Stat 140: R Commands for Loading and Wrangling Data

Loading Data and Taking a First Look	
Load a package	`library(dplyr)`
Assign a value to a variable	`my_var <- 3`
Read a csv file	`nhanes <- read_csv("path/to/nhanes.csv")`
First few lines of data frame	`head(nhanes)`
Structure of data frame	`str(nhanes)`
Dimensions of data frame	`dim(nhanes)`, `nrow(nhanes)`, `ncol(nhanes)`
Working with Categorical Variables	
Convert a nominal categorical variable to factor	`nhanes <- nhanes %>% mutate(Gender = factor(Gender))`

Convert an ordinal categorical variable to factor	`nhanes <- nhanes %>% mutate(`
	`  Education = factor(Education,`
	`    levels = c("High School", "Some College", "College Grad"),`
	`    ordered = TRUE))`

View the distinct values of a variable (mainly useful for categorical variables)	`nhanes %>% distinct(Education)`
Count number of obs. in each level of a categorical variable	`nhanes %>% count(Education)`

Count number of obs. in each combination of levels of two categorical var's	`nhanes %>% count(Education, Gender) %>%`
	`  spread(Gender, n)`

Summarizing Quantitative Variables	

Mean of quantitative variable, for each level of a categorical variable	`nhanes %>%`
	`  group_by(MaritalStatus) %>%`
	`  summarize(mean_poverty_index = mean(Poverty, na.rm = TRUE))`

Data Wrangling	

Add or modify a variable in a data frame	`nhanes_modified <- nhanes %>%`
	`  mutate(Weight_pounds = Weight * 2.205)`

Filter observational units, character condition	`nhanes_fewer_obs_units <- nhanes %>%`
	`  filter(Education == "High School")`

Filter, character condition with multiple values	`nhanes_fewer_obs_units <- nhanes %>%`
	`  filter(Education == "High School" | Education == "Some College")`

Filter, numeric condition	`nhanes_fewer_obs_units <- nhanes %>%`
	`  filter(Age >= 22)`

Filter, multiple conditions	`nhanes_fewer_obs_units <- nhanes %>%`
	`  filter(Education == "High School", Age >= 22)`

Sort, ascending order	`nhanes_sorted <- nhanes %>% arrange(Age)`
Sort, descending order	`nhanes_sorted <- nhanes %>% arrange(desc(Age))`

In this document I am going to summarize the main commands and concepts for R that we have learned so far – along with a couple of others that you haven't seen but are closely related to what we've done so far. These are organized into four main groups:

1. R variables and the assignment operator
2. Basic interactions with data frames
    a. Reading data into R from spreadsheet files
    b. Getting a first look at what's in a data frame
    c. Converting categorical variables to factors
3. Summarizing categorical variables
4. Summarizing quantitative variables
5. Data wrangling

I will illustrate the ideas using the NHANES data we looked at in Lab 1.

1. R variables and the assignment operator

In R, we use the word "variable" in two ways. The first is a name that we've given a value that we want to be able to re-use later. In the example below, `my_var` is a variable. We have *assigned* the value 3 to it using the *assignment operator*, `<-` (a less than sign followed by a minus sign, to form an arrow).

```
my_var <- 3
```

We can see the value that's currently assigned to `my_var` by entering the name of the variable on its own line:

```
my_var
```

```
## [1] 3
```

We can also use that value in later calculations:

```
my_var * 2
```

```
## [1] 6
```

The second meaning of the word "variable" is more closely related to our use of the word in statistics: a column in a data frame. We'll look at that next.

