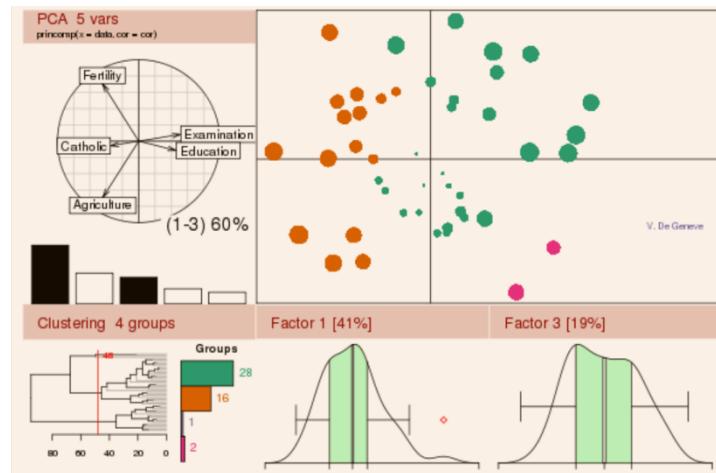
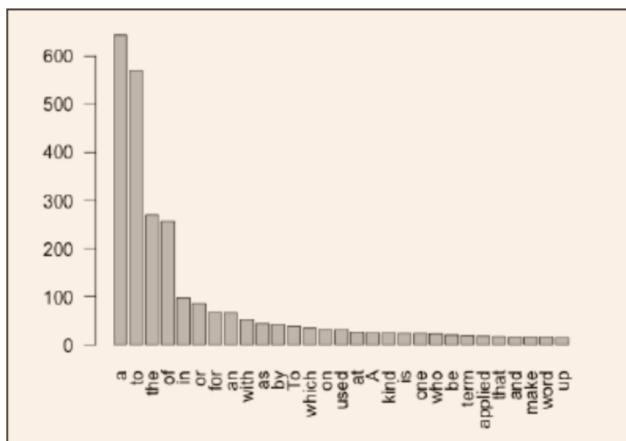


Data Science: Many Skills

Data Science: Many Skills

Data science is an emerging area of work concerned with the collection, preparation, analysis, visualization, management, and preservation of large collections of information. Although the term *data science* seems to connect most strongly with areas such as databases and computer science, many different kinds of skills—including nonmathematical skills—are needed.



What Is Data Science?

Data science is different from other areas such as mathematics and statistics.

→ Data scientists serve the needs and solve the problems of data users.

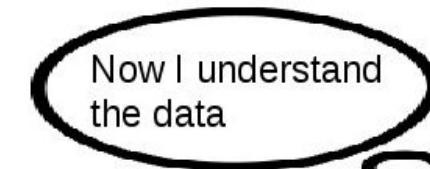
→ Before you can solve a problem, you need to identify it.

→ This process is not always as obvious as it might seem.

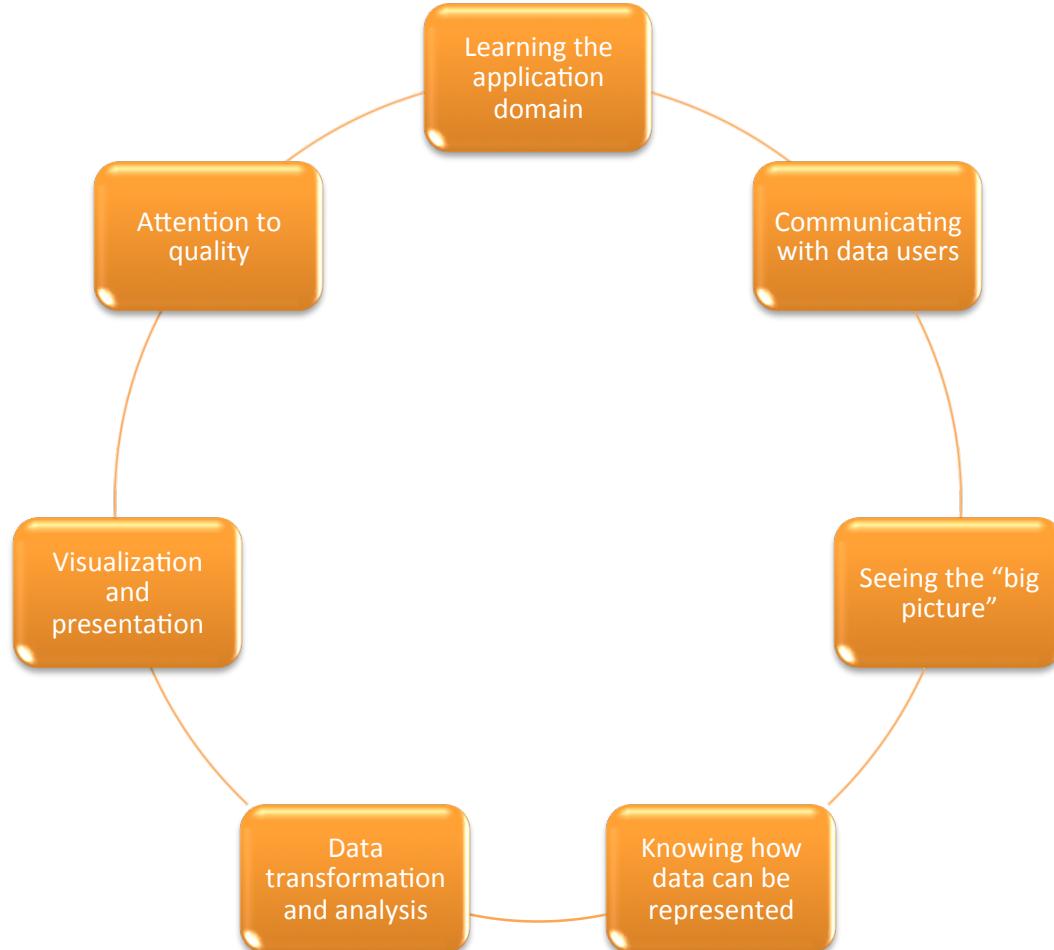


Data Science: Many Skills

Data Science → Perception

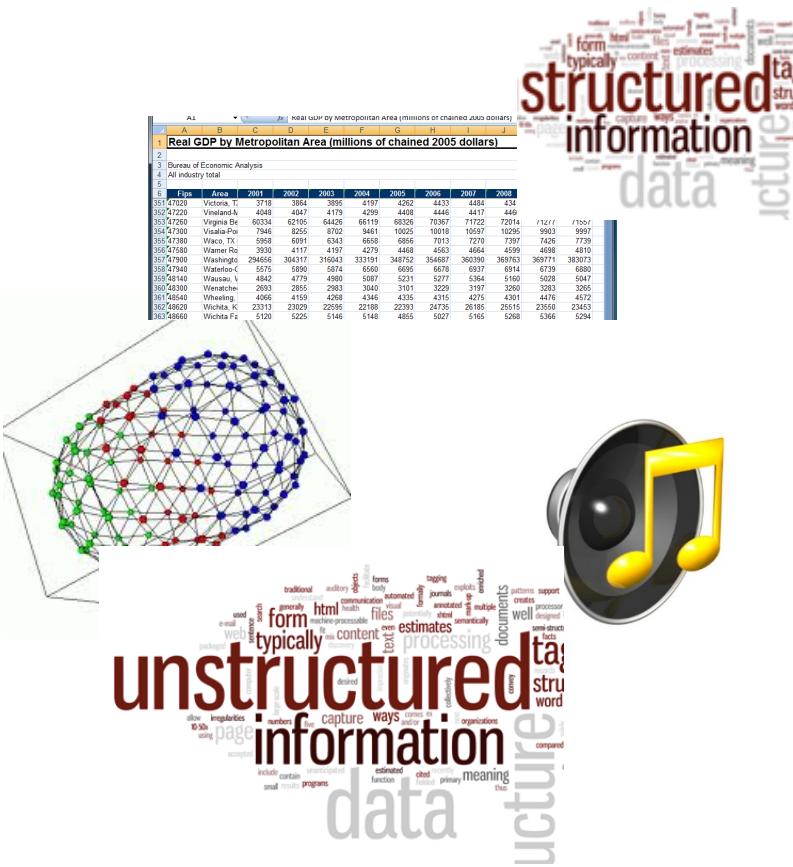


Skills Through the Life Cycle



Skills Across Data and Analysis

Contemporary Perspective



Data Science Is Multidisciplinary

By Brendan Tierney, 2012

Question

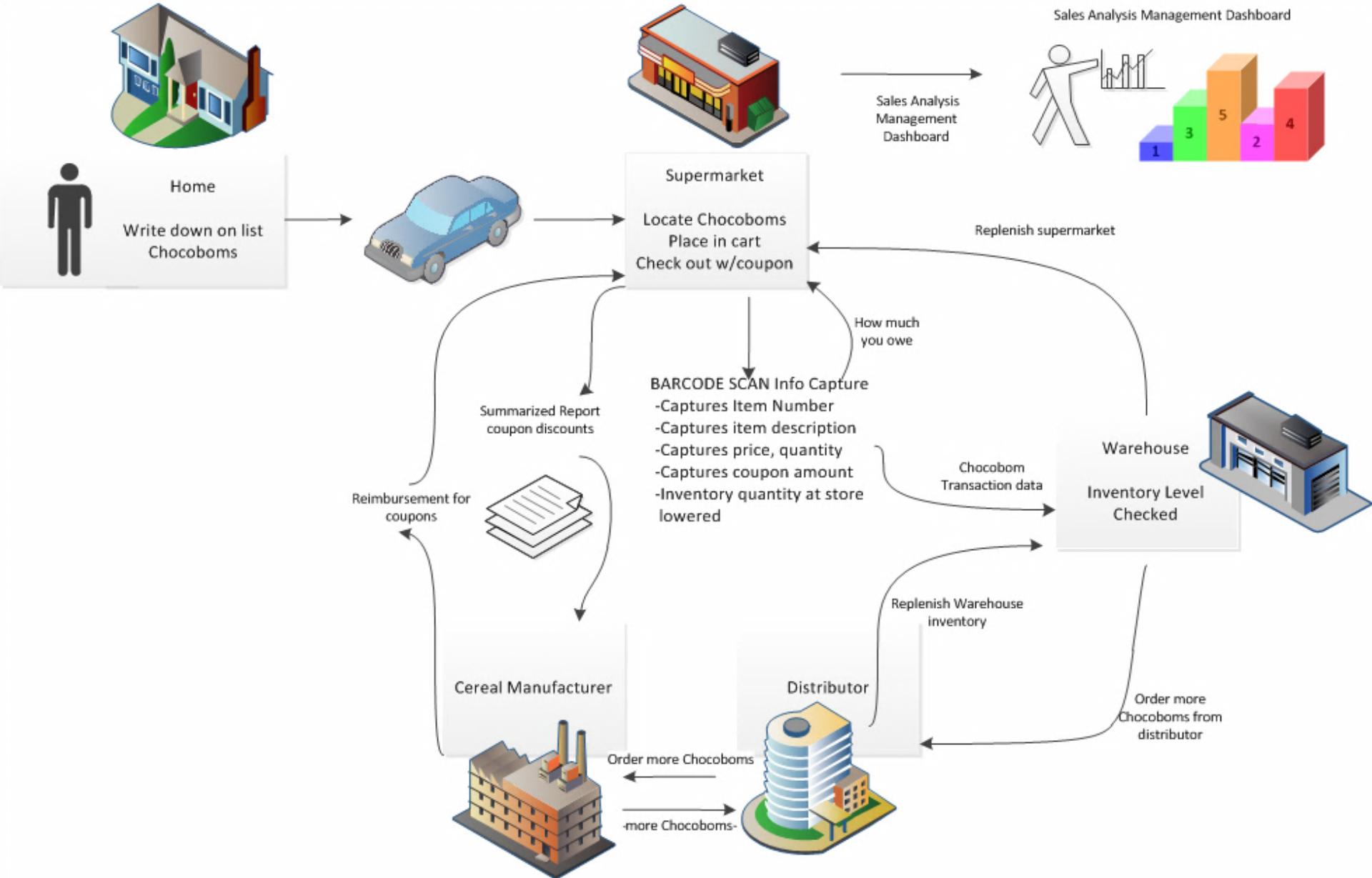
What is the difference between structured and unstructured data?

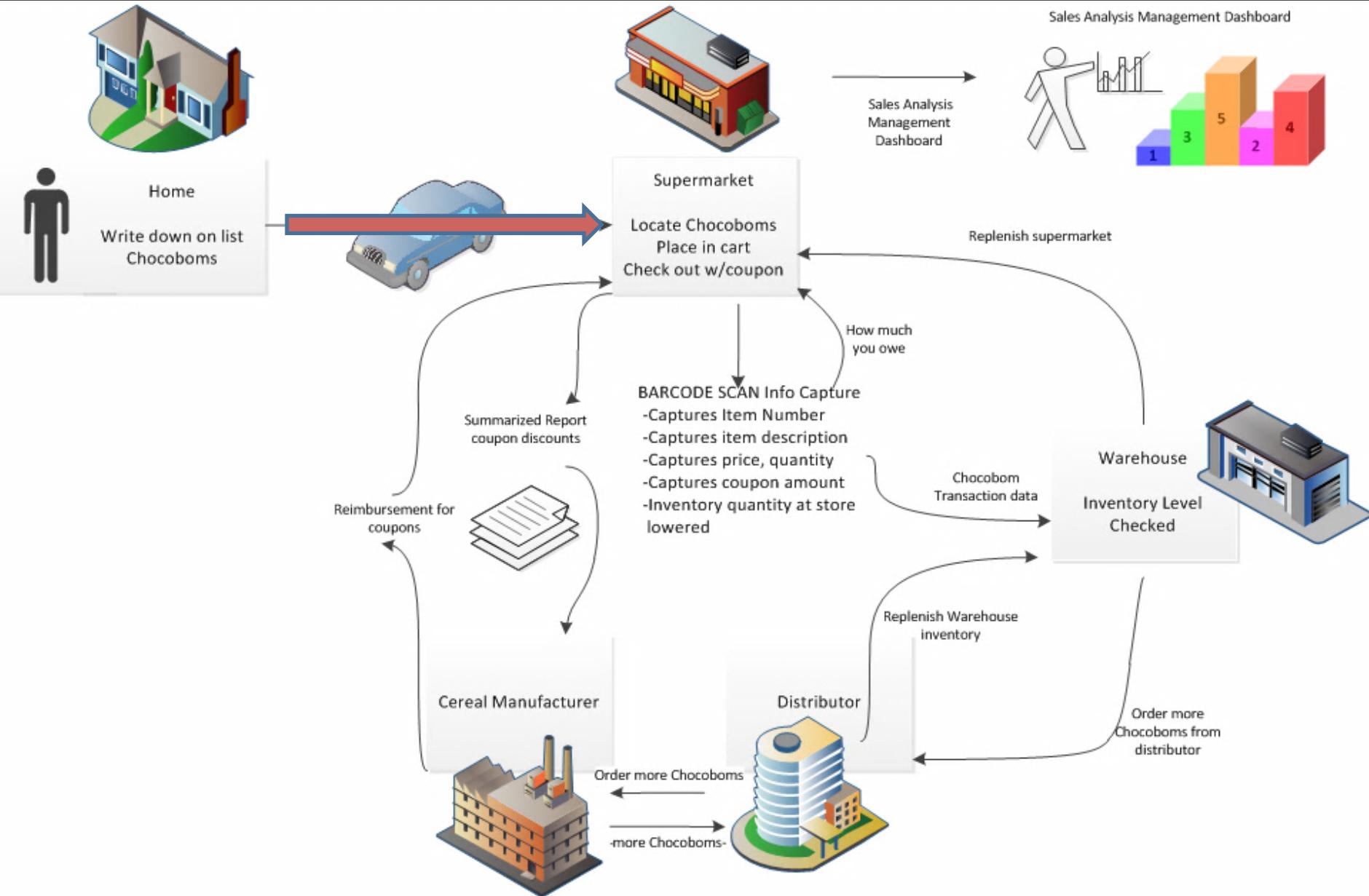
Provide some examples of each.

