SML 201 – Week 4

John D. Storey

Spring 2016

Contents

R Packages **2**

Install These Packages 2

Manipulating Data Frames **3**

`dplyr` Package 3

Grammar of `dplyr` 3

Grammar of `dplyr` 3

Example: Baby Names 3

`babynames` Object 4

Peek at the Data 4

`%>%` Operator 5

`filter()` 5

`arrange()` 6

`arrange()` 7

`rename()` 7

`select()` 7

Renaming with `select()` 8

`mutate()` 8

No. Individuals by Year and Sex 9

`summarize()` 9

`group_by()` 10

No. Individuals by Year and Sex 10

How Many Distinct Names? 11

Most Popular Names 11

Most Popular Names 12

Most Popular Female Names 12
Most Popular Male Names 13
Additional Examples 15
Additional `dplyr` Features 15

Getting Data In and Out of R **16**
Some Functions for Data In/Out 16
Key Arguments 16
CSV Files 16
`airquality` Data: Out 17
`airquality` Data: In 17
`readr` Package 18
Scraping from the Web (Ex. 1) 18
Scraping from the Web (Ex. 2) 18
APIs 19

Exploratory Data Analysis **19**

Numerical Summaries of Data **19**

Data Visualization Basics **19**

Extras **19**
License 19
Source Code 19
Session Information 20

R Packages

Install These Packages

Run this code in RStudio and let us know if you experience any errors.

```
pkgs <- c("dplyr", "babynames", "readr", "tidyr", "reshape2",
          "hexbin", "ggplot2", "ggthemes", "broom",
          "devtools", "RColorBrewer", "rvest", "xtable")
install.packages(pkgs)
```

Manipulating Data Frames

`dplyr` Package

`dplyr` is a package with the following description:

> A fast, consistent tool for working with data frame like objects, both in memory and out of memory.

This package offers a "grammar" for manipulating data frames.

Everything that `dplyr` does can also be done using basic R commands – however, it tends to be much faster and easier to use `dplyr`.

Grammar of `dplyr`

Verbs:

- `filter`: extract a subset of rows from a data frame based on logical conditions
- `arrange`: reorder rows of a data frame
- `rename`: rename variables in a data frame
- `select`: return a subset of the columns of a data frame, using a flexible notation

Partially based on *R Programming for Data Science*

Grammar of `dplyr`

Verbs (continued):

- `mutate`: add new variables/columns or transform existing variables
- `distinct`: returns only the unique values in a table
- `summarize`: generate summary statistics of different variables in the data frame, possibly within strata
- `group_by`: breaks down a dataset into specified groups of rows

Partially based on *R Programming for Data Science*

Example: Baby Names

```
> library("dplyr", verbose=FALSE)
> library("babynames")
> ls()
character(0)
> babynames <- babynames::babynames
> ls()
[1] "babynames"
```

babynames Object

```
> class(babynames)
[1] "tbl_df"     "tbl"        "data.frame"
> dim(babynames)
[1] 1825433       5
```

```
> babynames
Source: local data frame [1,825,433 x 5]

    year   sex      name     n       prop
   (dbl) (chr)     (chr) (int)      (dbl)
1   1880     F      Mary  7065 0.07238359
2   1880     F      Anna  2604 0.02667896
3   1880     F      Emma  2003 0.02052149
4   1880     F Elizabeth  1939 0.01986579
5   1880     F    Minnie  1746 0.01788843
6   1880     F  Margaret  1578 0.01616720
7   1880     F       Ida  1472 0.01508119
8   1880     F     Alice  1414 0.01448696
9   1880     F    Bertha  1320 0.01352390
10  1880     F     Sarah  1288 0.01319605
..   ...   ...       ...   ...         ...
```