2. Basic interactions with data frames

In R, the most common way to store data is in a data frame. You can think of a data frame as being like a spreadsheet. Each row corresponds to an observational unit, and each column corresponds to a variable.

a. Reading data into R from spreadsheet files

Usually, the data are stored in a spreadsheet-like file outside of R. The file format we'll work with most in this class is a csv file (csv stands for comma separated value). We can read in csv files using the `read_csv` function, which is in the `readr` package:

```
library(readr)
nhanes <- read_csv("http://www.evanlray.com/data/misc/nhanes/nhanes.csv")
```

```
## Parsed with column specification:
## cols(
##   ID = col_integer(),
##   Gender = col_character(),
##   Age = col_integer(),
##   Race = col_character(),
##   Education = col_character(),
##   MaritalStatus = col_character(),
##   HHIncome = col_character(),
##   Poverty = col_double(),
##   Weight = col_double(),
##   Length = col_double(),
##   Height = col_double(),
##   Diabetes = col_character(),
##   nPregnancies = col_integer(),
##   nBabies = col_integer(),
##   PregnantNow = col_character()
## )
```

If the data file was stored on your computer instead of on the class website, you would change the file location in these commands to where the file is located on your computer.

There are also functions to read in data from other file formats. For example, if your data were stored in an excel file (with a file extension like xlsx), you could use the `read_excel` function from the `readxl` package to read the data in. This function doesn't handle reading files from the internet very well yet, so we won't use it much in this class – but it's there if you need it later.

b. Getting a first look at what's in the data frame

There are a couple of questions I always ask myself whenever I'm thinking about a new data set:

1. How many observational units and variables are in this data set?
2. What are the variables and variable types?

We've talked about three functions that can be used to help answer these questions.

head

The `head` function shows you the first few rows of the data set (by default, the first 6 rows). It's good for getting a quick summary of what's in the data frame, but it will not tell you how many observational units there are.

```r
head(nhanes)
```

```
## # A tibble: 6 x 15
##      ID Gender   Age Race  Education MaritalStatus HHIncome Poverty Weight
##   <int> <chr>  <int> <chr> <chr>     <chr>         <chr>      <dbl>  <dbl>
## 1  3923 female    80 White High Sch~ Married       55000-6~    4.27   71.1
## 2  1548 male      42 Black 9 - 11th~ LivePartner   5000-99~    0.3   115.
## 3  1205 male       4 Hisp~ <NA>      <NA>          25000-3~    0.78   19.7
## 4  1519 male      12 Black <NA>      <NA>          75000-9~    2.96   63.7
## 5  4148 male       1 Black <NA>      <NA>          15000-1~    0.67   11.7
## 6  1681 female    14 White <NA>      <NA>          25000-3~    1.52   71.6
## # ... with 6 more variables: Length <dbl>, Height <dbl>, Diabetes <chr>,
## #   nPregnancies <int>, nBabies <int>, PregnantNow <chr>
```

