Automating Data Exploration with R

Reading Data

readLines

A first investigative exploration of a data set is the readLines function (https://stat.ethz.ch/R-manual/R-devel/library/base/html/readLines.html). It allows you to cull a small amount of lines from the top of a file, no matter how big the file is. This is a silly example but imagine if your file is over 10 gigabytes in size, why waste time and memory when you aren’t sure what it contains?

Let’s use the `readLines` function to open part of a text file off **CRAN**:

```
readLines('https://cran.r-project.org/src/base/README', n=20)
```

```
##  [1] ""
##  [2] "\t\t\tTHE BASIC R README"
##  [3] ""
##  [4] ""
##  [5] "\t   (See \"doc/FAQ\" and \"doc/RESOURCES\" for more detailed informatio
n"
##  [6] "\t\t\t\t      - these files are only in the tarballs)"
##  [7] "\t   (See \"INSTALL\"             for help on installation)"
##  [8] ""
##  [9] "1. INTRODUCTION"
## [10] ""
## [11] "This directory contains the source code tree for R, which is a"
## [12] "language which is not entirely unlike (versions 3 and 4 of) the S"
## [13] "language developed at AT&T Bell Laboratories by Rick Becker, John"
## [14] "Chambers and Allan Wilks."
## [15] ""
## [16] "R is free software distributed under a GNU-style copyleft."
## [17] ""
## [18] "The core of R is an interpreted computer language with a syntax"
## [19] "superficially similar to C, but which is actually a \"functional"
## [20] "programming language\" with capabilities similar to Scheme.  The"
```

This is a great way of discovering what data types are contained in a very large data set. You will see in the next few functions we use, knowing the data type in advance can speed things up tremendously.

read.table, read.csv, read.csv2

read.csv is the most common reader in R, followed closely by read.table . In fact, read.csv and read.csv2 are both wrappers over read.table (https://stat.ethz.ch/R-manual/R-devel/library/utils/html/read.table.html).

```
read.table(file, header = FALSE, sep = "", quote = "\"'",
           dec = ".", numerals = c("allow.loss", "warn.loss", "no.loss"),
           row.names, col.names, as.is = !stringsAsFactors,
           na.strings = "NA", colClasses = NA, nrows = -1,
           skip = 0, check.names = TRUE, fill = !blank.lines.skip,
           strip.white = FALSE, blank.lines.skip = TRUE,
           comment.char = "#",
           allowEscapes = FALSE, flush = FALSE,
           stringsAsFactors = default.stringsAsFactors(),
           fileEncoding = "", encoding = "unknown", text, skipNul = FALSE)
```

Let's use read.table . Here we'll read the Titanic dataset from the University of Colorado Denver. Eventhough it off the Internet, the same use applies to your hard-drive. We set the separator parameter of \t for tabular and request to consider the first row as headers:

```
Titanic_dataset <- read.table('http://math.ucdenver.edu/RTutorial/titanic.txt', se
p='\t', header=TRUE)
head(Titanic_dataset)
```

```
##                                             Name PClass   Age    Sex
## 1                   Allen, Miss Elisabeth Walton    1st 29.00 female
## 2                    Allison, Miss Helen Loraine    1st  2.00 female
## 3            Allison, Mr Hudson Joshua Creighton    1st 30.00   male
## 4 Allison, Mrs Hudson JC (Bessie Waldo Daniels)    1st 25.00 female
## 5                  Allison, Master Hudson Trevor    1st  0.92   male
## 6                             Anderson, Mr Harry    1st 47.00   male
##   Survived
## 1        1
## 2        0
## 3        0
## 4        0
## 5        1
## 6        1
```

Let's see if the function figured out the data types contained in the Titanic dataset:

```
str(Titanic_dataset)
```

```
## 'data.frame':    1313 obs. of  5 variables:
##  $ Name    : Factor w/ 1310 levels "Abbing, Mr Anthony",..: 22 25 26 27 24 31 4
5 46 50 54 ...
##  $ PClass  : Factor w/ 3 levels "1st","2nd","3rd": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Age     : num  29 2 30 25 0.92 47 63 39 58 71 ...
##  $ Sex     : Factor w/ 2 levels "female","male": 1 1 2 1 2 2 1 2 1 2 ...
##  $ Survived: int  1 0 0 0 1 1 1 0 1 0 ...
```

