Lab 9 - Data Transformation

William Bernard

October 29, 2017

Using your own dataset (which may include more than one table) carry out the following data cleaning steps. Knit together the PDF document and commit both the Lab 9 RMD file and the PDF document to Git. Push the changes to GitHub so both documents are visible in your public GitHub repository.

```
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
```

```
library(tidyr)
library(tidyverse)
```

```
## Loading tidyverse: ggplot2
## Loading tidyverse: tibble
## Loading tidyverse: readr
## Loading tidyverse: purrr
```

```
## Conflicts with tidy packages ----------------------------------------------
```

```
## filter(): dplyr, stats
## lag():    dplyr, stats
```

```
setwd("C:\\Users\\William Bernard\\Desktop\\William Bernard Poster Project v.2\\william_bernard_poster

# Read in your data with the appropriate function
changing_lives <- load("C:\\Users\\William Bernard\\Desktop\\americans's_changing_lives_data_set\\ICPSR


data_subset <- da04690.0001 %>%
  select(V6, V7, V103, V104, V220, V221, V222, V223, V224, V225, V301, V302, V303, V304, V305, V306, V3
```

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.

```
as_tibble(select(data_subset, V6, V7, V103))
```

```
## # A tibble: 3,617 x 3
##              V6           V7       V103
##          <fctr>       <fctr>     <fctr>
##  1  (1) CORRECT  (1) CORRECT (2) FEMALE
##  2  (1) CORRECT  (1) CORRECT   (1) MALE
```

```
##  3  (1) CORRECT  (1) CORRECT   (1) MALE
##  4  (1) CORRECT  (1) CORRECT   (1) MALE
##  5 (5) INCORRCT  (1) CORRECT (2) FEMALE
##  6  (1) CORRECT  (1) CORRECT   (1) MALE
##  7 (5) INCORRCT (5) INCORRCT (2) FEMALE
##  8 (5) INCORRCT  (1) CORRECT (2) FEMALE
##  9  (1) CORRECT  (1) CORRECT (2) FEMALE
## 10 (5) INCORRCT  (1) CORRECT   (1) MALE
## # ... with 3,607 more rows
```

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

```
as_tibble(select(data_subset, V441, V445, V446))
```

```
## # A tibble: 3,617 x 3
##     V441         V445         V446
##    <dbl>       <fctr>       <fctr>
##  1     2 (1) UNEXPECT (3) SOMEWELL
##  2    NA         <NA>         <NA>
##  3    NA         <NA>         <NA>
##  4    NA         <NA>         <NA>
##  5    NA         <NA>         <NA>
##  6    NA         <NA>         <NA>
##  7     2 (1) UNEXPECT (3) SOMEWELL
##  8    NA         <NA>         <NA>
##  9    NA         <NA>         <NA>
## 10     2 (1) UNEXPECT (1) VERYWELL
## # ... with 3,607 more rows
```

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

```
as_tibble(select(data_subset, V6, V7, V103, V441, V445, V446))
```

```
## # A tibble: 3,617 x 6
##              V6           V7       V103  V441         V445         V446
##          <fctr>       <fctr>     <fctr> <dbl>       <fctr>       <fctr>
##  1  (1) CORRECT  (1) CORRECT (2) FEMALE     2 (1) UNEXPECT (3) SOMEWELL
##  2  (1) CORRECT  (1) CORRECT   (1) MALE    NA         <NA>         <NA>
##  3  (1) CORRECT  (1) CORRECT   (1) MALE    NA         <NA>         <NA>
##  4  (1) CORRECT  (1) CORRECT   (1) MALE    NA         <NA>         <NA>
##  5 (5) INCORRCT  (1) CORRECT (2) FEMALE    NA         <NA>         <NA>
##  6  (1) CORRECT  (1) CORRECT   (1) MALE    NA         <NA>         <NA>
##  7 (5) INCORRCT (5) INCORRCT (2) FEMALE     2 (1) UNEXPECT (3) SOMEWELL
##  8 (5) INCORRCT  (1) CORRECT (2) FEMALE    NA         <NA>         <NA>
##  9  (1) CORRECT  (1) CORRECT (2) FEMALE    NA         <NA>         <NA>
## 10 (5) INCORRCT  (1) CORRECT   (1) MALE     2 (1) UNEXPECT (1) VERYWELL
## # ... with 3,607 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.

