Week 2

Week 2 of Getting and Cleaning Data: Extracting Data From Databases and the Web

- Welcome to Week 2 of Getting and Cleaning Data!
- The primary goal is to introduce you to the most common data storage systems and the appropriate tools to extract data from web or from databases like MySQL.
- Remember that the Course Project is open and ongoing. It is due BEFORE 11:30 PM UTC on the Sunday at the end of Week 3, but please don't put it off until the last minute.
- To access Course Project instructions and submission interface, click the Course Project link in the left navigation bar.
- With the skills you learn this week you should be able to start on the basic data extraction that will form the beginnings of your project.

The httr package (Hadley Wickham)

The aim of ***httr*** is to provide a wrapper for RCurl customised to the demands of modern web APIs.

Key features:

- Functions for the most important http verbs: `GET()`, `HEAD()`, `PATCH()`, `PUT()`, `DELETE()` and `POST()`.

  For more on these verbs - visit this site

  `http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html`

- Automatic connection sharing across requests to the same website (by default, curl handles are managed automatically), cookies are maintained across requests, and a up-to-date root-level SSL certificate store is used.
- Requests return a standard reponse object that captures the http status line, headers and body, along with other useful information.
- Response content is available with `content()` as a raw vector (as `= "raw"`), a character vector (as `= "text"`), or parsed into an R object (as = ”parsed”), currently for html, xml, json, png and jpeg.
- You can convert http errors into `R` errors with `stop_for_status()`.
- Config functions make it easier to modify the request in common ways: `set_cookies()`, `add_headers()`, `authenticate()`, use_proxy(), `verbose()`, `timeout()`, `content_type()`, accept(), `progress()`.
- Support for OAuth 1.0 and 2.0 with `oauth1.0_token()` and `oauth2.0_token()`.
- The demos directory has seven `OAuth` demos: three for 1.0 (twitter, vimeo and yahoo) and four for 2.0 (facebook, github, google, linkedin). `OAuth` credentials are automatically cached within a project.

httr wouldn’t be possible without the hard work of the authors of RCurl and curl. Thanks!

httr is inspired by http libraries in other languages, such as Resty, Requests and httparty.

Installation

To get the current released version from CRAN:

```
install.packages("httr")
```

To get the current development version from github:

```
# install.packages("devtools")
devtools::install_github("hadley/httr")
```

OAuth - Open Standard for Authorisation

- *OAuth* is an open standard for authorization. OAuth provides client applications a 'secure delegated access' to server resources on behalf of a resource owner.
- It specifies a process for resource owners to authorize third-party access to their server resources without sharing their credentials.
- Designed specifically to work with *Hypertext Transfer Protocol (HTTP)*, OAuth essentially allows access tokens to be issued to third-party clients by an authorization server, with the approval of the resource owner, or end-user.
- The client then uses the access token to access the protected resources hosted by the resource server.
- OAuth is commonly used as a way for web surfers to log into third party web sites using their Google, Facebook or Twitter accounts, without worrying about their access credentials being compromised.
- OAuth is a service that is complementary to, and therefore distinct from, OpenID.
- OAuth is also distinct from OATH, which is a reference architecture for authentication, not a standard.

References

`http://en.wikipedia.org/wiki/OAuth`

Useful Commands for Question 1

```
http://cran.r-project.org/web/packages/httr/httr.pdf
```

- `oauth_app` : Create an *OAuth* application.
- `oauth1.0_token` : Generate an *oauth1.0* token.
- `oauth2.0_token` : Generate an *oauth2.0* token.
- `oauth_endpoints` : Describe an OAuth endpoint.
- `stop_for_status` : Take action on http error.
- `warn_for_status` : Take action on http error.
- `content` : Extract content from a request
- `GET` : GET a url.
- `BROWSE` : Open specified url in browser.

Question 1

Question

- Register an application with the Github API here

  `https://github.com/settings/applications.`

- Access the API to get information on your instructors repositories
- (hint: this is the url you want ”https://api.github.com/users/jtleek/repos”).
- Use this data to find the time that the datasharing repo was created.
- What time was it created?
- This tutorial may be useful

  `(https://github.com/hadley/httr/blob/master/demo/oauth2-github.r).`

- You may also need to run the code in the base R package and not R studio.