Peek at the Data

```
> set.seed(201)
> sample_n(babynames, 10)
Source: local data frame [10 x 5]

    year   sex    name     n         prop
   (dbl) (chr)   (chr) (int)        (dbl)
1   1991     M  Esaias     5 2.359700e-06
```

```
2   1933     F   Maida    33 3.155410e-05
3   1967     M   Alvis    33 1.853916e-05
4   1905     M Gaylord    11 7.679151e-05
5   1993     F Kyleigh   157 7.965969e-05
6   1927     M   Della     8 6.886519e-06
7   1908     F Luberta    12 3.384753e-05
8   1968     F  Andrea  7086 4.145300e-03
9   1921     F Ardelle    50 3.907288e-05
10  1955     M  Dainel     7 3.351657e-06
> # try also sample_frac(babynames, 6e-6)
```

%>% Operator

Originally from R package `magrittr`. Provides a mechanism for chaining commands with a forward-pipe operator, %>%.

```
> x <- 1:10
>
> x %>% log(base=10) %>% sum
[1] 6.559763
>
> sum(log(x,base=10))
[1] 6.559763
```

```
> babynames %>% sample_n(5)
Source: local data frame [5 x 5]

   year   sex    name     n         prop
  (dbl) (chr)   (chr) (int)        (dbl)
1  1979     M   Sunil    42 2.344364e-05
2  1996     F  Kelina     5 2.608857e-06
3  1991     F  Gimena     7 3.443326e-06
4  1979     M Neilson     9 5.023636e-06
5  1984     F Romelia     5 2.774045e-06
```

filter()

```
> filter(babynames, year==1880, sex=="F")
Source: local data frame [942 x 5]

    year   sex      name     n       prop
   (dbl) (chr)     (chr) (int)      (dbl)
```

```
1   1880     F      Mary  7065 0.07238359
2   1880     F      Anna  2604 0.02667896
3   1880     F      Emma  2003 0.02052149
4   1880     F Elizabeth  1939 0.01986579
5   1880     F    Minnie  1746 0.01788843
6   1880     F  Margaret  1578 0.01616720
7   1880     F       Ida  1472 0.01508119
8   1880     F     Alice  1414 0.01448696
9   1880     F    Bertha  1320 0.01352390
10  1880     F     Sarah  1288 0.01319605
..   ...   ...       ...   ...        ...
> # same as filter(babynames, year==1880 & sex=="F")
```

```
> filter(babynames, year==1880, sex=="F", n > 5000)
Source: local data frame [1 x 5]

   year   sex  name     n       prop
  (dbl) (chr) (chr) (int)      (dbl)
1  1880     F  Mary  7065 0.07238359
```

arrange()

```
> arrange(babynames, name, year, sex)
Source: local data frame [1,825,433 x 5]

    year   sex  name     n         prop
   (dbl) (chr) (chr) (int)        (dbl)
1   2007     M Aaban     5 2.260251e-06
2   2009     M Aaban     6 2.834029e-06
3   2010     M Aaban     9 4.390297e-06
4   2011     M Aaban    11 5.429927e-06
5   2012     M Aaban    11 5.440091e-06
6   2013     M Aaban    14 6.961721e-06
7   2014     M Aaban    16 7.882569e-06
8   2011     F Aabha     7 3.622491e-06
9   2012     F Aabha     5 2.587144e-06
10  2014     F Aabha     9 4.642684e-06
..   ...   ...   ...   ...          ...
```

arrange()

```
> arrange(babynames, desc(name), desc(year), sex)
Source: local data frame [1,825,433 x 5]

    year   sex      name     n         prop
   (dbl) (chr)     (chr) (int)        (dbl)
1   2010     M     Zzyzx     5 2.439054e-06
2   2014     M     Zyyon     6 2.955964e-06
3   2010     F   Zyyanna     6 3.067323e-06
4   2009     M    Zyvion     5 2.361691e-06
5   2010     M Zytavious     6 2.926865e-06
6   2009     M Zytavious     7 3.306368e-06
7   2007     M Zytavious     6 2.712301e-06
8   2006     M Zytavious     7 3.196664e-06
9   2005     M Zytavious     5 2.352830e-06
10  2004     M Zytavious     6 2.841628e-06
..   ...   ...       ...   ...          ...
```

