NYC DATA SCIENCE ACADEMY

Data Science with R (Data Analytics)

Basic Data Elements

Outline For Today

- Data Transformation
  - Reshape
  - Split
  - Combine
- Character Manipulation
- Dates and timestamps
- Accessing Database
- Getting Data from Web
  - API data sources
  - Connecting to an external database
- Other Data Resources

Data Transformation Overview

Subset

Index operators [], subset(), and with() can be used to subset data according to various conditions.

```
#Three different ways to do the exact same thing!
data_sub = iris[iris$Species=='setosa', 3:5]
data_sub1 = subset(iris, Species=='setosa', 3:5)
data_sub2 = with(iris, iris[Species=='setosa', 3:5])
head(data_sub, 1); head(data_sub1, 1); head(data_sub2, 1)
```

	Petal.Length	Petal.Width	Species
1	1.4	0.2	setosa

	Petal.Length	Petal.Width	Species
1	1.4	0.2	setosa

	Petal.Length	Petal.Width	Species
1	1.4	0.2	setosa

Transform

The transform() function can be used to create a new column in an existing dataset.

```
iris_tr = transform(iris, v1=log(Sepal.Length))
head(iris_tr, 1)
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species	v1
1	5.1	3.5	1.4	0.2	setosa	1.6292

```
#Equivalent:
iris_tr1 = iris
iris_tr1$v1 = log(iris$Sepal.Length)
head(iris_tr1, 1)
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species	v1
1	5.1	3.5	1.4	0.2	setosa	1.6292

Discretize

The cut() function can be used to transform a numeric variable into a categorical variable.

```
groupvec = quantile(iris_tr$v1, c(0, 0.25, 0.50, 0.75, 1.0))
labels = c('A', 'B', 'C', 'D')
iris_tr$v2 = cut(iris_tr$v1, breaks=groupvec, labels=labels, include.lowest=TRUE)

groupvec
```

```
      0%      25%      50%      75%     100%
1.458615 1.629241 1.757858 1.856298 2.066863
```

```
iris_tr[c(1,6), ]
```

```
    Sepal.Length  Sepal.Width   Petal.Length  Petal.Width   Species   v1
v2
1   5.1           3.5           1.4           0.2           setosa    1.629   A
6   5.4           3.9           1.7           0.4           setosa    1.686   B
```

Set Levels of a Factor

```
vec = rep(c(0,1), c(4,6))
vec
```

```
[1] 0 0 0 0 1 1 1 1 1 1
```

```
#Converting to a factor and creating the labels/levels all at once.
vec_fac
vec_fac = factor(vec, labels=c('male','female'))
```

```
[1] male   male   male   male   female female female female female female
Levels: male female
```

Set Levels of a Factor

```
#First converting to a factor.
vec1 = factor(rep(c(0,1,3), c(4,6,2)))
vec1
```

```
[1] 0 0 0 0 1 1 1 1 1 1 3 3
Levels: 0 1 3
```

```
#Then creating the labels/levels.
levels(vec1) = c("male", "female", "male")
vec1
```

```
[1] male   male   male   male   female female female female female female male male
Levels: male female
```

Rename Levels of a Factor

```
vec2 = factor(rep(c('b','a'), c(4,6)))
vec2
```

```
[1] b b b b a a a a a a
Levels: a b
```

```
levels(vec2)
```

```
[1] "a" "b"
```

```
relevel(vec2, ref='b') #Changing the reference level.
```

```
[1] b b b b a a a a a a
Levels: b a
```

Data Reshape

What makes data wide or long?

Wide data has a column for each variable.

For example, this is **wide-format** data:

```
data = iris[, 1:4] #A wide dataset; columns are variables, rows are observations.
head(data, 5)
```

```
     Sepal.Length  Sepal.Width  Petal.Length  Petal.Width
1    5.1           3.5          1.4           0.2
2    4.9           3.0          1.4           0.2
3    4.7           3.2          1.3           0.2
4    4.6           3.1          1.5           0.2
5    5.0           3.6          1.4           0.2
```

What makes data wide or long?

Long-format data has a column for possible variable types and a column for the values of those variables.

For example, this is **Long-format** data:

	Species	variable	value
121	virginica	Sepal.Length	6.9
411	virginica	Petal.Length	5.1
549	versicolor	Petal.Width	1.1
170	setosa	Sepal.Width	3.8
63	versicolor	Sepal.Length	6.0
418	virginica	Petal.Length	6.7
314	setosa	Petal.Length	1.1
480	setosa	Petal.Width	0.2
567	virginica	Petal.Width	1.8
66	versicolor	Sepal.Length	6.7

Why we need to reshape the data?