Options

(i) 2012-06-20T18:39:06Z

(ii) 2014-03-05T16:11:46Z

(iii) 2014-01-04T21:06:44Z

(iv) 2013-11-07T13:25:07Z

The Basics of Structured Query Language (SQL)

A `SQL SELECT` statement can be broken down into numerous elements, each beginning with a keyword. Although it is not necessary, common convention is to write these keywords in all capital letters.

The most fundamental and common elements of a SELECT statement, namely

- `SELECT`
- `FROM`
- `WHERE`
- `ORDER BY`

The `SELECT ... FROM` Clause

The most basic `SELECT` statement has only 2 parts:

(1) what columns you want to return and

(2) what table(s) those columns come from.

The WHERE Clause

The next thing we want to do is to start limiting, or filtering, the data we fetch from the database. By adding a `WHERE` clause to the `SELECT` statement, we add one (or more) conditions that must be met by the selected data.

This will limit the number of rows that answer the query and are fetched. In many cases, this is where most of the ”action” of a query takes place.

What if we want to check if a column value is equal to more than one value? If it is only 2 values, then it is easy enough to test for each of those values, combining them with the `OR` operator.

The `ORDER BY` Clause

Until now, we have been discussing filtering the data: that is, defining the conditions that determine which rows will be included in the final set of rows to be fetched and returned from the database.

Once we have determined which columns and rows will be included in the results of our `SELECT` query, we may want to control the order in which the rows appearsorting the data.

To sort the data rows, we include the `ORDER BY` clause. The `ORDER BY` clause includes one or more column names that specify the sort order.

References

http://www.1keydata.com/sql/sql-commands.html

http://www.cs.utexas.edu/~mitra/csFall2013/cs329/lectures/sql.html

http://www.w3schools.com/sql/

The sqldf package

- ***sqldf*** is an R package for running SQL statements on R data frames, optimized for convenience.
- ***sqldf*** works with the SQLite, H2, PostgreSQL or MySQL databases.
- SQLite has the least prerequisites to install.
- H2 is just as easy if you have Java installed and also supports Date class and a few additional functions.
- PostgreSQL notably supports Windowing functions providing the SQL analogue of the R ave function.
- MySQL is a particularly popular database that drives many web sites.

```
# installs everything you need to use sqldf with SQLite
# including SQLite itself
install.packages("sqldf")
# shows built in data framesdata()
# load sqldf into workspace
library(sqldf)
sqldf("select * from iris limit 5")
sqldf("select count(*) from iris")
sqldf("select Species, count(*) from iris group by Species")


# create a data frame
DF <- data.frame(a = 1:5, b = letters[1:5])
sqldf("select * from DF")
sqldf("select avg(a) mean, variance(a) var from DF")
```

Question 2

The **sqldf** package allows for execution of SQL commands on `R` data frames.

We will use the **sqldf** package to practice the queries we might send with the `dbSendQuery` command in RMySQL.

Download the American Community Survey data and load it into an R object called `acs`.

`https://d396qusza40orc.cloudfront.net/getdata%2Fdata%2Fss06pid.csv`

Which of the following commands will select only the data for the probability weights `pwgtp1` with ages less than 50?

(i) `sqldf("select * from acs where AGEP < 50 and pwgtp1")`

(ii) `sqldf("select * from acs")`

(iii) `sqldf("select * from acs where AGEP < 50")`

(iv) `sqldf("select pwgtp1 from acs where AGEP < 50")`

```
library(sqldf)

## Data saved in local directory as "ss06pid.csv"

acs <- read.csv("./ss06pid.csv", header=T, sep=",")

names(acs)
```

Question 3

Using the same data frame you created in the previous problem, what is the equivalent function to `unique(acs$AGEP)`

(i) `sqldf("select unique AGEP from acs")`

(ii) `sqldf("select distinct pwgtp1 from acs")`

(iii) `sqldf("select AGEP where unique from acs")`

(iv) `sqldf("select distinct AGEP from acs")`

R commands for working with Text

Here are a small selection of useful commands for working with text

The nchar() function

The `nchar()` command returns the number of characters in the argument.

```
> string=c("kevin")
> nchar(string)
[1] 5
> nchar(1001)
[1] 4
```

The grep() function

The `grep()` command returns the location of a string that contains a specified substring, from a character vector. If you specify the additional argument ”`value=T`”, it will return that string.

```
> names(iris)
[1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  "Species"
>
> class(names(iris))
[1] "character"
> grep("Petal",names(iris))
[1] 3 4
>
> grep("Petal",names(iris),value=T)
[1] "Petal.Length" "Petal.Width"
>
```

