Lesson 04 - Functions and basic data management

Robin Donatello

Last Updated 01-06-2019

Introduction

In this lesson we will introduce how to use functions and their arguments, how to summarize data in a data frame, and to do basic data management tasks such as making new variables, recoding data and dealing with missing data.

Student Learning Objectives

After completing this lesson students will be able to

- Work with variables inside a data frame
- Identify when data values are missing
- Use function arguments to perform calculations in the presence of missing data.
- Make new variables inside a data frame.
- Selectively edit (and recode) data elements.

Preparation

Prior to this lesson students should

- Download the [04_fun_dm_notes.Rmd] R markdown file and save into your Math130 folder.
- Write the code to import the `NCbirths` data set into this notes file. Run it to make sure it works.
- Install the `dplyr` package.

```
library(dplyr)
NCbirths <- read.csv("../data/NCbirths.csv", header=TRUE)
```

Missing Data

Let's look at the `NCbirths` data set, specifically the first column containing data on the father's age (`fage`).

```
head(NCbirths)
```

```
##   fage mage      mature weeks    premie visits marital gained weight
## 1   NA   13 younger mom    39 full term     10 married     38   7.63
## 2   NA   14 younger mom    42 full term     15 married     20   7.88
## 3   19   15 younger mom    37 full term     11 married     38   6.63
## 4   21   15 younger mom    41 full term      6 married     34   8.00
## 5   NA   15 younger mom    39 full term      9 married     27   6.38
## 6   NA   15 younger mom    38 full term     19 married     22   5.38
##   lowbirthweight gender     habit  whitemom
## 1        not low   male nonsmoker not white
## 2        not low   male nonsmoker not white
```

```
## 3        not low female nonsmoker     white
## 4        not low   male nonsmoker     white
## 5        not low female nonsmoker not white
## 6            low   male nonsmoker not white
```

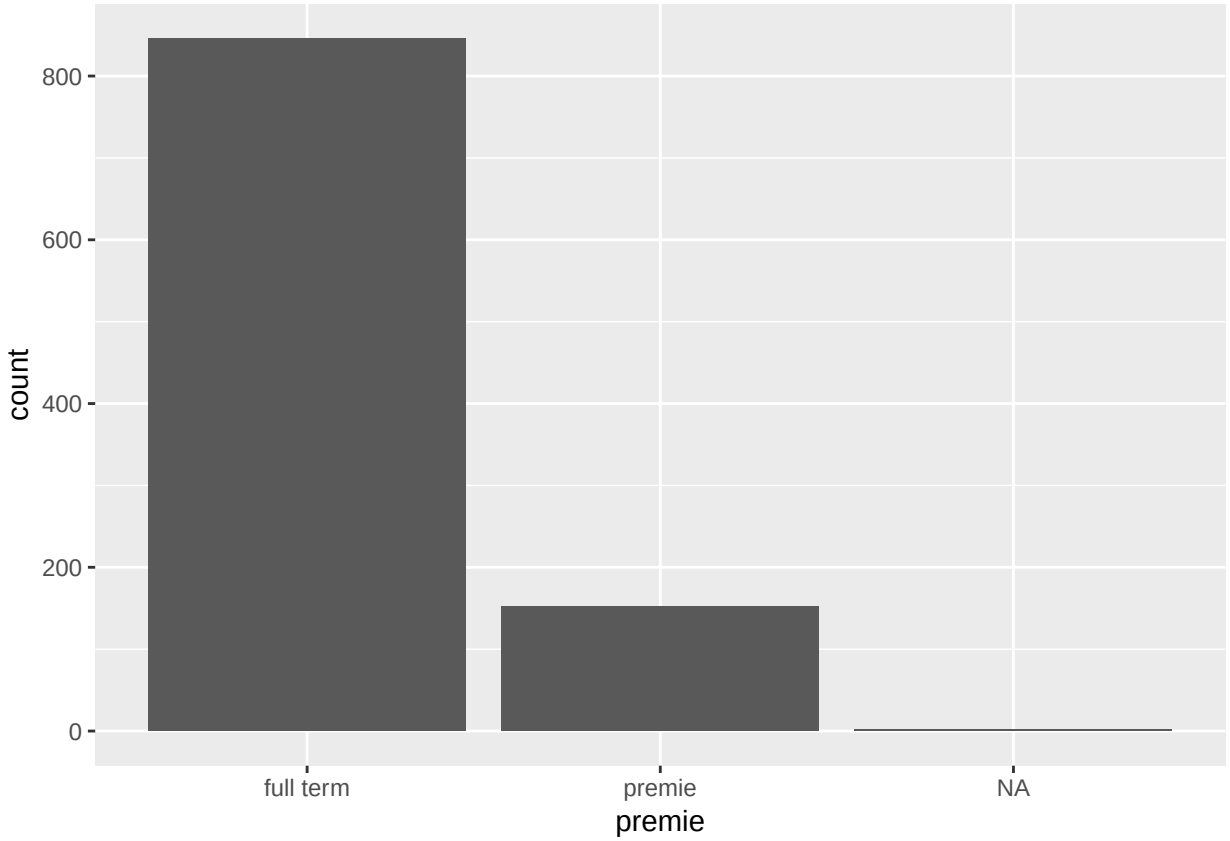
- R puts a `NA` as a placeholder when the value for that piece of data is missing.
- Missing data can be a result of many things: skip patterns in a survey (i.e. non-smokers don't get asked how many packs per week they smoke), errors in data reads from a machine, etc.