rename()

```
> rename(babynames, number=n)
Source: local data frame [1,825,433 x 5]

    year   sex      name number       prop
   (dbl) (chr)     (chr)  (int)      (dbl)
1   1880     F      Mary   7065 0.07238359
2   1880     F      Anna   2604 0.02667896
3   1880     F      Emma   2003 0.02052149
4   1880     F Elizabeth   1939 0.01986579
5   1880     F    Minnie   1746 0.01788843
6   1880     F  Margaret   1578 0.01616720
7   1880     F       Ida   1472 0.01508119
8   1880     F     Alice   1414 0.01448696
9   1880     F    Bertha   1320 0.01352390
10  1880     F     Sarah   1288 0.01319605
..   ...   ...       ...    ...        ...
```

select()

```
> select(babynames, sex, name, n)
Source: local data frame [1,825,433 x 3]

     sex      name     n
   (chr)     (chr) (int)
1      F      Mary  7065
2      F      Anna  2604
3      F      Emma  2003
4      F Elizabeth  1939
5      F    Minnie  1746
6      F  Margaret  1578
7      F       Ida  1472
8      F     Alice  1414
9      F    Bertha  1320
10     F     Sarah  1288
..   ...       ...   ...
> # same as select(babynames, sex:n)
```

Renaming with `select()`

```
> select(babynames, sex, name, number=n)
Source: local data frame [1,825,433 x 3]

     sex      name number
   (chr)     (chr)  (int)
1      F      Mary   7065
2      F      Anna   2604
3      F      Emma   2003
4      F Elizabeth   1939
5      F    Minnie   1746
6      F  Margaret   1578
7      F       Ida   1472
8      F     Alice   1414
9      F    Bertha   1320
10     F     Sarah   1288
..   ...       ...    ...
```

`mutate()`

```
> mutate(babynames, total_by_year=round(n/prop))
Source: local data frame [1,825,433 x 6]
```

```
    year   sex      name     n       prop total_by_year
   (dbl) (chr)     (chr) (int)      (dbl)         (dbl)
1   1880     F      Mary  7065 0.07238359         97605
2   1880     F      Anna  2604 0.02667896         97605
3   1880     F      Emma  2003 0.02052149         97605
4   1880     F Elizabeth  1939 0.01986579         97605
5   1880     F    Minnie  1746 0.01788843         97605
6   1880     F  Margaret  1578 0.01616720         97605
7   1880     F       Ida  1472 0.01508119         97605
8   1880     F     Alice  1414 0.01448696         97605
9   1880     F    Bertha  1320 0.01352390         97605
10  1880     F     Sarah  1288 0.01319605         97605
..   ...   ...       ...   ...        ...           ...
> # see also transmutate
```

No. Individuals by Year and Sex

Let's put a few things together now adding the function `distinct()`...

```
> babynames %>% mutate(total_by_year=round(n/prop)) %>%
+   select(sex, year, total_by_year) %>% distinct()
Source: local data frame [270 x 3]

     sex  year total_by_year
   (chr) (dbl)         (dbl)
1      F  1880         97605
2      M  1880        118400
3      F  1881         98856
4      M  1881        108284
5      F  1882        115698
6      M  1882        122033
7      F  1883        120064
8      M  1883        112480
9      F  1884        137588
10     M  1884        122741
..   ...   ...           ...
```

summarize()

```
> summarize(babynames, mean_n = mean(n), median_n = median(n),
+           number_sex = n_distinct(sex),
+           distinct_names = n_distinct(name))
```

```
Source: local data frame [1 x 4]

    mean_n median_n number_sex distinct_names
     (dbl)    (int)      (int)          (int)
1 184.6879       12          2          93889
```