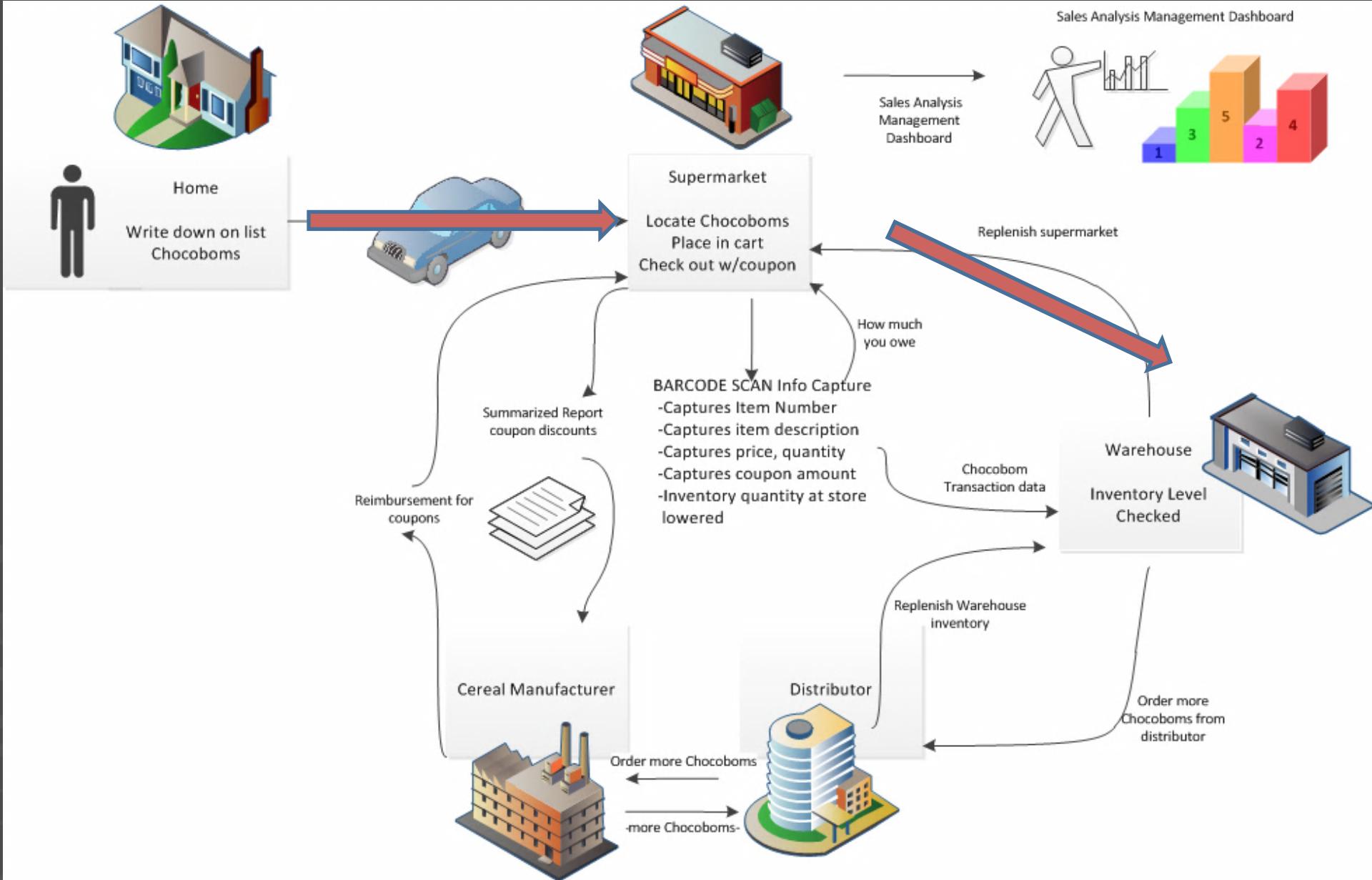
Data Science: Example

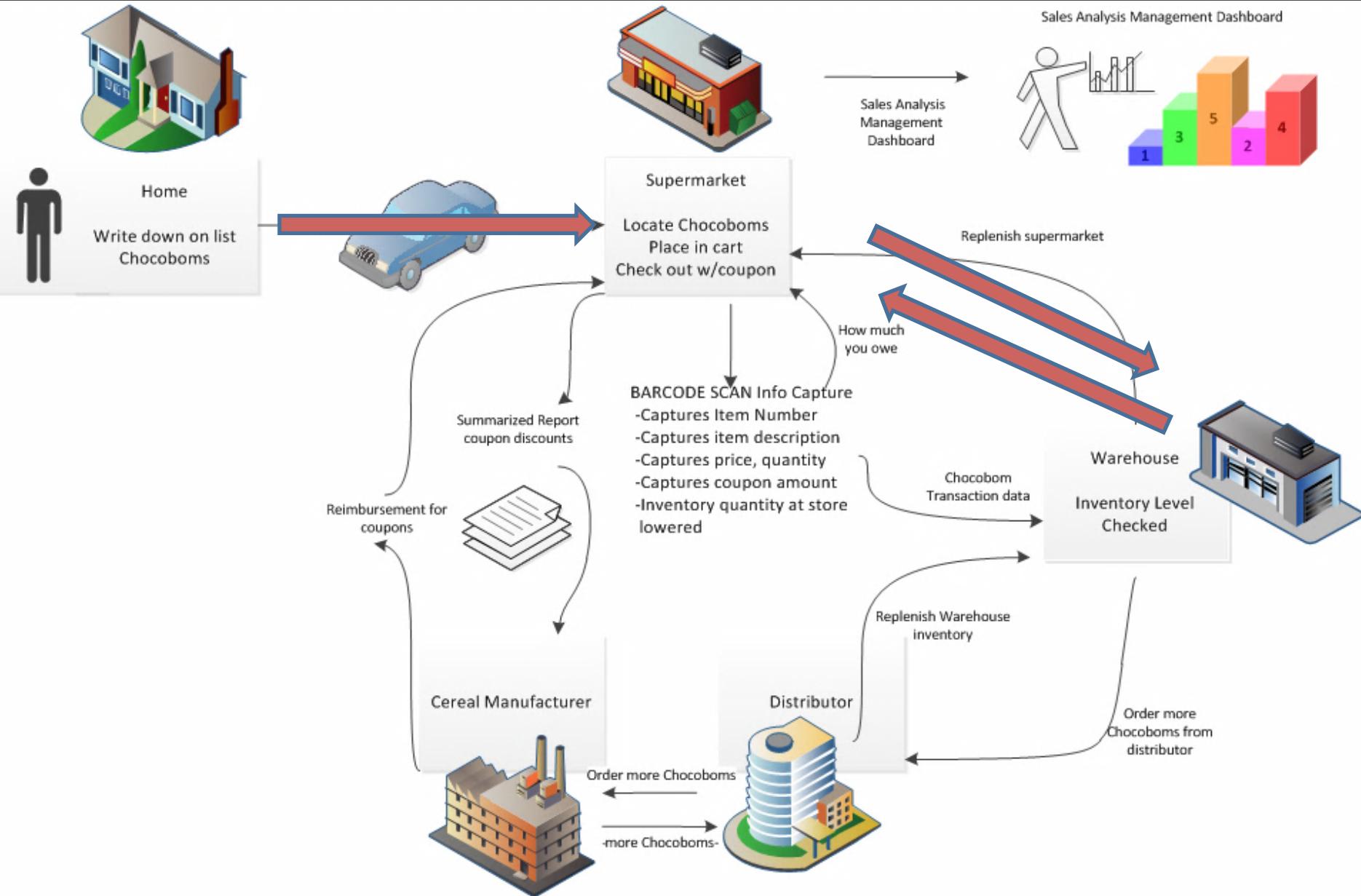
Data Science: An Example

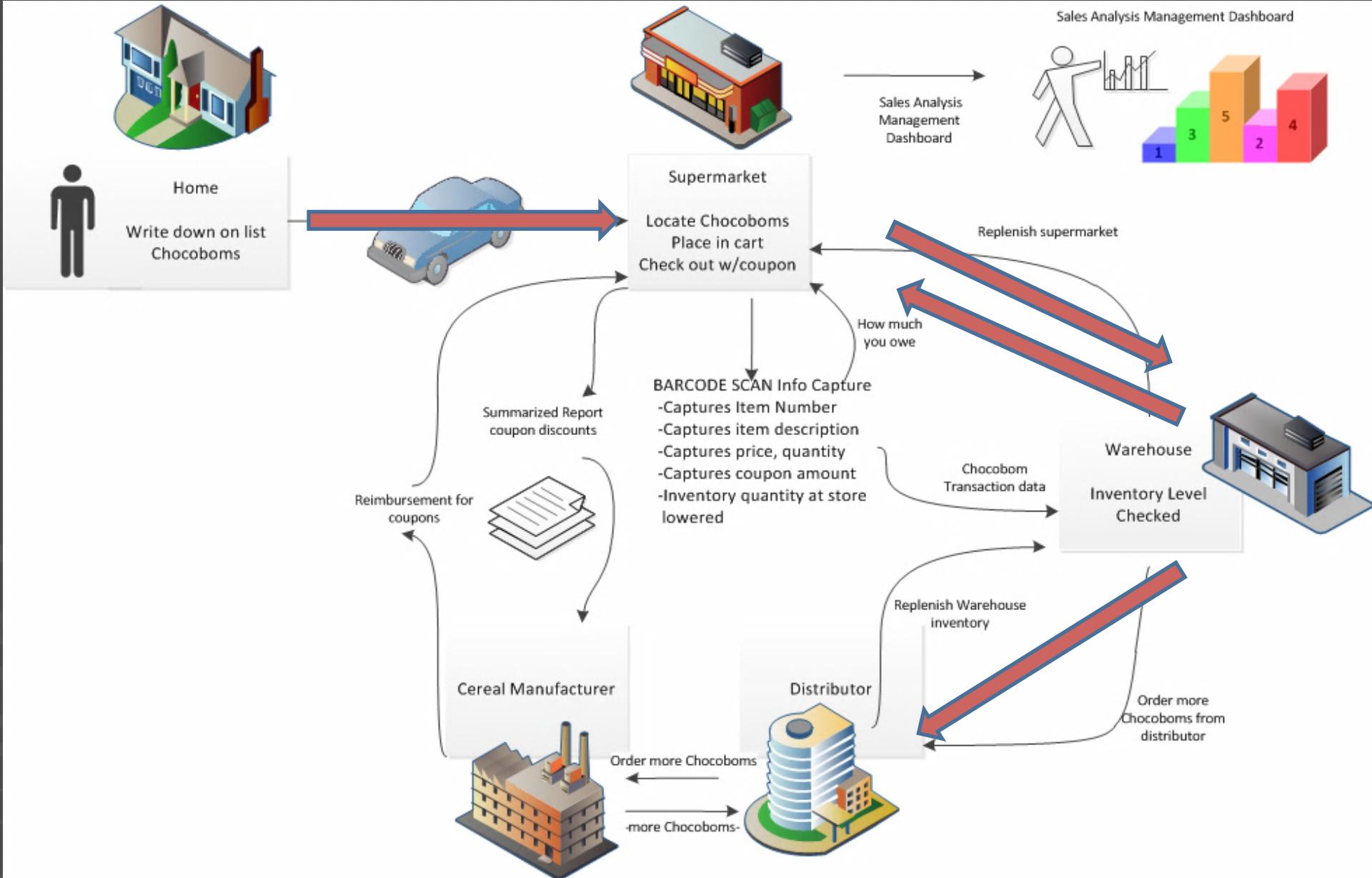


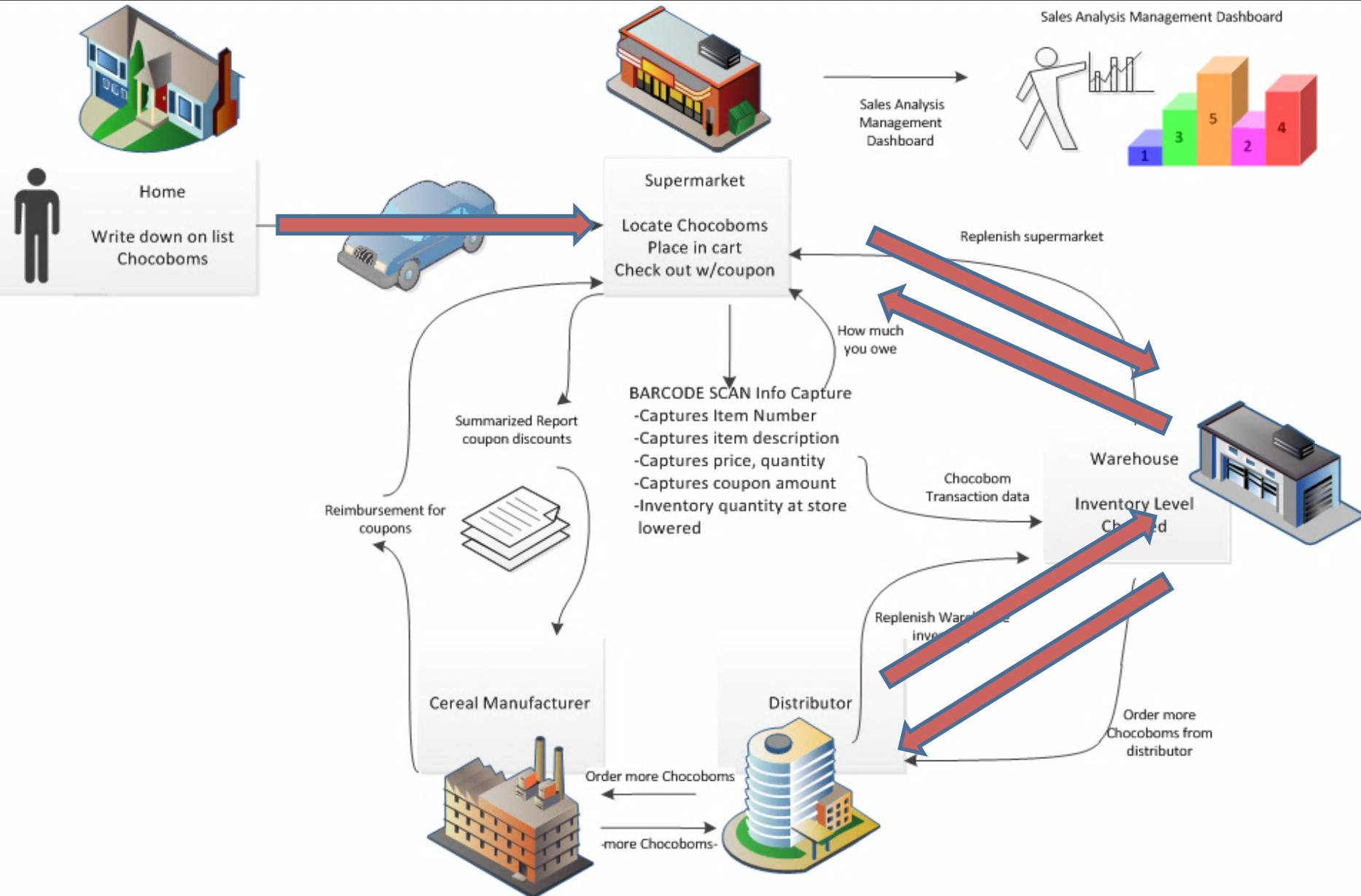


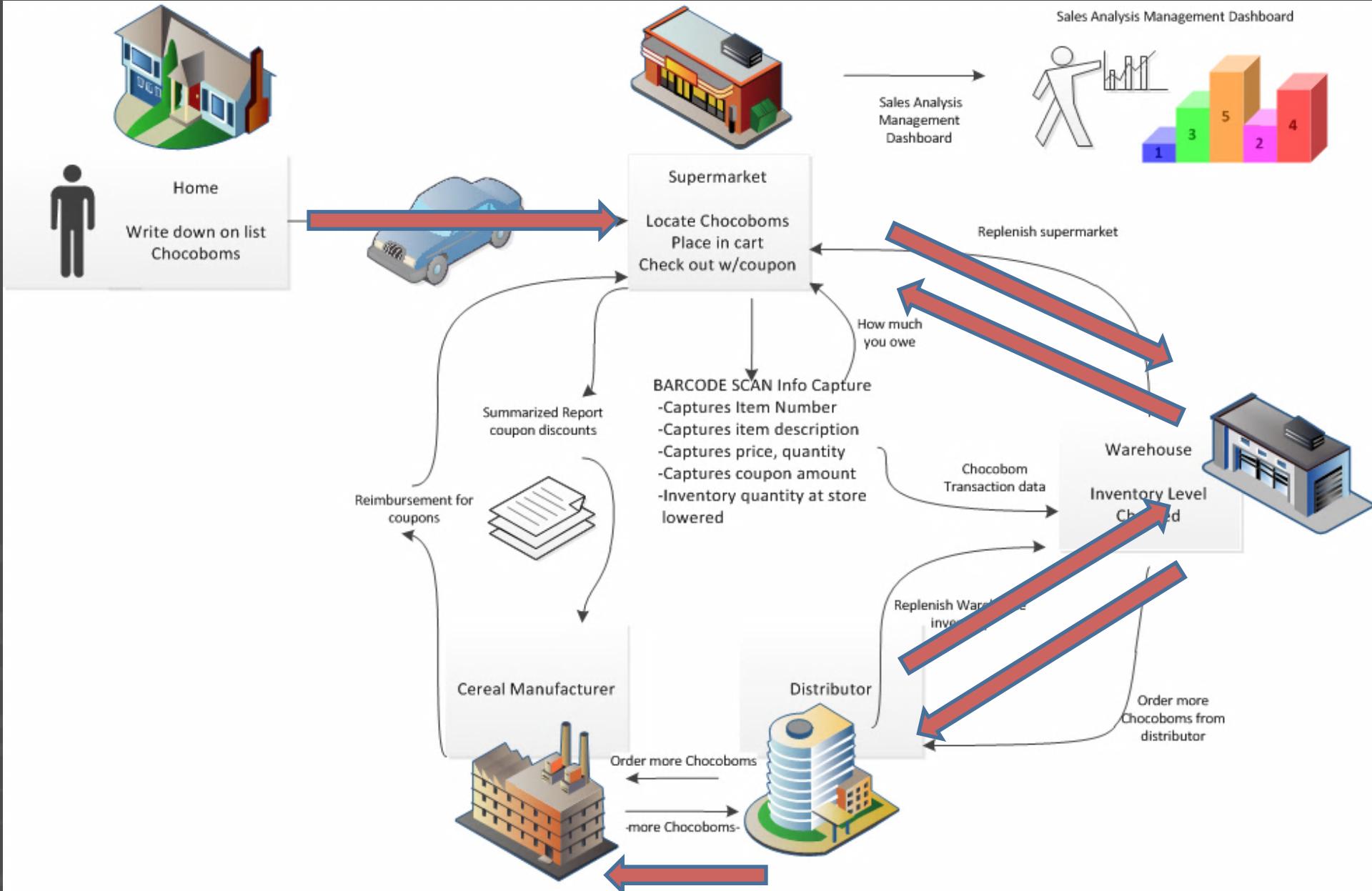


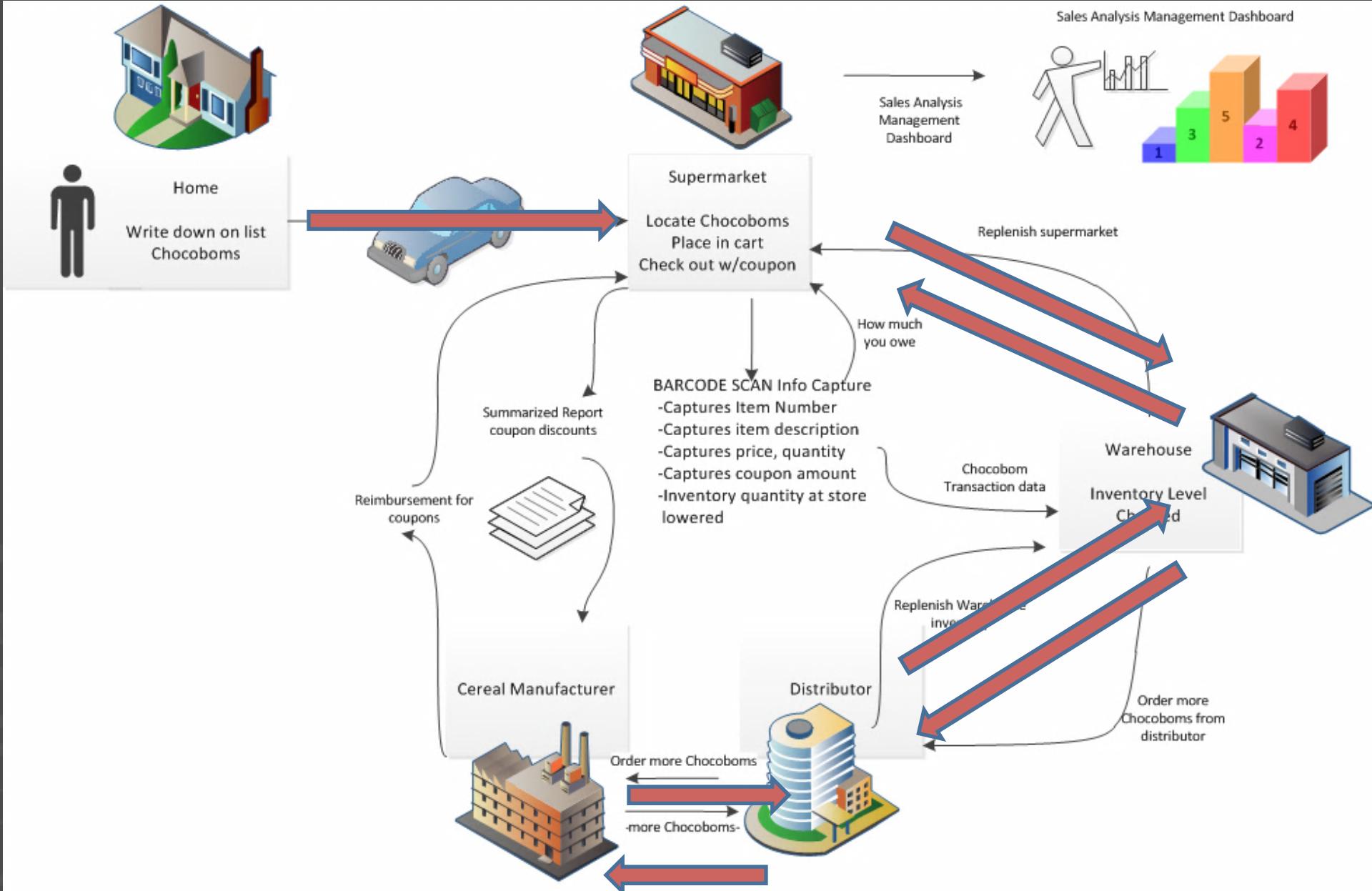


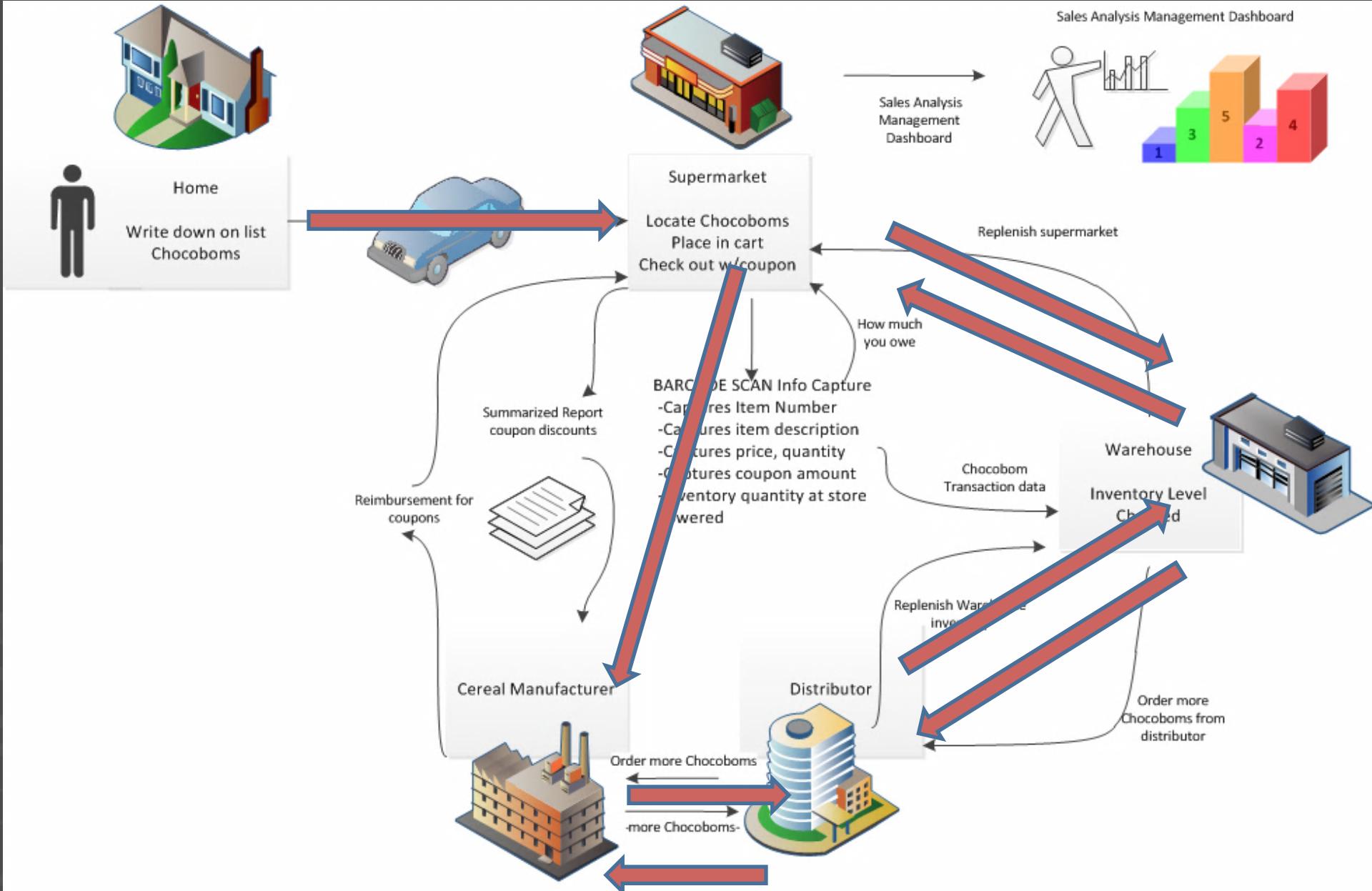


