Lab 9 - Data Transformation

Ramin Jabbarialghanab

October 29, 2017

1. In addition to simply naming variable names in select you can also use `:` to select a range of variables and `-` to exclude some variables, similar to indexing a `data.frame` with square brackets. You can use both variable's names as well as integer indexes.

a. Use `select()` to print out a tbl that contains only the first 3 columns of your dataset, called by name.

b. Print out a `tbl` with the last 3 columns of your dataset, called by name.

c. Find the most concise way to select the first 3 columns and the last 3 columns by name.

First three columns

```
library(readxl)
unemployment_women <- read_excel("~/Desktop/Autumn 2017/Statistics 321/unemployment rate/unemployment_w
    skip = 2)
install.packages("dplyr")
```

```
## Installing package into '/Users/Ramin/Desktop/Autumn 2017/Statistics 321/unemployment rate/unemployme
## (as 'lib' is unspecified)
```

```
##
## The downloaded binary packages are in
##  /var/folders/z9/b9hh4hpj6hl6x9r9dxjctv240000gn/T//RtmpyTxiey/downloaded_packages
```

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
dplyr::select(unemployment_women, `2006` : `2009`, -`2001`)
```

```
## # A tibble: 32 x 3
##                `2006`             `2008` `2009`
##                 <chr>              <chr>  <chr>
##  1               16.2               16.7   16.8
##  2                5.4 9.1999999999999993   11.6
##  3                7.3                9.1    9.6
##  4               11.4               10.4   13.8
##  5                 16                 15   20.7
##  6                  -                  -      -
##  7               18.8               25.7   14.9
##  8               14.5               14.6   18.3
##  9                 24               20.8   16.2
## 10 16.399999999999999               28.7   10.1
## # ... with 22 more rows
```

Last three columns

```
dplyr::select(unemployment_women, `2012`:`2014`)
```

```
## # A tibble: 32 x 3
##    `2012` `2013` `2014`
##     <dbl>  <dbl>  <dbl>
##  1   19.7   19.8   19.7
##  2   13.5   17.0   11.4
##  3    9.4    9.9   11.1
##  4   17.1   15.8   14.0
##  5   25.0   18.4   23.2
##  6   30.8   27.0   24.5
##  7   31.5   26.8   21.0
##  8   15.3   10.9   12.7
##  9   20.5   23.1   18.1
## 10   21.3   22.3   25.2
## # ... with 22 more rows
```

2. `dplyr` comes with a set of helper functions that can help you select groups of variables inside a `select()` call:

- `starts_with("X")`: every name that starts with "X",
- `ends_with("X")`: every name that ends with "X",
- `contains("X")`: every name that contains "X",
- `matches("X")`: every name that matches "X", where "X" can be a regular expression,
- `num_range("x", 1:5)`: the variables named x01, x02, x03, x04 and x05,
- `one_of(x)`: every name that appears in x, which should be a character vector.

Pay attention here: When you refer to columns directly inside `select()`, you don't use quotes. If you use the helper functions, you do use quotes.

a. Use `select()` and a helper function to print out a `tbl` that selects only variables that contain a specific character string.

b. Use `select()` and a helper function to print out a `tbl` that selects only variables that start with a certain letter or string of letters.

```
library(readxl)
unemployment_men <- read_excel("~/Desktop/Autumn 2017/Statistics 321/unemployment rate/unemployment_men
    skip = 2)
dplyr::select(unemployment_men, contains('10'))
```

```
## # A tibble: 32 x 1
##    `2010`
##     <chr>
##  1   11.9
##  2   10.1
##  3   12.4
##  4   12.9
##  5   13.1
##  6      -
##  7   14.1
##  8   12.4
##  9   12.7
## 10   11.1
## # ... with 22 more rows
```

```
dplyr::select(unemployment_men, starts_with("200"))
```

```
## # A tibble: 32 x 4
##    `2001`             `2006`             `2008` `2009`
##     <chr>              <chr>              <chr>  <chr>
##  1   13.2                 10                9.1   10.8
##  2      -                5.3                6.1    9.5
##  3      -               10.8               10.7     11
##  4      -               11.1 9.8000000000000007   11.3
##  5      -                9.6                  8    9.9
##  6      -                  -                  -      -
##  7      -                 12               11.3   11.9
##  8      - 9.8000000000000007                 10   10.5
##  9      -               10.9 9.3000000000000007   11.3
## 10      -               11.7               11.3      6
## # ... with 22 more rows
```

4. Are there any mutations you wish to carry out on your data (i.e. new variables you wish to create based upon the values of already existing variables)? If so, describe what they are and what you will name them. I do not need to mutate my data.

5. You can use mutate() to add multiple variables at once. To create more than one variable, place a comma between each variable that you define inside `mutate()`.

a. Carry out any and all of the mutations you wish to perform on your dataset and print the results to the console.