```
as_tibble(select(data_subset, contains("V")))
```

```
## # A tibble: 3,617 x 59
##              V6           V7       V103  V104        V220        V221
##          <fctr>       <fctr>     <fctr> <dbl>      <fctr>      <fctr>
##  1  (1) CORRECT  (1) CORRECT (2) FEMALE    69   (6) NEVER   (2) 1X/WK
##  2  (1) CORRECT  (1) CORRECT   (1) MALE    44  (2) 1X/DAY   (2) 1X/WK
##  3  (1) CORRECT  (1) CORRECT   (1) MALE    75  (5) <1X/WK  (1) >1X/WK
##  4  (1) CORRECT  (1) CORRECT   (1) MALE    25 (3) 2-3X/WK   (2) 1X/WK
##  5 (5) INCORRCT  (1) CORRECT (2) FEMALE    30 (3) 2-3X/WK  (1) >1X/WK
##  6  (1) CORRECT  (1) CORRECT   (1) MALE    57 (3) 2-3X/WK   (2) 1X/WK
##  7 (5) INCORRCT (5) INCORRCT (2) FEMALE    56   (6) NEVER   (2) 1X/WK
##  8 (5) INCORRCT  (1) CORRECT (2) FEMALE    37 (3) 2-3X/WK (3) 2-3X/MO
##  9  (1) CORRECT  (1) CORRECT (2) FEMALE    27 (1) >1X/DAY  (1) >1X/WK
## 10 (5) INCORRCT  (1) CORRECT   (1) MALE    73 (3) 2-3X/WK   (4) 1X/MO
## # ... with 3,607 more rows, and 53 more variables: V222 <fctr>,
## #   V223 <fctr>, V224 <fctr>, V225 <fctr>, V301 <fctr>, V302 <fctr>,
## #   V303 <fctr>, V304 <fctr>, V305 <fctr>, V306 <fctr>, V307 <fctr>,
## #   V308 <fctr>, V309 <fctr>, V310 <fctr>, V311 <fctr>, V314 <fctr>,
## #   V315 <fctr>, V316 <fctr>, V317 <fctr>, V322 <fctr>, V323 <fctr>,
## #   V325 <fctr>, V326 <fctr>, V328 <fctr>, V329 <fctr>, V330 <fctr>,
## #   V331 <fctr>, V332 <fctr>, V333 <fctr>, V334 <fctr>, V335 <fctr>,
## #   V401 <fctr>, V402 <fctr>, V405 <fctr>, V406 <fctr>, V407 <fctr>,
## #   V408 <fctr>, V410 <dbl>, V416 <fctr>, V419 <fctr>, V420 <dbl>,
## #   V425 <dbl>, V430 <fctr>, V431 <fctr>, V432 <fctr>, V433 <fctr>,
## #   V434 <fctr>, V437 <fctr>, V438 <fctr>, V440 <fctr>, V441 <dbl>,
## #   V445 <fctr>, V446 <fctr>
```

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.

