Introduction to Importing Data

Connecting R to External Data Sources

- R data import/export:
  - Range of methods for obtaining data from a wide variety of programs and formats
  - http://cran.r-project.org/doc/manuals/R-data.html
- Two threads:
  - Data in a discrete “flat” file format
  - Data in nondiscrete format, i.e., “system” oriented such as a relational database

R Data Import / Export

cran.r-project.org/doc/manuals/R-data.html#SQL-queries

R Data Import/Export

Table of Contents

R Supplied Data Sets

- R-supplied data sets—discrete files

> data()

```
Data sets in package 'datasets':

AirPassengers                   Monthly Airline Passenger Numbers 1949-1960
BJsales                         Sales Data with Leading Indicator
BJsales.lead (BJsales)          Sales Data with Leading Indicator
BOD                             Biochemical Oxygen Demand
CO2                             Carbon Dioxide Uptake in Grass Plants
ChickWeight                     Weight versus age of chicks on different diets
DNase                           Elisa assay of DNase
EuStockMarkets                  Daily Closing Prices of Major European Stock Indices, 1991-1998
Formaldehyde                    Determination of Formaldehyde
HairEyeColor                    Hair and Eye Color of Statistics Students
Harman23.cor                    Harman Example 2.3
Harman74.cor                    Harman Example 7.4
Indometh                        Pharmacokinetics of Indomethacin
InsectSprays                    Effectiveness of Insect Sprays
JohnsonJohnson                  Quarterly Earnings per Johnson & Johnson Share
```

R Supplied Data Sets - Example

- R-supplied data sets—discrete files

  > BOD # Biochemical Oxygen Demand

```
> BOD
  Time demand
1    1    8.3
2    2   10.3
3    3   19.0
4    4   16.0
5    5   15.6
6    7   19.8
>
```

```
> summary(BOD)
      Time           demand     
 Min.   :1.000   Min.   : 8.30  
 1st Qu.:2.250   1st Qu.:11.62  
 Median :3.500   Median :15.80  
 Mean   :3.667   Mean   :14.83  
 3rd Qu.:4.750   3rd Qu.:18.25  
 Max.   :7.000   Max.   :19.80  
> str(BOD)
'data.frame':   6 obs. of  2 variables:
 $ Time  : num  1 2 3 4 5 7
 $ demand: num  8.3 10.3 19 16 15.6 19.8
 - attr(*, "reference")= chr "A1.4, p. 270"
>
```

R Supplied Data Sets - Example

- R-supplied data sets—discrete files

  > help(BOD)

BOD {datasets}

Biochemical Oxygen Demand

Description

The BOD data frame has 6 rows and 2 columns giving the biochemical oxygen demand versus time in an evaluation of water quality.

Usage

```
BOD
```

Format

This data frame contains the following columns:

Time

A numeric vector giving the time of the measurement (days).

demand

A numeric vector giving the biochemical oxygen demand (mg/l).

Source

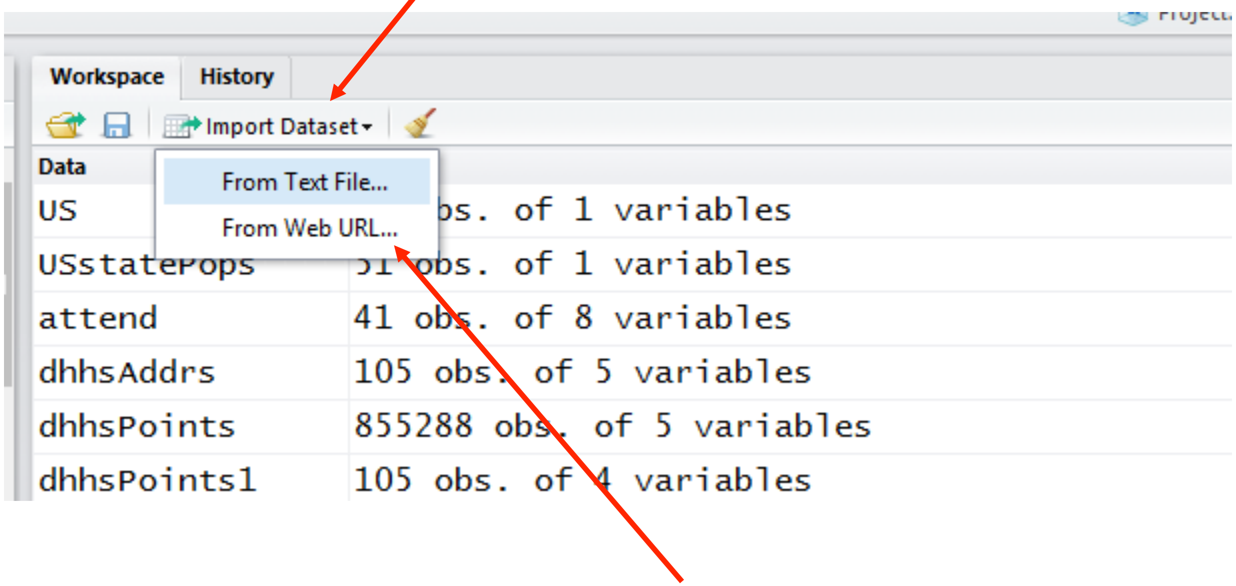
Bates, D.M. and Watts, D.G. (1988), *Nonlinear Regression Analysis and Its Applications*, Wiley, Appendix A1.4.

