Lab 9 - Data Transformation

Johnathan Hsu

November 3rd, 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.

```
# package
library(tidyverse)
```

```
## -- Attaching packages ----------------------------------------------------------------------- tidyv
## √ ggplot2 2.2.1     √ purrr   0.2.4
## √ tibble  1.3.4     √ dplyr   0.7.4
## √ tidyr   0.7.2     √ stringr 1.2.0
## √ readr   1.1.1     √ forcats 0.2.0
## -- Conflicts -------------------------------------------------------------------------- tidyverse_c
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
# read data
data_95_96 <- read.csv("~/monitoring-federal-criminal-sentences/clean_data/initial_data/95-96.csv")

# making copy for this lab
test <- as_tibble(data_95_96)
tbl_df(select(data_95_96, MONRACE, MONSEX, DISPOSIT))
```

```
## # A tibble: 42,436 x 3
##            MONRACE MONSEX    DISPOSIT
##             <fctr> <fctr>      <fctr>
##  1           Black Female Guilty plea
##  2           Black   Male Guilty plea
##  3           Black   Male Guilty plea
##  4 White/Caucasian   Male Guilty plea
##  5 White/Caucasian   Male Guilty plea
##  6 White/Caucasian Female Guilty plea
##  7 White/Caucasian Female Guilty plea
##  8 White/Caucasian   Male Guilty plea
##  9 White/Caucasian   Male Guilty plea
## 10 White/Caucasian   Male Guilty plea
## # ... with 42,426 more rows
```

```
tbl_df(select(data_95_96, XCRHISSR, DISTRICT, MONCIRC))
```

```
## # A tibble: 42,436 x 3
##    XCRHISSR DISTRICT MONCIRC
##       <fctr>    <int>   <int>
##  1        3       23       4
##  2        4       45       6
##  3        3       25       4
```

```
##  4        3        4       1
##  5        1       70       9
##  6        5       70       9
##  7        1       70       9
##  8        1       41       5
##  9        3       46       6
## 10        4       42       5
## # ... with 42,426 more rows
```

```
efficient_df <- data_95_96 %>%
  select(MONRACE, DISPOSIT, MONSEX, EDUCATN, XCRHISSR, DISTRICT, MONCIRC, STATMAX, STATMIN, TOTPRISN)
```

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.

```
select(test, contains("STAT"))
```

```
## # A tibble: 42,436 x 10
##    STATMIN STATMAX NOSTATS1                                    STAT1C
##     <fctr>  <fctr>   <fctr>                                     <fctr>
##  1     120     540        2 Not Appl; Count contains 2 or less statu
##  2       0     252        2 Not Appl; Count contains 2 or less statu
##  3       0     240        3                                  18 par 2
##  4      60     480        3                                  18 par 2
##  5       0      36        2 Not Appl; Count contains 2 or less statu
##  6       0      60        2 Not Appl; Count contains 2 or less statu
##  7       0     240        1 Not Appl; Count contains 2 or less statu
##  8       0      60        1 Not Appl; Count contains 2 or less statu
##  9       0      60        1 Not Appl; Count contains 2 or less statu
## 10      60     480        1 Not Appl; Count contains 2 or less statu
## # ... with 42,426 more rows, and 6 more variables: NOSTATS2 <fctr>,
## #   STAT2C <fctr>, NOSTATS3 <fctr>, STAT3C <fctr>, NOSTATS4 <fctr>,
## #   STAT4C <fctr>
```

```
select(test, starts_with("MON"))
```

```
## # A tibble: 42,436 x 3
##    MONCIRC         MONRACE MONSEX
##      <int>          <fctr> <fctr>
##  1       4           Black Female
##  2       6           Black   Male
##  3       4           Black   Male
##  4       1 White/Caucasian   Male
```

```
##  5       9 White/Caucasian   Male
##  6       9 White/Caucasian Female
##  7       9 White/Caucasian Female
##  8       5 White/Caucasian   Male
##  9       6 White/Caucasian   Male
## 10       5 White/Caucasian   Male
## # ... with 42,426 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.

Yes - I want to find the guideline range by subtracting the statutory minimum from the maximum, I'll name it GUIDERANGE.

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.