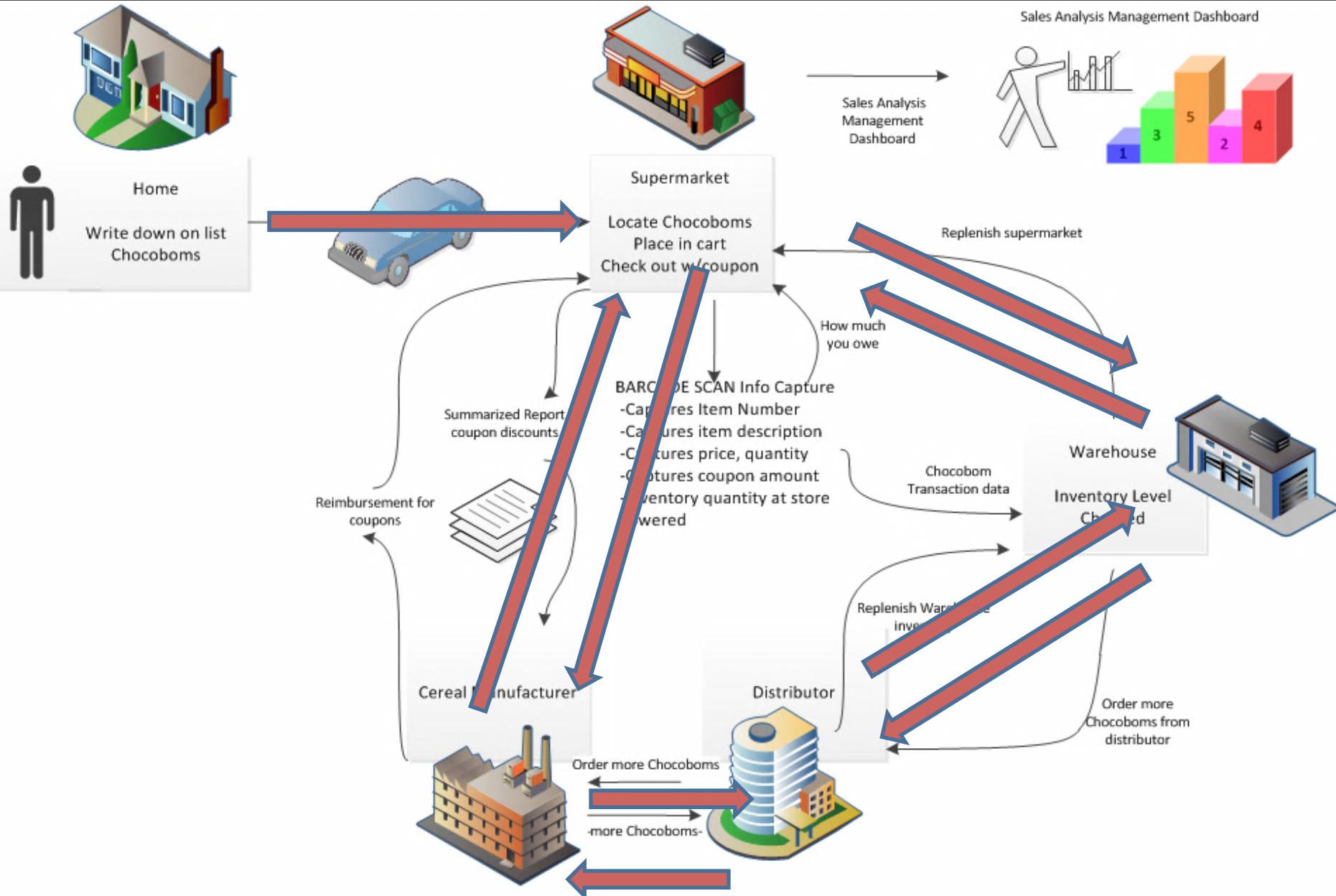


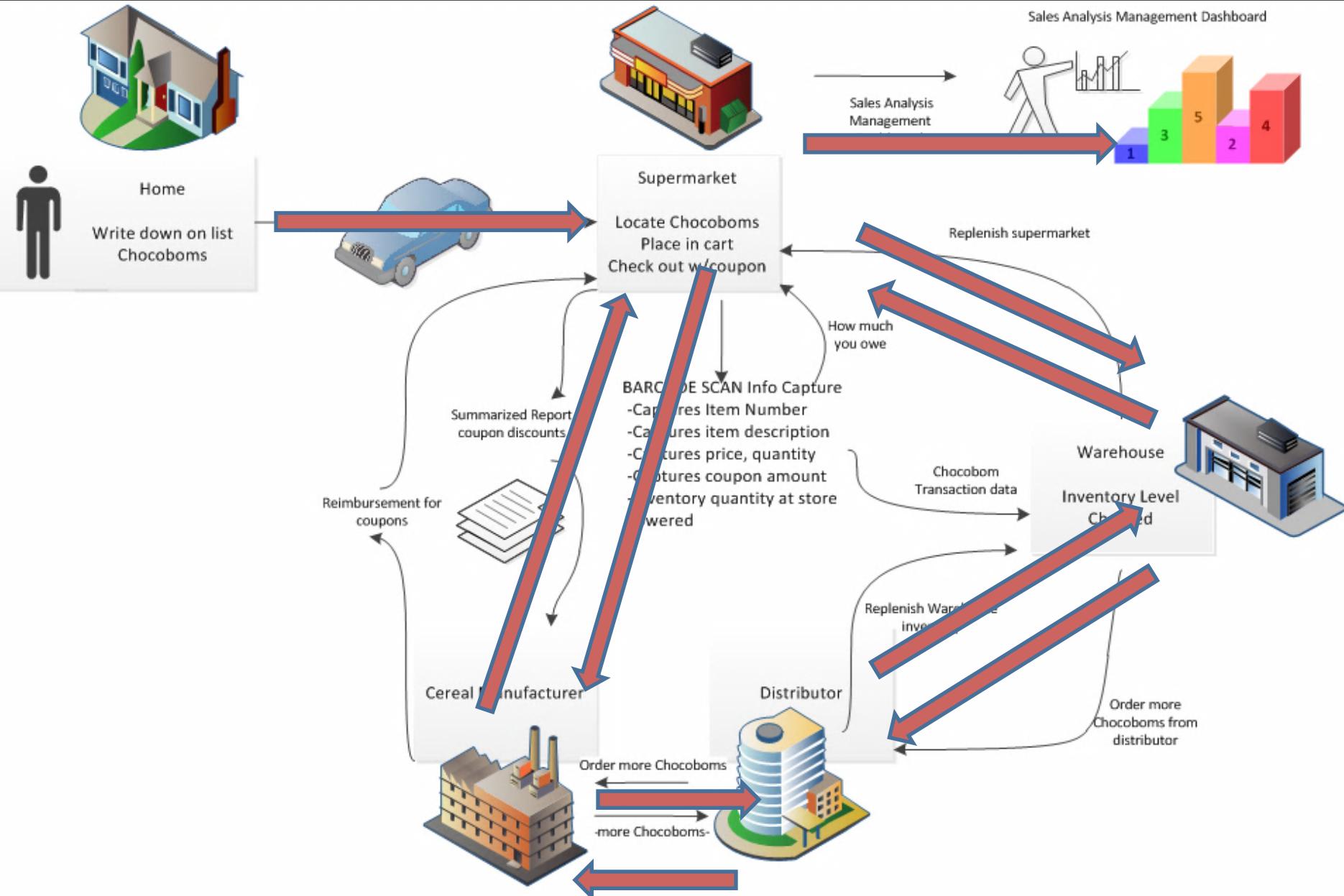


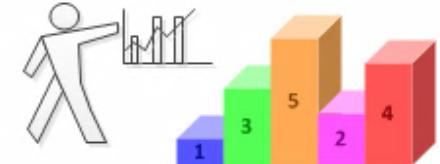










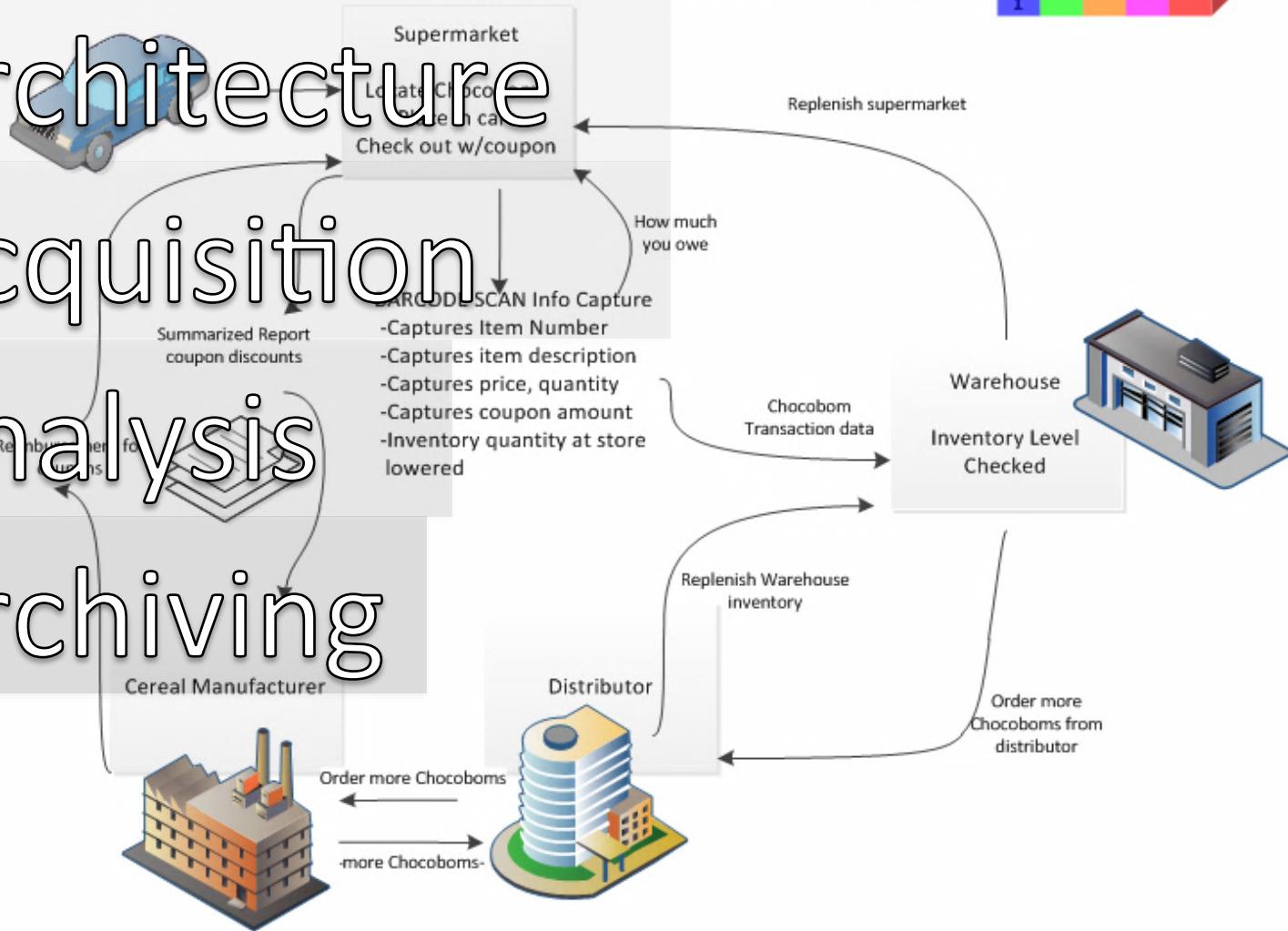


Data Architecture

Data Acquisition

Data Analysis

Data Archiving



Question

Where might there be data analysis in this process?

Provide some examples.

