Stat 140: R Commands and Concepts So Far

January 30, 2018

Task	R Command
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")`
Display first few lines of a data frame	`head(nhanes)`
Display more detail about a data frame	`str(nhanes)`
Number of rows and columns in a data frame	`dim(nhanes)`, `nrow(nhanes)`, `ncol(nhanes)`
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 observations in each level of a categorical variable	`tally( ~ Education, data = nhanes)`
Count number of observations in each combination of levels of two categorical variables	`tally(Education ~ Gender, data = nhanes)`

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 observational units, ascending order	`nhanes_sorted <- nhanes %>% arrange(Age)`
Sort observational units, 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 data
4. 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)
```

```
## Warning: package 'readr' was built under R version 3.4.2
```

```
nhanes <- read_csv("http://www.evanlray.com/stat140_s2018/lecture/20180125_intro_to_r/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
##   <int>  <chr> <int> <chr>        <chr>         <chr>       <chr>   <dbl>
## 1 62163   male    14 Asian         <NA>          <NA>  more 99999    4.07
## 2 62172 female    43 Black  High School  NeverMarried 20000-24999    2.02
## 3 62174   male    80 White College Grad       Married 65000-74999    4.30
## 4 62174   male    80 White College Grad       Married 65000-74999    4.30
## 5 62175   male     5 White         <NA>          <NA> 10000-14999    0.39
## 6 62176 female    34 White College Grad       Married  more 99999    5.00
## # ... with 7 more variables: Weight <dbl>, 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  62163 62172 62174 62174 62175 62176 62178 62180 62186 62190 ...
##  $ Gender       : chr  "male" "female" "male" "male" ...
##  $ Age          : int  14 43 80 80 5 34 80 35 17 15 ...
##  $ Race         : chr  "Asian" "Black" "White" "White" ...
##  $ Education    : chr  NA "High School" "College Grad" "College Grad" ...
##  $ MaritalStatus: chr  NA "NeverMarried" "Married" "Married" ...
##  $ HHIncome     : chr  "more 99999" "20000-24999" "65000-74999" "65000-74999" ...
##  $ Poverty      : num  4.07 2.02 4.3 4.3 0.39 5 0.05 0.87 0.53 0.54 ...
##  $ Weight       : num  49.4 98.6 95.8 95.8 23.9 68.7 85.9 89 63.5 38.5 ...
##  $ Length       : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ Height       : num  169 172 168 168 120 ...
##  $ Diabetes     : chr  "No" "No" "No" "No" ...
##  $ nPregnancies : int  NA 3 NA NA NA 5 NA NA NA NA ...
##  $ nBabies      : int  NA 2 NA NA NA 2 NA NA NA NA ...
##  $ PregnantNow  : chr  NA "No" 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.

```r
dim(nhanes)
```

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

```r
nrow(nhanes)
```

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

```r
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)
```

```
## Warning: package 'dplyr' was built under R version 3.4.2
```

```
##
## 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           <NA>
## 2    High School
## 3   College Grad
## 4   Some College
## 5      8th Grade
## 6 9 - 11th Grade
```

3. Summarizing Categorical Data

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 `tally` function in the `mosaic` package.

```
library(mosaic)
tally( ~ Education, data = nhanes)
```

```
## Education
##      8th Grade 9 - 11th Grade    High School   Some College   College Grad
##            212            405            679           1160           1128
##           <NA>
##           1416
```

```
tally(Education ~ Gender, data = nhanes)
```

```
##                   Gender
## Education          female male
##   8th Grade            91  121
##   9 - 11th Grade      174  231
##   High School         338  341
##   Some College        615  545
##   College Grad        584  544
##   <NA>                693  723
```

The first argument to the `tally` function is a formula. With one variable, the variable goes after the `~` (this character is a tilde; it's found in the top left corner of my keyboard). With two variables, one variable goes before the `~` and the second goes after it. The second argument is the data frame where these variables are found.

4. 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  62163 62172 62174 62174 62175 62176 62178 62180 62186 62190 ...
##  $ Gender       : Factor w/ 2 levels "female","male": 2 1 2 2 2 1 2 2 1 1 ...
##  $ Age          : int  14 43 80 80 5 34 80 35 17 15 ...
##  $ Race         : chr  "Asian" "Black" "White" "White" ...
##  $ Education    : Ord.factor w/ 5 levels "8th Grade"<"9 - 11th Grade"<..: NA 3 5 5 NA 5 3 5 NA NA ...
##  $ MaritalStatus: chr  NA "NeverMarried" "Married" "Married" ...
##  $ HHIncome     : chr  "more 99999" "20000-24999" "65000-74999" "65000-74999" ...
##  $ Poverty      : num  4.07 2.02 4.3 4.3 0.39 5 0.05 0.87 0.53 0.54 ...
##  $ Weight       : num  49.4 98.6 95.8 95.8 23.9 68.7 85.9 89 63.5 38.5 ...
##  $ Length       : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ Height       : num  169 172 168 168 120 ...
##  $ Diabetes     : chr  "No" "No" "No" "No" ...
##  $ nPregnancies : int  NA 3 NA NA NA 5 NA NA NA NA ...
##  $ nBabies      : int  NA 2 NA NA NA 2 NA NA NA NA ...
##  $ PregnantNow  : chr  NA "No" NA NA ...
##  $ Weight_pounds: num  108.9 217.4 211.2 211.2 52.7 ...
```

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
##   <int> <fctr> <int>  <chr>       <ord>         <chr>       <chr>   <dbl>
## 1 62172 female    43  Black High School  NeverMarried 20000-24999    2.02
## 2 62178   male    80  White High School       Widowed      0-4999    0.05
## 3 62223   male    54   Asian High School       Married        <NA>      NA
## 4 62231 female    48 Mexican High School       Married  more 99999    3.92
## 5 62308 female    78   White High School       Married 35000-44999    2.64
## 6 62308 female    78   White High School       Married 35000-44999    2.64
## # ... with 7 more variables: Weight <dbl>, 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
##   <int> <fctr> <int>    <chr>        <ord>         <chr>       <chr>
## 1 62172 female    43    Black  High School  NeverMarried 20000-24999
## 2 62178   male    80    White  High School       Widowed      0-4999
## 3 62206 female    35    White Some College       Married 75000-99999
## 4 62208   male    38 Hispanic Some College       Married 35000-44999
## 5 62223   male    54    Asian  High School       Married        <NA>
## 6 62231 female    48  Mexican  High School       Married  more 99999
## # ... with 8 more variables: Poverty <dbl>, Weight <dbl>, 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
##   <int> <fctr> <int> <chr>        <ord>         <chr>       <chr>   <dbl>
## 1 62172 female    43 Black  High School  NeverMarried 20000-24999    2.02
## 2 62174   male    80 White College Grad       Married 65000-74999    4.30
## 3 62174   male    80 White College Grad       Married 65000-74999    4.30
## 4 62176 female    34 White College Grad       Married  more 99999    5.00
## 5 62178   male    80 White  High School       Widowed      0-4999    0.05
## 6 62180   male    35 White College Grad       Married 20000-24999    0.87
## # ... with 7 more variables: Weight <dbl>, 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)
```

```r
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:

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