str

The `str` function will print out some more detailed information about the data frame, including how many observational units and variables there are, and the type of each variable – but its output is a little less well organized.

```r
str(nhanes)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    5000 obs. of  15 variables:
##  $ ID           : int  3923 1548 1205 1519 4148 1681 3710 3733 2552 373 ...
##  $ Gender       : chr  "female" "male" "male" "male" ...
##  $ Age          : int  80 42 4 12 1 14 56 0 33 46 ...
##  $ Race         : chr  "White" "Black" "Hispanic" "Black" ...
##  $ Education    : chr  "High School" "9 - 11th Grade" NA NA ...
##  $ MaritalStatus: chr  "Married" "LivePartner" NA NA ...
##  $ HHIncome     : chr  "55000-64999" "5000-9999" "25000-34999" "75000-99999" ...
##  $ Poverty      : num  4.27 0.3 0.78 2.96 0.67 1.52 5 2.46 NA 2.81 ...
##  $ Weight       : num  71.1 115.4 19.7 63.7 11.7 ...
##  $ Length       : num  NA NA NA NA 84.2 NA NA 61.7 NA NA ...
##  $ Height       : num  162 165 110 170 NA ...
##  $ Diabetes     : chr  "Yes" "Yes" "No" "No" ...
##  $ nPregnancies : int  5 NA NA NA NA NA 2 NA NA 3 ...
##  $ nBabies      : int  4 NA NA NA NA NA 2 NA NA 2 ...
##  $ PregnantNow  : chr  NA NA NA NA ...
##  - attr(*, "spec")=List of 2
##   ..$ cols   :List of 15
##   .. ..$ ID           : list()
##   .. .. ..- attr(*, "class")= chr  "collector_integer" "collector"
##   .. ..$ Gender       : list()
##   .. .. ..- attr(*, "class")= chr  "collector_character" "collector"
##   .. ..$ Age          : list()
##   .. .. ..- attr(*, "class")= chr  "collector_integer" "collector"
##   .. ..$ Race         : list()
##   .. .. ..- attr(*, "class")= chr  "collector_character" "collector"
##   .. ..$ Education    : list()
##   .. .. ..- attr(*, "class")= chr  "collector_character" "collector"
##   .. ..$ MaritalStatus: list()
##   .. .. ..- attr(*, "class")= chr  "collector_character" "collector"
##   .. ..$ HHIncome     : list()
##   .. .. ..- attr(*, "class")= chr  "collector_character" "collector"
##   .. ..$ Poverty      : list()
##   .. .. ..- attr(*, "class")= chr  "collector_double" "collector"
##   .. ..$ Weight       : list()
##   .. .. ..- attr(*, "class")= chr  "collector_double" "collector"
##   .. ..$ Length       : list()
##   .. .. ..- attr(*, "class")= chr  "collector_double" "collector"
##   .. ..$ Height       : list()
##   .. .. ..- attr(*, "class")= chr  "collector_double" "collector"
##   .. ..$ Diabetes     : list()
##   .. .. ..- attr(*, "class")= chr  "collector_character" "collector"
##   .. ..$ nPregnancies : list()
##   .. .. ..- attr(*, "class")= chr  "collector_integer" "collector"
##   .. ..$ nBabies      : list()
##   .. .. ..- attr(*, "class")= chr  "collector_integer" "collector"
##   .. ..$ PregnantNow  : list()
##   .. .. ..- attr(*, "class")= chr  "collector_character" "collector"
##   ..$ default: list()
##   .. ..- attr(*, "class")= chr  "collector_guess" "collector"
##   ..- attr(*, "class")= chr "col_spec"
```

`dim`, `nrow`, and `ncol`

The `dim` function will tell you how many rows (i.e., how many observational units) and columns (i.e., how many variables) are in the data frame (in that order). The `nrow` function will tell you how many rows there are, and the `ncol` function will tell you how many columns there are.

```
dim(nhanes)
```

```
## [1] 5000   15
```

```
nrow(nhanes)
```

```
## [1] 5000
```

```
ncol(nhanes)
```

```
## [1] 15
```

c. Converting categorical variables to factors

When you first read a data set in, quantitative data types will usually be assigned the correct data type in R, but categorical variables will typically be stored as a character data type in R. We'll need to tell R that these are categorical variables by converting them to `factors`. A factor is just R's name for a categorical variable.

Remember that we divide categorical variables into two sub-types:

1. Nominal, where there is no specific order to the categories (for example think of eye color – the categories might be blue, green, brown, etc., and there is no specific order to those categories)
2. Ordinal, where there is a specific order to the categories (for example think of education level – the categories might be "less than high school degree", "some college", "college degree", "graduate degree")

The difference in reading these into R is in whether or not we need to specify an `ordered = TRUE` argument to the `factor` function.