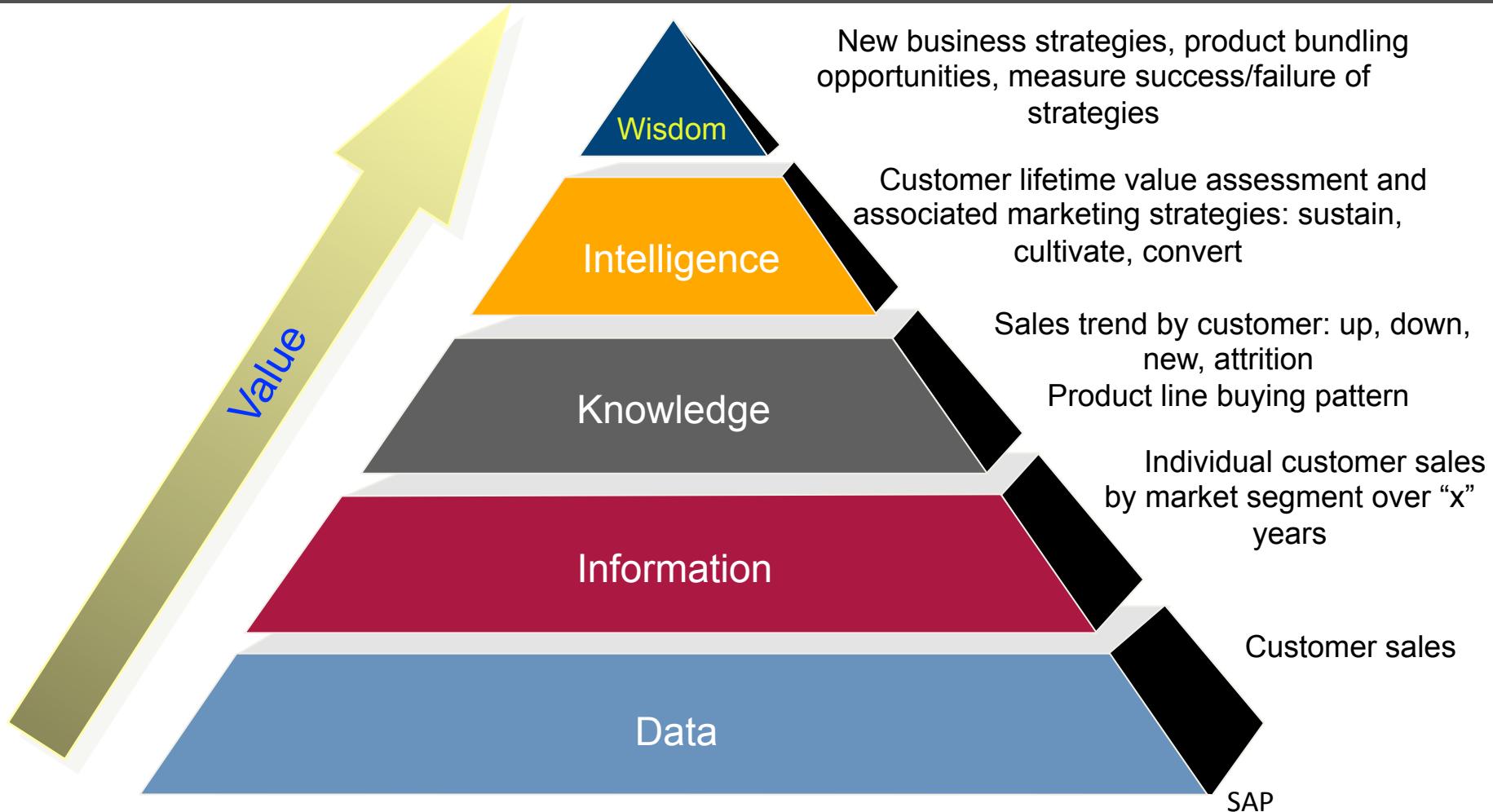
Data Science Overview, Part II

Data Science: About Data

Data comes from the Latin word *datum*, meaning “a thing given.” Although the term *data* has been used as early as the 1500s, modern usage started in the 1940s and 1950s as the age or era of computers started to emerge within a context of data input, process, data output.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	1	2	3	4	5	6	7	10	11	12	13	14	15	16	17
3	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
4	1	10	11	100	101	110	111	1000	1001	1010	1011	1100	1101	1110	1111

Getting Value from Data



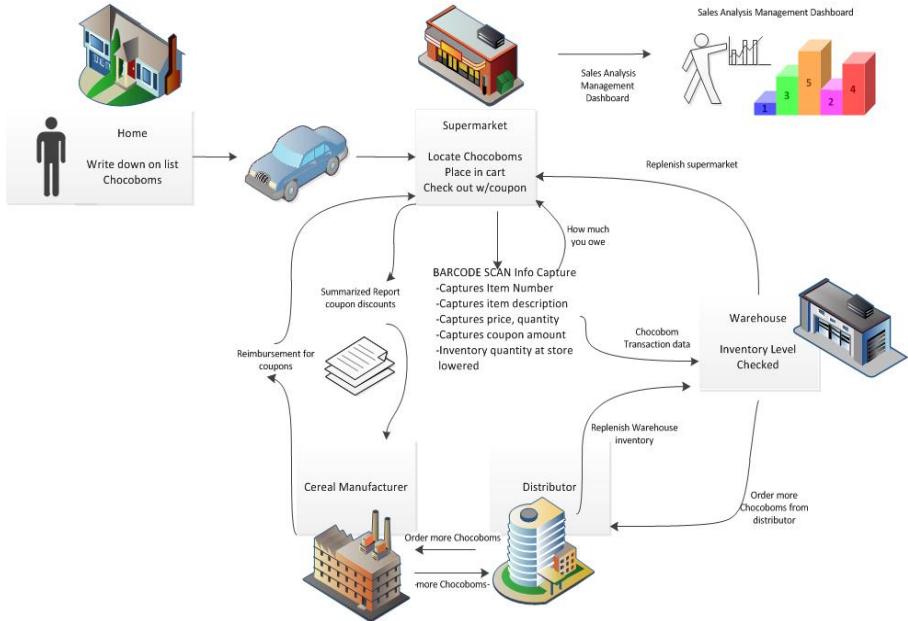
Data Science: Identifying Data Problems

Data science is different from other areas such as mathematics and statistics. Data science is an applied activity, and data scientists serve the needs and solve the problems of data users. Before you can solve a problem, you need to identify it, and this process is not always as obvious as it might seem. In this segment we discuss the identification of data problems.



Thinking About the Problem Domain

- Scope



- Focus area

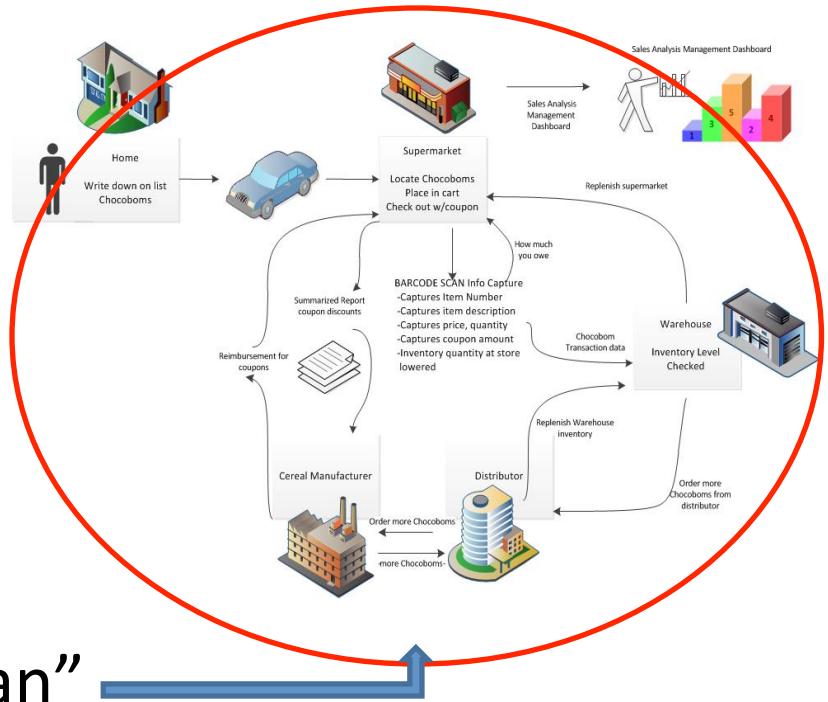
- Context

- Mitigate “boiling the ocean”



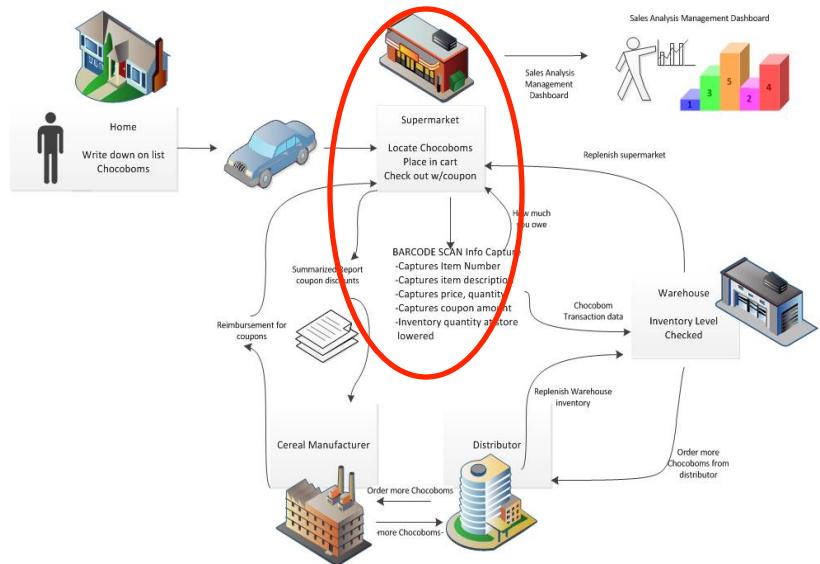
Get Focus

- Problem domain
 - Scope
 - Boundaries
 - Focus area
 - Context
 - Mitigate “boiling the ocean”



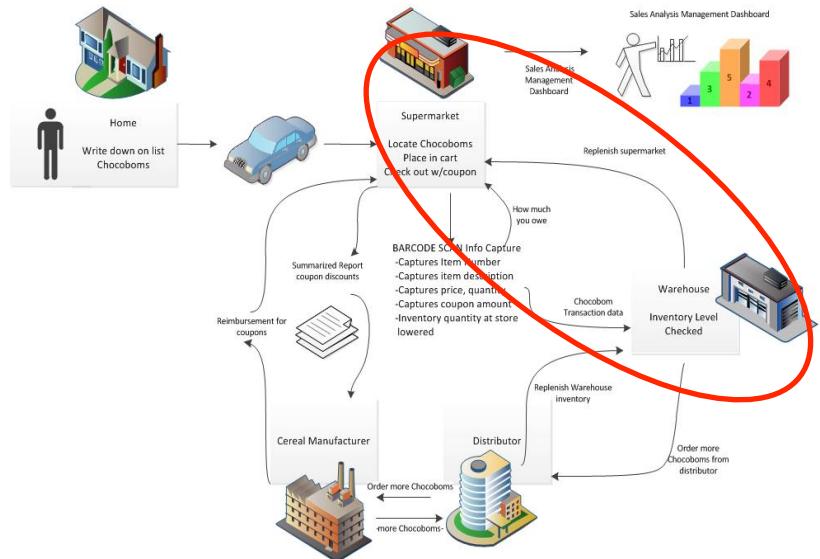
Get Focus

- Problem domain
 - Scope
 - Boundaries
 - Focus area
 - Context
 - Mitigate “boiling the ocean”



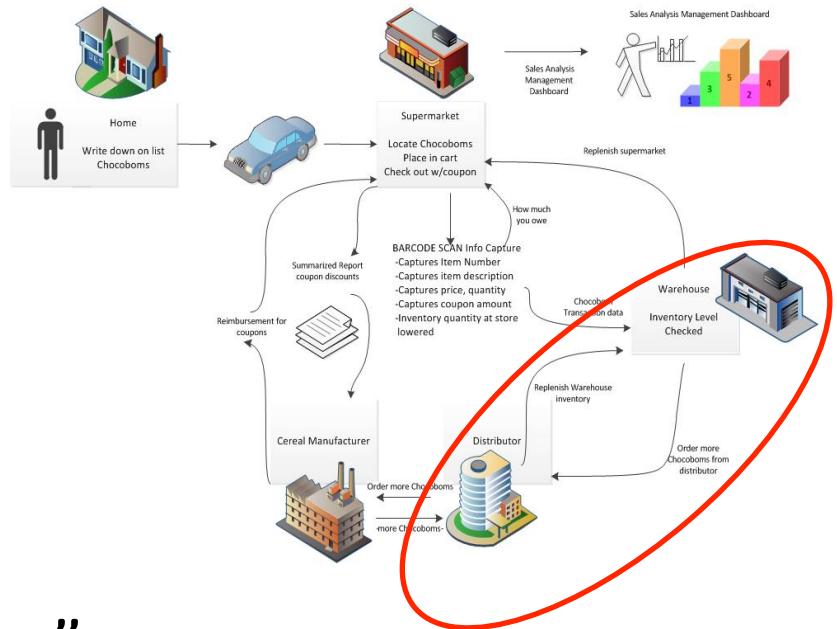
Get Focus

- Problem domain
 - Scope
 - Boundaries
 - Focus area
 - Context
 - Mitigate “boiling the ocean”



Get Focus

- Problem domain
 - Scope
 - Boundaries
 - Focus area
 - Context
 - Mitigate “boiling the ocean”



The Data Science Process

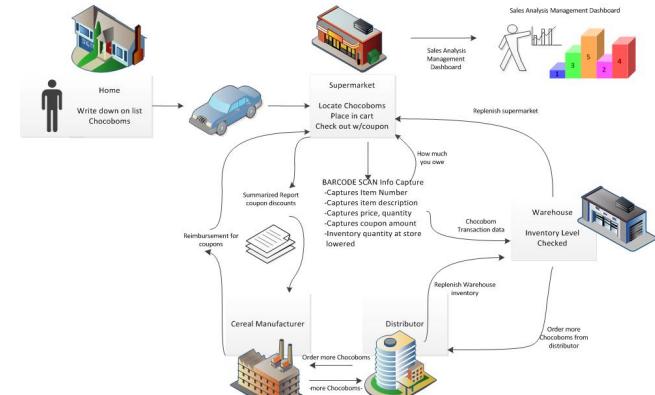
- Domain analysis/decomposition
- Identify subject matter expert(s)
- Question/interview/observation process
 - Stories
 - Anomalies
 - Risks and uncertainty

The Value of Stories

- Via Stories, Understand:
 - What people do
 - How they do it
 - Information produced and consumed
 - Process and information touch points
 - Decisions made
 - Challenges associated with all of the above

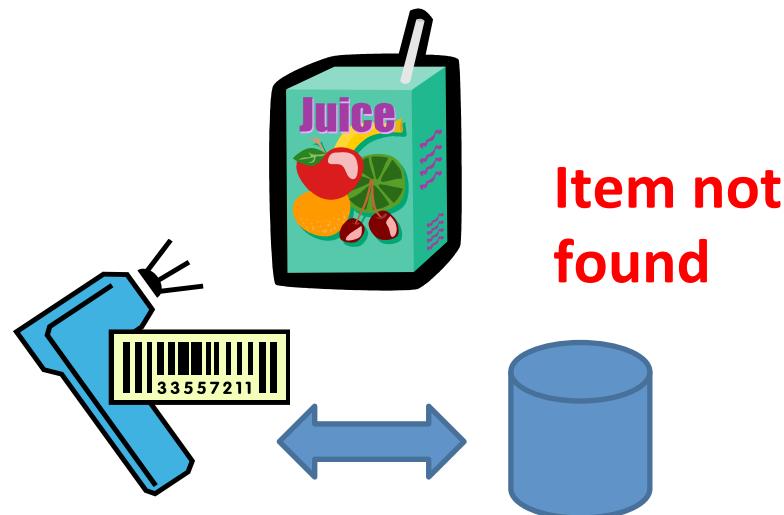
Stories: An Example

- Via Stories, Understand:
 - What activity performed
 - How the activity is performed
 - Information produced and consumed
 - Process and information touch points
 - Decisions made
 - Challenges associated with all of the above
- As the “cashier” my job is to scan the bar code on each item to capture product and price information. I also scan coupons to capture item discount information, which will reduce a customer’s total bill.
- As a cashier the challenges I encounter are item bar codes that are not legible, item bar codes that are invalid, and coupon rejections.



Anomalies

- Determine typical events, activities, processes.
- Identify the exceptions.
- Is it an **error** or an **interesting situation**

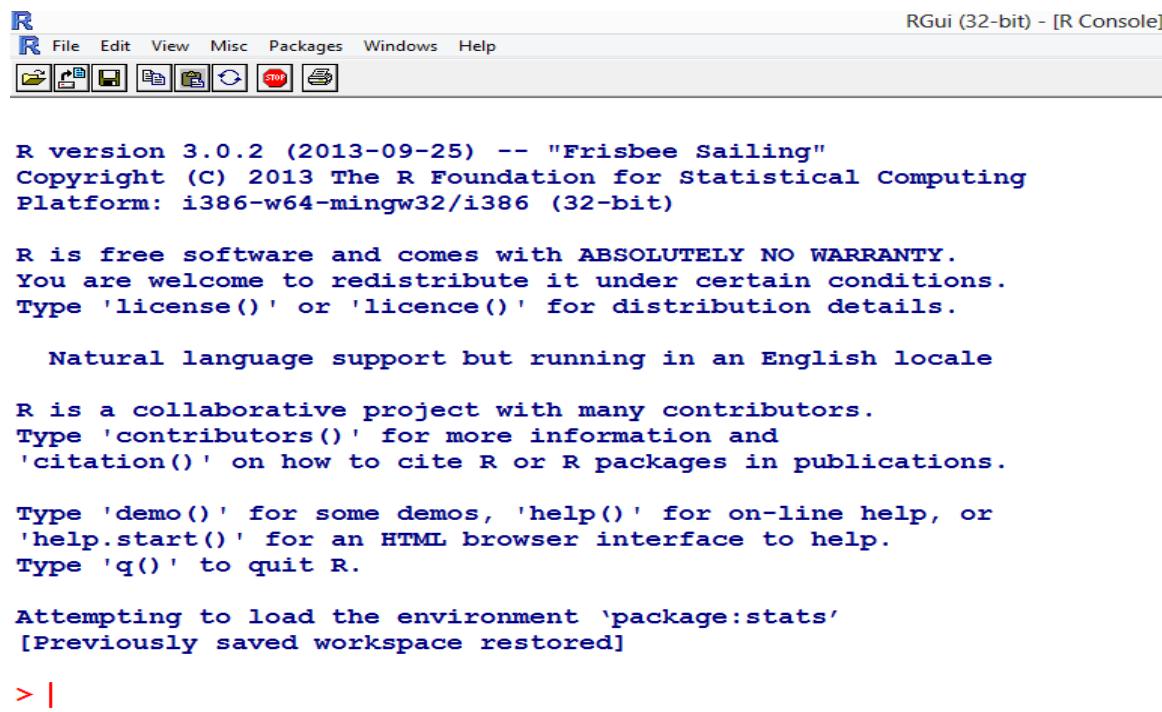


Key Themes to Review

- **Domain identification** and **understanding** is essential such that we zero in on the right problem or opportunity.
- **Identify SMEs** and engage them in “storytelling” about their activities such that you can get a sense of who, what, why they do what they do to also include surfacing anomalies and risk situations.