```
as.numeric(unemployment_men$`2006`)
```

```
## Warning: NAs introduced by coercion
```

```
##  [1] 10.0  5.3 10.8 11.1  9.6   NA 12.0  9.8 10.9 11.7  9.1  7.7  5.2 11.1
## [15] 10.6  9.0 12.0 12.3  9.8 10.8 10.8 10.0 15.2 13.4  7.9  9.1 14.6  5.7
## [29] 11.1  7.6 13.4  5.9
```

```
mutate(unemployment_men,

       `2001` = as.numeric(`2001`),
       `2006` = as.numeric(`2006`),
       `2008` = as.numeric(`2008`),
       `2009` = as.numeric(`2009`),
       `2010` = as.numeric(`2010`),
       men_change_overtime = `2006` - `2014`)
```

```
## Warning in eval(substitute(expr), envir, enclos): NAs introduced by
## coercion
```

```
## Warning in eval(substitute(expr), envir, enclos): NAs introduced by
## coercion

## Warning in eval(substitute(expr), envir, enclos): NAs introduced by
## coercion

## Warning in eval(substitute(expr), envir, enclos): NAs introduced by
## coercion

## Warning in eval(substitute(expr), envir, enclos): NAs introduced by
## coercion
```

```
## # A tibble: 32 x 11
##           `Province | Year` `2001` `2006` `2008` `2009` `2010` `2011`
##                       <chr>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
##  1                   Total   13.2   10.0    9.1   10.8   11.9   10.5
##  2         East Azarbayejan     NA    5.3    6.1    9.5   10.1    8.0
##  3         West Azarbayejan     NA   10.8   10.7   11.0   12.4   13.0
##  4                 Ardebil     NA   11.1    9.8   11.3   12.9   12.5
##  5                 Esfahan     NA    9.6    8.0    9.9   13.1   10.8
##  6                  Alborz     NA     NA     NA     NA     NA   16.3
##  7                    Ilam     NA   12.0   11.3   11.9   14.1   12.6
##  8                 Bushehr     NA    9.8   10.0   10.5   12.4   10.7
##  9                  Tehran     NA   10.9    9.3   11.3   12.7    9.6
## 10 Chaharmahal & Bakhtiyari     NA   11.7   11.3    6.0   11.1   12.1
## # ... with 22 more rows, and 4 more variables: `2012` <dbl>, `2013` <dbl>,
## #   `2014` <dbl>, men_change_overtime <dbl>
```

6. `R` comes with a set of logical operators that you can use inside `filter()`:

- `x < y`, `TRUE` if `x` is less than `y`
- `x <= y`, `TRUE` if `x` is less than or equal to `y`
- `x == y`, `TRUE` if `x` equals `y`
- `x != y`, `TRUE` if `x` does not equal `y`
- `x >= y`, `TRUE` if `x` is greater than or equal to `y`
- `x > y`, `TRUE` if `x` is greater than `y`
- `x %in% c(a, b, c)`, `TRUE` if `x` is in the vector `c(a, b, c)`