Accessing Discrete Files

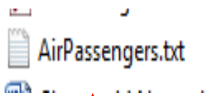
- Connecting R to external data sources—discrete files
  - Utilize RStudio: Import Dataset option
    - Tab delimited
    - Comma delimited
    - Decimal
  - Examples
    - Using RStudio import function on comma- and tab-delimited data sets

Import via RStudio

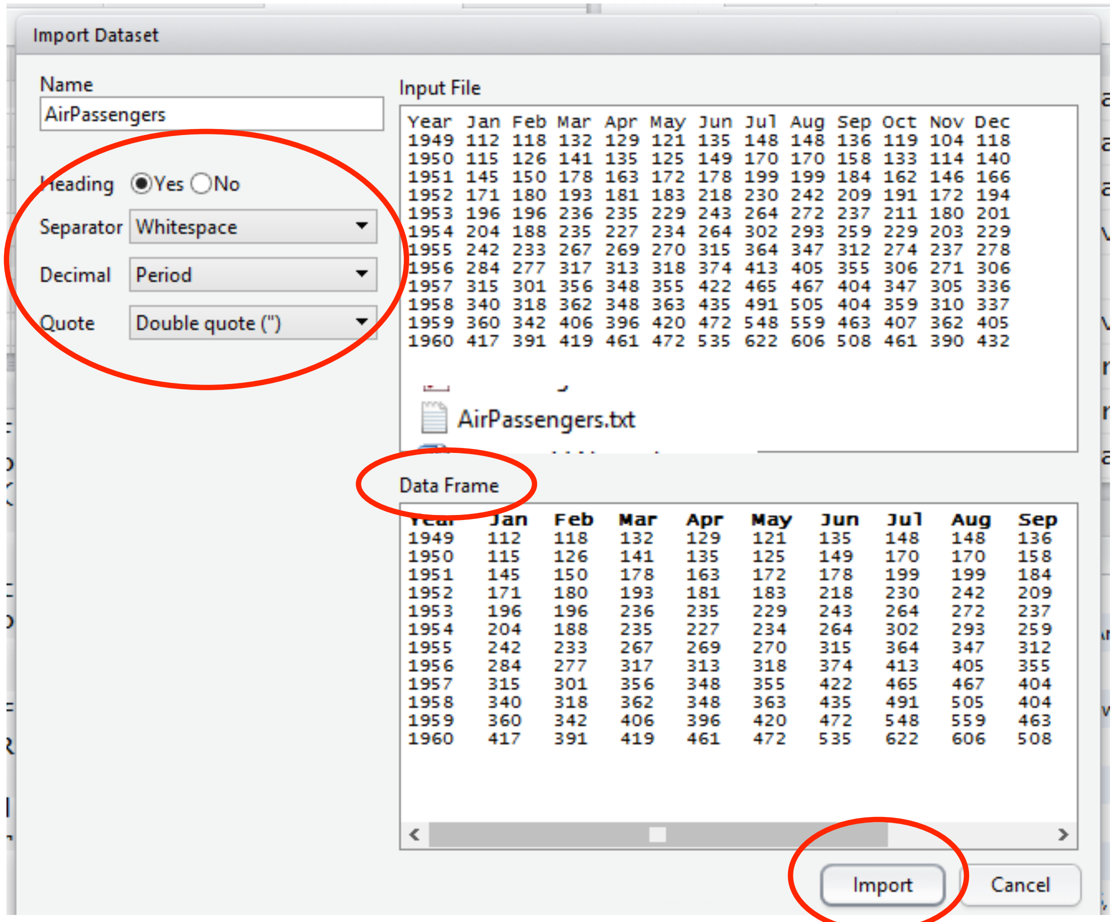
Upper right quadrant of RStudio



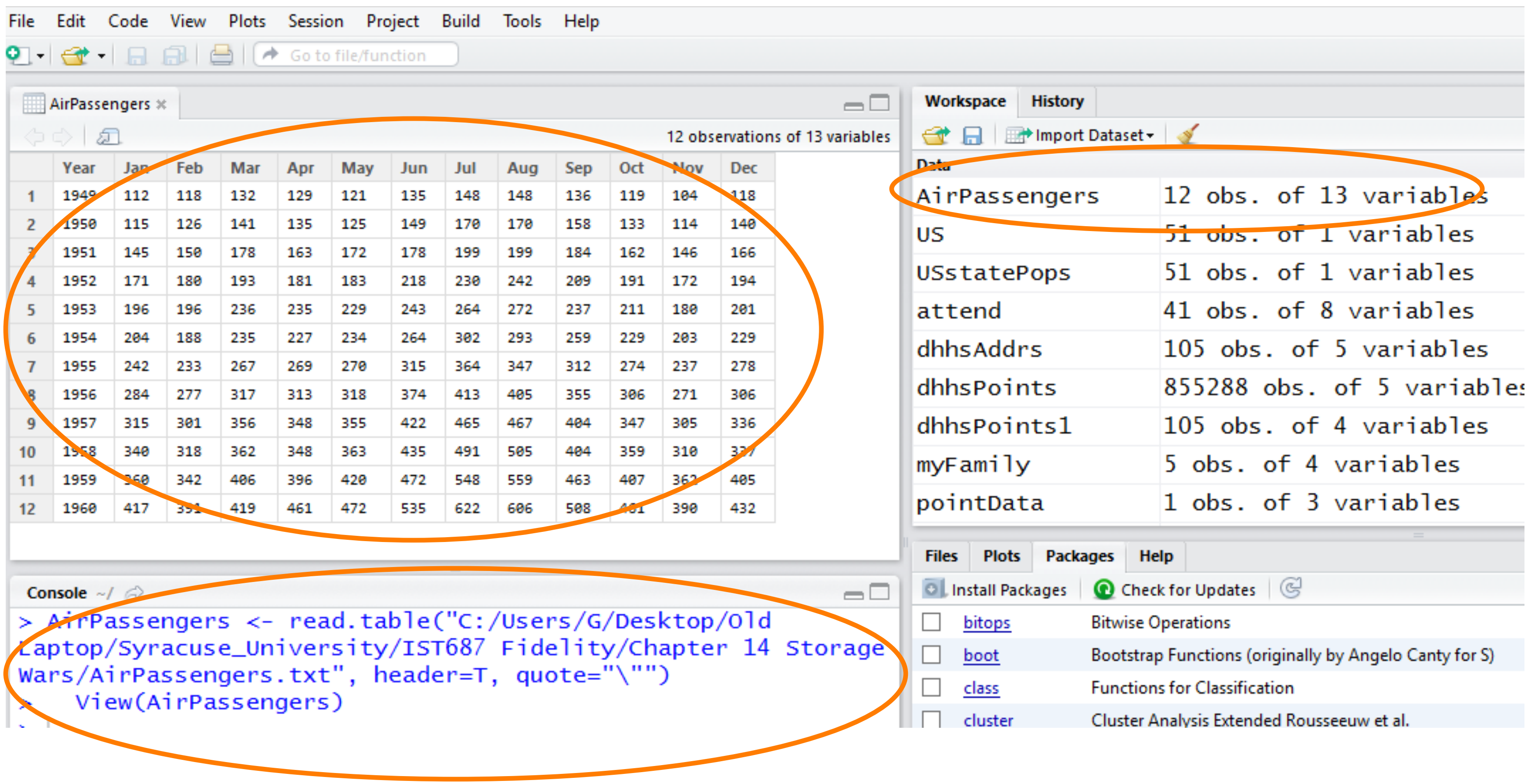
Import via Rstudio - Example



text file



Import via Rstudio - Example



Importing Spreadsheets

Connecting R to Discrete Files

- R packages
  - RODBC (Windows)
  - xlsReadWrite (Windows)
  - xlsx (Mac)
  - XLConnect (Mac)
  - gdata
    - read.xls function
    - http://cran.r-project.org/web/packages/gdata/gdata.pdf

    > ls("package:gdata")   #list content of gdata package

Connecting R to Discrete Files

- Read/load census data via read.xls (included in the R gdata package)
- One can look at the data at Census.gov before executing read.xls
- Get gdata package ready, then Read Data:

> **install.packages**(“gdata”)

> **library**(“gdata”)

> testFrame<-**read.xls**("http://www.census.gov/popest/data/state/totals/2011/tables/NST-EST2011-01.xls")

Example Data – From Census.gov

- http://www.census.gov/popest/data/state/totals/2011/tables/NST-EST2011-01.xls