Getting Started With R

Data Science: Getting Started with R



R Gui (32-bit) - [R Console]

R version 3.0.2 (2013-09-25) -- "Frisbee Sailing"
Copyright (C) 2013 The R Foundation for Statistical Computing
Platform: i386-w64-mingw32/i386 (32-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

Attempting to load the environment 'package:stats'
[Previously saved workspace restored]

> |

R is an **open-source** software program, developed by volunteers as a service to the community of scientists, researchers, and data analysts who use it. R is **free** to download and use. **Lots of advice is available** online to help users learn R, which is good because it is a powerful and complex program, in reality a full-featured programming language dedicated to data.

Some Basic Information on R

- Command line oriented
- Not especially good at giving feedback or error messages
- One Needs to know the data

An Introduction to Data: Strings

Character string



“this is a piece of text”



011101000110100001101001011100110010000001101001011100110010
0000011000010010000001110000011010010110010110001101100101
00100000011011110110011000100000011101000110010101110000111
0100

R code:

`myText <- “this is a piece of text”`

An Introduction to Data: Type / Mode

List of integers

43, 42, 12, 8, 5

Each integer represents the age of a family member.

Integer list is all the same “type/mode.”

R refers to a list as a “vector.”

R code for this vector looks like

`c (43, 42, 12, 8, 5)`

`myFamilyAges <- c (43, 42, 12, 8, 5)`

Data set



Getting Started in R

- Start up R
- Create/key in at the command prompt ‘>’
 - > MyText<- “this is a piece of text”
Retrieve MyText
Retrieve mytext
- Create/key in at the command prompt ‘>’
 - > myFamilyAges <-c(**43, 42, 12, 8, 5**)
Retrieve myFamilyAges
- End R q() at the command prompt ‘>’
- Respond ‘yes’ to save your work space

The Initial Console View



> |

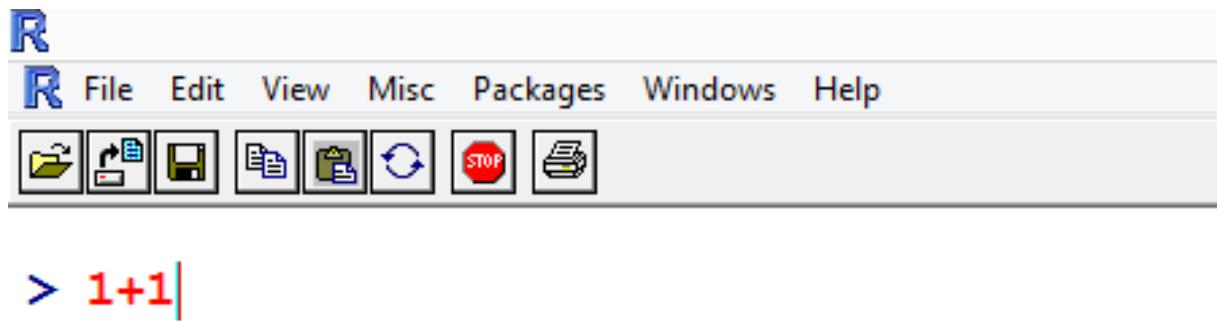
Clear the Console

Edit → Clear Console or Control L

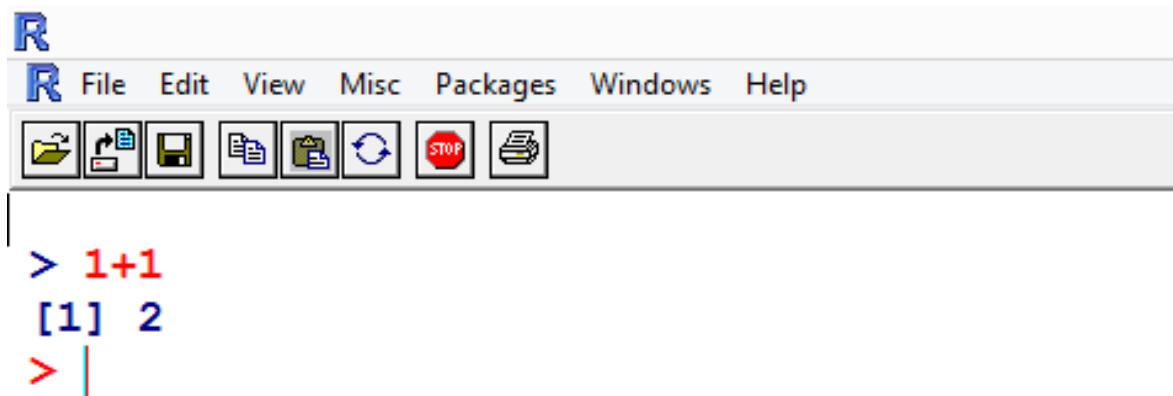


> |

Adding Two Numbers



Adding Two Numbers



The image shows a screenshot of the R software environment. At the top is a menu bar with the 'R' logo, followed by 'File', 'Edit', 'View', 'Misc', 'Packages', 'Windows', and 'Help'. Below the menu is a toolbar with various icons: a folder, a copy/paste pair, a file, a clipboard, a circular arrow, a red stop sign, and a printer. The main window contains a command-line interface. The user has entered the command `> 1+1`, which has been evaluated and returned the result `[1] 2`. A cursor is visible at the end of the line.

```
> 1+1
[1] 2
> |
```

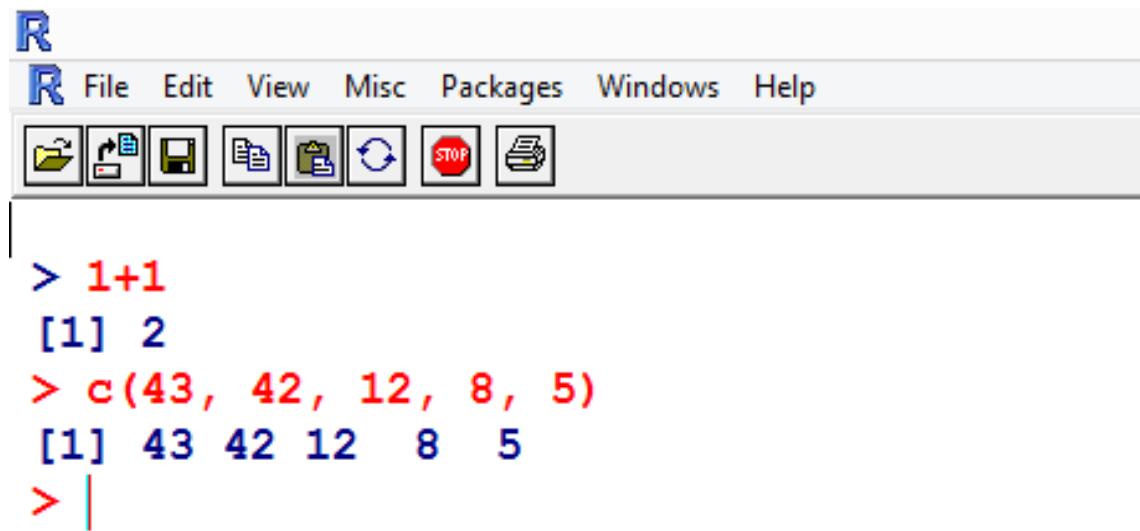
Creating a Vector



```
> 1+1  
[1] 2  
> c(43, 42, 12, 8, 5)|
```

Note that the ‘c’ command stands for concatenate or combine

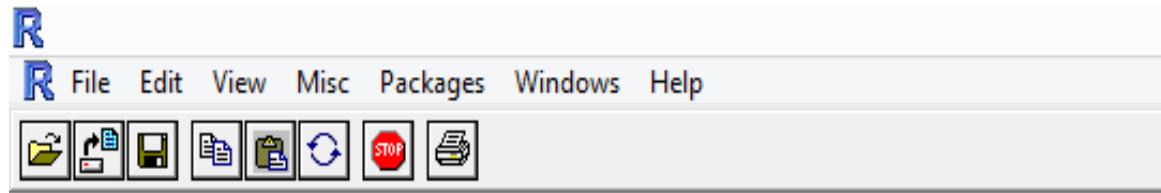
Creating a Vector



The image shows a screenshot of the R software environment. At the top is a menu bar with the following items: File, Edit, View, Misc, Packages, Windows, and Help. Below the menu bar is a toolbar with several icons: a folder, a file, a database, a clipboard, a refresh, and a stop sign. The main area is a command-line interface where the user has entered some R code:

```
> 1+1  
[1] 2  
> c(43, 42, 12, 8, 5)  
[1] 43 42 12 8 5  
> |
```

Storing a Vector



```
> 1+1
[1] 2
> c(43, 42, 12, 8, 5)
[1] 43 42 12  8  5
> myFamilyAges <- c(43, 42, 12, 8, 5)
> |
```

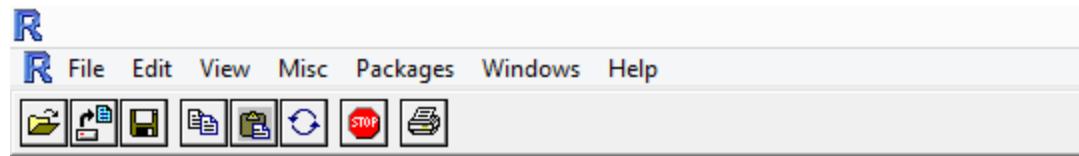
Storing a Vector



```
R
R File Edit View Misc Packages Windows Help
[1] [2] [3] [4] [5] [6] [7] [8]

> 1+1
[1] 2
> c(43, 42, 12, 8, 5)
[1] 43 42 12 8 5
> myFamilyAges <- c(43, 42, 12, 8, 5)
> myFamilyAges
[1] 43 42 12 8 5
> |
```

Some Simple Functions on a Vector



```
> 1+1
[1] 2
> c(43, 42, 12, 8, 5)
[1] 43 42 12 8 5
> myFamilyAges <- c(43, 42, 12, 8, 5)
> myFamilyAges
[1] 43 42 12 8 5
> sum(myFamilyAges)
[1] 110
> mean(myFamilyAges)
[1] 22
> range(myFamilyAges)
[1] 5 43
> |
```

sum

mean

range

Errors in R



```
R
R File Edit View Misc Packages Windows Help
[File, Edit, View, Misc, Packages, Windows, Help icons]

> 1+1
[1] 2
> c(43, 42, 12, 8, 5)
[1] 43 42 12 8 5
> myFamilyAges <- c(43, 42, 12, 8, 5)
> myFamilyAges
[1] 43 42 12 8 5
> sum(myFamilyAges)
[1] 110
> mean(myFamilyAges)
[1] 22
> range(myFamilyAges)
[1] 5 43
> fish(myFamilyAges)
Error: could not find function "fish"
> |
```

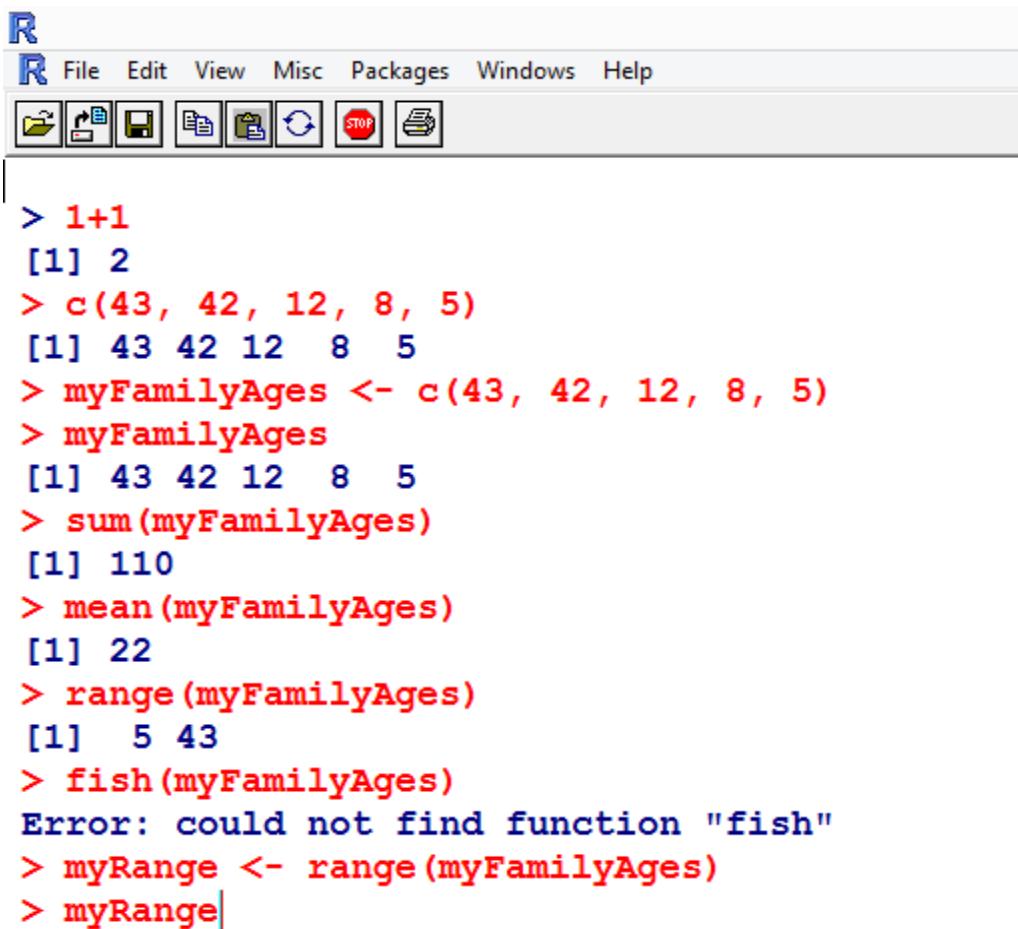
Storing Function Results



```
R
R File Edit View Misc Packages Windows Help
[<--> <--> <--> <--> <--> <--> STOP <-->]

> 1+1
[1] 2
> c(43, 42, 12, 8, 5)
[1] 43 42 12 8 5
> myFamilyAges <- c(43, 42, 12, 8, 5)
> myFamilyAges
[1] 43 42 12 8 5
> sum(myFamilyAges)
[1] 110
> mean(myFamilyAges)
[1] 22
> range(myFamilyAges)
[1] 5 43
> fish(myFamilyAges)
Error: could not find function "fish"
> myRange <- range(myFamilyAges)
> |
```

Storing Function Results



```
R
File Edit View Misc Packages Windows Help
[<--> <--> <--> <--> <--> <--> STOP <--> ]
```

```
> 1+1
[1] 2
> c(43, 42, 12, 8, 5)
[1] 43 42 12 8 5
> myFamilyAges <- c(43, 42, 12, 8, 5)
> myFamilyAges
[1] 43 42 12 8 5
> sum(myFamilyAges)
[1] 110
> mean(myFamilyAges)
[1] 22
> range(myFamilyAges)
[1] 5 43
> fish(myFamilyAges)
Error: could not find function "fish"
> myRange <- range(myFamilyAges)
> myRange
```



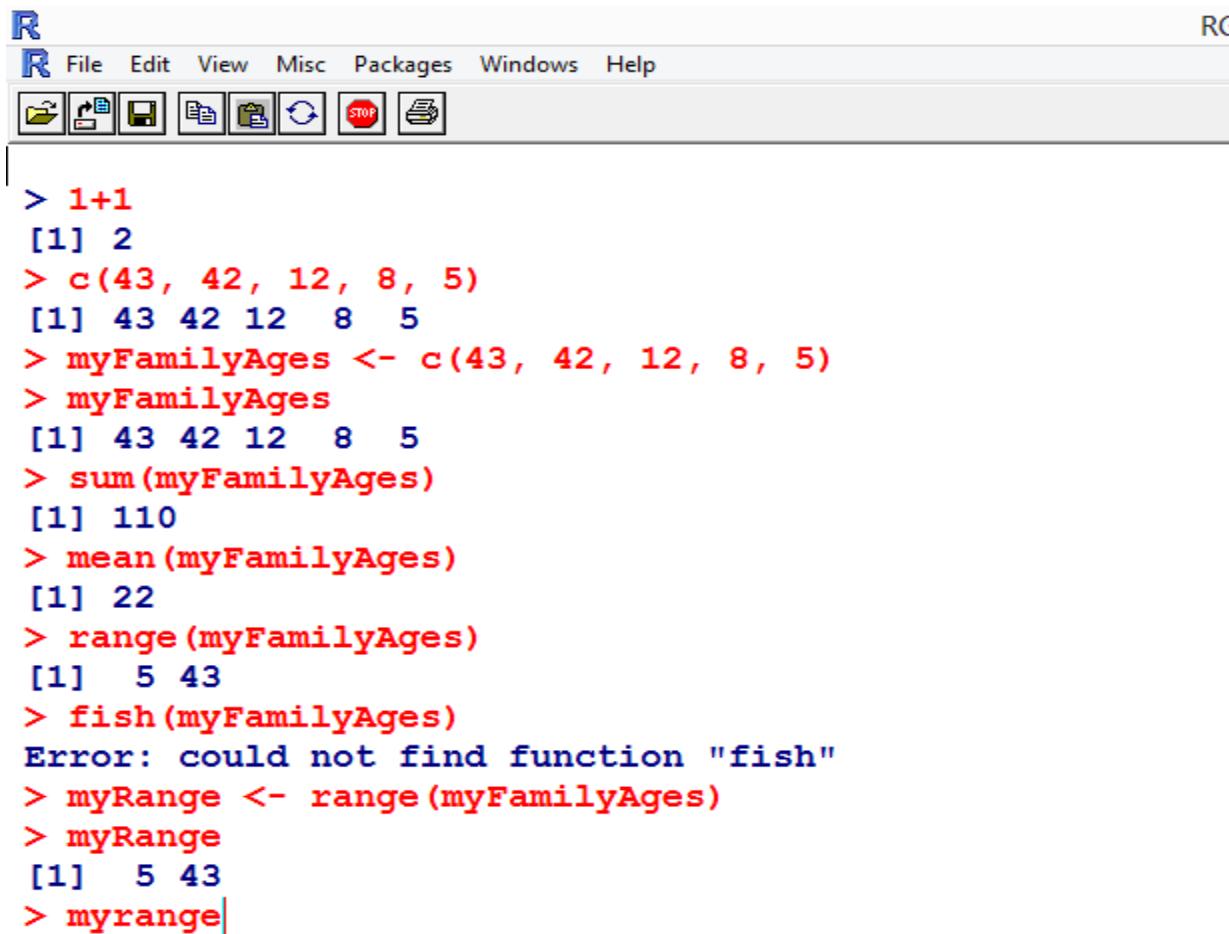
Storing Function Results



```
R
R File Edit View Misc Packages Windows Help
[1] [2] [3] [4] [5] [6] [7] [8]

> 1+1
[1] 2
> c(43, 42, 12, 8, 5)
[1] 43 42 12 8 5
> myFamilyAges <- c(43, 42, 12, 8, 5)
> myFamilyAges
[1] 43 42 12 8 5
> sum(myFamilyAges)
[1] 110
> mean(myFamilyAges)
[1] 22
> range(myFamilyAges)
[1] 5 43
> fish(myFamilyAges)
Error: could not find function "fish"
> myRange <- range(myFamilyAges)
> myRange
[1] 5 43
> |
```

