
##  9            West Midlands 43.99924       0     235
## 10 Yorkshire and The Humber 45.05660       0     270
```

Data manipulation: dplyr

```
sum_jsa = my_data %>%
  group_by(location.region) %>%
  summarise(mean.jsa = mean(jsa.fem),
            min.jsa = min(jsa.fem),
            max.jsa = max(jsa.fem)) %>%
  filter(location.region != "Unspecified")

sum_jsa
```

```
## # A tibble: 9 x 4
##            location.region mean.jsa min.jsa max.jsa
##                      <chr>    <dbl>   <dbl>   <dbl>
## 1            East Midlands 38.54505       0     200
## 2          East of England 33.36542       0     250
## 3                   London 42.06765       0     160
## 4               North East 51.70144       0     265
## 5               North West 39.80883       0     320
## 6               South East 26.16772       0     375
## 7               South West 27.69028       0     210
## 8            West Midlands 43.99924       0     235
## 9 Yorkshire and The Humber 45.05660       0     270
```

Data manipulation: dplyr

- Change the types of variables

```
glimpse(my_data)
```

```
## Observations: 97,416
## Variables: 5
## $ location.id     <chr> "1-125058834", "1-125058834", "
## $ location.region <chr> "East of England", "East of Eng
## $ date            <chr> "2010-10-01", "2010-12-21", NA
## $ jsa.fem         <int> 55, 55, 55, 55, 40, 40, 40, 40
## $ unemp_level     <chr> "high", "high", "high", "high"
```

- `location.region` and `unemp_level` are represent categories. `date` represents dates.

```
my_data =  my_data %>%
  mutate_at(vars(location.region, unemp_level), f
  mutate_at(vars(date), funs(as.Date))
```

Data manipulation: dplyr

```
glimpse(my_data)
```

```
## Observations: 97,416
## Variables: 5
## $ location.id     <chr> "1-125058834", "1-125058834", 
## $ location.region <fctr> East of England, East of Engl
## $ date            <date> 2010-10-01, 2010-12-21, NA, N
## $ jsa.fem         <int> 55, 55, 55, 55, 40, 40, 40, 40
## $ unemp_level     <fctr> high, high, high, high, high,
```

Data manipulation: dplyr

- `dplyr` has functions for linking datasets

Function	Description
inner(join)	return all rows from x where there are matching values in y, and all columns from x and y
left_join()	return all rows from x, and all columns from x and y
right_join()	return all rows from y, and all columns from x and y
semi_join()	return all rows from x where there are matching values in y, keeping just columns from x
anti_join()	return all rows from x where there are not matching values in y, keeping just columns from x
full_join()	return all rows and all columns from both x and y

Data manipulation: dplyr

- Linking two datasets

```
library(dplyr)
library(rio)

d1  = import("data/pop_link.csv")

d2 = import("data/claim_link.csv")
```

Data manipulation: dplyr

```
glimpse(d1)
```

```
## Observations: 326
## Variables: 8
## $ `Local Authority` <chr> "Babergh", "Basildon", "Bedf
## $ oslaua            <chr> "E07000200", "E07000066", "E
## $ `All Ages`        <int> 88845, 180521, 163924, 14998
## $ `Aged 65-69`      <int> 6947, 9808, 8600, 9443, 9917
## $ `Aged 70-74`      <int> 5045, 6816, 6223, 6461, 7249
## $ `Aged 75-79`      <int> 3907, 5850, 5177, 4864, 5977
## $ `Aged 80-84`      <int> 2856, 4493, 4000, 3561, 4358
## $ `Aged 85+`        <int> 2961, 3844, 3910, 3886, 4225
```

```
glimpse(d2)
```

```
## Observations: 326
## Variables: 6
## $ `Local Authority` <chr> "Babergh", "Basildon", "Bedf
## $ code_la           <chr> "E07000200", "E07000066", "E
## $ `2014`            <int> 88845, 180521, 163924, 14998
## $ `2015`            <int> 88990, 181859, 166167, 15094
## $ `2016`            <int> 89237, 183308, 168303, 15197
## $ `2017`            <int> 89549, 184789, 170394, 15303
```

Data manipulation: dplyr

- Add information from d2 to d1