Problem 1 `R` can't do arithmetic on missing data. So `5 + NA = NA`, and if you were to try to calculate the `mean()` of a variable, you'd also get `NA`.

```
mean(NCbirths$fage)
```

```
## [1] NA
```

Problem 2 Some plots will show `NA` as it's own category, which is undesirable.



Identifying missing values

1. Look at the raw data using `head()` or `str()`
2. Look at data summaries using `table()` for categorical data and `summary()` for numerical data.

```
table(NCbirths$habit, useNA="always")
```

```
##
## nonsmoker    smoker      <NA>
##       873       126         1
```

```
summary(NCbirths$fage)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##   14.00   25.00   30.00   30.26   35.00   55.00     171
```

Later lessons will discuss how to work around these missing values.

Is it NA?

The `is.na()` function returns TRUE or FALSE for each element in the provided vector for whether or not that element is missing.

```
is.na(c("green", NA, 3))
```

```
## [1] FALSE  TRUE FALSE
```

This can be used to count the number of missing values in a variable

```
sum(is.na(NCbirths$fage))
```

```
## [1] 171
```

There are 171 records in this data set where the age for the father is not present.

Functions

Functions take inputs, called **arguments** and provide outputs, or results. A few functions you have already used are `head()`, `table()` and `subset()`. Let's look at the `mean` function again by typing ?mean.

The **Usage** section of the documentation includes two versions of the mean() function; What's the difference? The first function

```
mean(x,...)
```

is the most general definition of the mean function. This section also shows you what the default values for each argument are. This is a very important piece to pay attention. Sometimes the default behaviors are not what you want to happen.

```
mean(x, trim = 0, na.rm = FALSE, ...)
```

In the **Arguments** section the help file defines what each argument does.

- `x` is the object that you want to take the mean of
- `trim` is a number from 0 to 0.5 that defines the fraction of observations to be excluded from each side before the mean is calculated.
- `na.rm` is a logical value (`TRUE`/`FALSE`) that tells `R` whether NA values should be stripped before the computation proceeds.
- `...` is called the ellipsis, and it is a way for R to pass arguments to or from other methods without the function having to name them explicitly.

Argument ordering

A function's arguments can be named, or can be referred to by position. As an example, let's look at the variable on the `NCbirths` data set that contains data on the fathers age: `NCbirths$fage`.

If we were to calculate the mean, we'd get NA.

```
mean(NCbirths$fage)
```

```
## [1] NA
```

We need to tell R to *remove* the `NA`'s from the data prior to calculating the mean.

```
mean(NCbirths$fage, na.rm=TRUE)
```

```
## [1] 30.25573
```

It worked fine because I named the argument to remove missing values. What if I didn't state what that argument was?

```
mean(NCbirths$fage, TRUE)
```

```
## Error in mean.default(NCbirths$fage, TRUE): 'trim' must be numeric of length one
```

R is expecting a value for trim as the second argument and doesn't know what to do with the value `TRUE`. If you name the arguments, then the order is irrelevant.

```
mean(na.rm=TRUE, x=NCbirths$fage, trim=.1)
```

```
## [1] 30.01053
```

But let's not get that crazy.