Caps are NOT Ignored



The screenshot shows the RStudio interface with the R logo in the top-left corner and "RG" in the top-right corner. The menu bar includes File, Edit, View, Misc, Packages, Windows, and Help. Below the menu is a toolbar with various icons. The main area is a console window displaying R code and its output:

```
> 1+1
[1] 2
> c(43, 42, 12, 8, 5)
[1] 43 42 12 8 5
> myFamilyAges <- c(43, 42, 12, 8, 5)
> myFamilyAges
[1] 43 42 12 8 5
> sum(myFamilyAges)
[1] 110
> mean(myFamilyAges)
[1] 22
> range(myFamilyAges)
[1] 5 43
> fish(myFamilyAges)
Error: could not find function "fish"
> myRange <- range(myFamilyAges)
> myRange
[1] 5 43
> myrange|
```

Caps are NOT Ignored



```
R
R File Edit View Misc Packages Windows Help
[< > <= <= <= STOP <=]

> 1+1
[1] 2
> c(43, 42, 12, 8, 5)
[1] 43 42 12 8 5
> myFamilyAges <- c(43, 42, 12, 8, 5)
> myFamilyAges
[1] 43 42 12 8 5
> sum(myFamilyAges)
[1] 110
> mean(myFamilyAges)
[1] 22
> range(myFamilyAges)
[1] 5 43
> fish(myFamilyAges)
Error: could not find function "fish"
> myRange <- range(myFamilyAges)
> myRange
[1] 5 43
> myrange
Error: object 'myrange' not found
> |
```

Some Takeaways

- The use of the “c()” function.
- The vector is a basic form of data storage.
- Numbers & text can be collected in lists (Vector)
- A vector can be stored in a named location using the arrow “<-” (e.g. myFamilyAges)
- You can get the data object a named location by typing the name.

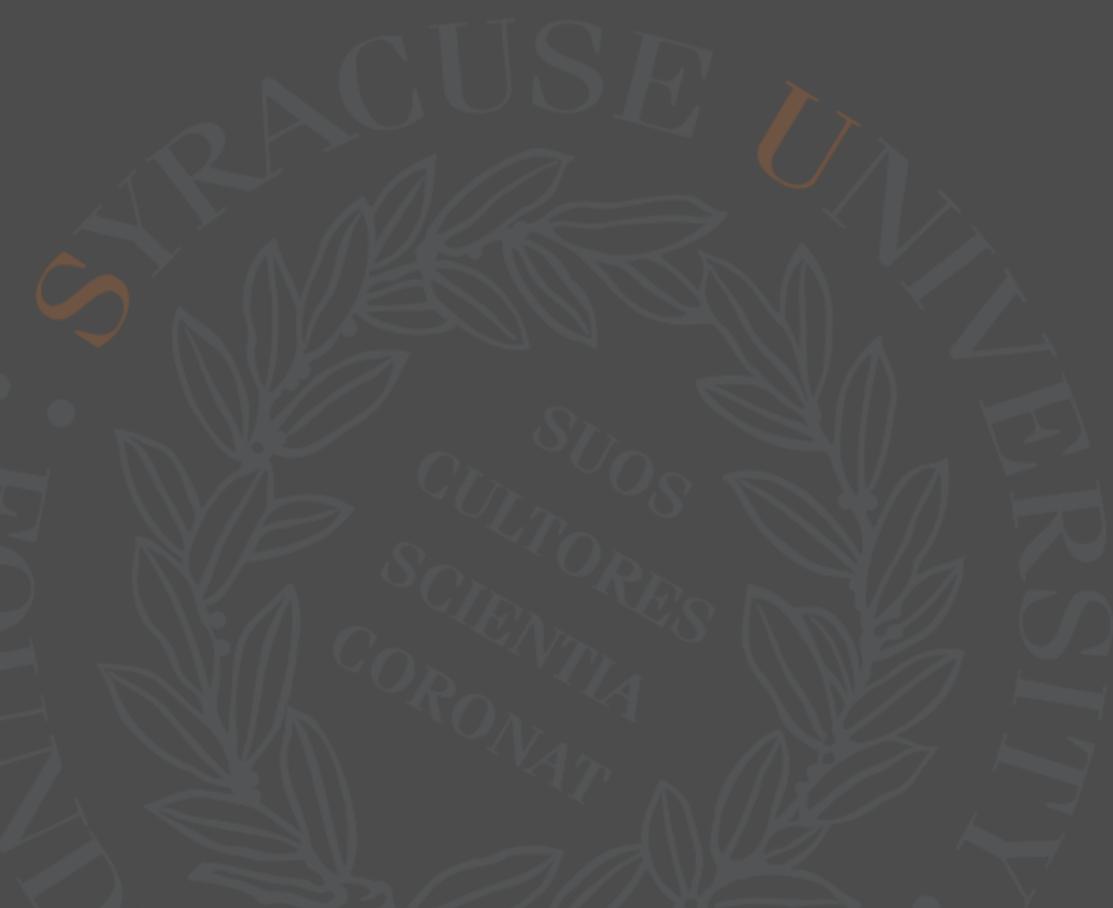
Some Takeaways

- Vector has a length (number of items in it).
- Vector has a mode (type of data).
- Capitalization matters.
- Legal functions include sum, mean, range, and fish is not a legal function.
- Typically more than one variable or data set is used for analysis.

Data Science

Data Science Fun Meter

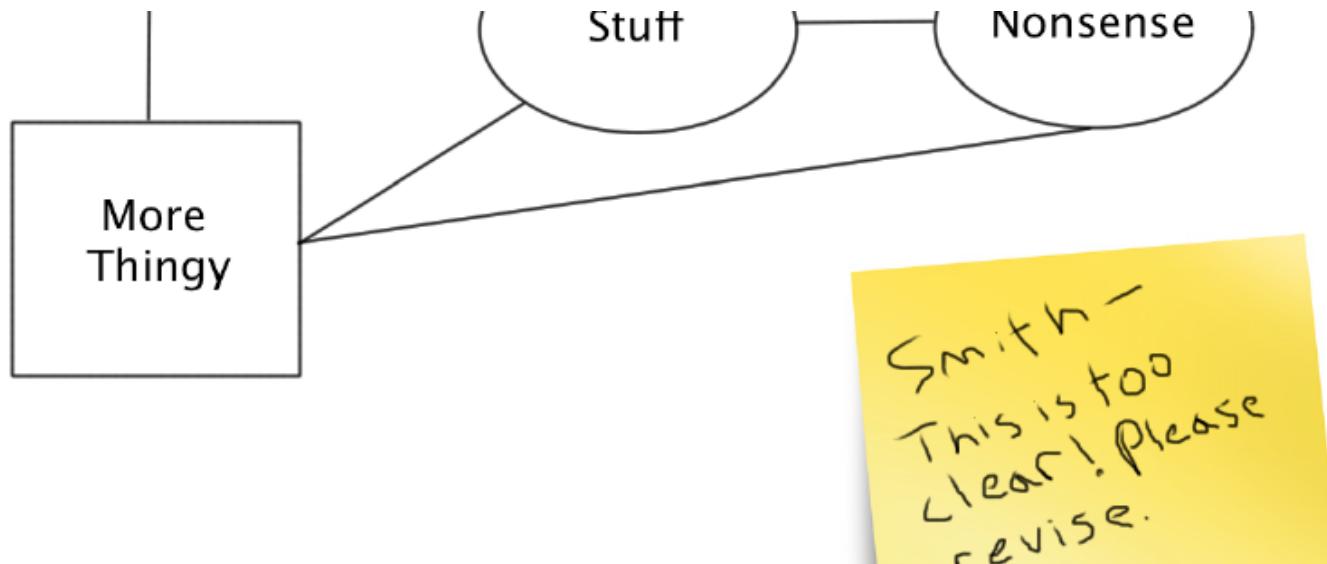




School of Information Studies
SYRACUSE UNIVERSITY

Data Modeling Overview

Data Science: Following the Data



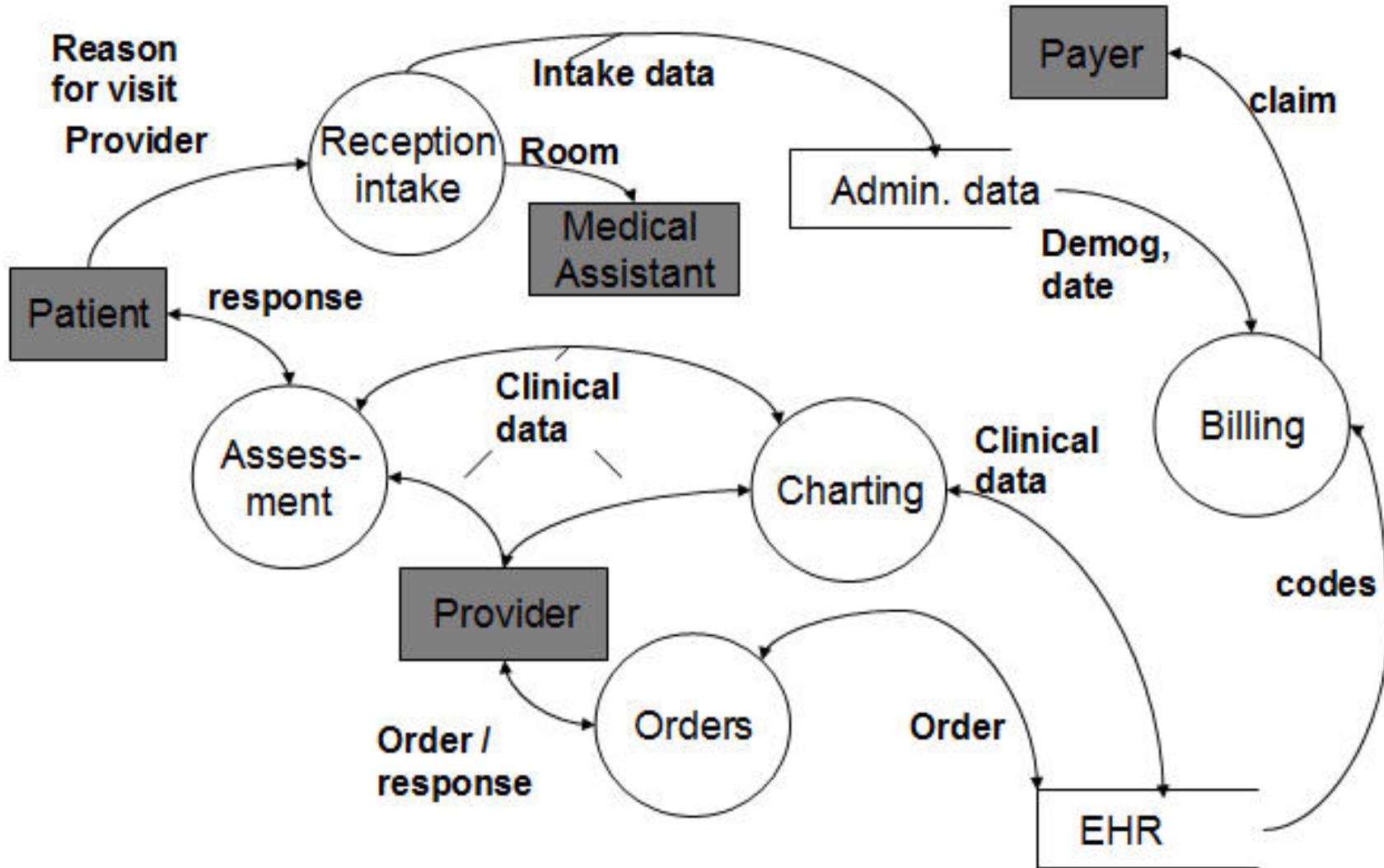
An old adage in detective work is to “follow the money.” In data science, one key to success is to “follow the data.” In most cases, a data scientist will not help to design an information system from scratch. Instead, there will be several or many legacy systems where data resides; a big part of the challenge to the data scientist lies in integrating those systems.

Introduction to Data Models & Systems

- Context for more functional use of R
 - Systems Analysis & Design 101
 - Process model (data flow diagram)
 - Data model (entity relationship diagram)
 - Data model (star schema)
 - Graphical user interface (GUI)
- Reference to the “systems type” pyramid

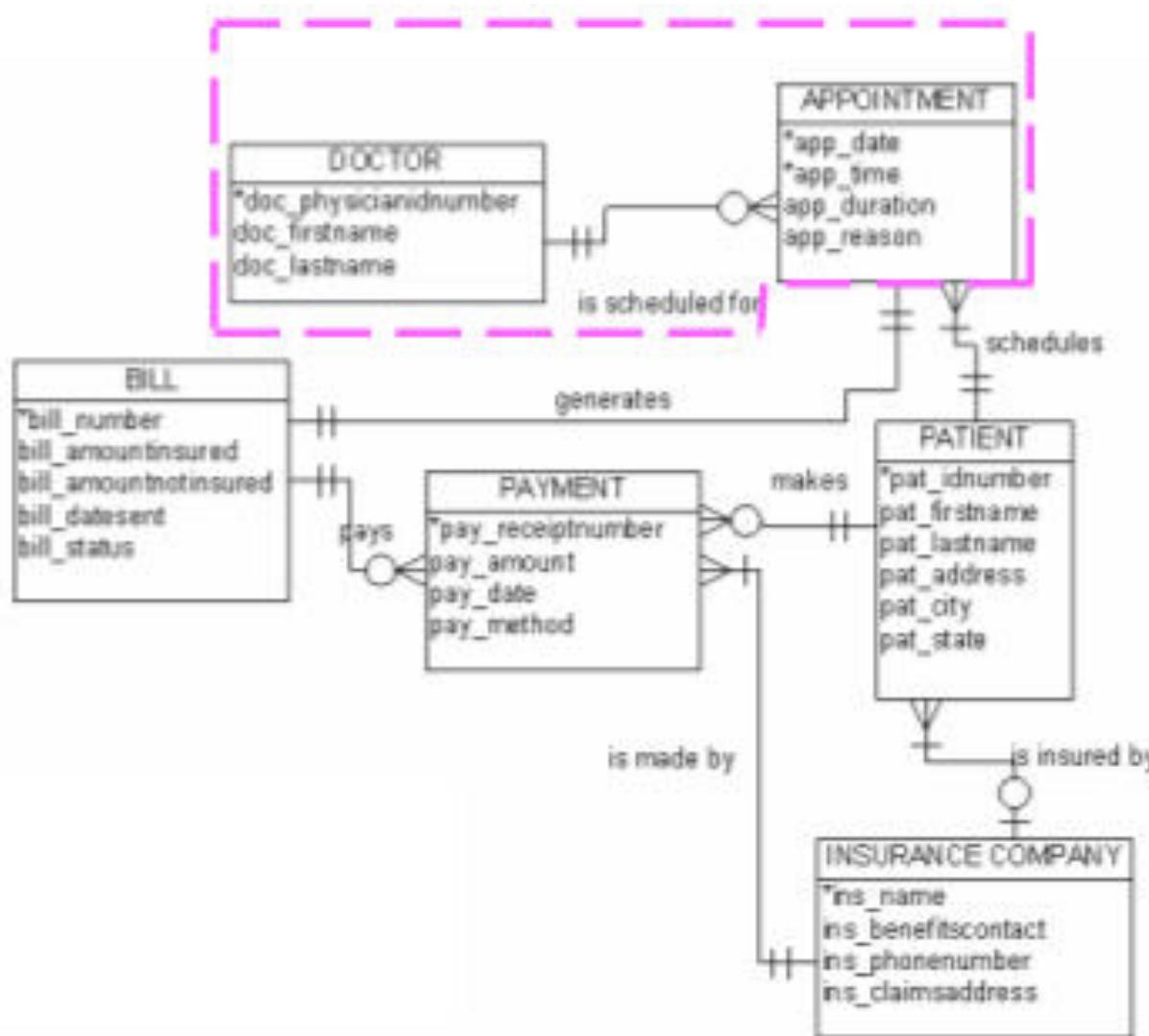
Note that might be a review for many of you

Data Flow Diagram (DFD)