```
test <- mutate(test, GUIDERANGE = as.numeric(STATMAX) - as.numeric(STATMIN))

# but that won't do me any good because I need to be able to compare the sentence that I can use to ref

efficient_df$DOWNADJUSTMENT <- as.numeric(as.numeric(efficient_df$STATMIN) > as.numeric(efficient_df$TO

efficient_df$UPWARDADJUSTMENT <- as.numeric(as.numeric(efficient_df$STATMAX) < as.numeric(efficient_df$
```

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 need to filter out the prison times that are just time sentences (excluding death penalty, and LWOP).

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

```
  filter(test, MONSEX == "Male")
```

```
## # A tibble: 35,893 x 221
##        X USSCIDN  RECDATE DISTRICT MONCIRC SENTDATE
##    <int>   <int>    <int>    <int>   <int>    <int>
##  1     2  248876 19960205       45       6 19960110
##  2     3  248986 19960205       25       4 19960122
##  3     4  252632 19951010        4       1 19951002
##  4     5  252645 19951006       70       9 19951002
##  5     8  252705 19951006       41       5 19951002
##  6     9  252719 19951006       46       6 19951002
##  7    10  252774 19951006       42       5 19951002
##  8    11  252781 19951030       35       5 19951018
##  9    13  252783 19951030        8       2 19951005
## 10    14  252784 19951030        8       2 19951017
## # ... with 35,883 more rows, and 215 more variables: OLDLAW <fctr>,
```

```
## #   JANDCDOC <fctr>, PLEADOC <fctr>, PSRDOC <fctr>, REASDOC <fctr>,
## #   INDIDOC <fctr>, PETTYBC <fctr>, RESENT <fctr>, POOFFICE <fctr>,
## #   DEFCNSUL <fctr>, MONRACE <fctr>, HISPORIG <fctr>, MONSEX <fctr>,
## #   EDUCATN <fctr>, CITIZEN <fctr>, CITWHERE <fctr>, AGE <fctr>,
## #   OFFTYPE <int>, OFFCASE <fctr>, OFFCODC1 <fctr>, OFFCODC2 <fctr>,
## #   OFFCODC3 <fctr>, OFFCODC4 <fctr>, NOCOUNTS <fctr>, DISPOSIT <fctr>,
## #   TOTPRISN <fctr>, TOTDAYS <fctr>, BOPCONF <fctr>, PROBATN <fctr>,
## #   SUPREL <fctr>, MOCOMCON <fctr>, MOINTCON <fctr>, MOHOMDET <fctr>,
## #   HRCOMSRV <fctr>, TYPEOTHS <fctr>, FINE <fctr>, COSTSUP <fctr>,
## #   TOTREST <fctr>, FINEWAIV <fctr>, STATMIN <fctr>, STATMAX <fctr>,
## #   NOSTATS1 <fctr>, TITLE1A <fctr>, SECT1A <fctr>, SUBSEC1A <fctr>,
## #   TITLE1B <fctr>, SECT1B <fctr>, SUBSEC1B <fctr>, STAT1C <fctr>,
## #   NOSTATS2 <fctr>, TITLE2A <fctr>, SECT2A <fctr>, SUBSEC2A <fctr>,
## #   TITLE2B <fctr>, SECT2B <fctr>, SUBSEC2B <fctr>, STAT2C <fctr>,
## #   NOSTATS3 <fctr>, TITLE3A <fctr>, SECT3A <fctr>, SUBSEC3A <fctr>,
## #   TITLE3B <fctr>, SECT3B <fctr>, SUBSEC3B <fctr>, STAT3C <fctr>,
## #   NOSTATS4 <fctr>, TITLE4A <fctr>, SECT4A <fctr>, SUBSEC4A <fctr>,
## #   TITLE4B <fctr>, SECT4B <fctr>, SUBSEC4B <fctr>, STAT4C <fctr>,
## #   ACCGDLN <fctr>, XFOLSOR <fctr>, XCRHISSR <fctr>, XMAXSOR <fctr>,
## #   XGLMAXPS <fctr>, XMINSOR <int>, XGLMINPS <int>, REASON1 <fctr>,
## #   REASON2 <fctr>, REASON3 <fctr>, DEPART <fctr>, PSROFFLV <fctr>,
## #   PSRCHCAT <fctr>, SORCHPT <fctr>, PSRCHPT <fctr>, SOURCES <fctr>,
## #   AMENDYR <fctr>, GDLINE1 <fctr>, GDLINE2 <fctr>, GDLINE3 <fctr>,
## #   GDLINE4 <fctr>, GDLINE5 <fctr>, LOSSGNS1 <fctr>, LOSSGNS2 <fctr>,
## #   LOSSGNS3 <fctr>, LOSSGNS4 <fctr>, LOSSGNS5 <fctr>, ...
```