```
unemployment_men %>%
  filter(`2014` > 8.8) %>%
  arrange(`2014`)
```

```
## # A tibble: 16 x 10
##           `Province | Year` `2001`             `2006`             `2008`
##                       <chr>  <chr>               <chr>              <chr>
##  1        Khorasan-e-Razavi      -                 7.7                7.9
##  2                     Fars      -                12.3                 10
##  3                   Zanjan      -                10.6                8.5
##  4         West Azarbayejan      -                10.8               10.7
##  5                   Alborz      -                   -                  -
##  6                  Esfahan      -                 9.6                  8
##  7                Kordestan      -                10.8               12.5
##  8     Sistan & Baluchestan      -                  12                 11
##  9                  Ardebil      -                11.1 9.8000000000000007
## 10                   Qazvin      - 9.8000000000000007 8.1999999999999993
## 11  Kohgiluyeh & Boyerahmad      -                13.4               10.8
## 12                    Gilan      -                 9.1               10.6
## 13                 Lorestan      -                14.6               13.6
## 14 Chaharmahal & Bakhtiyari      -                11.7               11.3
## 15               Kermanshah      -                15.2               11.1
## 16           North Khorasan      -                 5.2                6.1
## # ... with 6 more variables: `2009` <chr>, `2010` <chr>, `2011` <dbl>,
## #   `2012` <dbl>, `2013` <dbl>, `2014` <dbl>
```

For this chunck, I would now which provinces have emigration rate more than average emigration rate in 2014.

```
unemployment_men %>%
  filter(`2006` > 10) %>%
```

```
  arrange(`2006`)
```

```
## # A tibble: 29 x 10
##           `Province | Year` `2001` `2006`             `2008` `2009`
##                       <chr>  <chr>  <chr>              <chr>  <chr>
##  1                  Zanjan      -   10.6                8.5      7
##  2         West Azarbayejan      -   10.8               10.7     11
##  3                     Qom      -   10.8                8.4   17.7
##  4               Kordestan      -   10.8               12.5    5.7
##  5                  Tehran      -   10.9 9.3000000000000007   11.3
##  6                 Ardebil      -   11.1 9.8000000000000007   11.3
##  7               Khuzestan      -   11.1 10.199999999999999    7.8
##  8                 Markazi      -   11.1               10.7     11
##  9 Chaharmahal & Bakhtiyari      -   11.7               11.3      6
## 10                   Ilam      -     12               11.3   11.9
## # ... with 19 more rows, and 5 more variables: `2010` <chr>, `2011` <dbl>,
## #   `2012` <dbl>, `2013` <dbl>, `2014` <dbl>
```

Here, I am going to know which provinces are about the average rate of emigration.

```
unemployment_men %>%
  filter(`2006` == 10) %>%
  arrange(`2006`)
```

```
## # A tibble: 2 x 10
##   `Province | Year` `2001` `2006` `2008`             `2009` `2010` `2011`
##               <chr>  <chr>  <chr>  <chr>              <chr>  <chr>  <dbl>
## 1             Total   13.2     10    9.1               10.8   11.9   10.5
## 2            Kerman      -     10    7.5 9.1999999999999993    8.1    9.2
## # ... with 3 more variables: `2012` <dbl>, `2013` <dbl>, `2014` <dbl>
```

This chunk shows which provinces have the exact average emigration rate for 2006.

a. What are some potential subsets of your data that seem interesting and worth investigation to you?
b. Use at least two of the logical operators presented above to print these subsets of your data.

7. `R` also comes with a set of boolean operators that you can use to combine multiple logical tests into a single test. These include `&` (and), `|` (or), and `!` (not). Instead of using the `&` operator, you can also pass several logical tests to `filter()`, separated by commas. `is.na()` will also come in handy.

```
unemployment_men  %>%
  filter(`2006` > 12 & `2006` < 8) %>%
  arrange(`2006`)
```

```
## # A tibble: 12 x 10
##           `Province | Year` `2001` `2006` `2008`             `2009`
##                       <chr>  <chr>  <chr>  <chr>              <chr>
##  1                   Fars      -   12.3     10               10.9
##  2 Kohgiluyeh & Boyerahmad      -   13.4   10.8               16.3
##  3                 Hamedan      -   13.4   13.5                 13
##  4               Lorestan      -   14.6   13.6 8.8000000000000007
##  5             Kermanshah      -   15.2   11.1                5.7
##  6           North Khorasan      -    5.2    6.1               11.9
##  7        East Azarbayejan      -    5.3    6.1                9.5
##  8              Mazandaran      -    5.7    5.5               13.7
##  9                   Yazd      -    5.9    5.9                7.3
## 10               Hormozgan      -    7.6      8                6.1
```

```
## 11       Khorasan-e-Razavi      -    7.7    7.9               5.3
## 12                 Golestan      -    7.9    6.4              14.9
## # ... with 5 more variables: `2010` <chr>, `2011` <dbl>, `2012` <dbl>,
## #   `2013` <dbl>, `2014` <dbl>
```

a. Use R's logical and boolean operators to select just the rows in your data that meet a specific boolean condition.
b. Print out all of the observations in your data in which none of variables are NA.