Long-form data and wide-form data are used for different purposes in data analysis. The ultimate shape you want to get your data into will depend on what you are doing with it.

It turns out that you need wide-format data for some types of data analysis and long-format data for others. In reality, you need long-format data much more commonly than wide-format data. For example, `ggplot2` requires long-format data (technically tidy data), `plyr` requires long-format data, and most modelling functions (such as `lm()`, `glm()`, and `gam()`) require long-format data. But people often find it easier to record their data in wide format.

So it's important to get comfortable converting back-and-forth between the two.

Long and Wide-Form Data

Use the stack() function to reshape to long form; use unstack() for wide form. Stacking vectors concatenates multiple vectors into a single vector along with a factor indicating where each observation originated. Unstacking reverses this operation.

```
data_l = stack(data) #A long dataset; treats all elements as pieces of data.
data_w = unstack(data_l) #A wide dataset; columns are variables, rows are
obs.
str(data_l); str(data_w)
```

```
'data.frame':   600 obs. of  2 variables:
 $ values: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ ind   : Factor w/ 4 levels "Petal.Length",..: 3 3 3 3 3 3 3 3 3 3 ...
```

```
'data.frame':   150 obs. of  4 variables:
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
```

Long and Wide-Form Data

Let's take the last two columns of iris and reshape them to wide form.

```
subdata = iris[ ,4:5] #This subset of the overall data is in long form.
subdata[c(1,51, 101), ]
```

	Petal.Width	Species
1	0.2	setosa
51	1.4	versicolor
101	2.5	virginica

```
subdata_w = unstack(subdata) #Each factor becomes a column in the wide form.
head(subdata_w, 1)
```

	setosa	versicolor	virginica
1	0.2	1.4	2.5

Restructuring with *reshape2*

The reshape2 package lets you flexibly restructure and aggregate data using just two functions:

melt() (takes wide format data and "melts" it into long format data)

cast() (takes long format data and "casts" it into wide format data).

- dcast() reshapes a molten data into a data frame
- acast() reshape a molten data into a vector/matrix/array

Think of working with metal: if you melt metal, it drips and becomes long. If you cast it into a mould, it becomes wide. Since you will most commonly work with data.frame objects, we'll explore the dcast function.

Restructuring with *reshape2*

Let's take the long-format `subset` data and `cast` it into a wide-format data frame.

`dcast` uses a formula to describe the shape of the data. The arguments on the left refer to the ID variables and the arguments on the right refer to the measured variables.

```
#install.packages("reshape2")
library(reshape2)                #Cast long format to wide
format.
dcast(data=subdata,              #Specifying the data to manipulate.
      formula=Species ~ .,       #Species should be the main column; nothing else.
      value.var='Petal.Width',   #Values to fill in should come from Petal.Width.
      fun=mean)                  #Aggregate the values by the mean.
```

```
     Species      .
1    setosa       0.246
2    versicolor   1.326
3    virginica    2.026
```

Restructuring with *reshape2*

To make use of the function we need to specify a data frame, the id variables (which will be left at their settings) and the measured variables (columns of data) to be stacked. The default assumption on measured variables is that it is all columns that are not specified as id variables.

```
iris_long = melt(data=iris,          #Melt wide format to long format.
                id.vars ='Species')          #The main identification variable is
Species.
set.seed(5)
i = sample(nrow(iris_long), 5)
iris_long[i, ]
```

```
        Species      variable       value
121     virginica    Sepal.Length   6.9
411     virginica    Petal.Length   5.1
549     versicolor   Petal.Width    1.1
170     setosa       Sepal.Width    3.8
63      versicolor   Sepal.Length   6.0
```

Restructuring with *reshape2*

Using dcast to get the mean (by species) of each variable:

```
dcast(data=iris_long,                    #Specifying the data to manipulate.
      formula=Species ~ variable,        #Species is main col.; swing variable col.
      value.var = 'value',               #Values to fill in should come from value col.
      fun=mean)                          #Aggregate the values by the mean.
```

	Species	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
1	setosa	5.01	3.43	1.46	0.246
2	versicolor	5.94	2.77	4.26	1.326
3	virginica	6.59	2.97	5.55	2.026

In-class exercise using *Tips dataset*

1. Come up one or two questions you want to answer from Tips dataset.
2. Present your code and results.

Example : Is there a gender difference in the tipping habits?

