Volpe R Course: Session 3, Data Exploration

Instructors: Don Fisher, Dan Flynn, Jessie Yang
Course webpage: `http://bit.ly/volpeR`

9/19/2017

Overview

Last session:

- Writing functions
- Writing loops

Today: Data exploration

- Aggregating data
- Matching
- Basic data analysis
- Saving work and sharing

Homework

Quick review of some of your solutions!

Loops

Write a loop to perform some task on your data.

```r
indexvec <- vector()

for(indexvalue in 1:15){
  indexvec <- c(indexvec,
 sample(homework2$Numeric1,size=1))
}

indexvec
```

Functions

Write a function to perform some task on your data.

```r
GradeServ <- function(emp){
  mg <- mean(emp$Grade)
  ms <- mean(emp$Years.Of.Service)
  c(mg, ms)
}
GradeServ(emp)
```

Data exploration

Read in the Hubway data from the course page. We will use this data on Hubway bike-share trips (from `https://www.thehubway.com/system-data`) to demonstrate first steps in exploring a new data set (with all the issues that real-world data often have!).

Read in the data, after downloading to your H: drive:

```r
setwd("H:/R")

d <- read.csv("Hubway_Sample.csv")

head(d)
```

```
##   tripduration     starttime      stoptime start.station.id
## 1          370 7/1/2016 0:00 7/1/2016 0:07               39
## 2          211 7/1/2016 0:01 7/1/2016 0:04               60
## 3          720 7/1/2016 0:01 7/1/2016 0:13               36
## 4          720 7/1/2016 0:01 7/1/2016 0:13               36
## 5          445 7/1/2016 0:01 7/1/2016 0:08               36
## 6          320 7/1/2016 0:02 7/1/2016 0:07               26
##                               start.station.name start.station.latitude
## 1                  Washington St. at Rutland St.               42.33849
## 2 Charles Circle - Charles St. at Cambridge St.               42.36062
## 3      Boston Public Library - 700 Boylston St.               42.34967
## 4      Boston Public Library - 700 Boylston St.               42.34967
## 5      Boston Public Library - 700 Boylston St.               42.34967
## 6                  Washington St. at Waltham St.               42.34152
```

Aggregating data

Aggregating data is done most often with these functions: `aggregate`, `tapply`, and `apply`.

Use `aggregate` to make two-way (or more) summary tables:

```r
aggregate(d$tripduration,
          by = list(d$birth.year, d$gender), FUN = mean)
```

Use `tapply` to make one-way summary tables:

```r
tapply(d$tripduration, d$gender, mean)
```

Aggregating data

Use apply to look across multiple columns or rows:

```
apply(d[6:7],      # data frame to look at (should be two-dimensional)
      2,                    # 1 is for rows, 2 is for columns
      mean)                 # Function
```

Aggregating data

For these data, the start and stop time should be read in as a `datetime` class vectors, which have different properties from character or factor vectors. Use the `strptime` function to convert to `datetime` class, which in R is called POSIXlt.

```
class(d$starttime)
```

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

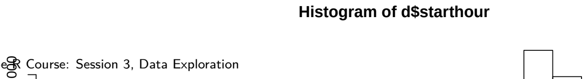
```
d$starttime[1]
```

```
## [1] 7/1/2016 0:00
## 33764 Levels: 7/1/2016 0:00 7/1/2016 0:01 7/1/2016 0:02 ... 7/9/2016 9:59
```

```
# Pay attention to the capitalization!
d$starttime <- strptime(d$starttime, format = "%m/%d/%Y %H:%M")
class(d$starttime)
```

```
## [1] "POSIXlt" "POSIXt"
```

```
# Create starthour
d$starthour <- as.numeric(format(d$starttime, "%H"))
hist(d$starthour) # see the clear peaks for communiting times.
```



Aggregating data

Let's aggregate the trip duration by the new variable start hour and by gender.

```r
# Two index vectors, using aggregate
aggregate(d$tripduration,
          by = list(d$starthour, d$gender), FUN = mean)
```

Exercise:

Create a table of average trip duration by *month* of the year.

Hint: look at `?strptime` for format ideas

Optional advanced exercise:

Carry out a statistical test to assess if trip duration differs by month of the year. Do you format *month* as a factor or a numeric variable? Which test would you use to assess that?

Matching and merging

Matching is a powerful tool to work with character strings, and comes in handy for both cleaning data and creating subsets of data. The most common functions are:

- `grep` to find the elements that partially match one input
- `sub` to find and change the elements that match one input
- `match` to find the elements that match a vector of inputs

```r
boylston.stations <- grep("Boylston", d$start.station.name)

length(boylston.stations)
```

```
## [1] 14677
```

```r
length(boylston.stations) / nrow(d)
```

```
## [1] 0.08768408
```

```r
# make a subset based on this search:

db <- d[boylston.stations,]
dim(db)
```

```
## [1] 14677    16
```

Regular expressions

The use of grep above is an example of *regular expression*. These are very flexible and form the basis for all search engines. In R, there are number of ways to use regular expressions, with grep and sub being two of the most common ones. regexpr also provides additional options for those familiar with Perl or Python.

```
text1 <- c("Testing", "matching ", " and substitutions")
sub(" +$", "", text1) # find and replace trailing whitespace
```

```
## [1] "Testing"            "matching"           " and substitutions"
```

```
grep("^ ", text1) # find elements which start with a whitespace
```

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

```
grep("^[A-Z]", text1) # find elements which start with a capital letter
```

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

Merging data frames

Merging data between different sources requires having a common column name or row name. The two functions most often used are merge and match. Type the following:

```
example(merge)
authors
books

merge(authors, books, by.x = "surname", by.y = "name")
# and do it again again, adding `all.y = TRUE`
```

We can use match to carry out a similar operation. match is more flexible than merge, and therefore more difficult to use!

```
m1 <- match(authors$surname, books$name) # returns index for the second element

books[m1,]

newdata <- data.frame(authors, books[m1,])
```

Merging data frames

`match` can also be used in many other contexts, besides merging data frames. Additional functions for matching to know are `which`, `grep`, and `intersect`.

```r
which(books$name == "Ripley")
```

```
## [1] 4 5
```

```r
grep("Rip", books$name) # partial string matching
```

```
## [1] 4 5
```

```r
intersect(books$name, authors$surname)
```

```
## [1] "Tukey"    "Venables" "Tierney"  "Ripley"   "McNeil"
```

```r
# unique shared elements, order unimportant.
```

Saving your work

After doing all this work, you will want to save the results. You can save R objects in your current workspace to file type called `.RData`. Then you can load these R objects back in at a later session, without having to carry out this preparation work.

For example, to save all objects in the current workspace:

```
save(list = ls(), file = "My_Prepped_Data.RData")
```

and load it back in using

```
load("My_Prepped_Data.RData")
```

To save specific R objects, you will name them in the list as strings, such as

```
save(list = c('d', 'books'), file = "My_Prepped_Data.RData")
```

You can also consider making a script that does all this preparation work, and save it as a file called something like `Analysis Prep.R`. Then, in your data analysis script, you can automatically run that script by inserting the command: `source('Analysis Prep.R')`.

Homework

Using the Hubway sample data, or your own data if you like, try to do the following:

- Make summaries of your data using `aggregate`, `tapply`, and `apply`
- Make a subset of your data using `grep`
- Make two different subsets of your data, then merge them together using `merge`

Optional advanced homework:

- Examine the data for outliers. What values are too extreme to be 'real', and how would you filter your dataset to exclude them?
- Analyze your dataset using the statistical functions `t.test`, `aov`, or `lm`, as appropriate.