group_by()

```
> babynames %>% group_by(year, sex)
Source: local data frame [1,825,433 x 5]
Groups: year, sex [270]

    year   sex      name     n       prop
   (dbl) (chr)     (chr) (int)      (dbl)
1   1880     F      Mary  7065 0.07238359
2   1880     F      Anna  2604 0.02667896
3   1880     F      Emma  2003 0.02052149
4   1880     F Elizabeth  1939 0.01986579
5   1880     F    Minnie  1746 0.01788843
6   1880     F  Margaret  1578 0.01616720
7   1880     F       Ida  1472 0.01508119
8   1880     F     Alice  1414 0.01448696
9   1880     F    Bertha  1320 0.01352390
10  1880     F     Sarah  1288 0.01319605
..   ...   ...       ...   ...         ...
```

No. Individuals by Year and Sex

```
> babynames %>% group_by(year, sex) %>%
+   summarize(total_by_year=sum(n))
Source: local data frame [270 x 3]
Groups: year [?]

    year   sex total_by_year
   (dbl) (chr)         (int)
1   1880     F         90993
2   1880     M        110491
3   1881     F         91954
4   1881     M        100745
5   1882     F        107850
6   1882     M        113688
7   1883     F        112321
```

```
8   1883     M        104629
9   1884     F        129022
10  1884     M        114445
..   ...    ...           ...
```

Compare to earlier slide. Why the difference?

How Many Distinct Names?

```
> babynames %>% group_by(sex) %>%
+   summarize(mean_n = mean(n),
+             distinct_names_sex = n_distinct(name))
Source: local data frame [2 x 3]

    sex   mean_n distinct_names_sex
  (chr)    (dbl)              (int)
1     F 154.4542              64911
2     M 228.6588              39199
```

Most Popular Names

```
> top_names <- babynames %>% group_by(year, sex) %>%
+   summarize(top_name = name[which.max(n)])
>
> head(top_names)
Source: local data frame [6 x 3]
Groups: year [3]

   year   sex top_name
  (dbl) (chr)    (chr)
1  1880     F     Mary
2  1880     M     John
3  1881     F     Mary
4  1881     M     John
5  1882     F     Mary
6  1882     M     John
```

Most Popular Names

Recent Years

```
> tail(top_names, n=10)
Source: local data frame [10 x 3]
Groups: year [5]

    year   sex top_name
   (dbl) (chr)    (chr)
1   2010     F Isabella
2   2010     M    Jacob
3   2011     F   Sophia
4   2011     M    Jacob
5   2012     F   Sophia
6   2012     M    Jacob
7   2013     F   Sophia
8   2013     M     Noah
9   2014     F     Emma
10  2014     M     Noah
```

Most Popular Female Names

1990s

```
> top_names %>% filter(year >= 1990 & year < 2000, sex=="F")
Source: local data frame [10 x 3]
Groups: year [10]

    year   sex top_name
   (dbl) (chr)    (chr)
1   1990     F  Jessica
2   1991     F   Ashley
3   1992     F   Ashley
4   1993     F  Jessica
5   1994     F  Jessica
6   1995     F  Jessica
7   1996     F    Emily
8   1997     F    Emily
9   1998     F    Emily
10  1999     F    Emily
```