R, at its heart, is a functional programming language (cite: Adv. R by Wickham). We interact with the program, and data, using functions.

Summarizing data

There are two main methods to summarize data. Both were introduced in the last lesson: `table()` for categorical factor variables, and `summary()` for numeric variables.

Frequency Tables for categorical data

Let's look at the variable for whether or not the baby was born underweight. R acknowledges it is a factor variable already.

```
class(NCbirths$lowbirthweight)
```

```
## [1] "factor"
```

You can create a frequency table by using the `table()` function. The `useNA="always"` argument tells `R` to always include an entry for missing values `<NA>`, even if there are none.

```
table(NCbirths$lowbirthweight, useNA="always")
```

```
##
##     low not low    <NA>
##     111     889       0
```

Summary statistics for numerical data

Numerical variables can be summarized using statistics such as the min, max, mean and median. The function `summary()` prints out the five number summary, and includes the mean.

```
summary(NCbirths$visits)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##     0.0    10.0    12.0    12.1    15.0    30.0       9
```

Mothers included in this data set visited the doctor on average 12.1 times during their pregnancy.

Recoding data

Sometimes we have a need to edit, or change data. We can change values of existing data by using logical statements.

Recall the Data Camp lesson on Matricies. To choose all observations (rows) of a `data` set, where a `variable` is equal to some `value`, you would type

```
data[data$variable==value]
```

We can use this method of selecting rows, to change data in those specific rows.

Example 1: Too low birthweight

Let's look at the numerical distribution of birthweight of the baby.

```
summary(NCbirths$weight)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   6.380   7.310   7.101   8.060  11.750
```

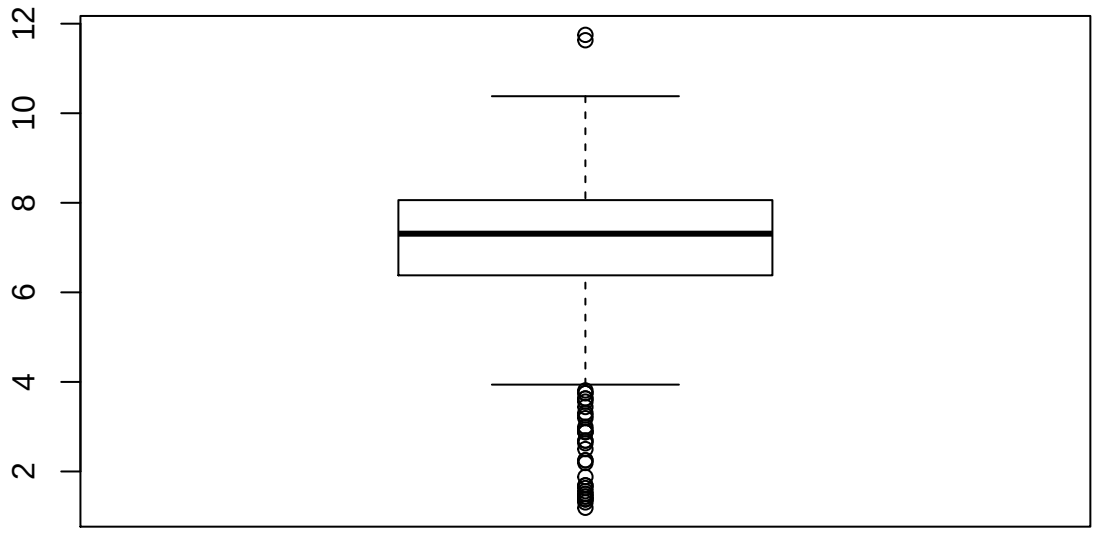
The value of 1 lb seems very low. The researchers you are working with decide that is a mistake and should be excluded from the data. We would then set all records where `weight=1` to missing.

```
NCbirths$weight[NCbirths$weight==1] <- NA
```

The specific variable `NCbirths$weight` is on the far left, outside the `[]`. So just the variable `weight` is being changed.