Using the built-in tips dataset, get the mean tip amount by sex:

```
dcast(tips, sex ~ ., value.var='tip', fun=mean)
#Manipulate the tips data; sex is the main column; don't keep anything else; fill in
#values based on the tip column; aggregate the results by the mean.
```

```
    sex      .
1   Female   2.833448
2   Male     3.089618
```

Get means by sex and party size ("swing over" the size variable).

```
dcast(tips, sex ~ size, value.var='tip', fun=mean)
```

```
    sex      1      2      3      4      5      6
1   Female   1.28   2.53   3.25   4.02   5.14   4.60
2   Male     1.92   2.61   3.48   4.17   3.75   5.85
```

Example : Is there a gender difference in the tipping habits?

We want to aggregate the average total_bill by sex. How can we do that?

```
dcast(tips, sex ~ . , value.var='total_bill', fun=mean)
```

	sex	.
1	Female	18.05690
2	Male	20.74408

Split and Combine Data

Merge Two Data Frames

```
datax = data.frame(id=c(1,2,3), gender=c('M', 'F', 'M'))
datay = data.frame(id=c(3,1,2), name=c('tom','john','mary'))
datax; datay
```

```
    id  gender
1   1   M
2   2   F
3   3   M
```

```
    id  name
1   3   tom
2   1   john
3   2   mary
```

Merge Two Data Frames

Merging two data frames by common columns or row names (similar to *JOIN* in SQL):

```
merge(datax, datay, by='id')
```

```
   id  gender  name
1  1   M       john
2  2   F       mary
3  3   M       tom
```

Split Into Groups

The split() function divides the data into groups defined by the factor specified in the second argument.

```
iris_split = split(iris, iris$Species)
class(iris_split)
```

```
[1] "list"
```

```
attributes(iris_split)
```

```
$names
[1] "setosa"     "versicolor" "virginica"
```

```
str(iris_split)
```

Split Into Groups

```
List of 3
 $ setosa    :'data.frame':	50 obs. of  5 variables:
  ..$ Sepal.Length: num [1:50] 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
  ..$ Sepal.Width : num [1:50] 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
  ..$ Petal.Length: num [1:50] 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
  ..$ Petal.Width : num [1:50] 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
  ..$ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
 $ versicolor:'data.frame':	50 obs. of  5 variables:
  ..$ Sepal.Length: num [1:50] 7 6.4 6.9 5.5 6.5 5.7 6.3 4.9 6.6 5.2 ...
  ..$ Sepal.Width : num [1:50] 3.2 3.2 3.1 2.3 2.8 2.8 3.3 2.4 2.9 2.7 ...
  ..$ Petal.Length: num [1:50] 4.7 4.5 4.9 4 4.6 4.5 4.7 3.3 4.6 3.9 ...
  ..$ Petal.Width : num [1:50] 1.4 1.5 1.5 1.3 1.5 1.3 1.6 1 1.3 1.4 ...
  ..$ Species     : Factor w/ 3 levels "setosa","versicolor",..: 2 2 2 2 2 2 2 2 2 2 ...
 $ virginica :'data.frame':	50 obs. of  5 variables:
  ..$ Sepal.Length: num [1:50] 6.3 5.8 7.1 6.3 6.5 7.6 4.9 7.3 6.7 7.2 ...
  ..$ Sepal.Width : num [1:50] 3.3 2.7 3 2.9 3 3 2.5 2.9 2.5 3.6 ...
  ..$ Petal.Length: num [1:50] 6 5.1 5.9 5.6 5.8 6.6 4.5 6.3 5.8 6.1 ...
  ..$ Petal.Width : num [1:50] 2.5 1.9 2.1 1.8 2.2 2.1 1.7 1.8 1.8 2.5 ...
  ..$ Species     : Factor w/ 3 levels "setosa","versicolor",..: 3 3 3 3 3 3 3 3 3 3 ...
```

'Unsplit'

unsplit reverses the split operation:

```
iris_unsplit = unsplit(iris_split, iris$Species)
class(iris_unsplit)
```

```
[1] "data.frame"
```

```
iris_unsplit[c(1,51, 101), ]
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
51	7.0	3.2	4.7	1.4	versicolor
101	6.3	3.3	6.0	2.5	virginica

Split Example

Using the tips dataset, let's split by sex and get some statistics from each subgroup.

