Intro to DS
Importing Data
Jeff Saltz and Jeff Stanton

School of Information Studies
SYRACUSE UNIVERSITY
ischool.syr.edu

Summary of Previous Learning

Here's what you should know and be able to do at this point in the course:

1. List major skills needed by data scientists
2. Describe a DS project with domain analysis and SMEs
3. Explain basic concepts of data modeling
4. Explain and use a data frame in R
5. Define the most common descriptive statistics and calculate them with R using appropriate functions
6. Demonstrate the development of a simple function in R
7. Describe the effects of randomness on sampling
8. Create and interpret a sampling distribution including defining the law of large numbers and the central limit theorem
9. Visualize a distribution and interpret a histogram

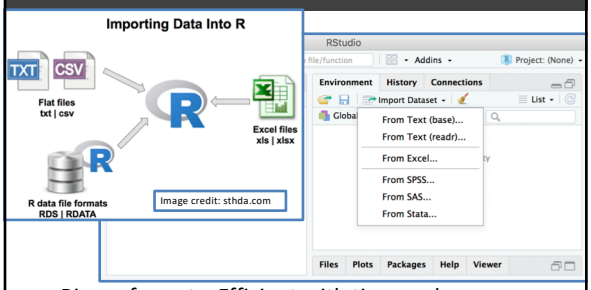
Importing Data

School of Information Studies
SYRACUSE UNIVERSITY
ischool.syr.edu

Objectives for this Session

- Recognizing different data types, data sets, and data bases as sources for R analysis; explaining their formats
- Building and executing relevant R code to import different data types, data sets, and accessing remote and/or local databases
  - Excel
  - JSON
  - Database

Binary vs Human Readable



- Binary formats: Efficient with time and space
- Human readable: Easy to diagnose and maintain

read.csv vs read_csv

read.csv

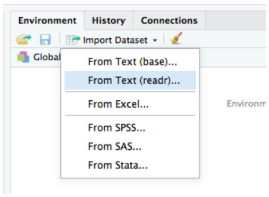
- Built into base R
- Slow with large datasets
- No diagnostics
- Produces a dataframe
- Converts char to factor
- Munges column names
- Inherits some default behaviors from OS

read_csv

- Part of the readr package
- 10x faster with large data
- Reports cols/parsing errors
- Produces a "tibble"
- Leaves char as char
- Does not change col names
- Works the same across operating systems

Conclusion: Use read_csv() from the tidyverse (readr) package whenever possible.

Environment Tab/From Text (readr)



Don't forget to copy the code:

- From: History or the console
- To: the source code file

→ so that your file import is included in your code (for homework, projects, and professional work)

Using readxl to Read an Excel File

```
#install.packages("readxl")
library("readxl")
library("tidyverse")

#Define the URL across multiple lines - make it easier to cut & paste code
part1 <-"http://www2.census.gov/programs-surveys/popest/tables/2010-2011/state/totals/"
part2 <- "nst-est2011-01.xls"
dataFile <- paste0(part1, part2)

#Download the file from the web, into tmpExcelFile
download.file(dataFile, "tmpExcelFile.xls")

#Now read the Excel file into R
testFrame <- read_excel("tmpExcelFile.xls")
```

Using Glimpse

glimpse(testFrame)

- Not that helpful (yet)

```
Rows: 66
Columns: 5
$ `table with row headers in column A and column headers in rows 3 through 4. (leading dots indicate sub-parts)` <chr> "Table 1. …
$ ...2                                                                                                           <chr> NA, "40269…
$ ...3                                                                                                           <chr> NA, NA, "E…
$ ...4                                                                                                           <chr> NA, "Popul…
$ ...5                                                                                                           <dbl> NA, NA, 20…
```

Take an Initial Look at the DataFrame

View(testFrame)

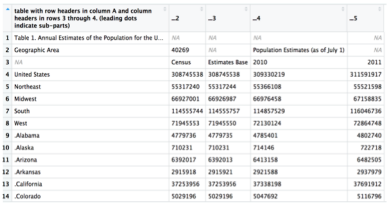
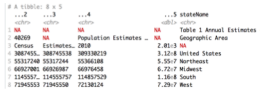


	table with row headers in column A and column headers in rows 3 through 4. (leading dots indicate sub-parts)	...2	...3	...4	...5
1	Table 1. Annual Estimates of the Population for the U...	NA	NA	NA	NA
2	Geographic Area	40269	NA	Population Estimates (as of July 1)	NA
3	NA	Census	Estimates Base	2010	2011
4	United States	308745538	308745538	309330219	311591917
5	Northeast	55317240	55317244	55366108	55521598
6	Midwest	66927001	66926987	66976458	67158835
7	South	114555744	114555757	114857529	116046736
8	West	71945553	71945550	72130124	72864748
9	Alabama	4779736	4779735	4785401	4802740
10	Alaska	710231	710231	714146	722718
11	Arizona	6392017	6392013	6413158	6482505
12	Arkansas	2915918	2915921	2921588	2937979
13	California	37253956	37253956	37338198	37691912
14	Colorado	5029196	5029196	5047692	5116796