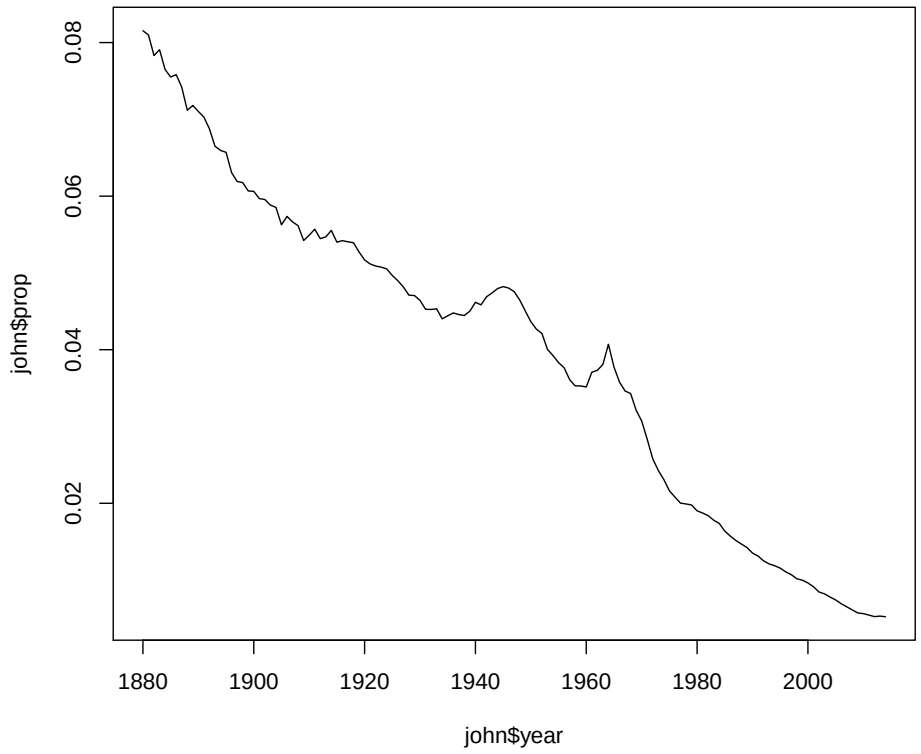
Most Popular Male Names

1990s

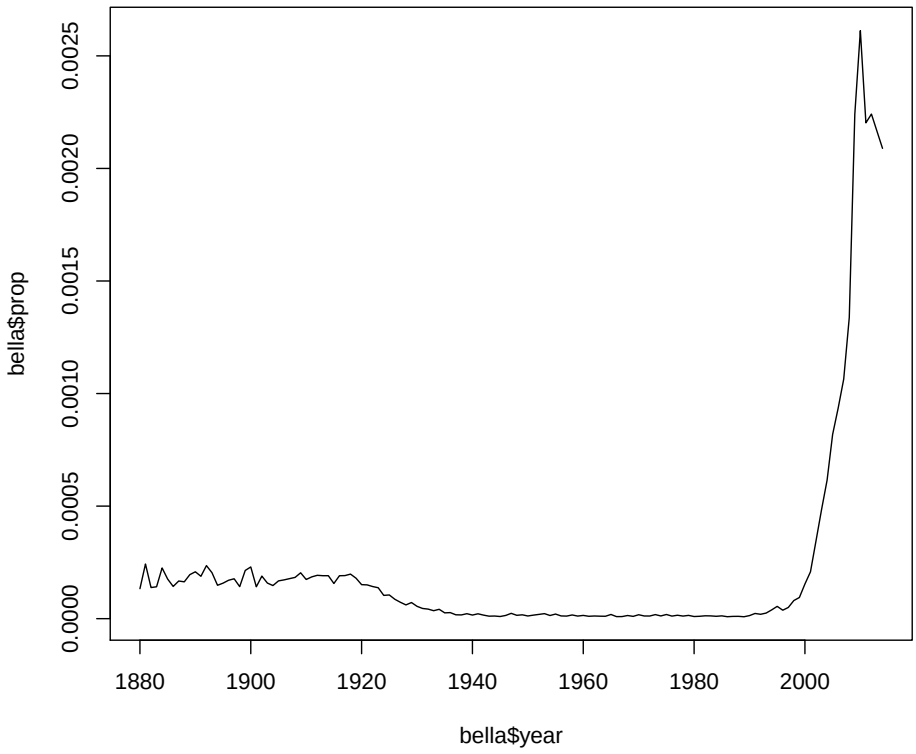
```
> top_names %>% filter(year >= 1990 & year < 2000, sex=="M")
Source: local data frame [10 x 3]
Groups: year [10]

    year   sex top_name
   (dbl) (chr)    (chr)
1   1990     M  Michael
2   1991     M  Michael
3   1992     M  Michael
4   1993     M  Michael
5   1994     M  Michael
6   1995     M  Michael
7   1996     M  Michael
8   1997     M  Michael
9   1998     M  Michael
10  1999     M    Jacob
```

```
> # Analyzing the name 'John'
> john <- babynames %>% filter(sex=="M", name=="John")
> plot(john$year, john$prop, type="l")
```




```
> # Analyzing the name 'Bella'
> bella <- babynames %>% filter(sex=="F", name=="Bella")
> plot(bella$year, bella$prop, type="l")
```