```
as_tibble(select(data_subset, num_range("V", 200:500)))
```

```
## # A tibble: 3,617 x 55
##           V220        V221        V222         V223         V224
##         <fctr>      <fctr>      <fctr>       <fctr>       <fctr>
##  1   (6) NEVER   (2) 1X/WK   (6) NEVER    (4) NEVER    (4) NEVER
##  2  (2) 1X/DAY   (2) 1X/WK (3) 2-3X/MO (2) SOMETIME (2) SOMETIME
##  3  (5) <1X/WK  (1) >1X/WK   (6) NEVER    (1) OFTEN    (1) OFTEN
##  4 (3) 2-3X/WK   (2) 1X/WK   (6) NEVER (2) SOMETIME   (3) RARELY
##  5 (3) 2-3X/WK  (1) >1X/WK   (6) NEVER (2) SOMETIME (2) SOMETIME
##  6 (3) 2-3X/WK   (2) 1X/WK   (6) NEVER (2) SOMETIME    (4) NEVER
##  7   (6) NEVER   (2) 1X/WK  (1) >1X/WK    (1) OFTEN    (4) NEVER
##  8 (3) 2-3X/WK (3) 2-3X/MO  (1) >1X/WK   (3) RARELY    (4) NEVER
##  9 (1) >1X/DAY  (1) >1X/WK   (4) 1X/MO   (3) RARELY    (1) OFTEN
## 10 (3) 2-3X/WK   (4) 1X/MO  (5) <1X/MO    (1) OFTEN    (1) OFTEN
## # ... with 3,607 more rows, and 50 more variables: V225 <fctr>,
## #   V301 <fctr>, V302 <fctr>, V303 <fctr>, V304 <fctr>, V305 <fctr>,
## #   V306 <fctr>, V307 <fctr>, V308 <fctr>, V309 <fctr>, V310 <fctr>,
## #   V311 <fctr>, V314 <fctr>, V315 <fctr>, V316 <fctr>, V317 <fctr>,
```

```
## #   V322 <fctr>, V323 <fctr>, V325 <fctr>, V326 <fctr>, V328 <fctr>,
## #   V329 <fctr>, V330 <fctr>, V331 <fctr>, V332 <fctr>, V333 <fctr>,
## #   V334 <fctr>, V335 <fctr>, V401 <fctr>, V402 <fctr>, V405 <fctr>,
## #   V406 <fctr>, V407 <fctr>, V408 <fctr>, V410 <dbl>, V416 <fctr>,
## #   V419 <fctr>, V420 <dbl>, V425 <dbl>, V430 <fctr>, V431 <fctr>,
## #   V432 <fctr>, V433 <fctr>, V434 <fctr>, V437 <fctr>, V438 <fctr>,
## #   V440 <fctr>, V441 <dbl>, V445 <fctr>, V446 <fctr>
```

```
as_tibble(select(data_subset, V303, V304, V305, V306))
```

```
## # A tibble: 3,617 x 4
##                                                                                V303
##                                                                              <fctr>
##  1                                                                             <NA>
##  2                                                                             <NA>
##  3                                                                             <NA>
##  4                                                          (58) TRAVEL; VACATIONS
##  5  (29) SCHOOL/EDUCATION (OF R); EDUCATIONAL ACCOMPLISHMENT, FINISHING SCHOOL
##  6                                                                             <NA>
##  7                                              (69) OTHER LEISURE ACTIVITIES
##  8                                                   (60) HOBBIES AND CRAFTS
##  9 "(19) \"FAMILY\" -- IN GENERAL OR NA 01-09; \"RELATIVES\"; SPENDING TIME WI
## 10 "(31) HOUSE/HOME; OWNING HOME; \"MY HOME\" -- NFS; SECOND/VACATION HOME, HO
## # ... with 3,607 more rows, and 3 more variables: V304 <fctr>,
## #   V305 <fctr>, V306 <fctr>
```

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.