In both cases, we will use the `mutate` function to modify the data frame. The `mutate` function will be described more later in this document. It is in the `dplyr` package, so we need to load that package before we can use it:

```
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
```

Converting a nominal categorical variable to a `factor`

```
nhanes <- nhanes %>%
    mutate(Gender = factor(Gender))
```

Converting an ordinal categorical variable to an *ordered* `factor`

```
nhanes <- nhanes %>%
    mutate(
        Education = factor(Education,
            levels = c("8th Grade", "9 - 11th Grade", "High School", "Some College", "College Grad"),
            ordered = TRUE)
    )
```

For an ordinal variable, we need to add two more arguments to the call to `factor`:

- specify the `levels` of the variable in order tell R what order they come in.
- `ordered = TRUE` argument to tell R that it needs to pay attention to and remember the order we specified above.

Listing distinct values of a variable

In order to know what to list for the possible levels of an ordinal categorical variable, you can use the `distinct` function to list the distinct values of the variable:

```
nhanes %>% distinct(Education)
```

```
## # A tibble: 6 x 1
##   Education
##   <ord>
## 1 High School
## 2 9 - 11th Grade
## 3 <NA>
## 4 Some College
## 5 College Grad
## 6 8th Grade
```

3. Summarizing Categorical Variables

It is often helpful to obtain counts of how many observational units fall into each category of a categorical variable, or into each combination of categories for two categorical variables. We will do this with the `count` function:

```
nhanes %>% count(Education)
```

```
## # A tibble: 6 x 2
##   Education          n
##   <ord>          <int>
## 1 8th Grade        212
## 2 9 - 11th Grade   405
## 3 High School      679
## 4 Some College    1160
## 5 College Grad    1128
## 6 <NA>            1416
```

```
nhanes %>% count(Education, Gender)
```

```
## # A tibble: 12 x 3
##    Education      Gender     n
##    <ord>          <fct>  <int>
##  1 8th Grade      female    91
##  2 8th Grade      male     121
##  3 9 - 11th Grade female   174
##  4 9 - 11th Grade male     231
##  5 High School    female   338
##  6 High School    male     341
##  7 Some College   female   615
##  8 Some College   male     545
##  9 College Grad   female   584
## 10 College Grad   male     544
## 11 <NA>           female   693
## 12 <NA>           male     723
```

Sometimes for two variables, it's helpful to convert the summaries above into a contingency table, with one variable in the rows and the other in the columns. We can do this by adding on a call to the `spread` function, which is in the `tidyr` package:

```
library(tidyr)
```

```
##
## Attaching package: 'tidyr'

## The following object is masked from 'package:Matrix':
##
##     expand
```

```
nhanes %>%
  count(Education, Gender) %>%
  spread(Gender, n)
```

```
## # A tibble: 6 x 3
##   Education      female  male
##   <ord>           <int> <int>
## 1 8th Grade          91   121
## 2 9 - 11th Grade    174   231
## 3 High School       338   341
## 4 Some College      615   545
## 5 College Grad      584   544
## 6 <NA>              693   723
```

4. Summarizing Quantitative Variables

We can use the `summarize` function to calculate summaries of quantitative variables in a data set:

```
nhanes %>%
  summarize(
    mean_poverty_index = mean(Poverty, na.rm = TRUE),
    median_poverty_index = median(Poverty, na.rm = TRUE),
    q1_poverty_index = quantile(Poverty, probs = 0.25, na.rm = TRUE),
    q3_poverty_index = quantile(Poverty, probs = 0.75, na.rm = TRUE),
    iqr_poverty_index = IQR(Poverty, na.rm = TRUE),
    var_poverty_index = var(Poverty, na.rm = TRUE),
    sd_poverty_index = sd(Poverty, na.rm = TRUE)
  )
```

```
## # A tibble: 1 x 7
##   mean_poverty_index median_poverty_ind~ q1_poverty_index q3_poverty_index
##                <dbl>               <dbl>            <dbl>            <dbl>
## 1               2.76                 2.6             1.19             4.76
## # ... with 3 more variables: iqr_poverty_index <dbl>,
## #   var_poverty_index <dbl>, sd_poverty_index <dbl>
```

If you don't need to worry about missing values in your data set, you don't need the `na.rm = TRUE` part in the calls above. Note that ordinarily, you'd probably only need to compute a couple of these summaries.