But what about other weights that aren't quite as low as 1, but still unusually low? The boxplot below shows outlying values as dots on the low end of birthweight.

```
boxplot(NCbirths$weight)
```

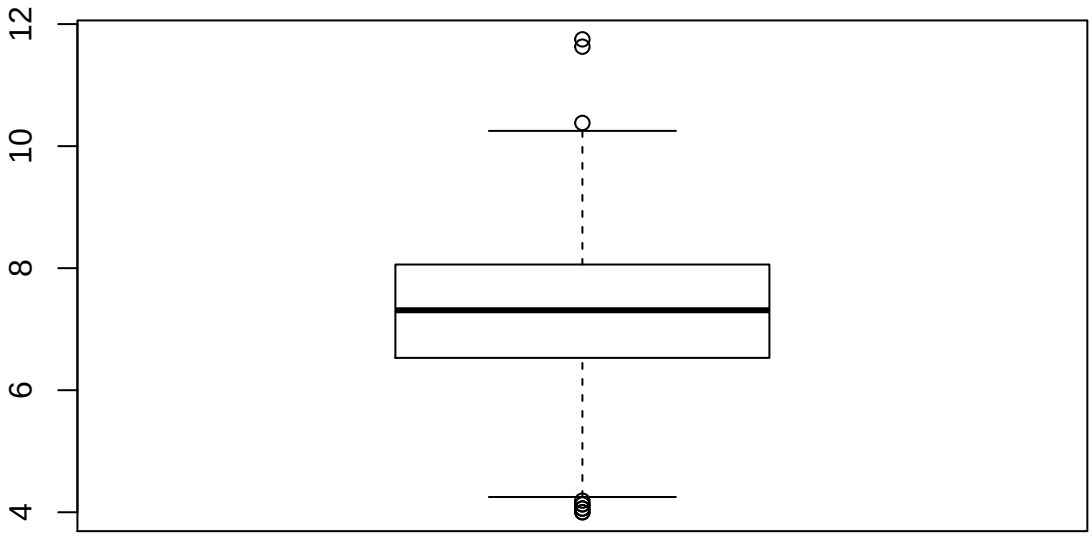


Let's set all birth weights less than 4 lbs to missing.

```
NCbirths$weight[NCbirths$weight < 4] <- NA
```

It is important to plot the data again, to make sure that there are no values below 4 now.

```
boxplot(NCbirths$weight)
```



Creating new variables

New variables should be added to the data frame. This can be done using `$` sign notation.

The new variable you want to create goes on the left side of the assignment operator `<-`, and how you want to create that new variable goes on the right side.

```
data$new_variable <- creation statement
```

Example: basic arithmetic on existing variables

As a pregnancy progresses, both the mother and the baby gain weight. The variable `gained` is the total amount of weight the mother gained in her pregnancy. The variable `weight` is how much the baby weighed at birth.

The following code creates a new variable `wtgain_mom` the weight gained by the mother, that is not due to the baby by subtracting `weight` from `gained`.

```
NCbirths$wtgain_mom <- NCbirths$gained - NCbirths$weight
```

To confirm this variable was created correctly, we look at the data contained in three variables in question.

```
head(NCbirths[,c('gained', 'weight', 'wtgain_mom')])
```

```
##   gained weight wtgain_mom
```

```
## 1     38   7.63      30.37
## 2     20   7.88      12.12
## 3     38   6.63      31.37
## 4     34   8.00      26.00
## 5     27   6.38      20.62
## 6     22   5.38      16.62
```

Dichtomizing data

The `ifelse()` is hands down the easiest way to create a binary variable (dichotomizing, only 2 levels)

Let's add a variable to identify if a mother in the North Carolina births data set was underage at the time of birth.

- We will define this as if the mothers age `mage` is under 18.
- We will create a new factor variable with two values: `underage` and `adult`.
- We will use the `ifelse()` function.
  - It's set of arguments are: `ifelse(logical, value if TRUE, value if FALSE)`
  - The `logical` argument is a statement that resolves as a `boolean` variable, as either TRUE or FALSE.
  - The second argument is what you want the resulting variable to contain if the logical argument is `TRUE`
  - The last argument is what you want the resulting variable to contain if the logical argument is `FALSE`

Make a new variable `underage` on the `NCbirths` data set. If `mage` is under 18, then the value of this new variable is `underage`, else it is labeled as `adult`.

```
NCbirths$underage <- ifelse(NCbirths$mage <= 18, "underage", "adult")
```

You should always make sure your code works as intended.