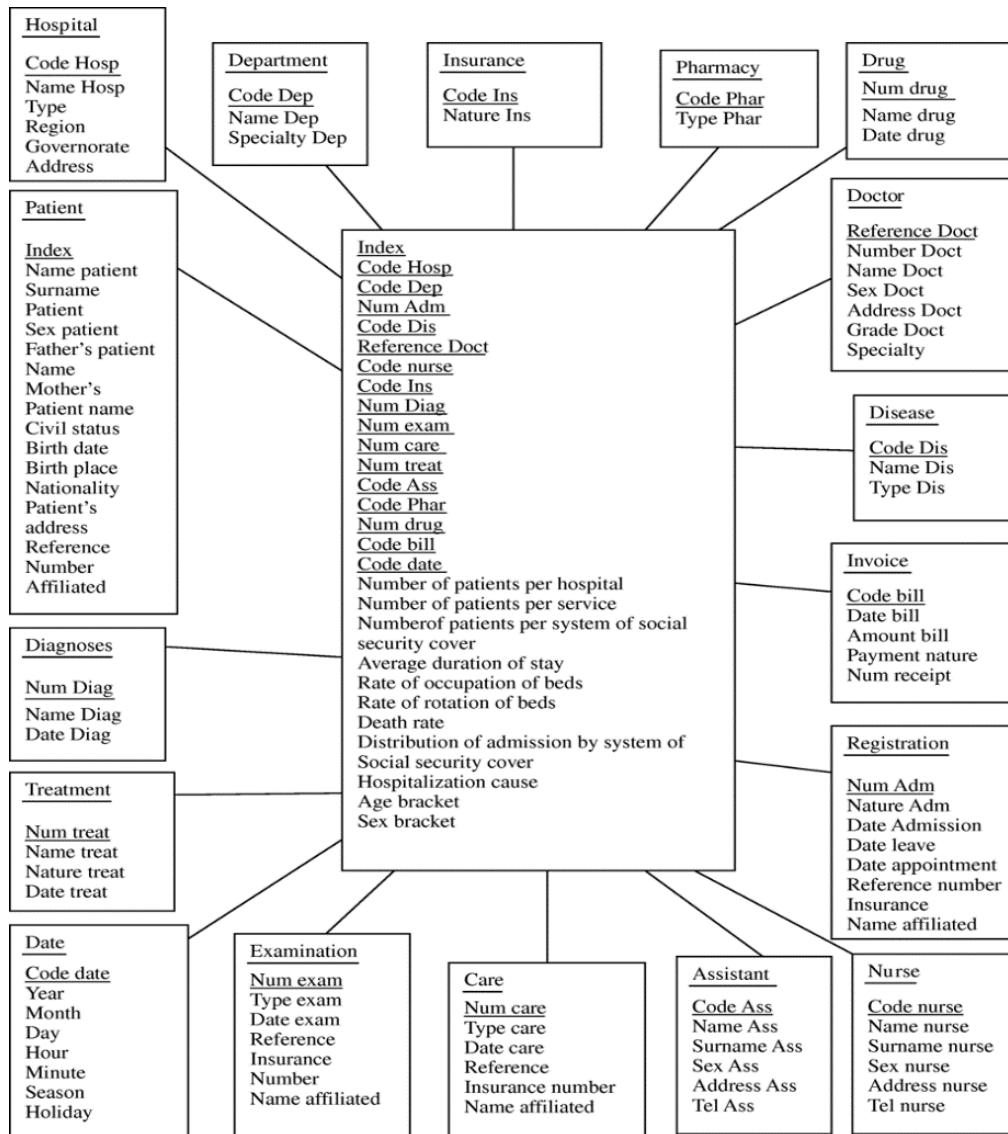


Entity Relationship Diagram (ERD)

A doctor can be scheduled for many appointments but may not have any scheduled at all. Each appointment is scheduled with exactly 1 doctor.

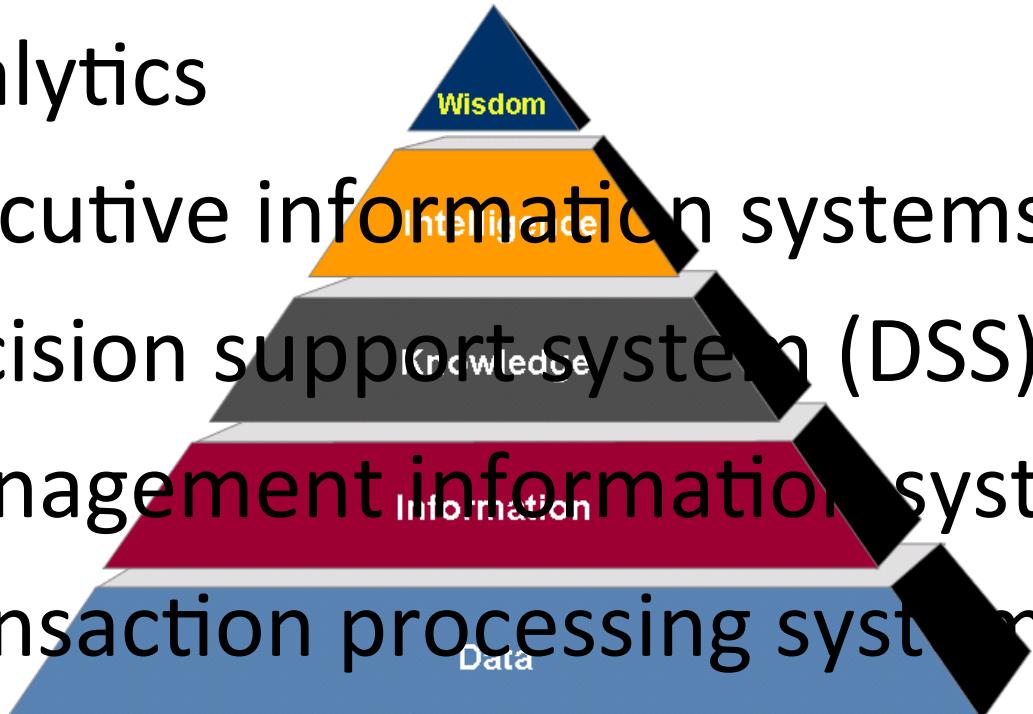


Star Schema



Information System Types

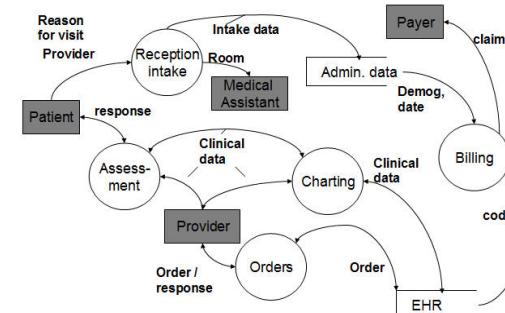
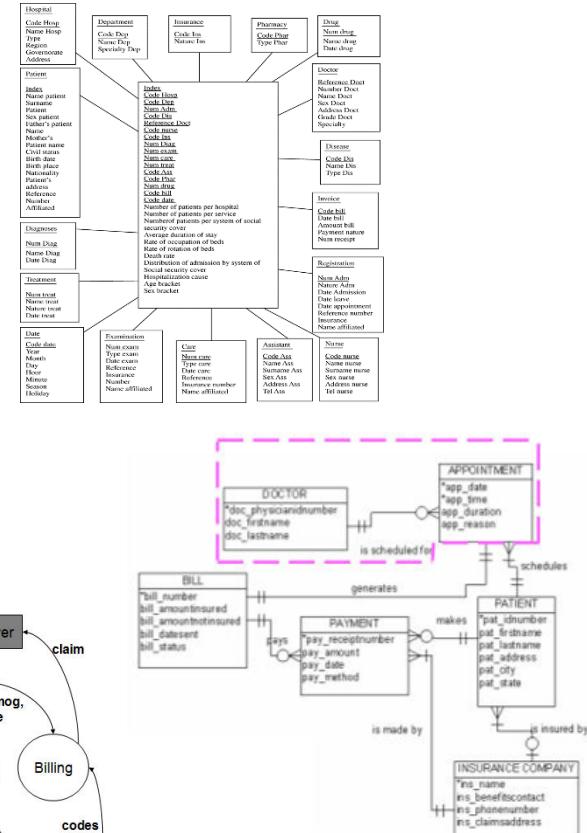
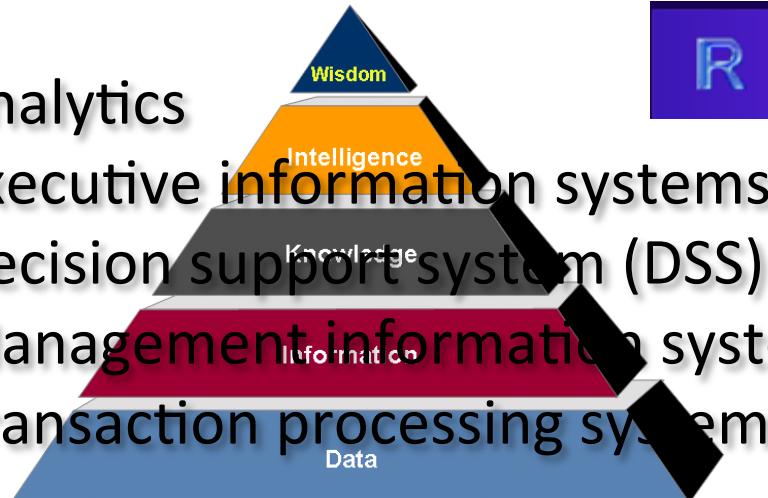
- Analytics
- Executive information systems (EIS)
- Decision support system (DSS)
- Management information system
- Transaction processing system



Data Science

Information Systems Types

- Analytics
- Executive information systems (EIS)
- Decision support system (DSS)
- Management information system
- Transaction processing system



Question

Why do data modeling—why is it useful?

Rows and Columns

Rows and Columns



One of the most basic and widely used methods of representing data is to use rows and columns, where each row is a case or instance and each column is a variable and attribute. Most spreadsheets arrange their data in rows and columns, although spreadsheets don't usually refer to these as cases or variables. R represents rows and columns in an object called a *data frame*.

Thinking About Data

- Know your data
 - Context
 - Content
 - Mode
- Data organization to facilitate R analysis
 - Rows and columns
 - Consistent mode type by attribute/variable

An Example Dataset: Context

NAME	AGE	GENDER	WEIGHT
Dad	43	Male	188
Mom	42	Female	136
Sis	12	Female	83
Bro	8	Male	61
Dog	5	Female	44

An Example Dataset: Characteristics

Two-dimensions: rows and columns

NAME	AGE	GENDER	WEIGHT
Dad	43	Male	188
Mom	42	Female	136
Sis	12	Female	83
Bro	8	Male	61
Dog	5	Female	44

An Example Dataset: Characteristics

- Rows (data)
 - Cases
 - Instances
 - Observations

NAME	AGE	GENDER	WEIGHT
Dad	43	Male	188
Mom	42	Female	136
Sis	12	Female	83
Bro	8	Male	61
Dog	5	Female	44

Note: Name Age Gender Weight is **not** a data row.

An Example Dataset: Characteristics

- Columns (data)
 - Variable name
 - Attributes
 - Variables

NAME	AGE	GENDER	WEIGHT
Dad	43	Male	188
Mom	42	Female	136
Sis	12	Female	83
Bro	8	Male	61
Dog	5	Female	44

An Example Dataset: Characteristics

- Columns (data)
 - **Attributes**
 - **Variables**
 - Variable name

NAME	AGE	GENDER	WEIGHT
Dad	43	Male	188
Mom	42	Female	136
Sis	12	Female	83
Bro	8	Male	61
Dog	5	Female	44

- Note: Name Age Gender Weight are **not** data

An Example Dataset: Characteristics

- Each row has a unique identifier (case label).

NAME	AGE	GENDER	WEIGHT
Dad	43	Male	188
Mom	42	Female	136
Sis	12	Female	83
Bro	8	Male	61
Dog	5	Female	44

An Example Dataset: Characteristics

- Each column has the same type/mode of data.
- Each column has the same number of entries.

NAME	AGE	GENDER	WEIGHT
Dad	43	Male	188
Mom	42	Female	136
Sis	12	Female	83
Bro	8	Male	61
Dog	5	Female	44

Creating a dataset in R

- Data set: How does this get built in R?
 - Create a vector for each variable (column).
 - Create a data frame to combine individual vectors.

NAME	AGE	GENDER	WEIGHT
Dad	43	Male	188
Mom	42	Female	136
Sis	12	Female	83
Bro	8	Male	61
Dog	5	Female	44

Question:

- How would you represent the following data in a data frame?
 - Students in a class
 - For each student, we have a student ID and a GPA.
 - Student 1: ID: N1; GPA: 3.8
 - Student 2: ID: N2; GPA: 4.0
 - Student 3: ID: N3; GPA: 3.3
 - Student 4: ID: N4; GPA: 3.5
 - Student 5: ID: N5; GPA: 3.9
- Create a grid (table) to show how you would represent this information. (Submit a simple table as a spreadsheet.)

Answer:

- How would you represent the following data in a data frame?

Student ID	Student GPA
N1	3.8
N2	4.0
N3	3.3
N4	3.5
N5	3.9

Data Frames in R

Creating a Dataframe in R

The respective variable columns have been built as vectors and displayed below.

R Gui (32-bit) - [R Console]

R File Edit View Misc Packages Windows Help

[File, Edit, View, Misc, Packages, Windows, Help, Stop, Help]

```
> myFamilyNames <- c("Dad", "Mom", "Sis", "Bro", "Dog")
> myFamilyNames
[1] "Dad" "Mom" "Sis" "Bro" "Dog"
> myFamilyAges <- c(43, 42, 12, 8, 5)
> myFamilyAges
[1] 43 42 12 8 5
> myFamilyGenders <- c("Male", "Female", "Female", "Male", "Female")
> myFamilyGenders
[1] "Male" "Female" "Female" "Male" "Female"
> myFamilyWeights <- c(188, 136, 83, 61, 44)
> myFamilyWeights
[1] 188 136 83 61 44
> myFamily <- data.frame(myFamilyNames, myFamilyAges, myFamilyGenders, myFamilyWeights)
> |
```

The columns have been combined and assigned a label via the `data.frame` function.

Viewing a Dataframe

Display the contents of the data object MyFamily.

```
> myFamily <- data.frame(myFamilyNames, myFamilyAges, myFamilyGenders, myFamilyWeights)
> myFamily
myFamilyNames myFamilyAges myFamilyGenders myFamilyWeights
1          Dad        43       Male            188
2          Mom        42     Female            136
3          Sis        12     Female             83
4          Bro         8       Male             61
5          Dog         5     Female             44
> |
```

Using the R “Str” (Structure) Command

```
> myFamily
   myFamilyNames myFamilyAges myFamilyGenders myFamilyWeights
1      Dad          43        Male         188
2      Mom          42       Female        136
3      Sis          12       Female         83
4      Bro           8        Male          61
5      Dog           5       Female         44
> str(myFamily)
'data.frame': 5 obs. of 4 variables:
 $ myFamilyNames : Factor w/ 5 levels "Bro", "Dad", "Dog", ...: 2 4 5 1
 $ myFamilyAges  : num 43 42 12 8 5
 $ myFamilyGenders: Factor w/ 2 levels "Female", "Male": 2 1 1 2 1
 $ myFamilyWeights: num 188 136 83 61 44
> |
```

What does
the structure
function tell
us about the
data object
myFamily?

- Confirmation that MyFamily is a data frame;
- MyFamily has five observations (cases/instances) and four variables.
- “\$” for each variable/component column with descriptive information.
- Each of the variables has a mode or type (same mode within a variable/column).
- Variable is either a “factor” or “num”.
- “Factor” variable has a “level”.
- “Level” describes the options within a variable.
- “num” variable indicates “numeric”.

Using the R Summary Command

```
> summary(myFamily)
   myFamilyNames  myFamilyAges  myFamilyGenders  myFamilyWeights
   Bro:1          Min.   : 5      Female:3        Min.   : 44.0
   Dad:1          1st Qu.: 8     Male   :2        1st Qu.: 61.0
   Dog:1          Median :12    Median :83.0
   Mom:1          Mean   :22    Mean   :102.4
   Sis:1          3rd Qu.:42    3rd Qu.:136.0
                  Max.   :43    Max.   :188.0
```

What does the
summary
function tell us
about the data
object
myFamily?