If we want to compute these summaries separately for each level of a categorical variable, we can `group_by` that categorical variable:

```
nhanes %>%
  group_by(MaritalStatus) %>%
  summarize(
    mean_poverty_index = mean(Poverty, na.rm = TRUE),
    median_poverty_index = median(Poverty, na.rm = TRUE),
    q1_poverty_index = quantile(Poverty, probs = 0.25, na.rm = TRUE),
    q3_poverty_index = quantile(Poverty, probs = 0.75, na.rm = TRUE),
    iqr_poverty_index = IQR(Poverty, na.rm = TRUE),
    var_poverty_index = var(Poverty, na.rm = TRUE),
    sd_poverty_index = sd(Poverty, na.rm = TRUE)
  )
```

```
## # A tibble: 7 x 8
##   MaritalStatus mean_poverty_index median_poverty_index q1_poverty_index
##   <chr>                      <dbl>                <dbl>            <dbl>
## 1 Divorced                    2.54                 2.2              1.19
## 2 LivePartner                 2.46                 1.95             1
## 3 Married                     3.36                 3.64             1.85
## 4 NeverMarried                2.42                 1.97             0.9
## 5 Separated                   1.87                 1.36             0.96
## 6 Widowed                     2.23                 1.79             1.06
## 7 <NA>                        2.38                 1.88             0.93
## # ... with 4 more variables: q3_poverty_index <dbl>,
## #   iqr_poverty_index <dbl>, var_poverty_index <dbl>,
## #   sd_poverty_index <dbl>
```

In this class, we will use the following summary functions:

- Summaries of center:
  - `mean` calculates the mean

    - `median` calculates the median
- Summaries of spread:
    - `var` calculates the variance
    - `sd` calculates the standard deviation
    - `IQR` calculates the interquartile range
- Other:
    - `quantile(..., probs = 0.25)` calculates the 25th percentile

5. Data Wrangling

In this class, we will learn about a few of the most common operations you may want to perform on data sets. Here are the ones we've talked about so far; we'll add a couple more to this list later:

- a. Add new **variables** or modify existing **variables** (remember that variables correspond to columns of the data frame):

    - `mutate`: add a new variable or modify an existing variable

- b. Keep a subset of **observational units** (rows):

    - `filter`: keep only a subset of the observational units in the data frame that meet conditions you specify

- c. Arrange the **observational units** (rows) in order:

    - `arrange`: sort the observations in order according to one of the variables

All of these functions are in the `dplyr` package, so we'll need to load that package:

```
library(dplyr)
```

a. mutate

The basic use of `mutate` looks like this:

```
<name of modified data frame> <- <original data frame> %>%
    mutate(
        <new/modified variable 1> = <how to calculate new/modified variable 1>,
        <new/modified variable 2> = <how to calculate new/modified variable 2>
    )
```

Note that the `mutate` function does not necessarily modify the original data frame: it creates a second copy, and leaves the original as it was.

Suppose we want to convert the subjects' weight in kilograms to a weight in pounds, and add the weight in pounds to the data frame as a new variable called `Weight_pounds`. Here's how we can do that (there are 2.205 pounds in a kilogram):

```
nhanes_with_weight_in_pounds <- nhanes %>%
    mutate(Weight_pounds = Weight * 2.205)
```