```
as_tibble(mutate(data_subset, reason_sat = V303, V304, V305, V306))
```

```
## # A tibble: 3,617 x 60
##              V6           V7       V103  V104        V220        V221
##          <fctr>       <fctr>     <fctr> <dbl>      <fctr>      <fctr>
##  1  (1) CORRECT  (1) CORRECT (2) FEMALE    69   (6) NEVER   (2) 1X/WK
##  2  (1) CORRECT  (1) CORRECT   (1) MALE    44  (2) 1X/DAY   (2) 1X/WK
##  3  (1) CORRECT  (1) CORRECT   (1) MALE    75  (5) <1X/WK  (1) >1X/WK
##  4  (1) CORRECT  (1) CORRECT   (1) MALE    25 (3) 2-3X/WK   (2) 1X/WK
##  5 (5) INCORRCT  (1) CORRECT (2) FEMALE    30 (3) 2-3X/WK  (1) >1X/WK
##  6  (1) CORRECT  (1) CORRECT   (1) MALE    57 (3) 2-3X/WK   (2) 1X/WK
##  7 (5) INCORRCT (5) INCORRCT (2) FEMALE    56   (6) NEVER   (2) 1X/WK
##  8 (5) INCORRCT  (1) CORRECT (2) FEMALE    37 (3) 2-3X/WK (3) 2-3X/MO
##  9  (1) CORRECT  (1) CORRECT (2) FEMALE    27 (1) >1X/DAY  (1) >1X/WK
## 10 (5) INCORRCT  (1) CORRECT   (1) MALE    73 (3) 2-3X/WK   (4) 1X/MO
## # ... with 3,607 more rows, and 54 more variables: V222 <fctr>,
## #   V223 <fctr>, V224 <fctr>, V225 <fctr>, V301 <fctr>, V302 <fctr>,
## #   V303 <fctr>, V304 <fctr>, V305 <fctr>, V306 <fctr>, V307 <fctr>,
## #   V308 <fctr>, V309 <fctr>, V310 <fctr>, V311 <fctr>, V314 <fctr>,
## #   V315 <fctr>, V316 <fctr>, V317 <fctr>, V322 <fctr>, V323 <fctr>,
## #   V325 <fctr>, V326 <fctr>, V328 <fctr>, V329 <fctr>, V330 <fctr>,
## #   V331 <fctr>, V332 <fctr>, V333 <fctr>, V334 <fctr>, V335 <fctr>,
## #   V401 <fctr>, V402 <fctr>, V405 <fctr>, V406 <fctr>, V407 <fctr>,
## #   V408 <fctr>, V410 <dbl>, V416 <fctr>, V419 <fctr>, V420 <dbl>,
## #   V425 <dbl>, V430 <fctr>, V431 <fctr>, V432 <fctr>, V433 <fctr>,
## #   V434 <fctr>, V437 <fctr>, V438 <fctr>, V440 <fctr>, V441 <dbl>,
```