Renaming the stateName Column

```
#Create a better column name
testFrame$stateName <- pull(testFrame, 1)
testFrame <- testFrame[,-1]

#Remove the '.'
testFrame$stateName <- str_replace(testFrame$stateName,"\\.","")
head(testFrame, 8)
```



```
# A tibble: 8 x 5
  ...2     ...3     ...4                  ...5 stateName
  <chr>    <chr>    <chr>                 <dbl> <chr>
1 NA       NA       NA                       NA Table 1. Annual Estimates
2 40269    NA       Population Estimates …   NA Geographic Area
3 Census   Estimates… 2010                  2.01e-3 NA
4 308745538 308745538 309330219            3.12e-4 United States
5 55317240 55317244 55366108               5.55e-7 Northeast
6 66927001 66926987 66976458               6.72e-7 Midwest
7 114555744 114555757 114857529            1.16e-6 South
8 71945553 71945550 72130124               7.29e-7 West
```

```
#Remove the first eight rows
testFrame < slice(testFrame, -1:-8)
```

Continue Cleaning the DataFrame (cont.)

```
#Remove the bottom lines
testFrame <- slice(testFrame, -52:-58)

#Rename the columns
testFrame <- testFrame %>% rename(april10census = '...2')
testFrame <- testFrame %>% rename(april10base = '...3')
testFrame <- testFrame %>% rename(july10pop = '...4')
testFrame <- testFrame %>% rename(july11pop = '...5')

#Make the populations numeric
testFrame <- testFrame %>%
      mutate_at(vars(april10census,april10base,july10pop,july11pop),
                as.numeric)
```

The Cleaned DataFrame

```
#Look at the first few rows
head(testFrame)
```

```
# A tibble: 6 x 5
  april10census april10base july10pop july11pop stateName
          <dbl>       <dbl>     <dbl>     <dbl> <chr>
1       4779736     4779735   4785401   4802740 Alabama
2        710231      710231    714146    722718 Alaska
3       6392017     6392013   6413158   6482505 Arizona
4       2915918     2915921   2921588   2937979 Arkansas
5      37253956    37253956  37338198  37691912 California
6       5029196     5029196   5047692   5116796 Colorado
```

Using group_by

```
#Remove the District of Columbia column
testFrame <- testFrame %>%  filter(stateName != "District of Columbia")

#Add the region for each state
testFrame <- testFrame %>% add_column(region=state.region)

#Calculate the region mean
testFrame %>% group_by(region) %>%
      summarise( regionMean = mean(july11pop),  .groups="drop")
```

```
# A tibble: 4 x 2
  region        regionMean
  <fct>              <dbl>
1 Northeast      6169066.
2 South          7214296.
3 North Central  5596570.
4 West           5604981.
```

Using group_by (cont.)

```
#Store the region mean as a new column
testFrame <-testFrame %>%  group_by(region) %>%
          mutate(regionMean = mean(july11pop))

#Look at the updated dataframe
head(testFrame, 2)
```

```
# A tibble: 2 x 7
# Groups:   region [2]
  april10census april10base july10pop july11pop stateName region regionMean
          <dbl>       <dbl>     <dbl>     <dbl> <chr>     <fct>       <dbl>
1       4779736     4779735   4785401   4802740 Alabama   South     7214296.
2        710231      710231    714146    722718 Alaska    West      5604981.
```

Question:

Why is storing data in spreadsheets (or related file formats such as CSV) sometimes not practical or not appropriate?

R & SQL

Did you Take IST659 or IST359?

- IST 359 - Introduction to Database Management Systems
- IST 659 - Data Administration Concepts and Database Management
- Not pre-requisites for this course: so brief review of SQL follows. . .

Structured Query Language

- *A Relational Model of Data for Large Shared Data Banks*, E. F. Codd (1970)
- "Relation" is a math term that essentially means a table with rows and columns
- Structured Query Language is a standard method of interacting with a relational database

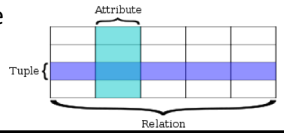


Image credit: Wikipedia

SQL Capabilities

- Select
- Update
- Insert into
- Delete
- Join

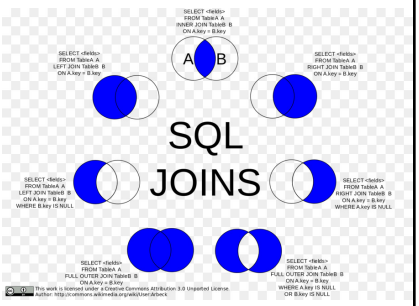


Image credit: Wikimedia (CC)