```
library(reshape2) #Not using the reshape2 functions, just want the tips dataset.
tips
tips_by_sex = split(tips, tips$sex)
head(tips_by_sex[[1]], 2)
```

	total_bill	tip	sex	smoker	day	time	size
1	17.0	1.01	Female	No	Sun	Dinner	2
5	24.6	3.61	Female	No	Sun	Dinner	4

```
ratio_fun = function(x) {
  sum(x$tip) / sum(x$total_bill)
}
```

Split Example

Apply the ratio_fun() function to each member of the split list:

```
result = lapply(tips_by_sex, ratio_fun)
result
```

```
$Female
[1] 0.1569178

$Male
[1] 0.1489398
```

Character Manipulation

Character Manipulation

Here are some basic functions for manipulating character data:

- nchar(): Count the number of characters
- strsplit(): Split the elements of a character vector
- paste(): Concatenate strings
- substr(): Substrings of a character vector
- gsub(): Replacement
- grep(): Pattern matching

Character Functions: *nchar*

Count the number of characters

```
fruit = 'apple orange grape banana'
nchar(fruit)
```

```
[1] 25
```

Character Functions: *strsplit*

Split the elements of a character vector

```
split.string.list = strsplit(fruit, split=' ')
split.string.list
```

```
[[1]]
[1] "apple"  "orange" "grape"  "banana"
```

```
fruitvec = unlist(split.string.list)
fruitvec
```

```
[1] "apple"  "orange" "grape"  "banana"
```

Character Functions: *paste*

Concatenate strings

```
paste(fruitvec, collapse = ' ')
```

```
[1] "apple orange grape banana"
```

```
paste(fruitvec, collapse=',')
```

```
[1] "apple,orange,grape,banana"
```

```
paste(fruitvec, 'extra')
```

```
[1] "apple extra"  "orange extra" "grape extra"  "banana extra"
```

```
paste(fruitvec, 'extra', sep = '.')
```

```
[1] "apple.extra"  "orange.extra" "grape.extra"  "banana.extra"
```

Example: Formulas Creation

Paste is often useful when you want to create objects programmatically, such as generating a formula or creating variables on the fly.

```
n = 1:20
xvar = paste('x', n, sep = '')
right = paste(xvar, collapse = ' + ')
left = 'y ~'
my_formula = paste(left, right)
my_formula
```

```
[1] "y ~ x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9 + x10 + x11 + x12 + x13
+ x14 + x15 + x16 + x17 + x18 + x19 + x20"
```

Character Functions: *substr*

Substrings of a character vector

```
substr(fruit, 1, 5)
```

```
[1] "apple"
```

```
substr(fruit, 1, 7)
```

```
[1] "apple o"
```

```
substr(fruit, 9, 21)
```

```
[1] "ange grape ba"
```

Character Functions: *gsub*

Replacement

```
gsub('apple', 'strawberry', fruit)
```

```
[1] "strawberry orange grape banana"
```

```
gsub('a', '?', fruit)
```

```
[1] "?pple or?nge gr?pe b?n?n?"
```

```
gsub('an', 'HA', fruit)
```

```
[1] "apple orHAge grape bHAHAa"
```

Character Functions: *grep*

Pattern Matching

```
grep('grape', fruitvec)
```

```
[1] 3
```

```
grep('a', fruitvec)
```

```
[1] 1 2 3 4
```

```
grep('an', fruitvec)
```

```
[1] 2 4
```

Case Study: Exploring CRAN

Case Study: Exploring CRAN

We want to explore the information provided in packages on CRAN. The goal is to get a dataframe where one column contains the package name and another the author name.

```
#get packages table
#install.packages("XML")
library(XML)
web =
'http://cran.r-project.org/web/packages/available_packages_by_name.html'
packages = readHTMLTable(web, stringsAsFactors = FALSE)
```

Case Study: Exploring CRAN

Use str() to explore the packages object. The V1 column is what we need.

```
pnames = packages[[1]][ ,1]
length(pnames)
```

```
[1] 8586
```

```
pnames = pnames[2:11]
```

```
b = 'http://cran.r-project.org/web/packages/'
a = '/index.html'
urls = paste0(b, pnames, a)
```

Case Study: Exploring CRAN

Now urls is a character strings containing the ten web addresses. Let's explore one of them.

