Data Wrangling

Data Wrangling

In the real world, data science projects rarely involve data that can be easily imported ready for analysis. According to Wikipedia:

> Data munging or data wrangling is loosely the process of manually converting or mapping data from one "raw" form into another format that allows for more convenient consumption of the data with the help of semi-automated tools.

Our example dataset provides an example:

```r
url <- "https://raw.githubusercontent.com/datasciencelabs/data/master/bio260-heights.csv"
dat <- read.csv(url)
```

First note how we make assignments in R: we use `<-`. We can also use the equal sign `=` although here we try to stick to `<-` to make it very clear it is an assignment and not logical statement.

We also note that we have put the content of what comes out of `read.csv` into an *object*. We picked the object name `dat`.

So what is `dat` exactly? We can get a quick summary of what an object is with the function `str` (stands for structure)

```r
str(dat)
```

```
## 'data.frame':    148 obs. of  3 variables:
##  $ Timestamp                        : Factor w/ 143 levels "1/25/2016 14:39:56",..: 9 10 11 12 13 13
##  $ What.is.your.gender.             : Factor w/ 3 levels "Female","I prefer not to disclose",..: 1 1
##  $ What.is.your.height..in.inches..: Factor w/ 67 levels "167","168","172",..: 38 35 53 50 58 64 52
```

Here we see that this object is a `data.frame`. These are one of the most widely used data types in R. They are particularly useful for storing tables.

To see more of this object we can type it

Now we want to describe the heights. We could simply report the list of numbers. But there is a problem. Take a look at the entries:

```r
View(dat)
```

Notice these not all entries are numbers. Furthermore, they are not all in inches. So what to do? We need to wrangle

Extracting columns To extract columns from the data.frame we use the `$` character like this:

```r
dat$Timestamp
```

This now gives us a vector. We can access elements of the vector using the `[` symbol:

```
dat$Timestamp[2]
```

```
## [1] 1/25/2016 8:15:21
## 143 Levels: 1/25/2016 14:39:56 1/25/2016 16:32:52 ... 1/27/2016 9:34:14
```

Quick Review of Vectors Vector are a sequence of data elements of the same type. Many of the operations used to analyze data are applied to vectors. In R vectors can be numeric, characters or logical.

The most basic way to creat a vector is with the function `c`

```
x <- c(1,2,3,4,5)
```

Two very common ways of generating vectors are using `:` or the `seq` function:

```
x <- 1:5
x <- seq(1,5)
```

Vecotrs can have names

```
names(x) <- letters[1:5]
x
```

```
## a b c d e
## 1 2 3 4 5
```

Coercion Vectors need to be homogenous. But when R is instructed to create a vector of different types, it does not give an error. Instead it tries to *coerce* values to be the same. Here is an example:

```
height <- c(60, 59, 55, "5'5", 70)
height
```

```
## [1] "60"  "59"  "55"  "5'5" "70"
```

Note that no warning or error was given. It simply changed everything to a character. This is important to know because sometimes we make a mistake in entering data and receive no error message.

Data Manipulation wiht `dplyr`

R provides incredibly powerful and flexible language for data manipulation. However, the syntax is somewhat hard to get used to. We will therefore introducing a package that makes the syntax much more like the English language. This package is `dplyr` which you should install if you have not done so already.

```
library(dplyr)
```

When using `dplyr` we recommend reading in data with the functions in the `readr` package:

```
library(readr)
dat <- read_csv("https://raw.githubusercontent.com/datasciencelabs/data/master/bio260-heights.csv")
```

This object is now a special type of `data.frame` called `tbl_df` that has a nicer printing method. We can now simply evaluate an expression with just the object and see a meaningful summary instead of everything.

```
dat
```

```
## Source: local data frame [148 x 3]
##
##            Timestamp What is your gender? What is your height (in inches)?
##                (chr)                (chr)                            (chr)
## 1  1/25/2016 8:15:15               Female                               63
## 2  1/25/2016 8:15:21               Female                               62
## 3  1/25/2016 8:15:25                 Male                               69
## 4  1/25/2016 8:15:29               Female                               68
## 5  1/25/2016 8:15:37                 Male                            71.65
## 6  1/25/2016 8:15:37                 Male                               75
## 7  1/25/2016 8:15:39                 Male                          68.8976
## 8  1/25/2016 8:15:40                 Male                               74
## 9  1/25/2016 8:15:41               Female                               65
## 10 1/25/2016 8:15:44               Female                              5'4
## ..               ...                  ...                              ...
```

Selecting columns Right, we are interested in looking at heights. We can select just that column using:

```
select(dat, contains("height"))
```

```
## Source: local data frame [148 x 1]
##
##    What is your height (in inches)?
##                               (chr)
## 1                                63
## 2                                62
## 3                                69
## 4                                68
## 5                             71.65
## 6                                75
## 7                           68.8976
## 8                                74
## 9                                65
## 10                              5'4
## ..                              ...
```

We have a problem: this is a `character`. We want numbers.

Renaming columns

Before we continue it will be convenient to change the names of our columns to something more convenient.

```
names(dat) <- c("time","gender","height")
```

Vectorization

```
height <- c(60, 59, 55, "5'5", 70)
height[3]
```

```
## [1] "55"
```

```
as.numeric(height[3])
```

```
## [1] 55
```

One powerful feature of R is that we can *vectorize* most operation

```
 as.numeric(height)
```

```
## Warning: NAs introduced by coercion
```

```
## [1] 60 59 55 NA 70
```

Note now we do receive an warning. This is because R has no idea how to convert “5’5” to a number.