```
  filter(test, TOTPRISN <  9000)
```

```
## Warning in Ops.factor(TOTPRISN, 9000): '<' not meaningful for factors
```

```
## # A tibble: 0 x 221
## # ... with 221 variables: X <int>, USSCIDN <int>, RECDATE <int>,
## #   DISTRICT <int>, MONCIRC <int>, SENTDATE <int>, OLDLAW <fctr>,
## #   JANDCDOC <fctr>, PLEADOC <fctr>, PSRDOC <fctr>, REASDOC <fctr>,
## #   INDIDOC <fctr>, PETTYBC <fctr>, RESENT <fctr>, POOFFICE <fctr>,
## #   DEFCNSUL <fctr>, MONRACE <fctr>, HISPORIG <fctr>, MONSEX <fctr>,
## #   EDUCATN <fctr>, CITIZEN <fctr>, CITWHERE <fctr>, AGE <fctr>,
## #   OFFTYPE <int>, OFFCASE <fctr>, OFFCODC1 <fctr>, OFFCODC2 <fctr>,
## #   OFFCODC3 <fctr>, OFFCODC4 <fctr>, NOCOUNTS <fctr>, DISPOSIT <fctr>,
## #   TOTPRISN <fctr>, TOTDAYS <fctr>, BOPCONF <fctr>, PROBATN <fctr>,
## #   SUPREL <fctr>, MOCOMCON <fctr>, MOINTCON <fctr>, MOHOMDET <fctr>,
## #   HRCOMSRV <fctr>, TYPEOTHS <fctr>, FINE <fctr>, COSTSUP <fctr>,
## #   TOTREST <fctr>, FINEWAIV <fctr>, STATMIN <fctr>, STATMAX <fctr>,
## #   NOSTATS1 <fctr>, TITLE1A <fctr>, SECT1A <fctr>, SUBSEC1A <fctr>,
## #   TITLE1B <fctr>, SECT1B <fctr>, SUBSEC1B <fctr>, STAT1C <fctr>,
## #   NOSTATS2 <fctr>, TITLE2A <fctr>, SECT2A <fctr>, SUBSEC2A <fctr>,
## #   TITLE2B <fctr>, SECT2B <fctr>, SUBSEC2B <fctr>, STAT2C <fctr>,
## #   NOSTATS3 <fctr>, TITLE3A <fctr>, SECT3A <fctr>, SUBSEC3A <fctr>,
## #   TITLE3B <fctr>, SECT3B <fctr>, SUBSEC3B <fctr>, STAT3C <fctr>,
## #   NOSTATS4 <fctr>, TITLE4A <fctr>, SECT4A <fctr>, SUBSEC4A <fctr>,
## #   TITLE4B <fctr>, SECT4B <fctr>, SUBSEC4B <fctr>, STAT4C <fctr>,
## #   ACCGDLN <fctr>, XFOLSOR <fctr>, XCRHISSR <fctr>, XMAXSOR <fctr>,
## #   XGLMAXPS <fctr>, XMINSOR <int>, XGLMINPS <int>, REASON1 <fctr>,
## #   REASON2 <fctr>, REASON3 <fctr>, DEPART <fctr>, PSROFFLV <fctr>,
## #   PSRCHCAT <fctr>, SORCHPT <fctr>, PSRCHPT <fctr>, SOURCES <fctr>,
```

```
## #   AMENDYR <fctr>, GDLINE1 <fctr>, GDLINE2 <fctr>, GDLINE3 <fctr>,
## #   GDLINE4 <fctr>, ...
```

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.
b. Print out all of the observations in your data in which none of variables are `NA`. Selecting only male and people with actual prison time (below 9000).