- First let's look at the frequency table of `underage` and see if records exist with the new categories, and if there are any missing values.

```
table(NCbirths$underage, useNA="always")
```

```
##
##    adult underage     <NA>
##      925       75        0
```

- Next let's check it against the value of `mage` itself. Let's look at all rows where mothers age is either 18 or 19 `mage %in% c(18,19)`, and only the columns of interest.

```
NCbirths[NCbirths$mage %in% c(18,19),c('mage', 'underage')]
```

```
##     mage underage
## 38    18 underage
## 39    18 underage
## 40    18 underage
## 41    18 underage
## 42    18 underage
## 43    18 underage
## 44    18 underage
## 45    18 underage
## 46    18 underage
## 47    18 underage
## 48    18 underage
```

```
## 49    18 underage
## 50    18 underage
## 51    18 underage
## 52    18 underage
## 53    18 underage
## 54    18 underage
## 55    18 underage
## 56    18 underage
## 57    18 underage
## 58    18 underage
## 59    18 underage
## 60    18 underage
## 61    18 underage
## 62    18 underage
## 63    18 underage
## 64    18 underage
## 65    18 underage
## 66    18 underage
## 67    18 underage
## 68    18 underage
## 69    18 underage
## 70    18 underage
## 71    18 underage
## 72    18 underage
## 73    18 underage
## 74    18 underage
## 75    18 underage
## 76    19    adult
## 77    19    adult
## 78    19    adult
## 79    19    adult
## 80    19    adult
## 81    19    adult
## 82    19    adult
## 83    19    adult
## 84    19    adult
## 85    19    adult
## 86    19    adult
## 87    19    adult
## 88    19    adult
## 89    19    adult
## 90    19    adult
## 91    19    adult
## 92    19    adult
## 93    19    adult
## 94    19    adult
## 95    19    adult
## 96    19    adult
## 97    19    adult
## 98    19    adult
## 99    19    adult
## 100   19    adult
## 101   19    adult
## 102   19    adult
```

Figure 1:

```
## 103   19    adult
## 104   19    adult
## 105   19    adult
## 106   19    adult
## 107   19    adult
## 108   19    adult
## 109   19    adult
## 110   19    adult
```

Notice I snuck a new operator in on you - `%in%`. This is a way you can provide a list of values (a.k.a a vector) and say "if the value of the variable I want is %in% any of these options in this vector..." do the thing.

Chaining using the pipe %>%

The pipe is technically part of the `magrittr` package, but quite often we will use it in conjunction with functions from the `dplyr` package introduced in lesson 07. In fact it's so commonly used with `dplyr` functions that it the `magrittr` package is loaded when you load `dplyr` (which is what we did here). However, the pipe operator is so useful in many other places that it deserves it's own introduction here.

> "and then..."

This is what I read to myself when using the pipe (`%>%`). The pipe lets you chain functions together, as long as the first argument of the function is a `data.frame` or a variable in a `data.frame`.

Example: Frequency tables & summary statistics

Earlier we saw that to create a frequency table for categorical variable we can type:

```
table(NCbirths$mature)
```

```
##
```

```
##  mature mom younger mom
##         133         867
```

This is also accomplished by first stating the variable, then piping in the summary function.

```
NCbirths$mature %>% table()
```

```
## .
##  mature mom younger mom
##         133         867
```

```
NCbirths$mage %>% mean()
```

```
## [1] 27
```

So in my head i'm reading "Take the `mage` variable on the `NCbirths` data set, *and then* calculate the mean."

Seems kinda trivial here, the base function without the pipe is pretty easy. But I promise the usefulness will be apparent before the class is out.

Additional References

This lesson was an introduction to typical basic data management tasks. In a later lesson you will learn how to use functions in the `dplyr` package to perform the same, and more different data management tasks, in a more streamline manner. Both levels of techniques are equally useful depending on the task.

There are always many ways to approach a problem in R. Our goal in this class is to provide you with a few ways to think of a problem. You will find what fits best for your coding style as you progress.

There are several other Data Camp courses that all have the first chapter free that you can use to get a start on learning more about how to import and clean up data. Here are a few.

- Cleaning Data in R
- Tidy Data - The Journal, and the tutorial

Resources for handling missing data

- http://www.statmethods.net/input/missingdata.html
- https://stats.idre.ucla.edu/r/faq/how-does-r-handle-missing-values/
- http://faculty.nps.edu/sebuttre/home/R/missings.html