Missing values

Note in the the `NA` value in the object above.

These are missing values. We can find out which values are missing using the function

```
?is.na
```

Adding columns

```
dat <- mutate(dat, numeric_height=as.numeric(height),
              original=height)
```

```
## Warning in eval(substitute(expr), envir, enclos): NAs introduced by
## coercion
```

Subsetting Observations

To see all the row in which we have problems:

```
filter(dat, is.na(numeric_height))
```

```
## Source: local data frame [21 x 5]
##
##                  time                 gender height numeric_height
##                 (chr)                  (chr)  (chr)          (dbl)
## 1   1/25/2016 8:15:44                 Female    5'4             NA
## 2   1/25/2016 8:15:45                 Female   5'8"             NA
## 3   1/25/2016 8:15:45                 Female    5'5             NA
## 4   1/25/2016 8:29:00                   Male    5'7             NA
## 5  1/25/2016 14:39:56                 Female    5'6             NA
## 6  1/25/2016 22:02:03                   Male  5'11"             NA
## 7   1/26/2016 8:36:33 I prefer not to disclose  5'10"             NA
## 8   1/26/2016 9:49:15                   Male   5'7"             NA
## 9   1/26/2016 9:49:19                   Male    5'7             NA
## 10  1/26/2016 9:51:19                 Female    5'8             NA
## ..                ...                    ...    ...            ...
## Variables not shown: original (chr)
```

The Pipe

```
filter(dat, is.na(numeric_height)) %>% select(height)
```

```
## Source: local data frame [21 x 1]
##
##    height
##     (chr)
## 1     5'4
## 2    5'8"
## 3     5'5
## 4     5'7
## 5     5'6
## 6   5'11"
## 7   5'10"
## 8    5'7"
## 9     5'7
## 10    5'8
## ..    ...
```

Let's see more

```
filter(dat, is.na(numeric_height)) %>% select(height) %>% print(n=21)
```

```
## Source: local data frame [21 x 1]
##
##           height
##            (chr)
## 1            5'4
## 2           5'8"
## 3            5'5
## 4            5'7
## 5            5'6
## 6          5'11"
## 7          5'10"
## 8           5'7"
## 9            5'7
## 10           5'8
## 11        5' 11"
## 12          6'1"
## 13           69"
## 14         5' 7"
## 15        5'10''
## 16          5'10
## 17          5'10
## 18  5ft 9 inches
## 19 5 ft 9 inches
## 20           5'2
## 21          5'11
```

gsub One of the most useful functions for data wranglin is `gsub`. It let's us searches for characters and substitutes it for others. More general it searches for regular expression. We will learn about those later.

Here is an example:

```
x <- dat$height[109:116]
x
```

```
## [1] "5'10"          "70"            "67.7"          "62"
## [5] "5ft 9 inches"  "5 ft 9 inches" "5'2"           "74"
```

Note that we are using both `'` and `ft` as the same thing. To simplify the problem we want to substitute one for the other. `gsub` does the trick:

```
x <- gsub("ft", "'", x)
x
```

```
## [1] "5'10"         "70"           "67.7"         "62"
## [5] "5' 9 inches"  "5 ' 9 inches" "5'2"          "74"
```

The word inches is not doing anything here so we might as well remove it.

```
x <- gsub("inches","",x)
x
```

```
## [1] "5'10"   "70"     "67.7"   "62"     "5' 9 "  "5 ' 9 " "5'2"    "74"
```

We are now ready to start fixing the height data:

```
dat <- mutate(dat, height= gsub("ft","'",height) ) %>%
  mutate(height= gsub("\"|inches|\ |''","",height) )
```

Functions

Up to now we have used prebuilt functions. However, many times we have to construct our own. We can do this in R using the `function`:

```
avg <- function(x){
  return( sum(x) / length(x) )
}
avg( 1:5 )
```

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

Assessment: Construct a function that computes the variance defined as follows for a vector $x_1, \dots, x_n$:

$$\frac{1}{n}\sum_{i=1}^{n}(x_i-\mu)^2 \text{ with } \mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$

What is the variance of `1:5` ?

Assessment: Write a function `convert` that takes two character arguments, feet and inches as characters, and returns inches

Here we construct a more complicated function that changes 5'4 to `5*12+4`

```
fixheight <- function(x){
  y <- strsplit(x, "'")
  ret <- sapply(y, function(z){
    ifelse( length(z)>1, as.numeric(z[1])*12 + as.numeric(z[2]) ,
            as.numeric(z[1]))
  })
  return(ret)
}
```

We can now test the function

```
fixheight( "70")
```

```
## [1] 70
```

```
fixheight( "5'10")
```

```
## [1] 70
```

```
fixheight( c("5'9","70","5'11"))
```

```
## [1] 69 70 71
```

Finally we can mutate our data:

```r
dat <- mutate(dat, height=fixheight(height)) %>% select(-numeric_height)
```

The last call to select removes the now unecessary column `numeric_height`. Let's see the result:

```r
filter(dat, is.na(height)) %>% select(height)
```

```
## Source: local data frame [0 x 1]
##
## Variables not shown: height (dbl)
```

We have removed all the NAs