Table 1. Annual Estimates of the Population for the United States, Regions, States, and Puerto Rico: April 1, 2010 to July 1, 2011

Geographic Area	April 1, 2010		Population Estimates (as of July 1)	
	Census	Estimates Base	2010	2011
United States	308,745,538	308,745,538	309,330,219	311,591,917
Northeast	55,317,240	55,317,244	55,366,108	55,521,598
Midwest	66,927,001	66,926,987	66,976,458	67,158,835
South	114,555,744	114,555,757	114,857,529	116,046,736
West	71,945,553	71,945,550	72,130,124	72,864,748
Alabama	4,779,736	4,779,735	4,785,401	4,802,740
Alaska	710,231	710,231	714,146	722,718
Arizona	6,392,017	6,392,013	6,413,158	6,482,505
Arkansas	2,915,918	2,915,921	2,921,588	2,937,979
California	37,253,956	37,253,956	37,338,198	37,691,912
Colorado	5,029,196	5,029,196	5,047,692	5,116,796
Connecticut	3,574,097	3,574,097	3,575,498	3,580,709
Delaware	897,934	897,934	899,792	907,135
District of Columbia	601,723	601,723	604,912	617,996
Florida	18,801,310	18,801,311	18,838,613	19,057,542
Georgia	9,687,653	9,687,660	9,712,157	9,815,210
Hawaii	1,360,301	1,360,301	1,363,359	1,374,810
Idaho	1,567,582	1,567,582	1,571,102	1,584,985
Illinois	12,830,632	12,830,632	12,841,980	12,869,257
Indiana	6,483,802	6,483,800	6,490,622	6,516,922
Iowa	3,046,355	3,046,350	3,050,202	3,062,309
Kansas	2,853,118	2,853,118	2,859,143	2,871,238
Kentucky	4,339,367	4,339,362	4,347,223	4,369,356
Louisiana	4,533,372	4,533,372	4,545,343	4,574,836