```
table = readHTMLTable(urls[1], stringsAsFactors=FALSE, header=FALSE)
info = table[[1]]
paste0(info$V1, info$V2)
```

```
[1] "Version:2.0"
[2] "Depends:R (≥ 2.10), nnet, quantreg, MASS, locfit"
[3] "Published:2014-07-11"
[4] "Author:Katalin Csillery, Louisiane Lemaire, Michael Blum and Olivier
Francois"
[5] "Maintainer:Michael Blum  <michael.blum at imag.fr>"
[6] "License:GPL (≥ 3)"
[7] "NeedsCompilation:no"
[8] "In views:Bayesian"
[9] "CRAN checks:abc results"
```

Case Study: Exploring CRAN

Get information on ten packages using lapply

```
tables = lapply(urls, readHTMLTable, stringsAsFactors=FALSE, header=FALSE)
infos = lapply(tables, function(x) x[[1]])
infovec = lapply(infos, function(x) paste0(x$V1, x$V2))
```

Case Study: Exploring CRAN

Extract author names from infovec using grep and lapply. Finally, combine anamevec with pnames.

```
aname = lapply(infovec, function(x) x[grep('Author:', x)])
anamevec = lapply(aname, function(x) substr(x, 8, nchar(x)))
anamevec = unlist(anamevec)
name.df = data.frame(pnames, anamevec)
```

Manipulating Dates and Timestamps

Two Types of Date-related Data

- Date object: only contains date and time information
- Time Series object: normal data object with timestamp added

Date Objects

- Date class: Only date; no time information
- POSIXct class: Time (with timezone) included with date

Date Class Object

Use as.Date() to create date class object.

```
date1 = '1989-05-04'
date1 = as.Date(date1)
class(date1)
```

```
[1] "Date"
```

```
date1 = '05/04/1989'
date1 = as.Date(date1, format='%m/%d/%Y')
```

Date Class Object

Manipulating a date class object:

```
date2 = date1 + 31
date2 - date1
```

```
Time difference of 31 days
```

```
date2 > date1
```

```
[1] TRUE
```

Date Class Object

Count from 1970.01.01 in the date class object:

```
Sys.Date() - structure(0, class='Date')
```

```
Time difference of 16962 days
```

Date Class Object

Create a vector of date class objects:

```
dates = seq(date1, length=4, by='day')
format(dates, '%w')
```

```
[1] "4" "5" "6" "0"
```

```
weekdays(dates)
```

```
[1] "Thursday" "Friday"   "Saturday" "Sunday"
```

POSIXct Class Object

Use as.POSIXct to create POSIXct class objects.

```
time1 = '1989-05-04'
time1 = as.POSIXct(time1)
time1 = "2011-03-1 01:30:00"
time1 = as.POSIXct(time1, format="%Y-%m-%d %H:%M:%S")
time1 = as.POSIXct("2011-03-1 01:30:00", tz='GMT')
time2 = seq(from=time1, to=Sys.time(), by='month')
```

POSIXct Class Object

Create a POSIXct class object based on a numeric object.

The ISOdatetime function can convert a numeric object into a POSIXct object.

```
time1 = ISOdatetime(2011,1,1,0,0,0)
rtimes = ISOdatetime(2013, rep(4:5,5), sample(30,10), 0, 0, 0)
```

Time Series Object

The zoo and xts packages are recommended to deal with time series objects.

```
#install.packages("xts")
library(xts)
x = 1:4
y = seq(as.Date('2001-01-01'), length=4, by='day')
date1 = xts(x, y)
```

Time Series Object

Extract or modify the content or time index of a time series object:

```
value = coredata(date1)
coredata(date1) = 2:5
time = index(date1)
```

Time Series Object

While we are using the xts package, other data manipulation methods like subset and aggregate still work.

```
x = 5:2
y = seq(as.Date('2001-01-02'), length=4, by='day')
date2 = xts(x, y)
date3 = cbind(date1, date2)
names(date3) = c('v1', 'v2')
date4 = rbind(date1, date2)
names(date4) = 'value'
```

Time Series Object

The window function can help us extract or modify a subset of a time series object (as observed between defined *start* and *end* times).

```
window(date4, start=as.Date("2001-01-04"))
```

```
           value
2001-01-04   5
2001-01-04   3
2001-01-05   2
```

```
#lag() and diff() are still available
lag(date2)
diff(date2)
```

Case Study: Exploring Stock Data

Case Study: Exploring Stock Data

First load the quantmod package, download data from the Shanghai Stock Exchange composite index, and extract the 'close' column, which is the adjusted closing price.