Not bad, but feature `Names` shouldn't be a factor but a character. In this case, we could change it after the case or force the reader to make all text a `character` instead of `factor` by setting the `stringsAsFactors` to FALSE:

```
Titanic_dataset <- read.table('http://math.ucdenver.edu/RTutorial/titanic.txt', se
p='\t', header=TRUE, stringsAsFactors=FALSE)
str(Titanic_dataset)
```

```
## 'data.frame':    1313 obs. of  5 variables:
##  $ Name    : chr  "Allen, Miss Elisabeth Walton" "Allison, Miss Helen Loraine"
"Allison, Mr Hudson Joshua Creighton" "Allison, Mrs Hudson JC (Bessie Waldo Daniel
s)" ...
##  $ PClass  : chr  "1st" "1st" "1st" "1st" ...
##  $ Age     : num  29 2 30 25 0.92 47 63 39 58 71 ...
##  $ Sex     : chr  "female" "female" "male" "female" ...
##  $ Survived: int  1 0 0 0 1 1 1 0 1 0 ...
```

You can also pass custom column names directly to the `read.table` function:

```
actg320_colnames <- c('id','time','censor','time_d','censor_d','treatment','treatm
ent_group','strat2','sex','raceth','ivdrug','hemophil','karnof','cd4','priorzd
v','age')
actg320 <- read.table('https://www.umass.edu/statdata/statdata/data/actg320.dat',
col.names = actg320_colnames)
head(actg320)
```

```
##   id time censor time_d censor_d treatment treatment_group strat2 sex
## 1  1  189      0    189        0         0               1      1   1
## 2  2  287      0    287        0         0               1      1   2
## 3  3  242      0    242        0         1               2      0   1
## 4  4  199      0    199        0         0               1      1   1
## 5  5  286      0    286        0         1               2      0   1
## 6  6  285      0    285        0         1               2      0   1
##   raceth ivdrug hemophil karnof   cd4 priorzdv age
## 1      1      1        0    100 169.0       39  34
## 2      2      1        0     90 149.5       15  34
## 3      1      1        1    100  23.5        9  20
## 4      1      1        0     90  46.0       53  48
## 5      1      3        0     90  10.0       12  46
## 6      1      1        0     70   0.0       24  51
```

```
dim(actg320)
```

```
## [1] 1151   16
```

Here's a quick look at `read.csv` . For more differences between these readers see: read.table (http://www.inside-r.org/r-doc/utils/read.table)

```
Iris_dataset <- read.csv('http://archive.ics.uci.edu/ml/machine-learning-database
s/iris/bezdekIris.data', header=FALSE)
head(Iris_dataset)
```

```
##    V1  V2  V3  V4          V5
## 1 5.1 3.5 1.4 0.2 Iris-setosa
## 2 4.9 3.0 1.4 0.2 Iris-setosa
## 3 4.7 3.2 1.3 0.2 Iris-setosa
## 4 4.6 3.1 1.5 0.2 Iris-setosa
## 5 5.0 3.6 1.4 0.2 Iris-setosa
## 6 5.4 3.9 1.7 0.4 Iris-setosa
```

Before we move to more sophisticated readers, let's build a simple data frame to work with. We'll create a very small data set made of the following data types: integers, factors, doubles, and dates.

```
mix_dataset <- data.frame(
                id=c(10,20,30,40,50),
                gender=c('male','female','female','male','female'),
                some_date=c('2012-01-12','2012-01-12','2012-12-01','2012-05-30','2
013-12-12'),
                value=c(12.34, 32.2, 24.3, 83.1, 8.32),
                outcome=c(1,1,0,0,0))

write.csv(mix_dataset, 'mix_dataset.csv', row.names = FALSE)
```

We'll save it to your current working directory and read it back again. Let's look at it using `read.csv` :

```
mix_dataset <- read.csv('mix_dataset.csv', stringsAsFactors = FALSE)
str(mix_dataset)
```

```
## 'data.frame':    5 obs. of  5 variables:
##  $ id       : int  10 20 30 40 50
##  $ gender   : chr  "male" "female" "female" "male" ...
##  $ some_date: chr  "2012-01-12" "2012-01-12" "2012-12-01" "2012-05-30" ...
##  $ value    : num  12.34 32.2 24.3 83.1 8.32
##  $ outcome  : int  1 1 0 0 0
```

The point to note here is the our date field `some_date` was read as a character string using read.csv.