```
## #   V445 <fctr>, V446 <fctr>, reason_sat <fctr>
```

This new variable is a combination of five other variables used to assess respondants reasons for satisfaction in their lives. These variables are denote priority of the actions but are not all combined for easy viewing. This mutate will allow me to view all the reasons for their satisfaction easily but will most likely not be used for analysis considering there is a still a single factor denoting each data point.

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_tibble(mutate(data_subset, prob_worry = V307, V308, V309, V310, V311))
```

```
## # A tibble: 3,617 x 60
##              V6           V7       V103  V104        V220        V221
##          <fctr>       <fctr>     <fctr> <dbl>      <fctr>      <fctr>
##  1  (1) CORRECT  (1) CORRECT (2) FEMALE    69   (6) NEVER   (2) 1X/WK
##  2  (1) CORRECT  (1) CORRECT   (1) MALE    44  (2) 1X/DAY   (2) 1X/WK
##  3  (1) CORRECT  (1) CORRECT   (1) MALE    75  (5) <1X/WK  (1) >1X/WK
##  4  (1) CORRECT  (1) CORRECT   (1) MALE    25 (3) 2-3X/WK   (2) 1X/WK
##  5 (5) INCORRCT  (1) CORRECT (2) FEMALE    30 (3) 2-3X/WK  (1) >1X/WK
##  6  (1) CORRECT  (1) CORRECT   (1) MALE    57 (3) 2-3X/WK   (2) 1X/WK
##  7 (5) INCORRCT (5) INCORRCT (2) FEMALE    56   (6) NEVER   (2) 1X/WK
##  8 (5) INCORRCT  (1) CORRECT (2) FEMALE    37 (3) 2-3X/WK (3) 2-3X/MO
##  9  (1) CORRECT  (1) CORRECT (2) FEMALE    27 (1) >1X/DAY  (1) >1X/WK
## 10 (5) INCORRCT  (1) CORRECT   (1) MALE    73 (3) 2-3X/WK   (4) 1X/MO
## # ... with 3,607 more rows, and 54 more variables: V222 <fctr>,
## #   V223 <fctr>, V224 <fctr>, V225 <fctr>, V301 <fctr>, V302 <fctr>,
## #   V303 <fctr>, V304 <fctr>, V305 <fctr>, V306 <fctr>, V307 <fctr>,
## #   V308 <fctr>, V309 <fctr>, V310 <fctr>, V311 <fctr>, V314 <fctr>,
## #   V315 <fctr>, V316 <fctr>, V317 <fctr>, V322 <fctr>, V323 <fctr>,
## #   V325 <fctr>, V326 <fctr>, V328 <fctr>, V329 <fctr>, V330 <fctr>,
## #   V331 <fctr>, V332 <fctr>, V333 <fctr>, V334 <fctr>, V335 <fctr>,
## #   V401 <fctr>, V402 <fctr>, V405 <fctr>, V406 <fctr>, V407 <fctr>,
## #   V408 <fctr>, V410 <dbl>, V416 <fctr>, V419 <fctr>, V420 <dbl>,
## #   V425 <dbl>, V430 <fctr>, V431 <fctr>, V432 <fctr>, V433 <fctr>,
## #   V434 <fctr>, V437 <fctr>, V438 <fctr>, V440 <fctr>, V441 <dbl>,
## #   V445 <fctr>, V446 <fctr>, prob_worry <fctr>
```

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)`

a. What are some potential subsets of your data that seem interesting and worth investigation to you?

I would like to potentially subset my data based on age of respondant, and other qualifying factors like if they have lost a child.

b. Use at least two of the logical operators presented above to print these subsets of your data.

```
fifty_plus <- select(data_subset, contains("325"))

as_tibble(filter(fifty_plus, V325 == "(1) 50AOVER"))
```

```
## # A tibble: 2,067 x 1
##           V325
##         <fctr>
##  1 (1) 50AOVER
##  2 (1) 50AOVER
##  3 (1) 50AOVER
##  4 (1) 50AOVER
##  5 (1) 50AOVER
##  6 (1) 50AOVER
##  7 (1) 50AOVER
##  8 (1) 50AOVER
##  9 (1) 50AOVER
## 10 (1) 50AOVER
## # ... with 2,057 more rows
```

```
child_die <- select(data_subset, V440)

as_tibble(filter(child_die, V440 != "(5) NO"))
```

```
## # A tibble: 621 x 1
##       V440
##     <fctr>
##  1 (1) YES
##  2 (1) YES
##  3 (1) YES
##  4 (1) YES
##  5 (1) YES
##  6 (1) YES
##  7 (1) YES
##  8 (1) YES
##  9 (1) YES
## 10 (1) YES
## # ... with 611 more rows
```

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.