NST01

Viewing the testFrame

> View(testFrame)   # View the results of **read.xls**

table.with.row.headers.in.column.A.and.column.headers.in.rows.3.through.4...leading.dots.in	X	X.1	X.2	X.3	X.4	X.5	X.6	X.7	X.8
Table 1. Annual Estimates of the Population for the United States, Regions, State					NA	NA	NA	NA	NA
Geographic Area	April 1, 2010		Population Estimates (as of July 1)		NA	NA	NA	NA	NA
	Census	Estimates Base	2010	2011	NA	NA	NA	NA	NA
United States	308,745,538	308,745,538	309,330,219	311,591,917	NA	NA	NA	NA	NA
Northeast	55,317,240	55,317,244	55,366,108	55,521,598	NA	NA	NA	NA	NA
Midwest	66,927,001	66,926,987	66,976,458	67,158,835	NA	NA	NA	NA	NA
South	114,555,744	114,555,757	114,857,529	116,046,736	NA	NA	NA	NA	NA
West	71,945,553	71,945,550	72,130,124	72,864,748	NA	NA	NA	NA	NA
.Alabama	4,779,736	4,779,735	4,785,401	4,802,740	NA	NA	NA	NA	NA
.Alaska	710,231	710,231	714,146	722,718	NA	NA	NA	NA	NA
.Arizona	6,392,017	6,392,013	6,413,158	6,482,505	NA	NA	NA	NA	NA
.Arkansas	2,915,918	2,915,921	2,921,588	2,937,979	NA	NA	NA	NA	NA
.California	37,253,956	37,253,956	37,338,198	37,691,912	NA	NA	NA	NA	NA
.Colorado	5,029,196	5,029,196	5,047,692	5,116,796	NA	NA	NA	NA	NA
.Connecticut	3,574,097	3,574,097	3,575,498	3,580,709	NA	NA	NA	NA	NA
.Delaware	897,934	897,934	899,792	907,135	NA	NA	NA	NA	NA
.District of Columbia	601,723	601,723	604,912	617,996	NA	NA	NA	NA	NA
.Florida	18,801,310	18,801,311	18,838,613	19,057,542	NA	NA	NA	NA	NA
.Georgia	9,687,653	9,687,660	9,712,157	9,815,210	NA	NA	NA	NA	NA
.Hawaii	1,360,301	1,360,301	1,363,359	1,374,810	NA	NA	NA	NA	NA
.Idaho	1,567,582	1,567,582	1,571,102	1,584,985	NA	NA	NA	NA	NA
.Illinois	12,830,632	12,830,632	12,841,980	12,869,257	NA	NA	NA	NA	NA
.Indiana	6,483,802	6,483,800	6,490,622	6,516,922	NA	NA	NA	NA	NA
.Iowa									

Looking at the Structure of testFrame

> str(testFrame)   # structure of testFrame

```
> str(testFrame)
'data.frame':	65 obs. of  10 variables:
 $ table.with.row.headers.in.column.A.and.column.headers.in.rows.3.through.4...leading.dots.indicate.sub.parts.: Factor w/ 65 levels
"",".Alabama",..: 62 53 1 64 55 54 60 65 2 3 ...
 $ X                                                                                                            : Factor w/ 60 levels
"","1,052,567",..: 1 59 60 27 38 47 10 49 32 50 ...
 $ X.1                                                                                                          : Factor w/ 59 levels
"","1,052,567",..: 1 1 59 27 38 47 10 49 32 50 ...
 $ X.2                                                                                                          : Factor w/ 60 levels
"","1,052,528",..: 1 60 21 28 39 48 10 51 33 50 ...
 $ X.3                                                                                                          : Factor w/ 59 levels
"","1,051,302",..: 1 1 21 28 38 48 10 50 33 51 ...
 $ X.4                                                                                                          : logi  NA NA NA NA
```

Analyzing What was Read Into R

- View the results of read.xls using **View**(testFrame)
- **Compare** source data from Census.gov to what has been read into R - testFrame
- Use the structure function **str**(testFrame) to provide summary statistics about the data frame testFrame
- Key takeaways about testFrame
  - Variable names are not clear
  - Variable columns are of no use, i.e., x.4
  - Data is in character string format vs. numeric
  - The data set/data frame needs to be "cleansed" and "transformed" before starting any R analysis

The Cleansing and Transformation Process

- **Cleansing**
  - Remove header rows.
  - Remove unneeded columns.
  - Remove last few rows.
  - Copy first column to a column with a good name.
  - Remove first column.

The Cleansing and Transformation Process

- **Transformation**
  - Remove dots on front of state names.
  - Convert "factor"/character data to numeric via a custom developed function… Numberize.