```
mutate(unemployment_men,
       `2001` = as.numeric(`2001`),
       `2006` = as.numeric(`2006`),
       `2008` = as.numeric(`2008`),
       `2009` = as.numeric(`2009`),
       `2010` = as.numeric(`2010`))
```

```
## Warning in eval(substitute(expr), envir, enclos): NAs introduced by
## coercion

## Warning in eval(substitute(expr), envir, enclos): NAs introduced by
## coercion

## Warning in eval(substitute(expr), envir, enclos): NAs introduced by
## coercion

## Warning in eval(substitute(expr), envir, enclos): NAs introduced by
## coercion

## Warning in eval(substitute(expr), envir, enclos): NAs introduced by
## coercion
```

```
## # A tibble: 32 x 10
##           `Province | Year` `2001` `2006` `2008` `2009` `2010` `2011`
##                       <chr>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
##  1                    Total   13.2   10.0    9.1   10.8   11.9   10.5
##  2         East Azarbayejan     NA    5.3    6.1    9.5   10.1    8.0
##  3         West Azarbayejan     NA   10.8   10.7   11.0   12.4   13.0
##  4                  Ardebil     NA   11.1    9.8   11.3   12.9   12.5
##  5                  Esfahan     NA    9.6    8.0    9.9   13.1   10.8
##  6                   Alborz     NA     NA     NA     NA     NA   16.3
##  7                     Ilam     NA   12.0   11.3   11.9   14.1   12.6
##  8                  Bushehr     NA    9.8   10.0   10.5   12.4   10.7
##  9                   Tehran     NA   10.9    9.3   11.3   12.7    9.6
## 10 Chaharmahal & Bakhtiyari     NA   11.7   11.3    6.0   11.1   12.1
## # ... with 22 more rows, and 3 more variables: `2012` <dbl>, `2013` <dbl>,
## #   `2014` <dbl>
```

```
filter(unemployment_men, !is.na(`2001`) & !is.na(`2006`) & !is.na(`2008`) & !is.na(`2009`) & !is.na(`20
```

```
## # A tibble: 32 x 10
##           `Province | Year` `2001`             `2006`             `2008`
##                       <chr>  <chr>               <chr>              <chr>
##  1                    Total   13.2                  10                9.1
##  2         East Azarbayejan      -                 5.3                6.1
##  3         West Azarbayejan      -                10.8               10.7
##  4                  Ardebil      -                11.1 9.8000000000000007
```

```
##  5                  Esfahan      -                9.6                  8
##  6                   Alborz      -                  -                  -
##  7                     Ilam      -                 12               11.3
##  8                  Bushehr      - 9.8000000000000007                 10
##  9                   Tehran      -               10.9 9.3000000000000007
## 10 Chaharmahal & Bakhtiyari      -               11.7               11.3
## # ... with 22 more rows, and 6 more variables: `2009` <chr>, `2010` <chr>,
## #   `2011` <dbl>, `2012` <dbl>, `2013` <dbl>, `2014` <dbl>
```

8. `arrange()` can be used to rearrange rows according to any type of data. If you pass `arrange()` a character variable, for example, `R` will rearrange the rows in alphabetical order according to values of the variable. If you pass a factor variable, `R` will rearrange the rows according to the order of the levels in your factor (running `levels()` on the variable reveals this order).

By default, `arrange()` arranges the rows from smallest to largest. Rows with the smallest value of the variable will appear at the top of the data set. You can reverse this behavior with the `desc()` function. `arrange()` will reorder the rows from largest to smallest values of a variable if you wrap the variable name in `desc()` before passing it to `arrange()`.