```
filter(test, MONSEX == "Male" & TOTPRISN < 9000)
```

```
## Warning in Ops.factor(TOTPRISN, 9000): '<' not meaningful for factors
```

```
## # A tibble: 0 x 221
## # ... with 221 variables: X <int>, USSCIDN <int>, RECDATE <int>,
## #   DISTRICT <int>, MONCIRC <int>, SENTDATE <int>, OLDLAW <fctr>,
## #   JANDCDOC <fctr>, PLEADOC <fctr>, PSRDOC <fctr>, REASDOC <fctr>,
## #   INDIDOC <fctr>, PETTYBC <fctr>, RESENT <fctr>, POOFFICE <fctr>,
## #   DEFCNSUL <fctr>, MONRACE <fctr>, HISPORIG <fctr>, MONSEX <fctr>,
## #   EDUCATN <fctr>, CITIZEN <fctr>, CITWHERE <fctr>, AGE <fctr>,
## #   OFFTYPE <int>, OFFCASE <fctr>, OFFCODC1 <fctr>, OFFCODC2 <fctr>,
## #   OFFCODC3 <fctr>, OFFCODC4 <fctr>, NOCOUNTS <fctr>, DISPOSIT <fctr>,
## #   TOTPRISN <fctr>, TOTDAYS <fctr>, BOPCONF <fctr>, PROBATN <fctr>,
## #   SUPREL <fctr>, MOCOMCON <fctr>, MOINTCON <fctr>, MOHOMDET <fctr>,
## #   HRCOMSRV <fctr>, TYPEOTHS <fctr>, FINE <fctr>, COSTSUP <fctr>,
## #   TOTREST <fctr>, FINEWAIV <fctr>, STATMIN <fctr>, STATMAX <fctr>,
## #   NOSTATS1 <fctr>, TITLE1A <fctr>, SECT1A <fctr>, SUBSEC1A <fctr>,
## #   TITLE1B <fctr>, SECT1B <fctr>, SUBSEC1B <fctr>, STAT1C <fctr>,
## #   NOSTATS2 <fctr>, TITLE2A <fctr>, SECT2A <fctr>, SUBSEC2A <fctr>,
## #   TITLE2B <fctr>, SECT2B <fctr>, SUBSEC2B <fctr>, STAT2C <fctr>,
## #   NOSTATS3 <fctr>, TITLE3A <fctr>, SECT3A <fctr>, SUBSEC3A <fctr>,
## #   TITLE3B <fctr>, SECT3B <fctr>, SUBSEC3B <fctr>, STAT3C <fctr>,
## #   NOSTATS4 <fctr>, TITLE4A <fctr>, SECT4A <fctr>, SUBSEC4A <fctr>,
## #   TITLE4B <fctr>, SECT4B <fctr>, SUBSEC4B <fctr>, STAT4C <fctr>,
## #   ACCGDLN <fctr>, XFOLSOR <fctr>, XCRHISSR <fctr>, XMAXSOR <fctr>,
## #   XGLMAXPS <fctr>, XMINSOR <int>, XGLMINPS <int>, REASON1 <fctr>,
## #   REASON2 <fctr>, REASON3 <fctr>, DEPART <fctr>, PSROFFLV <fctr>,
## #   PSRCHCAT <fctr>, SORCHPT <fctr>, PSRCHPT <fctr>, SOURCES <fctr>,
## #   AMENDYR <fctr>, GDLINE1 <fctr>, GDLINE2 <fctr>, GDLINE3 <fctr>,
## #   GDLINE4 <fctr>, ...
```

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

My data would best be arranged by the year, because it will eventually be a panel data. I won't do it here because it's not merged yet, but if I bind it in the right order of years I shouldnt need to use this. I will

arrange it by the prison time here to practice.