Heavy-duty Readers

readr

Let's look at some readers that aren't part of the base package. readr {readr} (https://cran.r-project.org/web/packages/readr/README.html) is a relatively new package maintained by Hadley Wickham. It does a great job at inferring data types and is fast!

As you can see, `some_date` is correctly cast as Date, and it does pick up two integer fields:

```
# install.packages('readr')
library(readr)

mix_dataset <- read_csv('mix_dataset.csv')
str(mix_dataset)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    5 obs. of  5 variables:
##  $ id       : int  10 20 30 40 50
##  $ gender   : chr  "male" "female" "female" "male" ...
##  $ some_date: Date, format: "2012-01-12" "2012-01-12" ...
##  $ value    : num  12.34 32.2 24.3 83.1 8.32
##  $ outcome  : int  1 1 0 0 0
```

If you know in advance the data types in a data set, you can pass it along to the function to save it time and processing. In the first read, we force `some_date` to character, in the read we force the `id` field to be a numeric instead of an integer.

```
mix_dataset <- read_csv('mix_dataset.csv', col_types='nccnn')
str(mix_dataset)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    5 obs. of  5 variables:
##  $ id       : num  10 20 30 40 50
##  $ gender   : chr  "male" "female" "female" "male" ...
##  $ some_date: chr  "2012-01-12" "2012-01-12" "2012-12-01" "2012-05-30" ...
##  $ value    : num  12.34 32.2 24.3 83.1 8.32
##  $ outcome  : num  1 1 0 0 0
```

```
mix_dataset <- read_csv('mix_dataset.csv', col_types='ncDni')
str(mix_dataset)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    5 obs. of  5 variables:
##  $ id       : num  10 20 30 40 50
##  $ gender   : chr  "male" "female" "female" "male" ...
##  $ some_date: Date, format: "2012-01-12" "2012-01-12" ...
##  $ value    : num  12.34 32.2 24.3 83.1 8.32
##  $ outcome  : int  1 1 0 0 0
```

fread

fread {data.table} (http://www.inside-r.org/packages/cran/data.table/docs/fread) is the fastest of the bunch presented so far. Set parameter `data.table` =FALSE to return a data frame:

```
# install.packages('data.table')
library(data.table)
mix_dataset <- fread('mix_dataset.csv', showProgress=TRUE, data.table=FALSE)
str(mix_dataset)
```

```
## 'data.frame':    5 obs. of  5 variables:
##  $ id       : int  10 20 30 40 50
##  $ gender   : chr  "male" "female" "female" "male" ...
##  $ some_date: chr  "2012-01-12" "2012-01-12" "2012-12-01" "2012-05-30" ...
##  $ value    : num  12.34 32.2 24.3 83.1 8.32
##  $ outcome  : int  1 1 0 0 0
```

If you only want a subset of columns, you can choose them directly in the `fread` command using the `select` parameter (similarly you can use the `drop` parameter to remove features):

```
mix_dataset <- fread('mix_dataset.csv',  data.table=FALSE, select = c('value', 'ou
tcome'))
head(mix_dataset)
```

```
##   value outcome
## 1 12.34       1
## 2 32.20       1
## 3 24.30       0
## 4 83.10       0
## 5  8.32       0
```

For reference, if you want to load data from an Excel spreadsheet, here are some popular libraries: XLConnect (https://cran.r-project.org/web/packages/XLConnect/index.html), openxlsx (https://cran.r-project.org/web/packages/openxlsx/index.html), readxl (https://cran.r-project.org/web/packages/readxl/index.html).

Pipeline Check

So far we don’t really need to create any custom code, simply use read_csv or fread if you have large data and/or complex data types, otherwise stick to read.csv. If this is new data, use the `readLines` function:

```
path_and_file_name <- 'https://cran.r-project.org/src/base/README'
print(readLines(path_and_file_name, n=5))
```

```
## [1] ""
## [2] "\t\t\tTHE BASIC R README"
## [3] ""
## [4] ""
## [5] "\t   (See \"doc/FAQ\" and \"doc/RESOURCES\" for more detailed information"
```