- Recommend viewing "testFrame" at various cleansing and transformation steps to see the affect of the R statement

Cleansing Example

```
# remove 1st 3 rows,,,  column parameter empty
testFrame<-testFrame[-1:-3,]

# keep 1st 5 columns,,, row parameter empty
testFrame<-testFrame[,1:5]

# Look at the last 5 rows of testFrame
tail(testFrame,5)

# remove last 5 rows
testFrame<-testFrame[-58:-62,]

# view testFrame post Cleansing
testFrame
```

Transformation Example

```
testFrame$region <- testFrame[,1]    # give 1st Column a name .. region
testFrame<-testFrame[,-1]
testFrame$region <- str_replace(testFrame$region,"\\.","") # remove dots in front of state name
#
# Numberize() - Gets rid of commas and other junk and converts to numbers
# Assumes that the inputVector is a list of data that can be treated as character strings
Numberize <- function(inputVector)
{
  inputVector<-str_replace_all(inputVector,",","")      # remove commas
  inputVector<-str_replace_all(inputVector," ","")      # remove spaces
  return(as.numeric(inputVector))
}
#
#     Apply Numberize function to columns in testFrame and give columns a new name
testFrame$april10census <-Numberize(testFrame$X)
testFrame$april10base <-Numberize(testFrame$X.1)
testFrame$july10pop <-Numberize(testFrame$X.2)
testFrame$july11pop <-Numberize(testFrame$X.3)
testFrame                                        # look at testFame post transformation
```

Viewing the updated testFrame

> View(testFrame)

	row.names	X	X.1	X.2	X.3	region	april10census	april10base	july10pop	july11pop
1	4	308,745,538	308,745,538	309,330,219	311,591,917	United States	308745538	308745538	309330219	311591917
2	5	55,317,240	55,317,244	55,366,108	55,521,598	Northeast	55317240	55317244	55366108	55521598
3	6	66,927,001	66,926,987	66,976,458	67,158,835	Midwest	66927001	66926987	66976458	67158835
4	7	114,555,744	114,555,757	114,857,529	116,046,736	South	114555744	114555757	114857529	116046736
5	8	71,945,553	71,945,550	72,130,124	72,864,748	West	71945553	71945550	72130124	72864748
6	9	4,779,736	4,779,735	4,785,401	4,802,740	Alabama	4779736	4779735	4785401	4802740
7	10	710,231	710,231	714,146	722,718	Alaska	710231	710231	714146	722718
8	11	6,392,017	6,392,013	6,413,158	6,482,505	Arizona	6392017	6392013	6413158	6482505
9	12	2,915,918	2,915,921	2,921,588	2,937,979	Arkansas	2915918	2915921	2921588	2937979
10	13	37,253,956	37,253,956	37,338,198	37,691,912	California	37253956	37253956	37338198	37691912
11	14	5,029,196	5,029,196	5,047,692	5,116,796	Colorado	5029196	5029196	5047692	5116796
12	15	3,574,097	3,574,097	3,575,498	3,580,709	Connecticut	3574097	3574097	3575498	3580709
13	16	897,934	897,934	899,792	907,135	Delaware	897934	897934	899792	907135
14	17	601,723	601,723	604,912	617,996	District of Columbia	601723	601723	604912	617996
15	18	18,801,310	18,801,311	18,838,613	19,057,542	Florida	18801310	18801311	18838613	19057542
16	19	9,687,653	9,687,660	9,712,157	9,815,210	Georgia	9687653	9687660	9712157	9815210

Question:

Why are reading spreadsheets (or other files such as CSV) sometimes not practical / appropriate?

R & SQL

Nondiscrete data access

- Database connectivity packages
  - RMySQL
  - ROracle
  - RPostgresSQL
  - RSQlite
  - RMongo
  - RODBC

- Chapter examples
  - RODBC
    - MySQL
    - SQLServer 2012
    - Microsoft Access

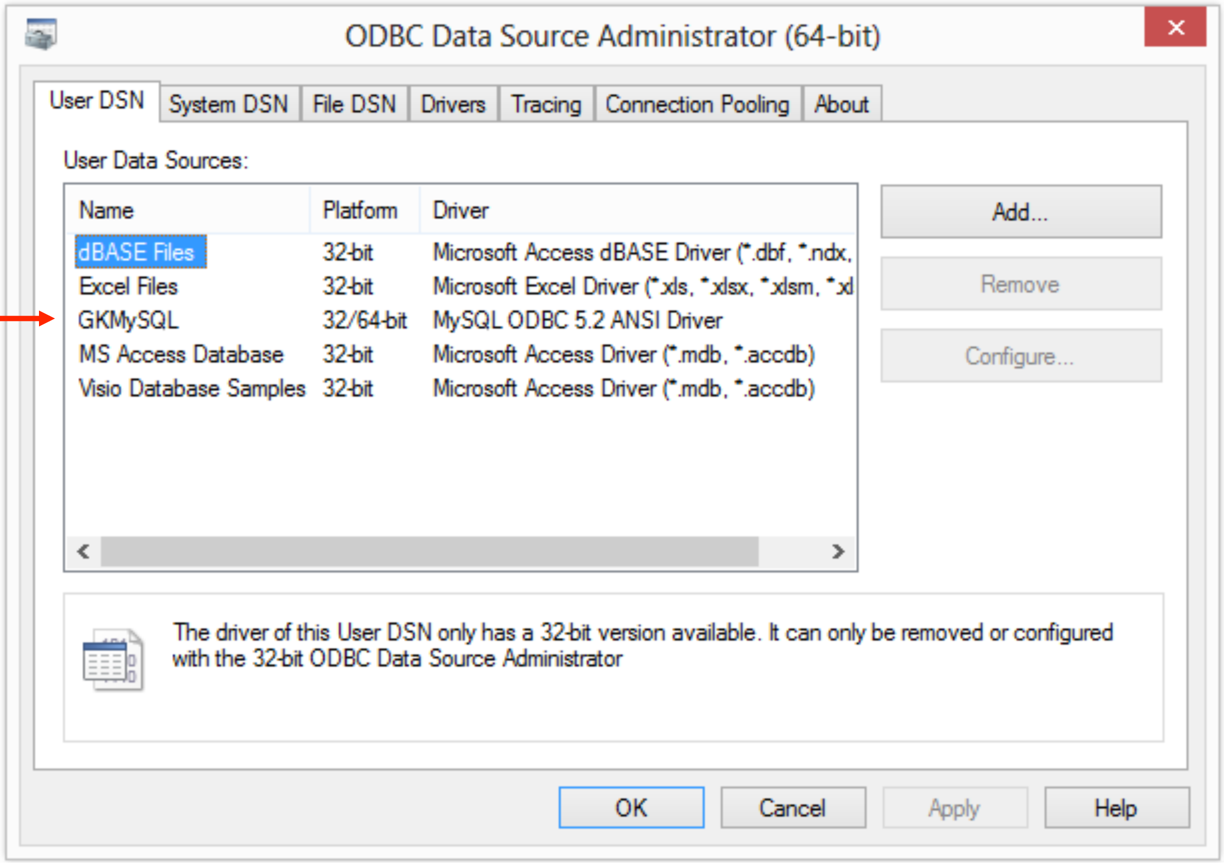
Environment Prerequisites (non R Activity)