```
arrange(unemployment_men, `2014`)
```

```
## # A tibble: 32 x 10
##    `Province | Year` `2001`             `2006`             `2008`
##                <chr>  <chr>              <chr>              <chr>
##  1              Yazd      -                5.9                5.9
##  2           Kerman       -                 10                7.5
##  3          Markazi       -               11.1               10.7
##  4           Tehran       -               10.9 9.3000000000000007
##  5  East Azarbayejan      -                5.3                6.1
##  6    South Khorasan      -                9.1                7.7
##  7           Hamedan      -               13.4               13.5
##  8        Hormozgan       -                7.6                  8
##  9             Ilam       -                 12               11.3
## 10          Bushehr       - 9.8000000000000007                 10
## # ... with 22 more rows, and 6 more variables: `2009` <chr>, `2010` <chr>,
## #   `2011` <dbl>, `2012` <dbl>, `2013` <dbl>, `2014` <dbl>
```

a. Which variable(s) in your dataset would be logical to arrange your data on? Explain your reasoning.
b. Arrange your data by this/these variables and print the results.

I would know which provinces have the least rate of emigration for 2014. For this purpose, I arranged for 2014.

```
arrange(unemployment_men,`2006`)
```

```
## # A tibble: 32 x 10
##    `Province | Year` `2001` `2006`             `2008`             `2009`
##                <chr>  <chr>  <chr>              <chr>              <chr>
##  1           Alborz      -      -                  -                  -
##  2            Total   13.2     10                9.1               10.8
##  3           Kerman       -     10                7.5 9.1999999999999993
##  4           Zanjan       -   10.6                8.5                  7
##  5  West Azarbayejan      -   10.8               10.7                 11
##  6              Qom       -   10.8                8.4               17.7
##  7         Kordestan      -   10.8               12.5                5.7
##  8            Tehran      -   10.9 9.3000000000000007               11.3
##  9          Ardebil       -   11.1 9.8000000000000007               11.3
```

```
## 10         Khuzestan      -   11.1 10.199999999999999                7.8
## # ... with 22 more rows, and 5 more variables: `2010` <chr>, `2011` <dbl>,
## #   `2012` <dbl>, `2013` <dbl>, `2014` <dbl>
```

I would know which provinces have the least rate of emigration in 2006.

9. You can use any function you like in `summarise()` so long as the function can take a vector of data and return a single number. `R` contains many aggregating functions, as `dplyr` calls them:

- `min(x)` - minimum value of vector `x`.
- `max(x)` - maximum value of vector `x`.
- `mean(x)` - mean value of vector `x`.
- `median(x)` - median value of vector `x`.
- `quantile(x, p)` - pth quantile of vector `x`.
- `sd(x)` - standard deviation of vector `x`.
- `var(x)` - variance of vector `x`.
- `IQR(x)` - Inter Quartile Range (IQR) of vector `x`.
- `diff(range(x))` - total range of vector `x`.

a. Pick at least one variable of interest to your project analysis.
b. Print out at least three summary statistics using `summarise()`.

```
summarise (unemployment_men, max = max(`2014`),
          min(`2014`),
          mean(`2014`),
          var(`2014`),
          IQR(`2014`))
```

```
## # A tibble: 1 x 5
##     max `min(\`2014\`)` `mean(\`2014\`)` `var(\`2014\`)` `IQR(\`2014\`)`
##   <dbl>           <dbl>            <dbl>           <dbl>           <dbl>
## 1  14.3             5.6          9.30625        5.276734           2.125
```

10. `dplyr` provides several helpful aggregate functions of its own, in addition to the ones that are already defined in `R`. These include:

- `first(x)` - The first element of vector `x`.
- `last(x)` - The last element of vector `x`.
- `nth(x, n)` - The nth element of vector `x`.
- `n()` - The number of rows in the data.frame or group of observations that `summarise()` describes.
- `n_distinct(x)` - The number of unique values in vector `x`.

Next to these `dplyr`-specific functions, you can also turn a logical test into an aggregating function with `sum()` or `mean()`. A logical test returns a vector of TRUE's and FALSE's. When you apply `sum()` or `mean()` to such a vector, `R` coerces each `TRUE` to a `1` and each `FALSE` to a `0`. `sum()` then represents the total number of observations that passed the test; `mean()` represents the proportion.

a. Print out a summary of your data using at least two of these `dplyr`-specific aggregate functions.
b. Why did you choose the ones you did? What did you learn about your data from these summaries?

I just this for practice to learn how that works.