Here's a look at the structure of the newly created data frame, `nhanes_with_weight_in_pounds`. Note the addition of a new variable at the end called `Weight_pounds`. If we were to look at the original `nhanes` data frame, we would see that it was not changed.

```
str(nhanes_with_weight_in_pounds)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    5000 obs. of  16 variables:
##  $ ID           : int  3923 1548 1205 1519 4148 1681 3710 3733 2552 373 ...
##  $ Gender       : Factor w/ 2 levels "female","male": 1 2 2 2 2 1 1 2 2 1 ...
##  $ Age          : int  80 42 4 12 1 14 56 0 33 46 ...
##  $ Race         : chr  "White" "Black" "Hispanic" "Black" ...
##  $ Education    : Ord.factor w/ 5 levels "8th Grade"<"9 - 11th Grade"<..: 3 2 NA NA NA NA 4 NA 2 4 ...
##  $ MaritalStatus: chr  "Married" "LivePartner" NA NA ...
##  $ HHIncome     : chr  "55000-64999" "5000-9999" "25000-34999" "75000-99999" ...
##  $ Poverty      : num  4.27 0.3 0.78 2.96 0.67 1.52 5 2.46 NA 2.81 ...
##  $ Weight       : num  71.1 115.4 19.7 63.7 11.7 ...
##  $ Length       : num  NA NA NA NA 84.2 NA NA 61.7 NA NA ...
##  $ Height       : num  162 165 110 170 NA ...
##  $ Diabetes     : chr  "Yes" "Yes" "No" "No" ...
##  $ nPregnancies : int  5 NA NA NA NA NA 2 NA NA 3 ...
##  $ nBabies      : int  4 NA NA NA NA NA 2 NA NA 2 ...
##  $ PregnantNow  : chr  NA NA NA NA ...
##  $ Weight_pounds: num  156.8 254.5 43.4 140.5 25.8 ...
```

b. filter

We often want to look at a subset of the observational units in a data frame. The `filter` command lets us do this by specifying values of the variables we want to keep. In this class, we will use a small amount of the filtering capabilities that R provides. Here are a few examples of some filters we will use. As with the `mutate` command, `filter` does not modify the original data set.

Filter according to the value of a categorical variable

In the command below we keep only observational units with an `Education` level of "High School". Note the use of two equals signs and quotes around the value we want to keep.

```
nhanes_fewer_obs_units <- nhanes %>%
    filter(Education == "High School")
```

```
head(nhanes_fewer_obs_units)
```

```
## # A tibble: 6 x 15
##      ID Gender   Age Race  Education MaritalStatus HHIncome Poverty Weight
##   <int> <fct>  <int> <chr> <ord>     <chr>         <chr>      <dbl>  <dbl>
## 1  3923 female    80 White High Sch~ Married       55000-6~    4.27   71.1
## 2  4880 male      80 White High Sch~ Married       45000-5~    3.48   86.4
## 3  1858 female    73 White High Sch~ Married       25000-3~    1.91   91.6
## 4   181 female    80 White High Sch~ Married       35000-4~    2.64   81.6
## 5  4991 male      80 White High Sch~ Married       55000-6~    4.08   71.5
## 6  1895 male      58 Mexi~ High Sch~ Married       20000-2~    1.56   80.7
## # ... with 6 more variables: Length <dbl>, Height <dbl>, Diabetes <chr>,
## #   nPregnancies <int>, nBabies <int>, PregnantNow <chr>
```

Filter according to the value of a categorical variable, keep multiple values

In the command below we keep only observational units with an `Education` level of "High School" or "Some College". Note the use of two equals signs and quotes around the values we want to keep. The vertical line in between the two possible values can be read as "or". On my keyboard, that symbol is above the backslash, on the right side of the keyboard.

```
nhanes_fewer_obs_units <- nhanes %>%
    filter(Education == "High School" | Education == "Some College")
```

```
head(nhanes_fewer_obs_units)
```

```
## # A tibble: 6 x 15
##      ID Gender   Age Race  Education MaritalStatus HHIncome Poverty Weight
##   <int> <fct>  <int> <chr> <ord>     <chr>         <chr>      <dbl>  <dbl>
## 1  3923 female    80 White High Sch~ Married       55000-6~    4.27   71.1
## 2  3710 female    56 White Some Col~ Married       more 99~    5      102.
## 3   373 female    46 White Some Col~ Divorced      45000-5~    2.81   90.9
## 4  4370 female    57 White Some Col~ Married       45000-5~    3.47   58.7
## 5  4880 male      80 White High Sch~ Married       45000-5~    3.48   86.4
## 6  1858 female    73 White High Sch~ Married       25000-3~    1.91   91.6
## # ... with 6 more variables: Length <dbl>, Height <dbl>, Diabetes <chr>,
## #   nPregnancies <int>, nBabies <int>, PregnantNow <chr>
```