- MySQL
  - Download/install MySQL.
  - Download/install Northwind database in your MySQL instance.
  - Configure ODBC for your MySQL instance.
- SQL Server 2012
  - Download/install SQL Server 2012.
  - Download/install Northwind database in your SQL Server instance.
  - Configure ODBC for your SQL Server instance.
- Microsoft Access
  - Download/install MS Access Northwind DB from IST687 course Blackboard: Resources -> Course Library -> Data Sets, Databases
  - Two versions of Northwind Access DB: .mdb .accdb
  - You can choose either.

Environment Prerequisites - MySQL

- Environment prerequisites (non R activity) MySQL

ODBC name will be used in R ODBC connect statement.



Environment Prerequisites - MySQL

Note: RODBC package must be loaded.

- MySQL R code

MySQL ODBC name

```
# establish R connection to GKMySQL
>conmysql <- odbcConnect("GKMySQL")
# assign SQL table list
>tblsmysql<-sqlTables(conmysql)
# View Northwind tables
>tblsmysql
# assign SQL Query script to datamysql
>datamysql<-sqlQuery(conmysql,paste("select * from
Products"))
```

Looking at datamysql

```
> datamysql
   ProductID                       ProductName SupplierID CategoryID      QuantityPerUnit UnitPrice
1          1                              Chai          1          1   10 boxes x 20 bags     18.00
2          2                             Chang          1          1   24 - 12 oz bottles     19.00
3          3                     Aniseed Syrup          1          2  12 - 550 ml bottles     10.00
4          4      Chef Anton's Cajun Seasoning          2          2       48 - 6 oz jars     22.00
5          5            Chef Anton's Gumbo Mix          2          2             36 boxes     21.35
6          6      Grandma's Boysenberry Spread          3          2       12 - 8 oz jars     25.00
7          7   Uncle Bob's Organic Dried Pears          3          7      12 - 1 lb pkgs.     30.00
8          8        Northwoods Cranberry Sauce          3          2      12 - 12 oz jars     40.00
9          9                   Mishi Kobe Niku          4          6     18 - 500 g pkgs.     97.00
```

```
UnitsInStock UnitsOnOrder ReorderLevel Discontinued
          39            0           10
          17           40           25
          13           70           25
          53            0            0
           0            0            0         \001
         120            0           25
          15            0           10
           6            0            0
          29            0            0         \001
          31            0            0
          22           30           30
          86            0            0
          24            0            5
```