Additional Examples

You should study additional tutorials of `dplyr` that utilize other data sets:

- Read the `dplyr` introductory vignette
- Read the examples given in *R Programming for Data Science*, the "Managing Data Frames with the `dplyr` Package" chapter

Additional `dplyr` Features

- We've only scratched the surface – many interesting demos of `dplyr` can be found online
- `dplyr` can work with other data frame backends such as SQL databases
- There is an SQL interface for relational databases via the `DBI` package
- `dplyr` can be integrated with the `data.table` package for large fast tables
- There is a healthy rivalry between `dplyr` and `data.table`

Getting Data In and Out of R

Some Functions for Data In/Out

- We have already used the `load()` function to load `.Rdata` files; the `save()` function saves R objects to `.RData` files
- The function `read.table()` is a standard function for reading in data from a **text** file
- Similarly `write.table()` is a standard function for writing data to a **text** file
- Study the help files:

```
> ?read.table
> ?write.table
```

Key Arguments

For `read.table`:

- `file` – the name of a file, or a connection
- `header` – logical indicating if the file has a header line
- `sep` – character string indicating how the values are separated
- `colClasses` – character vector indicating the class of each column
- `nrows` – maximum number of rows to be read in
- `skip` – number of lines to skip from beginning of file
- `stringsAsFactors` – a logical indicating if character strings should be coerced to factors

There are similar arguments for `write.table`.

CSV Files

- A CSV file is a "comma separated value" file, meaning the entries are separated by commas
- The functions `read.csv()` and `write.csv()` are specialized versions of `read.table()` and `write.table()` where essentially it sets `sep=","`
- Many data sets are distributed as `.csv` files, so these are worth knowing about
- Read the help files, `?read.csv` and `?write.csv`

`airquality` Data: Out

Let's write the `airquality` data frame to a tab-delimited text file (aka TSV) and a CSV file.

```
> data("airquality", package="datasets")
> head(airquality, n=8)
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
4    18     313 11.5   62     5   4
5    NA      NA 14.3   56     5   5
6    28      NA 14.9   66     5   6
7    23     299  8.6   65     5   7
8    19      99 13.8   59     5   8
>
> write.table(airquality, file="../data/airquality.txt",
+             sep="\t", row.names=FALSE)
> write.csv(airquality, file="../data/airquality.csv",
+           row.names=FALSE)
```

`airquality` Data: In

We will read in the two files we wrote in the previous slide. We first look at the top couple lines of each file using `readLines` to understand what is in the files.

```
> readLines(con="../data/airquality.txt", n=2)
[1] "\"Ozone\"\t\"Solar.R\"\t\"Wind\"\t\"Temp\"\t\"Month\"\t\"Day\""
[2] "41\t190\t7.4\t67\t5\t1"
> aq1 <- read.table(file="../data/airquality.txt", header=TRUE,
+                   sep="\t")
>
> readLines(con="../data/airquality.csv", n=2)
[1] "\"Ozone\",\"Solar.R\",\"Wind\",\"Temp\",\"Month\",\"Day\""
[2] "41,190,7.4,67,5,1"
> aq2 <- read.csv(file="../data/airquality.csv", header=TRUE)
>
> dim(aq1) == dim(aq2)
[1] TRUE TRUE
> sum(aq1 != aq2, na.rm=TRUE)
[1] 0
```

`readr` Package

There are a number of R packages that provide more sophisticated tools for getting data in and out of R, especially as data sets have become larger and larger.