a. Use `R`'s logical and boolean operators to select just the rows in your data that meet a specific boolean condition.

```
sat_r <- select(data_subset, V322)

as_tibble(filter(sat_r, V322 == "(1) COMPSAT" | V322 == "(2) VERYSAT" | V322 == "(3) SOMESAT"))
```

```
## # A tibble: 3,236 x 1
##           V322
##         <fctr>
##  1 (1) COMPSAT
##  2 (1) COMPSAT
##  3 (2) VERYSAT
##  4 (2) VERYSAT
```

```
##  5 (2) VERYSAT
##  6 (2) VERYSAT
##  7 (2) VERYSAT
##  8 (3) SOMESAT
##  9 (1) COMPSAT
## 10 (1) COMPSAT
## # ... with 3,226 more rows
```

b. Print out all of the observations in your data in which none of variables are `NA`.

```
sat_home <- select(data_subset, V322)

as_tibble(filter(sat_home, V322 == "(1) COMPSAT" | V322 == "(2) VERYSAT" | V322 == "(3) SOMESAT" & !is.
```

```
## # A tibble: 3,236 x 1
##           V322
##         <fctr>
##  1 (1) COMPSAT
##  2 (1) COMPSAT
##  3 (2) VERYSAT
##  4 (2) VERYSAT
##  5 (2) VERYSAT
##  6 (2) VERYSAT
##  7 (2) VERYSAT
##  8 (3) SOMESAT
##  9 (1) COMPSAT
## 10 (1) COMPSAT
## # ... with 3,226 more rows
```

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()`.

a. Which variable(s) in your dataset would be logical to arrange your data on? Explain your reasoning.

any data that has a number of factors for levels like completely satisfied, somewhat satisfied etc.

b. Arrange your data by this/these variables and print the results.

```
sat_r<- select(data_subset, V301, V302, V303, V304, V305, V306)

as_tibble(arrange(sat_r, V303, V304, V305, V306, V302, V301))
```

```
## # A tibble: 3,617 x 6
##           V301
##         <fctr>
##  1 (1) COMPSAT
##  2 (1) COMPSAT
##  3 (1) COMPSAT
##  4 (3) SOMESAT
##  5 (2) VERYSAT
##  6 (2) VERYSAT
```

```
##  7 (1) COMPSAT
##  8 (1) COMPSAT
##  9 (1) COMPSAT
## 10 (1) COMPSAT
## # ... with 3,607 more rows, and 5 more variables: V302 <fctr>,
## #   V303 <fctr>, V304 <fctr>, V305 <fctr>, V306 <fctr>
```

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.

```
age_respondants <- select(data_subset, V104)
```

b. Print out at least three summary statistics using `summarise()`.

```
age_respondants <- filter(age_respondants, !is.na(V104))

summarise(age_respondants, min = min(V104),
                           max = max(V104),
                           avg = mean(V104))
```

```
##   min max      avg
## 1  25  95 53.58356
```

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.

```
summarise(age_respondants, n_distinct(V104))
```

```
##   n_distinct(V104)
## 1               71
```

```
summarise(age_respondants, max(V104) >= 95)
```

```
##   max(V104) >= 95
## 1            TRUE
```

b. Why did you choose the ones you did? What did you learn about your data from these summaries?

I wanted to get a feeling for the variety of ages that were interviewed, and what the top end of the ages were. I learned that I have 71 distinct ages of respondents and that the max age is greater than or equal to 95