Filter according to the value of a quantitative variable

Here we keep only the observational units with an Age of at least 22:

```
nhanes_fewer_obs_units <- nhanes %>%
    filter(Age >= 22)
```

```
head(nhanes_fewer_obs_units)
```

```
## # A tibble: 6 x 15
##      ID Gender   Age Race  Education MaritalStatus HHIncome Poverty Weight
##   <int> <fct>  <int> <chr> <ord>     <chr>         <chr>      <dbl>  <dbl>
## 1  3923 female    80 White High Sch~ Married       55000-6~    4.27   71.1
## 2  1548 male      42 Black 9 - 11th~ LivePartner   5000-99~    0.3   115.
## 3  3710 female    56 White Some Col~ Married       more 99~    5     102.
## 4  2552 male      33 Mexi~ 9 - 11th~ Married       <NA>       NA      90.1
## 5   373 female    46 White Some Col~ Divorced      45000-5~    2.81   90.9
## 6  4370 female    57 White Some Col~ Married       45000-5~    3.47   58.7
## # ... with 6 more variables: Length <dbl>, Height <dbl>, Diabetes <chr>,
## #   nPregnancies <int>, nBabies <int>, PregnantNow <chr>
```

We could also use a variety of other conditions:

```
nhanes_fewer_obs_units <- nhanes %>%
    filter(Age < 22)
```

```
nhanes_fewer_obs_units <- nhanes %>%
    filter(Age <= 22)
```

```
nhanes_fewer_obs_units <- nhanes %>%
    filter(Age == 22)
```

```
nhanes_fewer_obs_units <- nhanes %>%
    filter(Age > 22)
```

Filter according to multiple conditions

If we have multiple conditions, they can be separated by commas in the call to the filter function:

```
nhanes_fewer_obs_units <- nhanes %>%
    filter(Education == "High School" | Education == "Some College", Age > 22)
```

```
head(nhanes_fewer_obs_units)
```

```
## # A tibble: 6 x 15
##      ID Gender   Age Race  Education MaritalStatus HHIncome Poverty Weight
##   <int> <fct>  <int> <chr> <ord>     <chr>         <chr>      <dbl>  <dbl>
## 1  3923 female    80 White High Sch~ Married       55000-6~    4.27   71.1
## 2  3710 female    56 White Some Col~ Married       more 99~    5     102.
## 3   373 female    46 White Some Col~ Divorced      45000-5~    2.81   90.9
## 4  4370 female    57 White Some Col~ Married       45000-5~    3.47   58.7
## 5  4880 male      80 White High Sch~ Married       45000-5~    3.48   86.4
## 6  1858 female    73 White High Sch~ Married       25000-3~    1.91   91.6
## # ... with 6 more variables: Length <dbl>, Height <dbl>, Diabetes <chr>,
## #   nPregnancies <int>, nBabies <int>, PregnantNow <chr>
```

c. `arrange`

The `arrange` function lets you sort the observational units in a data frame according to the values of one of the variables.