```
arrange(test, TOTPRISN)
```

```
## # A tibble: 42,436 x 221
##        X USSCIDN  RECDATE DISTRICT MONCIRC SENTDATE
##    <int>   <int>    <int>    <int>   <int>    <int>
##  1     7  252647 19951006       70       9 19951002
##  2   173  253622 19951012       70       9 19951002
##  3   183  253645 19951017       48       6 19951005
##  4   190  253663 19951017       34      11 19951003
##  5   684  254804 19951024       70       9 19951005
##  6   814  255101 19951026       44       6 19951006
##  7   817  255104 19951026       44       6 19951010
##  8   901  255241 19951027       50       6 19951018
##  9  1170  255686 19951030       24       4 19951023
## 10  1319  256002 19951031       68       8 19951026
## # ... with 42,426 more rows, and 215 more variables: OLDLAW <fctr>,
## #   JANDCDOC <fctr>, PLEADOC <fctr>, PSRDOC <fctr>, REASDOC <fctr>,
## #   INDIDOC <fctr>, PETTYBC <fctr>, RESENT <fctr>, POOFFICE <fctr>,
## #   DEFCNSUL <fctr>, MONRACE <fctr>, HISPORIG <fctr>, MONSEX <fctr>,
## #   EDUCATN <fctr>, CITIZEN <fctr>, CITWHERE <fctr>, AGE <fctr>,
## #   OFFTYPE <int>, OFFCASE <fctr>, OFFCODC1 <fctr>, OFFCODC2 <fctr>,
## #   OFFCODC3 <fctr>, OFFCODC4 <fctr>, NOCOUNTS <fctr>, DISPOSIT <fctr>,
## #   TOTPRISN <fctr>, TOTDAYS <fctr>, BOPCONF <fctr>, PROBATN <fctr>,
## #   SUPREL <fctr>, MOCOMCON <fctr>, MOINTCON <fctr>, MOHOMDET <fctr>,
## #   HRCOMSRV <fctr>, TYPEOTHS <fctr>, FINE <fctr>, COSTSUP <fctr>,
## #   TOTREST <fctr>, FINEWAIV <fctr>, STATMIN <fctr>, STATMAX <fctr>,
## #   NOSTATS1 <fctr>, TITLE1A <fctr>, SECT1A <fctr>, SUBSEC1A <fctr>,
## #   TITLE1B <fctr>, SECT1B <fctr>, SUBSEC1B <fctr>, STAT1C <fctr>,
## #   NOSTATS2 <fctr>, TITLE2A <fctr>, SECT2A <fctr>, SUBSEC2A <fctr>,
## #   TITLE2B <fctr>, SECT2B <fctr>, SUBSEC2B <fctr>, STAT2C <fctr>,
## #   NOSTATS3 <fctr>, TITLE3A <fctr>, SECT3A <fctr>, SUBSEC3A <fctr>,
## #   TITLE3B <fctr>, SECT3B <fctr>, SUBSEC3B <fctr>, STAT3C <fctr>,
## #   NOSTATS4 <fctr>, TITLE4A <fctr>, SECT4A <fctr>, SUBSEC4A <fctr>,
## #   TITLE4B <fctr>, SECT4B <fctr>, SUBSEC4B <fctr>, STAT4C <fctr>,
## #   ACCGDLN <fctr>, XFOLSOR <fctr>, XCRHISSR <fctr>, XMAXSOR <fctr>,
## #   XGLMAXPS <fctr>, XMINSOR <int>, XGLMINPS <int>, REASON1 <fctr>,
## #   REASON2 <fctr>, REASON3 <fctr>, DEPART <fctr>, PSROFFLV <fctr>,
## #   PSRCHCAT <fctr>, SORCHPT <fctr>, PSRCHPT <fctr>, SOURCES <fctr>,
## #   AMENDYR <fctr>, GDLINE1 <fctr>, GDLINE2 <fctr>, GDLINE3 <fctr>,
## #   GDLINE4 <fctr>, GDLINE5 <fctr>, LOSSGNS1 <fctr>, LOSSGNS2 <fctr>,
## #   LOSSGNS3 <fctr>, LOSSGNS4 <fctr>, LOSSGNS5 <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. Mode of the type of case (guilty plea, trial, dismissal…etc.), level of education of the defendant, and age.

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

```
# Create a mode funciton:

getmode <- function(v) {
   uniqv <- unique(v)
   uniqv[which.max(tabulate(match(v, uniqv)))]
}

# now summarise using 'getmode()' funciton

  summarise(test, getmode(test$DISPOSIT),
                  getmode(test$EDUCATN),
                  getmode(AGE))
```

```
## # A tibble: 1 x 3
##   `getmode(test$DISPOSIT)` `getmode(test$EDUCATN)` `getmode(AGE)`
##                      <fctr>                  <fctr>         <fctr>
## 1              Guilty plea    High school graduate             25
```

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?

```
test %>%
  filter(!is.na(TOTPRISN)) %>%
  summarise(last(TOTPRISN),
            n_distinct(DISTRICT))
```

```
## # A tibble: 1 x 2
##                 `last(TOTPRISN)` `n_distinct(DISTRICT)`
##                           <fctr>                 <int>
## 1 No prison or - 1 month ordered                    94
```

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?

I don't think this is applicable to my data.

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.

I don't think there are any variables that can be effectively mutated and included in my model. I think the adjustment variables will be useful, however, as it speaks on discretion taken by judges at sentencing.