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.

```
as_tibble(data_subset %>%
  group_by(V301) %>%
  mutate(prob_worry = V307, V308, V309, V310, V311))
```

```
## # A tibble: 3,617 x 60
## # Groups:   V301 [6]
##              V6           V7       V103  V104        V220        V221
##          <fctr>       <fctr>     <fctr> <dbl>      <fctr>      <fctr>
##  1  (1) CORRECT  (1) CORRECT (2) FEMALE    69   (6) NEVER   (2) 1X/WK
##  2  (1) CORRECT  (1) CORRECT   (1) MALE    44  (2) 1X/DAY   (2) 1X/WK
##  3  (1) CORRECT  (1) CORRECT   (1) MALE    75  (5) <1X/WK  (1) >1X/WK
##  4  (1) CORRECT  (1) CORRECT   (1) MALE    25 (3) 2-3X/WK   (2) 1X/WK
##  5 (5) INCORRCT  (1) CORRECT (2) FEMALE    30 (3) 2-3X/WK  (1) >1X/WK
##  6  (1) CORRECT  (1) CORRECT   (1) MALE    57 (3) 2-3X/WK   (2) 1X/WK
##  7 (5) INCORRCT (5) INCORRCT (2) FEMALE    56   (6) NEVER   (2) 1X/WK
##  8 (5) INCORRCT  (1) CORRECT (2) FEMALE    37 (3) 2-3X/WK (3) 2-3X/MO
##  9  (1) CORRECT  (1) CORRECT (2) FEMALE    27 (1) >1X/DAY  (1) >1X/WK
## 10 (5) INCORRCT  (1) CORRECT   (1) MALE    73 (3) 2-3X/WK   (4) 1X/MO
## # ... with 3,607 more rows, and 54 more variables: V222 <fctr>,
## #   V223 <fctr>, V224 <fctr>, V225 <fctr>, V301 <fctr>, V302 <fctr>,
## #   V303 <fctr>, V304 <fctr>, V305 <fctr>, V306 <fctr>, V307 <fctr>,
## #   V308 <fctr>, V309 <fctr>, V310 <fctr>, V311 <fctr>, V314 <fctr>,
## #   V315 <fctr>, V316 <fctr>, V317 <fctr>, V322 <fctr>, V323 <fctr>,
## #   V325 <fctr>, V326 <fctr>, V328 <fctr>, V329 <fctr>, V330 <fctr>,
## #   V331 <fctr>, V332 <fctr>, V333 <fctr>, V334 <fctr>, V335 <fctr>,
## #   V401 <fctr>, V402 <fctr>, V405 <fctr>, V406 <fctr>, V407 <fctr>,
## #   V408 <fctr>, V410 <dbl>, V416 <fctr>, V419 <fctr>, V420 <dbl>,
## #   V425 <dbl>, V430 <fctr>, V431 <fctr>, V432 <fctr>, V433 <fctr>,
## #   V434 <fctr>, V437 <fctr>, V438 <fctr>, V440 <fctr>, V441 <dbl>,
## #   V445 <fctr>, V446 <fctr>, prob_worry <fctr>
```

b. What do the results tell you about different groups in you data?

It tells me about the overall life satisfaction of individuals in the study and the reasons attributed to that satisfaction.

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?

I want to combine the satisfaction and worry components.

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.

```
as_tibble(data_subset %>%
  mutate(prob_worry = V307, V308, V309, V310, V311) %>%
  #I have mutated the data pertaining to the things that worry the individuals to be shown all at once
  mutate(reason_sat = V303, V304, V305, V306) %>%
           group_by(V301) %>%
  #the same has been done for the reason they are satisfied
  filter(V301 == "(1) COMPSAT" & V328 == "(1) ST AGREE") %>%
  #here I am filtering only the respondants who responded both completely satisfied with life and stron
  select(V301, V328, prob_worry, reason_sat))
```

```
## # A tibble: 593 x 4
## # Groups:   V301 [1]
##           V301         V328
##         <fctr>       <fctr>
##  1 (1) COMPSAT (1) ST AGREE
##  2 (1) COMPSAT (1) ST AGREE
##  3 (1) COMPSAT (1) ST AGREE
##  4 (1) COMPSAT (1) ST AGREE
##  5 (1) COMPSAT (1) ST AGREE
##  6 (1) COMPSAT (1) ST AGREE
##  7 (1) COMPSAT (1) ST AGREE
##  8 (1) COMPSAT (1) ST AGREE
##  9 (1) COMPSAT (1) ST AGREE
## 10 (1) COMPSAT (1) ST AGREE
## # ... with 583 more rows, and 2 more variables: prob_worry <fctr>,
## #   reason_sat <fctr>
```

```
#I have selecteed the relevant variables I wish to compare at this time.
```