Sort in ascending order (the default)

```r
nhanes_sorted <- nhanes %>%
    arrange(Age)
```

```r
head(nhanes_sorted)
```

```
## # A tibble: 6 x 15
##      ID Gender   Age Race  Education MaritalStatus HHIncome Poverty Weight
##   <int> <fct>  <int> <chr> <ord>     <chr>         <chr>      <dbl>  <dbl>
## 1  3733 male       0 White <NA>      <NA>          55000-6~    2.46    6.2
## 2  2361 female     0 White <NA>      <NA>          15000-1~    0.78    5.5
## 3  1441 female     0 Hisp~ <NA>      <NA>          10000-1~    0.37    6.3
## 4  3911 female     0 White <NA>      <NA>          35000-4~    1.83    7.7
## 5  1902 male       0 White <NA>      <NA>          75000-9~    3.44    5.6
## 6  1716 female     0 White <NA>      <NA>          25000-3~    1.03    9.5
## # ... with 6 more variables: Length <dbl>, Height <dbl>, Diabetes <chr>,
## #   nPregnancies <int>, nBabies <int>, PregnantNow <chr>
```

```r
head(nhanes)
```

```
## # A tibble: 6 x 15
##      ID Gender   Age Race  Education MaritalStatus HHIncome Poverty Weight
##   <int> <fct>  <int> <chr> <ord>     <chr>         <chr>      <dbl>  <dbl>
## 1  3923 female    80 White High Sch~ Married       55000-6~    4.27   71.1
## 2  1548 male      42 Black 9 - 11th~ LivePartner   5000-99~    0.3   115.
## 3  1205 male       4 Hisp~ <NA>      <NA>          25000-3~    0.78   19.7
## 4  1519 male      12 Black <NA>      <NA>          75000-9~    2.96   63.7
## 5  4148 male       1 Black <NA>      <NA>          15000-1~    0.67   11.7
## 6  1681 female    14 White <NA>      <NA>          25000-3~    1.52   71.6
## # ... with 6 more variables: Length <dbl>, Height <dbl>, Diabetes <chr>,
## #   nPregnancies <int>, nBabies <int>, PregnantNow <chr>
```

Sort in descending order

To sort in descending order, we wrap the variable we want to sort by in `desc()`:

```r
nhanes_sorted <- nhanes %>%
    arrange(desc(Age))
```

```r
head(nhanes_sorted)
```

```
## # A tibble: 6 x 15
##      ID Gender   Age Race  Education MaritalStatus HHIncome Poverty Weight
##   <int> <fct>  <int> <chr> <ord>     <chr>         <chr>      <dbl>  <dbl>
## 1  3923 female    80 White High Sch~ Married       55000-6~    4.27   71.1
## 2  4880 male      80 White High Sch~ Married       45000-5~    3.48   86.4
## 3   181 female    80 White High Sch~ Married       35000-4~    2.64   81.6
## 4  4991 male      80 White High Sch~ Married       55000-6~    4.08   71.5
## 5   244 male      80 White College ~ NeverMarried  35000-4~    3.21   72.1
## 6  3617 male      80 White College ~ Married       25000-3~    1.98   72.6
## # ... with 6 more variables: Length <dbl>, Height <dbl>, Diabetes <chr>,
## #   nPregnancies <int>, nBabies <int>, PregnantNow <chr>
```

```r
head(nhanes)
```

```
## # A tibble: 6 x 15
##      ID Gender   Age Race  Education MaritalStatus HHIncome Poverty Weight
##   <int> <fct>  <int> <chr> <ord>     <chr>         <chr>      <dbl>  <dbl>
## 1  3923 female    80 White High Sch~ Married       55000-6~    4.27   71.1
## 2  1548 male      42 Black 9 - 11th~ LivePartner   5000-99~    0.3   115.
## 3  1205 male       4 Hisp~ <NA>      <NA>          25000-3~    0.78   19.7
## 4  1519 male      12 Black <NA>      <NA>          75000-9~    2.96   63.7
## 5  4148 male       1 Black <NA>      <NA>          15000-1~    0.67   11.7
## 6  1681 female    14 White <NA>      <NA>          25000-3~    1.52   71.6
## # ... with 6 more variables: Length <dbl>, Height <dbl>, Diabetes <chr>,
## #   nPregnancies <int>, nBabies <int>, PregnantNow <chr>
```