SQL Syntax

- SQL is very English-language-like: verbs, nouns, adjectives

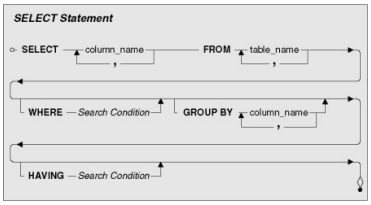
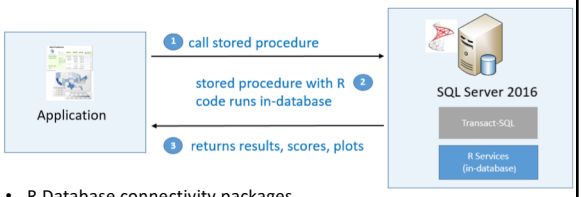


Image credit: TechTarget.com

Access to Data not Memory



- R Database connectivity packages
  - RMySQL
  - ROracle
  - RPostgresSQL
  - RSQlite
  - RMongo
  - RODBC
- Chapter examples
  - RODBC
    - MySQL
    - SQLServer 2012
    - Microsoft Access

Question

- Why might using SQL be useful?
- When R is accessing data using SQL (e.g., with MySQL), where do the data actually "live"?

Use sqldf for developing queries

sqldf supports access to a standard data frame using SQL

sqldf can connect to many different database types, including MySQL

If no DBs connected, defaults to memory-based SQL database

```
install.packages("sqldf")
library("sqldf")
sqldf('select mtcars.mpg from mtcars')
sqldf('select AVG(mtcars.mpg) from mtcars
      where cyl=4')
```

```
  AVG(mtcars.mpg)
1        26.66364
```

```
   mpg
1  21.0
2  21.0
3  22.8
4  21.4
5  18.7
6  18.1
7  14.3
8  24.4
9  22.8
10 19.2
11 17.8
12 16.4
13 17.3
14 15.2
15 10.4
16 10.4
17 14.7
18 32.4
19 30.4
20 33.9
21 21.5
22 15.5
23 15.2
24 13.3
25 19.2
26 27.3
27 26.0
28 30.4
29 15.8
30 19.7
31 15.0
32 21.4
>
```

Question

- When using sqldf, how is that different than accessing a SQL database?

JSON

Transactional Data Access

Remote applications as database "servers"

- Rationale
  - Data are too large to store in local memory
  - Data are too large to store on local disk
  - Privacy concerns limit the scope of results
  - Preference for analysis on current "official" source content vs. an archive copy
  - R is not designed to be a database manager

Common Vocabulary

- **HTML:** the code used to structure a web page and its content (Hypertext Markup Language)
- **JSON:** a human and machine readable data interchange format; defacto standard, not XML (JavaScript Object Notation)
- **XML:** a human and machine readable data interchange format; an international standard (eXtensible Markup Language)
- **SOAP:** a standardized method, using XML, for exchanging structured information among clients and servers (Simple object access protocol)
- **REST:** a software design style or strategy for organizing interactions between clients and servers; defacto standard (REpresentational State Transfer)

Example Architecture

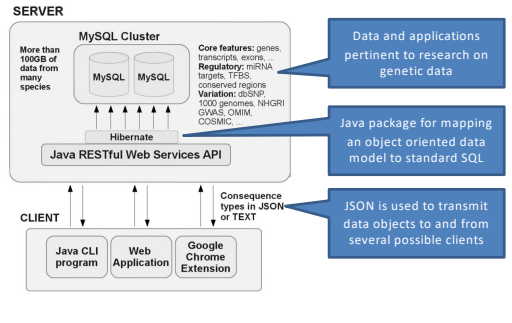
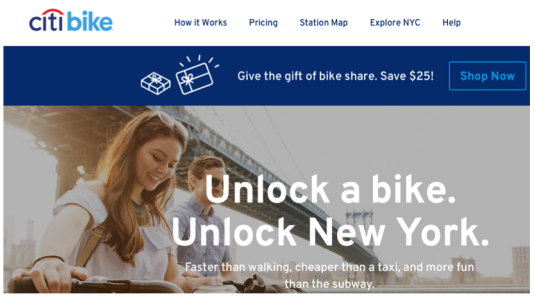