```
new_data = left_join(d1, d2, by = c("Local Author

glimpse(new_data)
```

```
## Observations: 326
## Variables: 13
## $ `Local Authority` <chr> "Babergh", "Basildon", "Bedf
## $ oslaua            <chr> "E07000200", "E07000066", "E
## $ `All Ages`        <int> 88845, 180521, 163924, 14998
## $ `Aged 65-69`      <int> 6947, 9808, 8600, 9443, 9917
## $ `Aged 70-74`      <int> 5045, 6816, 6223, 6461, 7249
## $ `Aged 75-79`      <int> 3907, 5850, 5177, 4864, 5977
## $ `Aged 80-84`      <int> 2856, 4493, 4000, 3561, 4358
## $ `Aged 85+`        <int> 2961, 3844, 3910, 3886, 4225
## $ code_la           <chr> "E07000200", "E07000066", "E
## $ `2014`            <int> 88845, 180521, 163924, 14998
## $ `2015`            <int> 88990, 181859, 166167, 15094
## $ `2016`            <int> 89237, 183308, 168303, 15197
## $ `2017`            <int> 89549, 184789, 170394, 15303
```

Data manipulation: dplyr

- Add information from d2 to d1

```
new_data = left_join(d1, d2,
                     by = c("Local Authority" = "
                            "oslaua" = "c

glimpse(new_data)
```

```
## Observations: 326
## Variables: 12
## $ `Local Authority` <chr> "Babergh", "Basildon", "Bedf
## $ oslaua            <chr> "E07000200", "E07000066", "E
## $ `All Ages`        <int> 88845, 180521, 163924, 14998
## $ `Aged 65-69`      <int> 6947, 9808, 8600, 9443, 9917
## $ `Aged 70-74`      <int> 5045, 6816, 6223, 6461, 7249
## $ `Aged 75-79`      <int> 3907, 5850, 5177, 4864, 5977
## $ `Aged 80-84`      <int> 2856, 4493, 4000, 3561, 4358
## $ `Aged 85+`        <int> 2961, 3844, 3910, 3886, 4225
## $ `2014`            <int> 88845, 180521, 163924, 14998
## $ `2015`            <int> 88990, 181859, 166167, 15094
## $ `2016`            <int> 89237, 183308, 168303, 15197
## $ `2017`            <int> 89549, 184789, 170394, 15303
```

Data manipulation: tidyr

- Tidyr is helpful for creating tidy datasets

```
- each variable is in a column

- each observation is in a row

- each value in a cell.
```

- Useful for reshaping data from wide to long formats and also for visualisations

```
- gather(): it makes “wide” data longer

- spread(): it makes “long” data wider
```

Data manipulation: tidyr

- Working example
  - transform a wide data frame into a long

```
library(rio)
library(tidyr)
library(dplyr)

d1  = import("data/pop_link.csv")

head(d1)
```

```
##    Local Authority    oslaua All Ages Aged 65-69 Aged 7(
## 1          Babergh E07000200    88845       6947        !
## 2         Basildon E07000066   180521       9808        (
## 3          Bedford E06000055   163924       8600        (
## 4        Braintree E07000067   149985       9443        (
## 5        Breckland E07000143   133986       9917        ]
## 6        Brentwood E07000068    75645       4464        :
##    Aged 80-84 Aged 85+
## 1        2856     2961
## 2        4493     3844
## 3        4000     3910
## 4        3561     3886
## 5        4358     4225
## 6        2368     2361
```

Data manipulation: tidyr