```
#install.packages("quantmod")
library(quantmod)
options(getSymbols.auto.assign=FALSE)
library(xts)
SSEC = getSymbols('^SSEC', src='yahoo', from='2000-01-01')
head(SSEC, 3)
```

```
           SSEC.Open SSEC.High SSEC.Low SSEC.Close
2000-01-04  1368.693  1407.518 1361.214   1406.371
2000-01-05  1407.829  1433.780 1398.323   1409.682
2000-01-06  1406.036  1463.955 1400.253   1463.942
           SSEC.Volume SSEC.Adjusted
2000-01-04           0      1406.371
2000-01-05           0      1409.682
2000-01-06           0      1463.942
```

Case Study: Exploring Stock Data

Compute the change rate, and find out the biggest change day:

```
data$ratio = with(data, diff(close)/close)
data.df = as.data.frame(data)
data.df[order(abs(data.df$ratio), decreasing=T), ][1:5, ]
```

```
             close         ratio
2007-02-27  2771.791   -0.09697993
2007-06-04  3670.401   -0.09000136
2001-10-23  1670.562    0.08972613
2008-09-19  2075.091    0.08638369
2008-04-24  3583.028    0.08503924
```

Case Study: Exploring Stock Data

Is there a calendar effect? To check, we aggregate *returns* by month

```
#install.packages("lubridate")
library(lubridate)
data$mday = month(data)
res = aggregate(data$ratio, data$mday, mean, na.rm=TRUE)
cat(format(res*100, digits=2, scientific=F))
```

```
0.00093  0.17295  0.02685  0.13156 -0.01909 -0.15709
0.05542 -0.09151 -0.01101 -0.06034  0.03629  0.13713
```

Case Study: Tencent QQ Data Analysis

Case Study: Tencent QQ Data Analysis

Tencent QQ is an instant messaging software service developed by Tencent. A QQ user can build a group to chat with friends. We'll explore QQ group chat data for this example. There are only two columns: the first column contains user IDs and the second column contains the chat times. So our questions are:

- Who chats the most in the group?
- When are people most likely to chat?

Case Study: Tencent QQ Data Analysis

```r
#install.packages(c("stringr","plyr"))
library(stringr)
library(plyr)
library(lubridate)
data = read.table('data/qqdata.csv', header=TRUE, sep=',',
                 stringsAsFactors=FALSE)
head(data, 3)
```

```
       id               time
1   8cha0   2011/7/8 12:11:13
2 2cha061   2011/7/8 12:11:49
3 6cha437   2011/7/8 12:13:36
```

Case Study: Tencent QQ Data Analysis

```
time = as.POSIXct(data$time, tz='GMT')
id = as.factor(data$id)
data1 = data.frame(id, time)

user = as.data.frame(table(data1$id))
user = user[order(user$Freq, decreasing=T), ]
user[1:5, ]  #getting the top five chat users
```

```
     Var1        Freq
91   7cha1       1511
86   6cha437     1238
66   4cha387     1100
98   8cha08      695
69   4cha69      533
```

Case Study: Tencent QQ Data Analysis

```
data1$hour = hour(data1$time)
hours = as.data.frame(table(data1$hour))
hours = hours[order(hours$Freq, decreasing=T), ]
data1$wday = wday(data1$time)
wdays = as.data.frame(table(data1$wday))
wdays = wdays[order(wdays$Freq, decreasing=T), ]
```

Accessing Databases

Accessing Databases

There are many database management systems (DBMS) for working with relational databases, and R can connect to all of the common ones.

The DBI package provides a unified syntax for accessing the following DBMS:

- SQLite
- MySQL
- MariaDB
- PostgreSQL
- Oracle

The RODBC package is an alternative that uses ODBC database connections

- This package is particularly useful for connecting to SQL Server/Access databases
- To use this package, you need to set up an ODBC data source on your machine

Example: Reading Data from an SQLite Database

```
#install.packages(c("DBI","RSQLite","learningr"))
library(DBI)
library(RSQLite)
driver = dbDriver("SQLite")
db_file = system.file("extdata", "crabtag.sqlite", package="learningr")
conn = dbConnect(driver, db_file)
query = "SELECT count(*) FROM Daylog"
(id_block = dbGetQuery(conn, query))
```

```
  count(*)
1      405
```

```
dbDisconnect(conn)
```

```
[1] TRUE
```

Getting Data from Web

The Different Routes of Data Retrieval

- If there is an API, use it.
- If there is no API but there is data in a table format, try readHTMLTable().
- If there is no API and the data is *unstructured*, you need to apply some form of web scraping.

Tools for Web-data Retrieval

- Packages for *retrieval*
    - RCurl
    - httr
- Packages for *parsing*
    - XML
    - rjson
    - RJSONIO
    - selectr
- Chrome Developer Tools

Using an API

Example: Get the weather data for your IP location from the Wunderground API.

- The RCurl package provides convenient functions to fetch URIs, get and post forms, etc.
- The RJSONIO package allows conversion to and from data in Javascript object notation (JSON) format