One of those packages is `readr`. It reads and writes data quickly, provides a useful status bar for large files, and does a good job at determining data types.

`readr` is organized similarly to the base R functions. For example, there are functions `read_table`, `read_csv`, `write_tsv`, and `write_csv`.

Scraping from the Web (Ex. 1)

There are several packages that facilitate "scraping" data from the web, including `rvest` demonstrated here.

```
> library("rvest")
> schedule <- read_html("http://sml201.github.io/schedule/")
> first_table <- html_table(schedule)[[1]]
> names(first_table) <- c("week", "topics", "reading")
> first_table[4,"week"]
[1] 4
> first_table[4,"topics"] %>% strsplit(split="  ")
[[1]]
[1] "Getting data in and out of R"
[2] "Exploring and visualizing data (part 1)"
> first_table[4,"reading"] %>% strsplit(split="  ")
[[1]]
[1] "ADS Ch. 4"           " EDAR, Ch. 3-9"
[3] "EDAS, Ch. 5, 10, 11" "RPDS, Ch. 6, 7"
> grep("EDAR", first_table$reading)
[1]  3  4  5 11
```

Scraping from the Web (Ex. 2)

The `rvest` documentation recommends SelectorGadget, which is "a javascript bookmarklet that allows you to interactively figure out what css selector you need to extract desired components from a page."

```
> usg_url <- "http://princetonusg.com/meet-your-usg-officers/"
> usg <- read_html(usg_url)
> officers <- html_nodes(usg, ".team-member-name") %>%
+             html_text
> head(officers, n=20)
```

```
 [1] "Ella Cheng"         "Aleksandra Czulak"
 [3] "Jeremy Burton"      "Hunter Dong"
 [5] "Sung Won Chang"     "Naimah Hakim"
 [7] "Jacob Cannon"       "Dallas Nan"
 [9] "Brandon McGhee"     "Christopher Hsu"
[11] "Ethan Marcus"       "Lavinia Liang"
[13] "Miranda Rosen"      "Shobhit Kumar"
[15] "Pooja Patel"        "Deana Davoudiasl"
[17] "Kristen Coke"       "Kishan Bhatt"
[19] "Cailin Hong"        "Paul Draper"
```

APIs

API stands for "application programming interface" which is a set of routines, protocols, and tools for building software and applications.

A specific website may provide an API for scraping data from that website.

There are R packages that provide an interface with specific APIs, such as the `twitteR` package.

Exploratory Data Analysis

Numerical Summaries of Data

Data Visualization Basics

Extras

License

https://github.com/SML201/lectures/blob/master/LICENSE.md

Source Code

https://github.com/SML201/lectures/tree/master/week4

Session Information

```
> sessionInfo()
R version 3.2.3 (2015-12-10)
Platform: x86_64-apple-darwin13.4.0 (64-bit)
Running under: OS X 10.11.3 (El Capitan)

locale:
[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods
[7] base

other attached packages:
[1] babynames_0.2.0 dplyr_0.4.3     rvest_0.3.1
[4] xml2_0.1.2      knitr_1.12.3    magrittr_1.5
[7] devtools_1.10.0

loaded via a namespace (and not attached):
 [1] Rcpp_0.12.3     R6_2.1.2        stringr_1.0.0
 [4] httr_1.1.0      highr_0.5.1     tools_3.2.3
 [7] parallel_3.2.3  DBI_0.3.1       selectr_0.2-3
[10] htmltools_0.3   yaml_2.1.13     lazyeval_0.1.10
[13] digest_0.6.9    assertthat_0.1  formatR_1.2.1
[16] curl_0.9.6      memoise_1.0.0   evaluate_0.8
[19] rmarkdown_0.9.2 stringi_1.0-1   XML_3.98-1.3
```