```
d1 %>% gather(age, number, `All Ages`: `Aged 85+`
```

```
##                     Local Authority   oslaua        age
## 1                           Babergh E07000200   All Ages
## 2                          Basildon E07000066   All Ages
## 3                           Bedford E06000055   All Ages
## 4                         Braintree E07000067   All Ages
## 5                         Breckland E07000143   All Ages
## 6                         Brentwood E07000068   All Ages
## 7                         Broadland E07000144   All Ages
## 8                        Broxbourne E07000095   All Ages
## 9                         Cambridge E07000008   All Ages
## 10                     Castle Point E07000069   All Ages
## 11           Central Bedfordshire E06000056   All Ages
## 12                       Chelmsford E07000070   All Ages
## 13                       Colchester E07000071   All Ages
## 14                          Dacorum E07000096   All Ages
## 15              East Cambridgeshire E07000009   All Ages
## 16               East Hertfordshire E07000242   All Ages
## 17                    Epping Forest E07000072   All Ages
## 18                          Fenland E07000010   All Ages
## 19                    Forest Heath E07000201   All Ages
## 20                   Great Yarmouth E07000145   All Ages
## 21                           Harlow E07000073   All Ages
## 22                        Hertsmere E07000098   All Ages
## 23                  Huntingdonshire E07000011   All Ages
## 24                          Ipswich E07000202   All Ages
## 25   King`s Lynn and West Norfolk E07000146   All Ages
## 26                            Luton E06000032   All Ages
## 27                           Maldon E07000074   All Ages
## 28                      Mid Suffolk E07000203   All Ages
## 29              North Hertfordshire E07000099   All Ages
## 30                    North Norfolk E07000147   All Ages
## 31                          Norwich E07000148   All Ages
## 32                     Peterborough E06000031   All Ages
```

Data manipulation: tidyr

- `tidyr` and `dplyr`

```
d1 %>% gather(age, number, `All Ages`: `Aged 85+
 arrange(`Local Authority`)
```

```
##                  Local Authority   oslaua         age
## 1                           Adur E07000223   All Ages
## 2                           Adur E07000223 Aged 65-69
## 3                           Adur E07000223 Aged 70-74
## 4                           Adur E07000223 Aged 75-79
## 5                           Adur E07000223 Aged 80-84
## 6                           Adur E07000223   Aged 85+
## 7                      Allerdale E07000026   All Ages
## 8                      Allerdale E07000026 Aged 65-69
## 9                      Allerdale E07000026 Aged 70-74
## 10                     Allerdale E07000026 Aged 75-79
## 11                     Allerdale E07000026 Aged 80-84
## 12                     Allerdale E07000026   Aged 85+
## 13                  Amber Valley E07000032   All Ages
## 14                  Amber Valley E07000032 Aged 65-69
## 15                  Amber Valley E07000032 Aged 70-74
## 16                  Amber Valley E07000032 Aged 75-79
## 17                  Amber Valley E07000032 Aged 80-84
## 18                  Amber Valley E07000032   Aged 85+
## 19                          Arun E07000224   All Ages
## 20                          Arun E07000224 Aged 65-69
## 21                          Arun E07000224 Aged 70-74
## 22                          Arun E07000224 Aged 75-79
## 23                          Arun E07000224 Aged 80-84
## 24                          Arun E07000224   Aged 85+
## 25                     Ashfield E07000170   All Ages
## 26                     Ashfield E07000170 Aged 65-69
## 27                     Ashfield E07000170 Aged 70-74
## 28                     Ashfield E07000170 Aged 75-79
## 29                     Ashfield E07000170 Aged 80-84
```

Exercise

- Create a data frame that contains information on the district regarding the number of inhabitants associated with each age range and the information corresponding to each year

```
##   Local Authority    oslaua         age number year cla
## 1            Adur E07000223   All Ages  63176 2014
## 2            Adur E07000223   All Ages  63176 2015
## 3            Adur E07000223   All Ages  63176 2016
## 4            Adur E07000223   All Ages  63176 2017
## 5            Adur E07000223 Aged 65-69   4310 2014
## 6            Adur E07000223 Aged 65-69   4310 2015
```

Exercise

```
new_df_long = new_data %>%
  select(oslaua, `2014`:`2017`) %>%
  gather(year, claimants, `2014`:`2017`)

d1_long = d1 %>% gather(age, number, `All Ages`:
  arrange(`Local Authority`)

exercise = left_join(d1_long, new_df_long, by = "

head(exercise)
```

```
##   Local Authority    oslaua         age number year cla
## 1            Adur E07000223    All Ages  63176 2014
## 2            Adur E07000223    All Ages  63176 2015
## 3            Adur E07000223    All Ages  63176 2016
## 4            Adur E07000223    All Ages  63176 2017
## 5            Adur E07000223 Aged 65-69   4310 2014
## 6            Adur E07000223 Aged 65-69   4310 2015
```

Thanks!

@EdudinGonzalo

`e.gonzalo-almorox@newcastle.ac.uk`