The `paste()` function

This command creates a string of specified components.
The default is to have whitespace between each component. This can be removed with the additional argument `sep=""`.

```
> x=5
> paste("file",x,".csv")
[1] "file 5 .csv"
>
> paste("file",x,".csv",sep="")
[1] "file5.csv"
```

```
> filenm(2)
[1] "file 2 .csv"
```

The `gsub()` function

The `R` command `gsub()` is used to replace a character or piece of text with another in a specified string

```
> string=c("kevin")
> gsub("k","s",string)
[1] "sevin"
```

The `sprintf()` function

This command returns a character vector containing a formatted combination of text and variable values. The structure of the command is `sprintf(format, input)`.

```
> sprintf("%f", pi)
[1] "3.141593"
> sprintf("%.3f", pi)   # 3decimal places
[1] "3.142"
> sprintf("%1.0f", pi)   # no decimal places
[1] "3"
> sprintf("%5.1f", pi)  # 5 characters with whitespace
[1] "  3.1"
> sprintf("%05.1f", pi)  #5 characters no whitespace
[1] "003.1"
> sprintf("%+f", pi)
[1] "+3.141593"
```

To express asingle or double digit character integer as three character number

```
> x = 4
> sprintf("%03d", x)
[1] "004"
>
> x=40
> sprintf("%03d", x)
[1] "040"
```

For character data (i.e. strings)

```
> sprintf("%s %d", "test", 1:3)
[1] "test 1" "test 2" "test 3"
```

N.B *s* for string and *d* for integers.

The `readLines()` function

This command is used to read some or all text lines from a connection (i.e. an internet connection or Database connection

The `list.files()` function

This command is used to produce a list of files in a specified directory.

```
getwd() # working directory
list.files(getwd()) # List all files in working directory
```

Question 4

How many characters are in the 10th, 20th, 30th and 100th lines of HTML from this page:

`http://biostat.jhsph.edu/~jleek/contact.html`

(Hint: the `nchar()` function in `R` may be helpful)

(i) 43 99 8 6

(ii) 45 31 7 31

(iii) 43 99 7 25

(iv) 45 31 7 25

(v) 45 0 2 2

(vi) 45 31 2 25

(vii) 45 92 7 2

Fixed width format

- This format has data where every field has a fixed width and for those fields where their width is less than the value, it is padded with pad characters.
- Every record ends with a new line character. The field width and pad character are both user configurable. This format may optionally contain a header at the top which corresponds to the properties of the business object.
- If the header is absent, then the order of the fields in the input data is the same as the order of properties in the business object. Typically, fixed width format contains data where every field has a different width. To enable this, the field width will be represented as a list property.
- Fixed width format may be transmitted in several forms. Fixed width format may come in a stream through data bindings such as HTTP, JMS and MQ as well as files.
- Fixed width format with no header and one record This is a fixed width format with one record and does not contain a header. In this case, the business object properties have to be in the order of the fields in the data.

```
8A7111John~~~~~~Doe~~~~~~~80000~
```

The corresponding business object for the record is as follows. Note the business object property names are in order of the data in the fixed width format. Note the lastName and firstName fields.

	CustomerBO
id	8A7111
firstName	John
lastName	Doe
salary	80000

Question 5

Read this data set into R and report the sum of the numbers in the fourth column.

`https://d396qusza40orc.cloudfront.net/getdata%2Fwksst8110.for`

Original source of the data:

`http://www.cpc.ncep.noaa.gov/data/indices/wksst8110.for`

(Hint this is a fixed width file (fwf) format)

(i) 32426.7

(ii) 35824.9

(iii) 222243.1

(iv) 36.5

(v) 28893.3

(vi) 101.83

Find out about fixed width files

```
help(read.fwf)
```

```
## Data Saved in Local Directory
## as "DSS3wk5q5.for"

data <- read.csv("./DSS3wk5q5.for", header = TRUE)
file_name <- "./DSS3wk5q5.for"
df <- read.fwf(file=file_name,
    widths=c(-1,9,-5,4,4,-5,4,4,-5,4,4,-5,4,4), skip=4)

## Carry Out usual Data Frame Inspection Procedures
```

```
ff <- tempfile()
cat(file=ff, "123456", "987654", sep="\n")
read.fwf(ff, width=c(1,2,3))    #> 1 23 456 \ 9 87 654
unlink(ff)
```