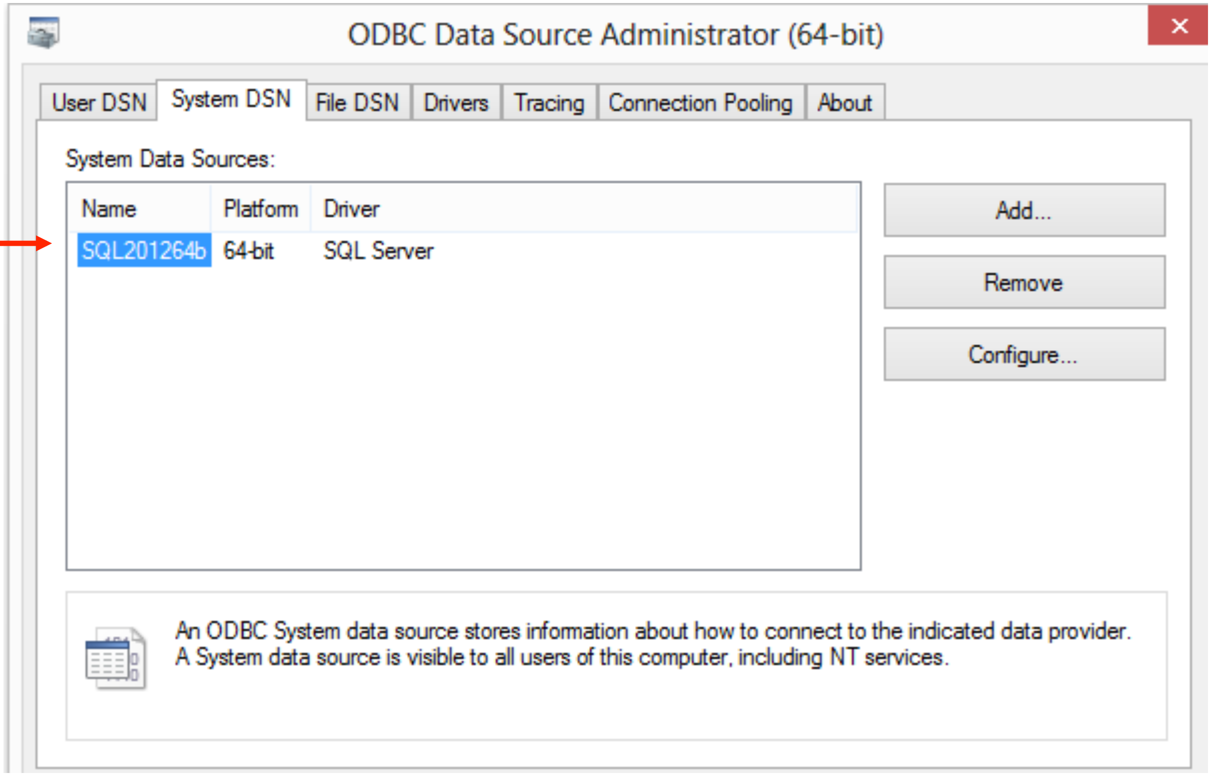
View (datamysql)

ProductID	ProductName	SupplierID	CategoryID	QuantityPerUnit	UnitPrice	UnitsInStock	UnitsOnOrder	ReorderLevel	Discontinued
1	Chai	1	1	10 boxes x 20 bags	18.00	39	0	10	
2	Chang	1	1	24 - 12 oz bottles	19.00	17	40	25	
3	Aniseed Syrup	1	2	12 - 550 ml bottles	10.00	13	70	25	
4	Chef Anton's Cajun Seasoning	2	2	48 - 6 oz jars	22.00	53	0	0	
5	Chef Anton's Gumbo Mix	2	2	36 boxes	21.35	0	0	0	
6	Grandma's Boysenberry Spread	3	2	12 - 8 oz jars	25.00	120	0	25	
7	Uncle Bob's Organic Dried Pears	3	7	12 - 1 lb pkgs.	30.00	15	0	10	
8	Northwoods Cranberry Sauce	3	2	12 - 12 oz jars	40.00	6	0	0	
9	Mishi Kobe Niku	4	6	18 - 500 g pkgs.	97.00	29	0	0	
10	Ikura	4	8	12 - 200 ml jars	31.00	31	0	0	
11	Queso Cabrales	5	4	1 kg pkg.	21.00	22	30	30	
12	Queso Manchego La Pastora	5	4	10 - 500 g pkgs.	38.00	86	0	0	
13	Konbu	6	8	2 kg box	6.00	24	0	5	
14	Tofu	6	7	40 - 100 g pkgs.	23.25	35	0	0	
15	Genen Shouyu	6	2	24 - 250 ml bottles	15.50	39	0	5	
16	Pavlova	7	3	32 - 500 g boxes	17.45	29	0	10	
17	Alice Mutton	7	6	20 - 1 kg tins	39.00	0	0	0	

Note: ‘view’ command shows formatted results in script window in Rstudio

Environment Prerequisites - SQLServer

ODBC name will be used in R ODBC connect statement.



SQLServer R Code

Note: RODBC package must be loaded

SQLServer ODBC name

```
# establish R connection to SQL201264b
>conSQL2012 <- odbcConnect(“SQL201264b")
# assign SQL table list
>tblsSQL2012<-sqlTables(conSQL2012)
# View Northwind tables
>tblsSQL2012
# assign SQL Query script   to dataSQL2012
>dataSQL2012<-sqlQuery(conSQL2012,paste("select * from
Products"))
# view output of SQL select
>dataSQL2012
>View(dataSQL2012)
```

SQLDF R Package

```
install.packages("sqldf")
library("sqldf")

sqldf('select mtcars.mpg from mtcars’)
```

```
   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
>
```

SQLDF R Package

- sqldf('select AVG(mtcars.mpg) from mtcars where cyl=4')

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

SAPPLY R Function

sapply(Variable, Function, optional parameters)

#Get the mean for each column in mtcars
sapply(mtcars, mean)

```
mpg        cyl       disp       hp         drat      wt        qsec       vs        am
20.090625  6.187500 230.721875 146.687500  3.596563  3.217250  17.848750  0.437500  0.406250
gear      carb
3.687500  2.812500
```