```
summarise (unemployment_men, first(`2014`),
           last(`2014`),
           nth(`2014`, 19))
```

```
## # A tibble: 1 x 3
##   `first(\`2014\`)` `last(\`2014\`)` `nth(\`2014\`, 19)`
##               <dbl>            <dbl>               <dbl>
## 1               8.8              5.6                10.6
```

11. You can also combine `group_by()` with `mutate()`. When you mutate grouped data, `mutate()` will calculate the new variables independently for each group. This is particularly useful when `mutate()` uses the `rank()` function, that calculates within-group rankings. `rank()` takes a group of values and calculates the rank of each value within the group, e.g.

```
rank(c(21, 22, 24, 23))
```

has the output

```
[1] 1 2 4 3
```

As with `arrange()`, `rank()` ranks values from the smallest to the largest.

a. Using the `%>%` operator, first group your dataset by a meaningful variable, then perform a mutation that you're interested in.
b. What do the results tell you about different groups in you data?

The result says that the rate of emitration is decreased in most of provinces.

```
group_by(unemployment_men, `2011` & `2014`) %>%
   mutate(difference = `2014` - `2011`)
```

```
## # A tibble: 32 x 12
## # Groups:   `2011` & `2014` [1]
##           `Province | Year` `2001`              `2006`              `2008`
##                       <chr>  <chr>               <chr>               <chr>
##  1                    Total   13.2                  10                 9.1
##  2         East Azarbayejan      -                  5.3                6.1
##  3         West Azarbayejan      -                 10.8               10.7
##  4                  Ardebil      -                 11.1 9.8000000000000007
##  5                  Esfahan      -                  9.6                  8
##  6                   Alborz      -                    -                  -
##  7                     Ilam      -                   12               11.3
##  8                  Bushehr      - 9.8000000000000007                 10
##  9                   Tehran      -                 10.9 9.3000000000000007
## 10 Chaharmahal & Bakhtiyari      -                 11.7               11.3
## # ... with 22 more rows, and 8 more variables: `2009` <chr>, `2010` <chr>,
## #   `2011` <dbl>, `2012` <dbl>, `2013` <dbl>, `2014` <dbl>, `\`2011\` &
## #   \`2014\`` <lgl>, difference <dbl>
```

12. The exercises so far have tried to get you to think about how to apply the five verbs of `dplyr` to your data.

a. Are there any specific transformations you want to make to your data? What are they and what aspect of your research question will they help you to answer?
b. In a code chunk below, carry out all the data transformations you wish to perform on your data. Utilize the `%>%` operator to tie multiple commands together and make your code more readable and efficient. Remember to comment your code so it is clear why you doing things a certain way.

Calculating the average of men's unemployment rate for last four years of the data

```
unemployment_men %>%
dplyr::select(`2011`:`2014`) %>%
summarise (mean(`2011`), mean(`2012`), mean(`2013`), mean(`2014`))
```

```
## # A tibble: 1 x 4
##   `mean(\`2011\`)` `mean(\`2012\`)` `mean(\`2013\`)` `mean(\`2014\`)`
##              <dbl>            <dbl>            <dbl>            <dbl>
## 1           10.525         10.30937         8.865625          9.30625
```

Calculating the average of women's unemployment rate for last four years of the data

```
unemployment_women %>%
dplyr::select(`2011`:`2014`) %>%
summarise (mean(`2011`), mean(`2012`), mean(`2013`), mean(`2014`))
```

```
## # A tibble: 1 x 4
##   `mean(\`2011\`)` `mean(\`2012\`)` `mean(\`2013\`)` `mean(\`2014\`)`
##              <dbl>            <dbl>            <dbl>            <dbl>
## 1         20.76562         20.46875          19.1375         19.63437
```

Calculating the average of total population's unemployment rate for last four years of the data

```
library(readxl)
unemployment_total <- read_excel("~/Desktop/Autumn 2017/Statistics 321/unemployment rate/unemployment_t
    skip = 2)
unemployment_total %>%
dplyr::select(`2011`:`2014`) %>%
summarise (mean(`2011`), mean(`2012`), mean(`2013`), mean(`2014`))
```

```
## # A tibble: 1 x 4
##   `mean(\`2011\`)` `mean(\`2012\`)` `mean(\`2013\`)` `mean(\`2014\`)`
##              <dbl>            <dbl>            <dbl>            <dbl>
## 1         12.21875         11.98438         10.51562         10.94063
```