Image credit: Medina et al., 2012

JSON Data Access Example



JSON Data Access Example, Part I

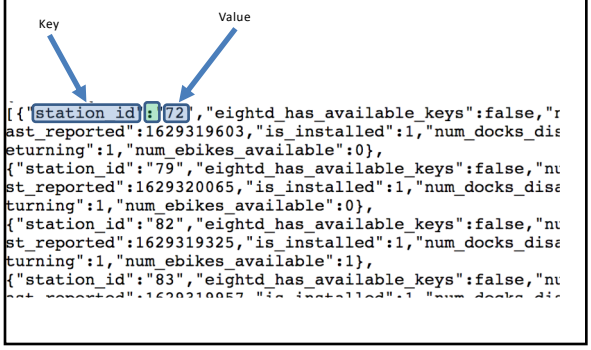
```
#For access to Internet data
library(RCurl)

#For decoding JSON
library(jsonlite)

#URL of JSON data
bikeURL <-
    'https://gbfs.citibikenyc.com/gbfs/en/station_status.json'
```

JSON Data – On the web

https://gbfs.citibikenyc.com/gbfs/en/station_status.json

JSON Data – On the web

https://gbfs.citibikenyc.com/gbfs/en/station_status.json



JSON Data Access Example, Part III

```
#Grab the JSON data
apiResult <- getURL(bikeURL)

#Parse the data
results <- fromJSON(apiResult)

#Look at our data
str(results)
```

```
List of 3
 $ data      :List of 1
  …$ stations:'data.frame':    1365 obs. of  13 variables:
```

Parsing JSON Data

```
#See the data generated
stations <- results$data$stations

glimpse(stations)
```

```
Rows: 1,367
Columns: 15
$ last_reported                <int> 1614275588, 1614275427, 1614276152, 1614274376, 1614274653, 1614276021, 1614275964, 161427…
$ station_id                   <chr> "72", "79", "82", "83", "116", "119", "120", "127", "128", "143", "144", "146", "150", "15…
$ num_bikes_disabled           <int> 3, 0, 2, 0, 3, 1, 1, 0, 3, 3, 1, 0, 0, 4, 1, 2, 1, 1, 3, 2, 3, 1, 1, 0, 1, 1, 0, 4, 2, 1, …
$ num_bikes_available          <int> 21, 14, 21, 34, 5, 17, 4, 4, 15, 8, 47, 19, 11, 25, 15, 39, 12, 20, 18, 45, 25, 5, 29, 3, …
$ station_status               <chr> "active", "active", "active", "active", "active", "active", "active", "active", "active", …
$ is_renting                   <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, …
$ num_docks_available          <int> 31, 19, 4, 28, 42, 35, 14, 27, 38, 13, 10, 20, 45, 4, 33, 22, 9, 14, 26, 0, 42, 29, 30, 34…
$ num_ebikes_available         <int> 0, 2, 0, 3, 0, 4, 2, 0, 7, 1, 2, 4, 1, 6, 1, 7, 1, 0, 2, 2, 1, 0, 0, 1, 4, 3, 2, 5, 1, 0, …
$ is_returning                 <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, …
$ num_docks_disabled           <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, …
$ legacy_id                    <chr> "72", "79", "82", "83", "116", "119", "120", "127", "128", "143", "144", "146", "150", "15…
$ eighthd_has_available_keys   <lgl> FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE,…
$ is_installed                 <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, …
$ eighthd_active_station_services <list> [NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL…
$ valet                        <df[,7]> <data.frame[44 x 7]>
```

Munging the Data

```
#Keep a subset of the columns
colsToKeep <- c('num_bikes_disabled','num_docks_disabled',
   'station_id', 'num_ebikes_available', 'num_bikes_available',
   'num_docks_available')

#Both of these commands do the same thing
stations1 <- stations[,colsToKeep]
stations2 <- stations %>% select(colsToKeep)

str(stations2)
```

```
'data.frame':   1367 obs. of  6 variables:
 $ num_bikes_disabled  : int  3 0 2 0 3 1 1 0 3 3 ...
 $ num_docks_disabled  : int  0 0 0 0 0 0 0 0 0 0 ...
 $ station_id          : chr  "72" "79" "82" "83" ...
 $ num_ebikes_available: int  0 2 0 3 0 4 2 0 7 1 ...
 $ num_bikes_available : int  21 14 21 34 5 17 4 4 15 8 ...
 $ num_docks_available : int  31 19 4 28 42 35 14 27 38 13 ...
```

Exploring the Parsed Data

```
#Keep a subset of the columns
mean(stations2$num_docks_available)
[1] 18.17557

mean(stations2$num_bikes_available)
[1] 10.71836

#How many stations have a bike available – could have used 'filter'
bikesAvailDF <- stations2[stations2$num_bikes_available>0,]
nrow(bikesAvailDF)
[1] 1255
```

Question

Why are these data available via JSON?

What would be some other good (and bad) alternatives for publishing these data?

Why did they make citibike data available at all?

Why JSON?

→JSON
- ✓ Easy to parse (easier than CSV file or XML data)
- ✓ Easy to update in real time

→Why make data available?
- ✓Let others develop tools