TAPPLY Function in R

```
# tapply(Summary Variable, Group Variable, Function)

# get the mean MPG for each CYL
tapply(mtcars$mpg, mtcars$cyl, mean)
       4        6        8
26.66364 19.74286 15.10000

#Use my own function (not mean)
tapply(mtcars$mpg, mtcars$cyl, meanPlusSD)

meanPlusSD <- function(v){
  t <- mean(v) + sd(v)
  return(t)
}
```

JSON

Nondiscrete Data Access

- What is nondiscrete data access?
  - Remote applications are database “servers”

- Rationale
  - Data is too large to store in local memory
  - Data is too large to store on local disk
  - Can’t make copies of large “system” databases
  - Preference that analysis is always on current “official” source content vs. a copy
  - R is not designed to be a database manager

Example: Google Geocode API

- Evaluate/test Google geocode api
- http://maps.googleapis.com/maps/api/geocode/json?address=1600+Pennsylvania+Avenue,+Washington,+DC&sensor=false
- Sample output on next slide

Example: Google Geocode API


```
   ],
   "formatted_address" : "1600 Pennsylvania Avenue Northwest, President's Park, Washington, DC 20500, USA",
   "geometry" : {
      "location" : {
         "lat" : 38.8978378,
         "lng" : -77.0365123
      },
      "location_type" : "ROOFTOP",
      "viewport" : {
         "northeast" : {
            "lat" : 38.89918678029149,
            "lng" : -77.03516331970849
         },
         "southwest" : {
            "lat" : 38.89648881970849,
            "lng" : -77.03786128029151
         }
      }
   },
   "partial_match" : true,
   "types" : [ "street_address" ]
}
```


JSON: Java Script Object Notation

Create a ‘MakeGeoURL’ Function

- Create a function to accept an ‘address’ argument, insert it into the Google geocode URL

```
MakeGeoURL <- function(address)
{
root <- "http://maps.google.com/maps/api/geocode/"
url <- paste(root, "json?address=",address, "&sensor=false", sep = "")
return(URLencode(url))
}
```

Testing MakeGeo URL

MakeGeoURL("1600 Pennsylvania Avenue,Washington, DC")

function

argument

"http://maps.google.com/maps/api/geocode/json?address=1600%20Pennsylvania%20Avenue,%20Washington,%20DC&sensor=false"

Function results

Using MakeGeo URL

- Create a function to
  - Take result of MakeGeoURL
  - Send it to the Internet via getURL and receive results
  - Isolate results (latitude, longitude coordinates) via fromJSON

Creating an 'Addr2latlng' Function

```
Addr2latlng <- function(address)
{
url <- MakeGeoURL(address)                # set up URL string

apiResult <- getURL(url)                  # send URL to internet

geoStruct <- fromJSON(apiResult, simplify = FALSE)
lat <- NA
lng <- NA
try(lat <- geoStruct$results[[1]]$geometry$location$lat
try(lng <- geoStruct$results[[1]]$geometry$location$lng
return(c(lat, lng))
}
```

Using Addr2latlng

```
>testData <- Addr2latlng( "1600 Pennsylvania Avenue, Washington, DC")

# See latitude and longitude coordinates
> str(testData)
num [1:2] 38.9 -77
```

Accessing Different JSON data

```
#An Example using citibike data from NYC
> bikeURL <-
    “https://www.citibikenyc.com/stations/json”

> apiResult <- getURL(bikeURL)
> Results <- fromJSON(apiResult)
> length(results)
[1] 2
```

Parsing JSON Data

```
#See when the data was generated
> when <- results[[1]]
> when [1]
"2016-01-03 11:56:40 AM"

#The next results is actually a list of stations
> stations <-results[[2]]
> length(stations)
[1] 508
```

Detailed Structure of the Data

```
> str(station[[1]])
List of 18

$id:                    num   72
$stationName:           chr ”W52 St & 11 Ave"
$availableDocks:        num 5
$totalDocks:            num 39
$latitude:              num 40.8
$longitude:             num -74
$statusValue:           chr  "In Service"
$statusKey:             num 1
$availableBikes:        num 34
$stAddress1:            chr ”W 52 St & 11 Ave"
$stAddress2:            chr "”
$city:                  chr ""
$postalCode:            chr ""
$location:              chr ""
$altitude:              chr  ""
$testStation:           logi FALSE
$lastCommunicationTime: chr "2016-01-03 11:53:24 AM"
$landMark:              chr ""
```

Converting from a List to Dataframe

```
#get size and names
>  numRows <- length(stations)
> nameList <- names(stations[[1]])

> dfStations <-data.frame(matrix(unlist(stations),
                   nrow=numRows,  byrow=T),
                   stringsAsFactors=FALSE)

#Finally, we need to name the columns :
> names(dfStations) <-nameList
```

Clean Up Dataframe

```
> df$availableDocks <-
        as.numeric(df$availableDocks)
> df$availableBikes <-
        as.numeric(df$availableBikes)
> df$totalDocks <- as.numeric(df$totalDocks)

> mean(df$availableDocks)
[1] 21.41142
```

Question

Why is the data available via JSON?

What are some other good (and bad) alternatives?

Why did they make citibike data available at all?

Why JSON?

→ JSON

- ✓ Easy to parse (easier than CSV file)
- ✓ Easy to update in real time

→ Why make data available?

- ✓ Let others develop tools