```r
head(nhanes_fewer_obs_units)
```

```
## # A tibble: 6 x 15
##      ID Gender   Age     Race    Education MaritalStatus     HHIncome
##   <int> <fctr> <int>    <chr>        <ord>         <chr>        <chr>
## 1 62172 female    43    Black  High School  NeverMarried  20000-24999
## 2 62178   male    80    White  High School       Widowed       0-4999
## 3 62206 female    35    White Some College       Married  75000-99999
## 4 62208   male    38 Hispanic Some College       Married  35000-44999
## 5 62223   male    54    Asian  High School       Married         <NA>
## 6 62231 female    48  Mexican  High School       Married   more 99999
## # ... with 8 more variables: Poverty <dbl>, Weight <dbl>, 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
##   <int> <fctr> <int>    <chr>     <ord>         <chr>       <chr>   <dbl>
## 1 62435 female     0    White      <NA>          <NA>        <NA>      NA
## 2 62451 female     0    White      <NA>          <NA>      0-4999    0.23
## 3 62468 female     0 Hispanic      <NA>          <NA> 20000-24999    0.89
## 4 62493 female     0    Other      <NA>          <NA> 35000-44999    2.65
## 5 62520   male     0    White      <NA>          <NA> 10000-14999    0.89
## 6 62656 female     0  Mexican      <NA>          <NA> 15000-19999    0.73
## # ... with 7 more variables: Weight <dbl>, 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
##   <int> <fctr> <int> <chr>        <ord>         <chr>       <chr>   <dbl>
## 1 62163   male    14 Asian         <NA>          <NA>  more 99999    4.07
## 2 62172 female    43 Black  High School  NeverMarried 20000-24999    2.02
## 3 62174   male    80 White College Grad       Married 65000-74999    4.30
## 4 62174   male    80 White College Grad       Married 65000-74999    4.30
## 5 62175   male     5 White         <NA>          <NA> 10000-14999    0.39
## 6 62176 female    34 White College Grad       Married  more 99999    5.00
## # ... with 7 more variables: Weight <dbl>, 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()`:

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

```
head(nhanes_sorted)
```

```
## # A tibble: 6 x 15
##      ID Gender   Age  Race      Education MaritalStatus    HHIncome
##   <int> <fctr> <int> <chr>          <ord>         <chr>       <chr>
## 1 62174   male    80 White   College Grad       Married 65000-74999
## 2 62174   male    80 White   College Grad       Married 65000-74999
## 3 62178   male    80 White    High School       Widowed      0-4999
## 4 62256   male    80 White      8th Grade       Married 20000-24999
## 5 62279   male    80 Black 9 - 11th Grade       Widowed        <NA>
## 6 62279   male    80 Black 9 - 11th Grade       Widowed        <NA>
## # ... with 8 more variables: Poverty <dbl>, Weight <dbl>, Length <dbl>,
## #   Height <dbl>, Diabetes <chr>, nPregnancies <int>, nBabies <int>,
## #   PregnantNow <chr>
```

```
head(nhanes)
```

```
## # A tibble: 6 x 15
##      ID Gender   Age  Race    Education MaritalStatus    HHIncome Poverty
##   <int> <fctr> <int> <chr>        <ord>         <chr>       <chr>   <dbl>
## 1 62163   male    14 Asian         <NA>          <NA>  more 99999    4.07
## 2 62172 female    43 Black  High School  NeverMarried 20000-24999    2.02
## 3 62174   male    80 White College Grad       Married 65000-74999    4.30
## 4 62174   male    80 White College Grad       Married 65000-74999    4.30
## 5 62175   male     5 White         <NA>          <NA> 10000-14999    0.39
## 6 62176 female    34 White College Grad       Married  more 99999    5.00
## # ... with 7 more variables: Weight <dbl>, Length <dbl>, Height <dbl>,
## #   Diabetes <chr>, nPregnancies <int>, nBabies <int>, PregnantNow <chr>
```