```
#install.packages(c("RCurl","RJSONIO"))
library(RCurl)
library(RJSONIO)
#the 'mykey' variable might need to be changed; create your own if it doesn't
work
mykey = 'a98d04ac43156c84'
url = paste0('http://api.wunderground.com/api/',
             mykey, '/conditions/forecast/q/autoip.json')
print(url)
```

```
[1]"http://api.wunderground.com/api/a98d04ac43156c84/conditions/forecast/q/aut
oip.json"
```

Extracting Data Using Wunderground API

```
fromurl = function(finalurl) {
    web = getURL(finalurl)
    raw = fromJSON(web)
    high = raw$forecast$simpleforecast$forecastday[[2]]$high['celsius']
    low = raw$forecast$simpleforecast$forecastday[[2]]$low['celsius']
    condition = raw$forecast$simpleforecast$forecastday[[2]]$conditions
    currenttemp = raw$current_observation$temp_c
    currentweather = raw$current_observation$weather
    city = as.character(raw$current_observation$display_location['full'])
    result = list(city=city, current=paste(currenttemp, '°C ',
                  currentweather, sep=''),
    tomorrow=paste(low, '-', high,'°C ', condition, sep=''))
    names(result) = c('city', 'current', 'tomorrow')
    return(result)
}
```

Local Weather Results

```
fromurl(url)
```

```
$city
[1] "New York, NY"

$current
[1] "27°C Overcast"

$tomorrow
[1] "23-32°C Clear"
```

Scraping Data From the Web

The RCurl and XML packages provide useful parsing functions:

- getURL: Download a web page
- readHTMLTable: Read data from one or more HTML tables
- htmlTreeParse/xmlTreeParse parses an XML or HTML file and generates an R structure representing the XML/HTML tree
- xmlApply applies a function to each of the children of an XMLNode
- getNodeSet: Find matching nodes in an internal XML tree/DOM

Example: Using readHTMLTable()

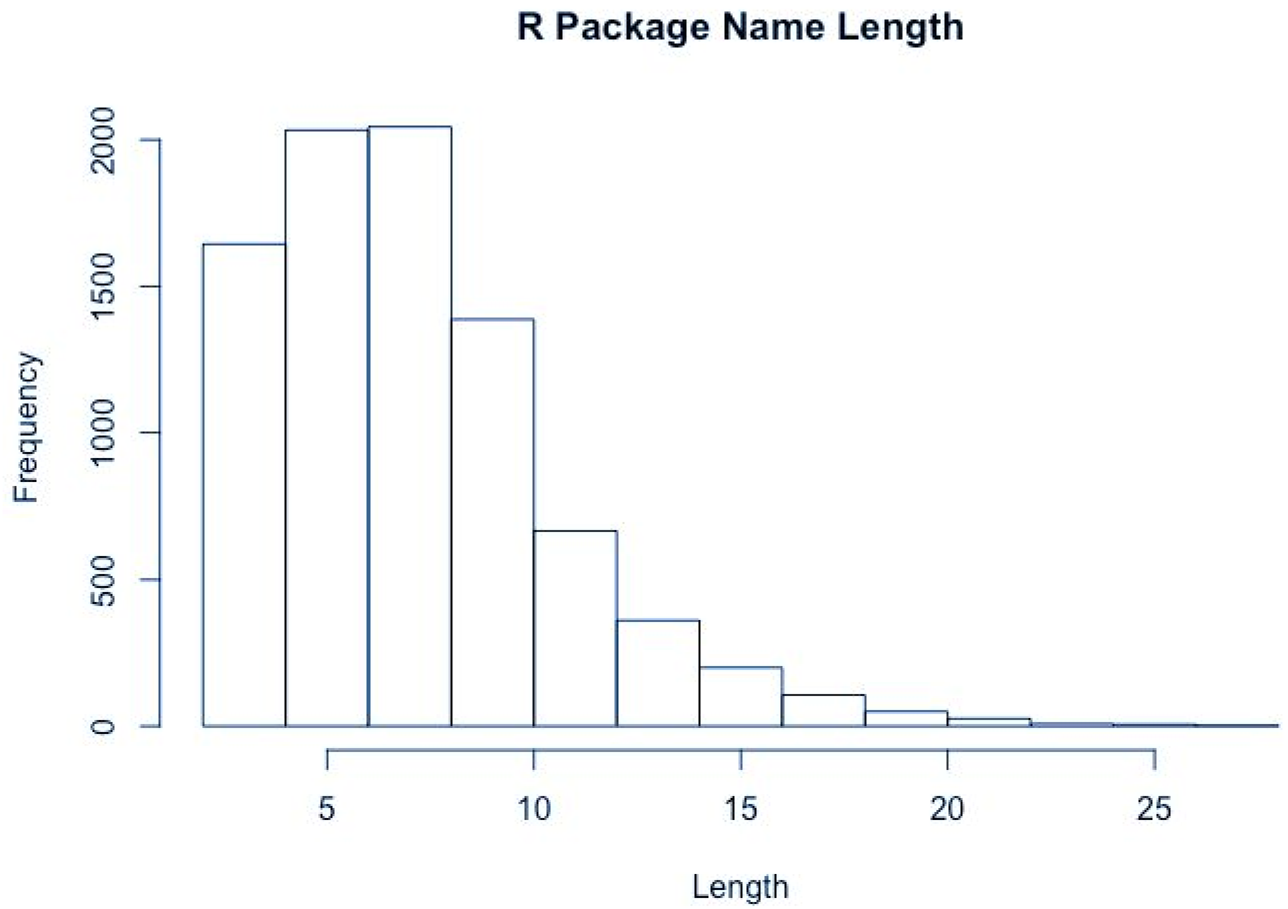
Use readHTMLTable to get data on Available CRAN Packages:

```
library(XML)
url =
'http://mirrors.ustc.edu.cn/CRAN/web/packages/available_packages_by_date.html'
tables = readHTMLTable(url,
                stringsAsFactors=FALSE,
                header=TRUE)
data = tables[[1]]
data[2,]
```

```
        Date        Package                                  Title
2       2015-06-21   FAMILY A Convex Formulation for Modeling Interactions with
Strong\nHeredity
```

Example: Using readHTMLTable()

```
res = nchar(data[,2])
hist(res, main="R Package Name Length", xlab="Length")
```



Example: Getting XML Data

```
library(XML)
url = "http://www.w3schools.com/xml/plant_catalog.xml"
xmlfile = xmlTreeParse(url)  #download and parse XML
xmltop = xmlRoot(xmlfile)     #get root node
xmlValue(xmltop[[10]][[1]])  #get leaf node data
```

```
[1] "Mayapple"
```

```
xmlValue(xmltop[['PLANT']][['COMMON']])  #get data from children of 'xmltop'
```

```
[1] "Bloodroot"
```

Example: Getting XML Data

```
xmlSApply(xmltop[[1]], xmlValue)  #get data from each child of XML nodes
```

```
plantcat = xmlSApply(xmltop, function(x) xmlSApply(x, xmlValue))
plantcat_df = data.frame(t(plantcat),row.names=NULL)
plantcat_df[1:5,1:4]
```

	COMMON	BOTANICAL	ZONE	LIGHT
1	Bloodroot	Sanguinaria canadensis	4	Mostly Shady
2	Columbine	Aquilegia canadensis	3	Mostly Shady
3	Marsh Marigold	Caltha palustris	4	Mostly Sunny
4	Cowslip	Caltha palustris	4	Mostly Shady
5	Dutchman's-Breeches	Dicentra cucullaria	3	Mostly Shady

Example: Getting HTML data

In the following exercise, we extract critic reviews for *The Shawshank Redemption.*

```
library(RCurl)
library(XML)
url = 'http://www.imdb.com/title/tt0111161/criticreviews?ref_=tt_ov_rt'
raw = getURL(url)
data = htmlParse(raw)
xpath = '//tr[@itemprop="reviews"]/td[2]/div'
nodes = getNodeSet(data, xpath)
text = sapply(nodes, xmlValue)
```

Other Data Resources

Government Open Data

- Data.gov
- Socrata
- Some cities have their own open data websites, like San Francisco: sfgov.org
- UN open data
- WHO open data
- US Census
- USGS

Economics and Finance

- OECD
- UMD
- World Bank
- CBOE Futures Exchange
- Google Finance
- Google Trends
- NASDAQ
- OANDA
- Yahoo Finance

Other Data Resources

- Programmable Web
- InfoChimps
- Google Public Data Explorer
- Junar
- The New York Times API
- The Guardian Data Blog