> |

- “Factor” variables list variable names (MyFamilyNames, myFamilyGenders, MyFamilyWeights) along with the number of occurrences of cases that are coded within that factor.
- Numeric variables have six different calculated quantities that help summarize the variable:
 - Min—minimum or lowest value of all cases
 - 1st Qu—dividing line at the top of the 1st quartile
 - Median—value of the case that splits the whole group in half
 - Mean—numeric average
 - 3rd Qu—3rd quartile
 - Max—max value of all cases

Accessing Dataframes as a Matrix

returns the data in the first row and first column

```
> myFamily[1,1]
```

#Returns the first row

```
> myFamily[1,]
```

#Returns the first column

```
> myFamily[,1]
```

#Returns everything but the first row (deletes first row)

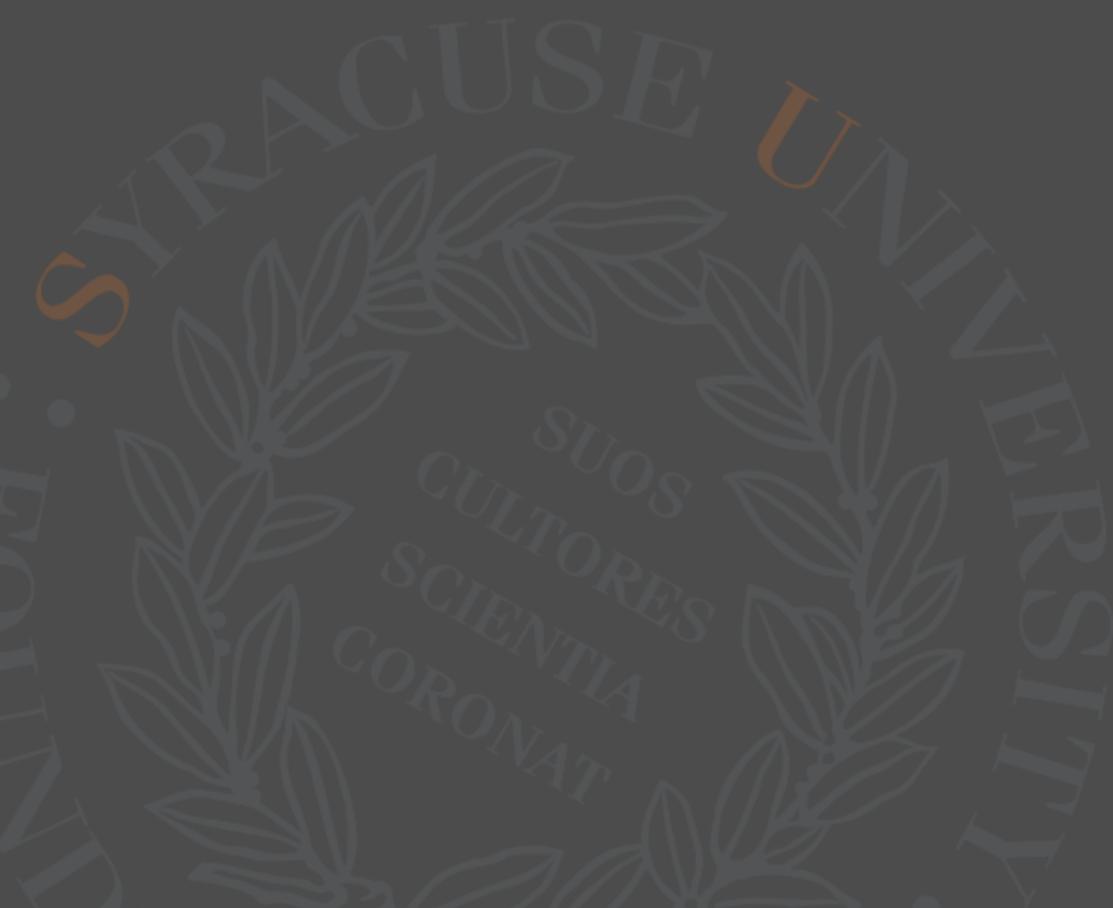
```
> myFamily[-1,]
```

#Returns everything but the first column

```
> myFamily[,-1]
```

R Takeaways

- A **vector** is a list of elements/things
- All the vectors things are the same type (**mode**)
- Data is in a **rectangular format** (rows & columns)
- A **data frame** stores these rectangular data sets
- **data.frame()** organizes vectors into a data frame
- **str()** and **summary()** provide info on a data frame
- A **factor** organizes groups of observations
- **Quartiles** divide a sorted vector into 4 groups.
- **Min()** and **max()** measure “**dispersion**”
- **Mean()** and **median()** measure “**central tendency**”



School of Information Studies
SYRACUSE UNIVERSITY

Writing Functions

Using R Functions

- R functions
 - Function from previous chapters
→ mean is an example (used on a vector)
 - Format
 - Function name: **mean**
 - Argument/input to function: any vector
 - Function **mean** returns the **mean** value of that vector
 - Bundle of code that can be used over again without retying

Writing R Functions

Writing a ‘MyMode’ function:

```
MyMode <- function(myVector) # create function MyMode  
{  
  return(myVector)  
}
```

Function name

Input argument

Send back result
of function

Curly brackets
contain function code
{ }

Writing a MyMode Function

```
> tinyData <- c(1,2,1,2,3,3,3,4,5,4,5) # create vector tinyData  
> tinyData  
[1] 1 2 1 2 3 3 3 4 5 4 5  
  
> MyMode(tinyData) #pass argument tinyData to function MyMode  
[1] 1 2 1 2 3 3 3 4 5 4 5  
# MyMode function designed to display  
content of argument passed to it
```

Writing a MyMode Function

```
MyMode <- function(myVector)
{
  uniqueValues <- unique(myVector)
  return(uniqueValues)
}
```

add unique function to
MyMode
Note: return argument
uniqueValues

```
> MyMode(tinyData)  
[1] 1 2 3 4 5
```

execute MyMode with new
unique function. Unique took
out redundancies from tinyData
vector passed to it.

Writing a MyMode Function

```
MyMode <- function(myVector)
```

```
{
```

```
uniqueValues <- unique(myVector)
```

```
uniqueCounts <- tabulate(myVector)
```

```
return(uniqueCounts)
```

```
}
```

add tabulate
function to MyMode
Note: uniqueCounts is
being returned

```
> MyMode(tinyData)
```

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

MyMode returns count
of how many times each
unique value of tinyData
occurs

Writing a MyMode Function

INDEX	1	2	3	4	5
uniqueValues	1	2	3	4	5
uniqueCounts	2	2	3	2	2

A red arrow points from the text "Most frequently occurring item is largest number in this row" to the value "3" in the third column of the "uniqueCounts" row. The value "3" is also circled in red.

Most frequently occurring item is largest number in this row

Writing a MyMode Function

```
MyMode <- function(myVector)
{
  uniqueValues <- unique(myVector)          # add which.max code
  uniqueCounts <- tabulate(myVector)
  return(uniqueValues[which.max(uniqueCounts)])
}                                              # return the uniqueValue that has the highest uniqueCount associated with it
```

```
> tinyData                                     # display content of tinyData
[1] 1 2 1 2 3 3 3 4 5 4 5
> MyMode(tinyData)                            # execute MyMode
[1] 3                                         uniqueValue that has the highest uniqueCount associated with it
```

Testing MyMode

```
> tinyData<-c(tinyData,5,5,5)          # add additional values to tinyData  
> tinyData                            # display content of tinyData  
[1] 1 2 1 2 3 3 3 4 5 4 5 5 5 5  
> MyMode(tinyData)  
[1] 5                                     uniqueValue that has the highest uniqueCount associated with it
```

Testing MyMode

```
tinyData<-c(tinyData,1,1,1)          # add additional values to tinyData  
> tinyData                            # display content of tinyData  
[1] 1 2 1 2 3 3 3 4 5 4 5 5 5 5 1 1 1  
> MyMode(tinyData)  
[1] 1                                     uniqueValue that has the highest uniqueCount associated with it
```

Issue:

- tinyData contains 5 1's and 5 5's; however our MyMode function is only returning 1 as the uniqueValue with the highest uniqueCount associated with it.
- which.max returns the first maximum it finds.
- We'll see a solution to this toward the end of our discussion; stay tuned.

Testing MyMode – Finding an Error

```
> tinyData<-c(tinyData,9,9,9,9,9,9,9,9)  
# add additional values to tinyData  
> MyMode(tinyData) # execute MyMode  
[1] NA # ????????????
```

What's causing the issue?

Writing Functions

Writing Functions – Fixing MyMode

```
> tinyData<-c(tinyData,9,9,9,9,9,9,9,9)  
# add additional values to tinyData  
> MyMode(tinyData) # execute MyMode  
[1] NA # ????????????
```

What's causing the issue?

Writing Functions – Fixing MyMode

```
> tinyData<-c(tinyData,9,9,9,9,9,9,9)  
> MyMode(tinyData)  
[1] NA
```

```
> tabulate(tinyData)  
[1] 5 2 3 2 5 0 0 0 7
```

tabulate returns count of how many times each unique value of tinyData occurs. Note 0's for 6, 7, 8 that were not present.

Writing Functions – Fixing MyMode

```
> tinyData<-c(tinyData,9,9,9,9,9,9,9)  
> MyMode(tinyData)  
[1] NA
```

```
> tabulate(tinyData)  
[1] 5 2 3 2 5 0 0 0 7
```

tabulate returns count of how many times each unique value of tinyData occurs. Note 0's for 6, 7, 8 that were not present.

```
> unique(tinyData)  
[1] 1 2 3 4 5 9
```

unique returns the six unique values in tinyData. It does not match the number of elements being returned by tabulate.

Writing Functions – Fixing MyMode

```
> tinyData<-c(tinyData,9,9,9,9,9,9,9)  
> MyMode(tinyData)  
[1] NA
```

```
> tabulate(tinyData) # tabulate returns count of how many times  
[1] 5 2 3 2 5 0 0 0 7 each unique value of tinyData occurs. Note  
0's for 6, 7, 8 that were not present.
```

```
MyMode <- function(myVector)  
{  
  uniqueValues <- unique(myVector)  
  uniqueCounts <- tabulate(myVector)  
  return(uniqueValues[which.max(uniqueCounts)])  
}  
# return the uniqueValue that has the highest uniqueCount associated with it
```

Writing Functions – Fixing MyMode

```
MyMode <- function(myVector)
{
  uniqueValues <- unique(myVector)
  uniqueCounts <- tabulate( match(myVector,uniqueValues))
  return(uniqueValues[which.max(uniqueCounts)])
}
```

```
> MyMode(tinyData)
[1] 9
```

Add new code to ‘tabulate’ such that instead of tabulating every possible value in tinyData, including the ones for which there is no data, only tabulate those items where there is a match between the list of unique values and what is in myVector.

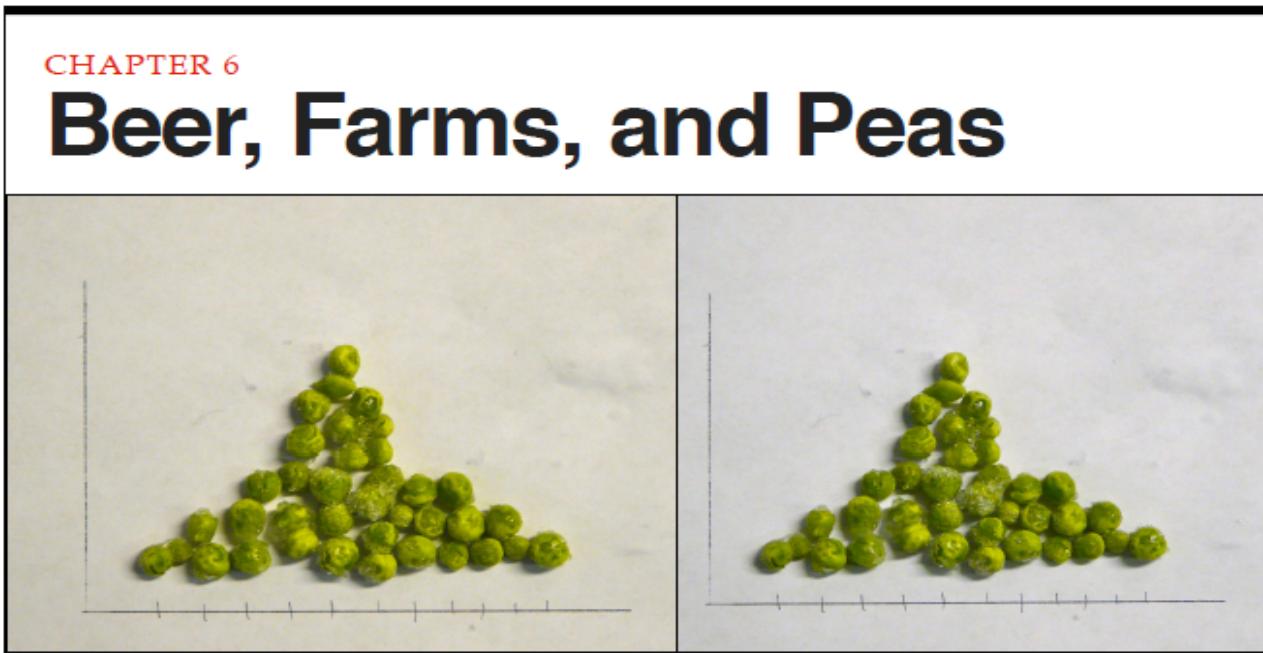
R Function – Most Frequent Value

```
> mfv(tinyData)          # mfv – most frequent value  
[1] 9
```

```
> multiData <- c(1,5,7,7,9,9,10)  
> mfv(multiData)        # create vector and assign to multiData  
[1] 7 9                 # note 2 7's and 2 9's  
> MyMode(multiData)     # mfv accurately displays 7 and 9  
[1] 7                   # our custom MyMode function only displays 7
```

Descriptive Stats Overview

Descriptive Stats Overview



Many of the simplest and most practical methods for summarizing collections of numbers come to us from four guys who were born in the 1800s at the start of the industrial revolution. A considerable amount of the work they did was focused on solving real-world problems in manufacturing and agriculture by using data to describe and draw inferences from what they observed.

Descriptive Stats - History

- Some history: contributions to the statistical party
 - Francis Galton: eugenics and peas
 - Karl Pearson: correlation and regression
 - William Sealy Gosset: small sample statistical techniques and beer
 - Ronald Fisher: analysis of variance and farms

Samples & Uncertainty

- Pervasive tenet: “sample of data”
- Whatever data you have there is more out there.
- Whatever data you have is just a snapshot or sample of what might be out there.
- There is always uncertainty in data, and we need to guard against putting too much stock in what a sample of data has to say.

Example Descriptive Stats

- Descriptive vs. inferential statistics
- Descriptive statistics covered previously
 - Mean
 - Median
 - Range
 - Mode
- New to this discussion
 - Variance
 - Standard deviation

Two Key Measures of Data

Measure
of Central
Tendency

- Mean
- Median
- Mode

Measure
of
Dispersion

- Range
- Variance
- Standard deviation

Question

Why is understanding central tendency
and measure of dispersion useful?

Descriptive Stats Example

Example - State Populations

- Residential population data set
 - State
 - Population value
- Loaded into R
 - USstatePops\$april10census

.Alaska	710,231
.Arizona	6,392,017
.Arkansas	2,915,918
.California	37,253,956
.Colorado	5,029,196
.Connecticut	3,574,097
.Delaware	897,934
.District of Columbia	601,723
.Florida	18,801,310
.Georgia	9,687,653
.Hawaii	1,360,301
.Idaho	1,567,582
.Illinois	12,830,632
.Indiana	6,483,802
.Iowa	3,046,355

Calculate Descriptive Stats Using R

R statistical functions

```
> mean(USstatePops$april10census)  
[1] 6053834
```

```
> median(USstatePops$april10census)  
[1] 4339367
```

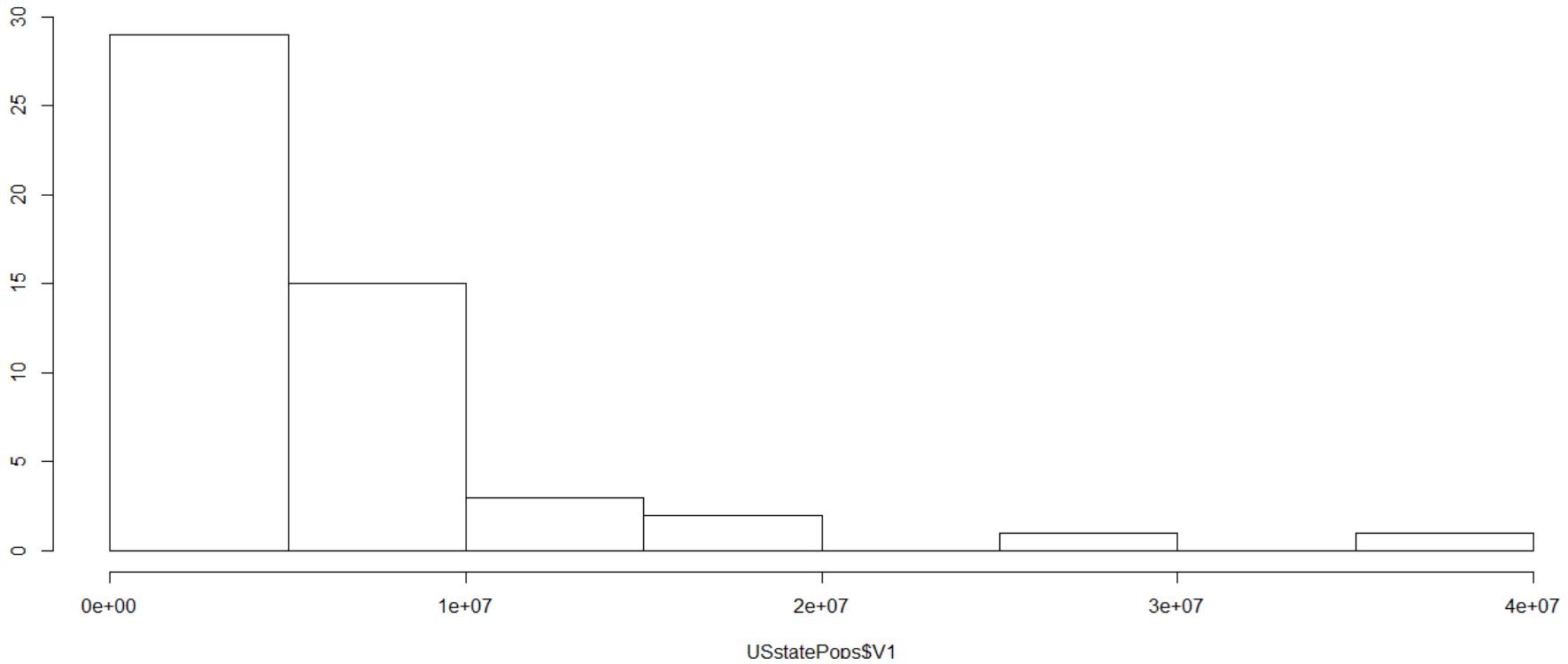
```
> mode(USstatePops$april10census)  
[1] "numeric"
```

Histograms

- Histogram
 - A picture that shows central tendency and dispersion
- R Function
Hist(USstatePops\$april10census)

Example Histogram

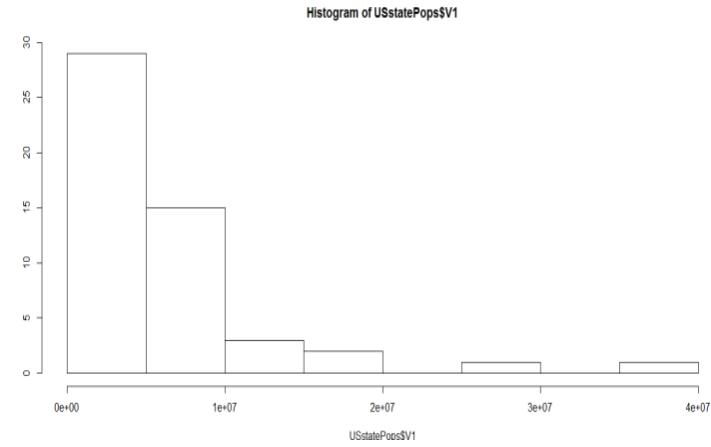
Histogram: `hist(USstatePops$April10census)`



Example Histogram - Explained

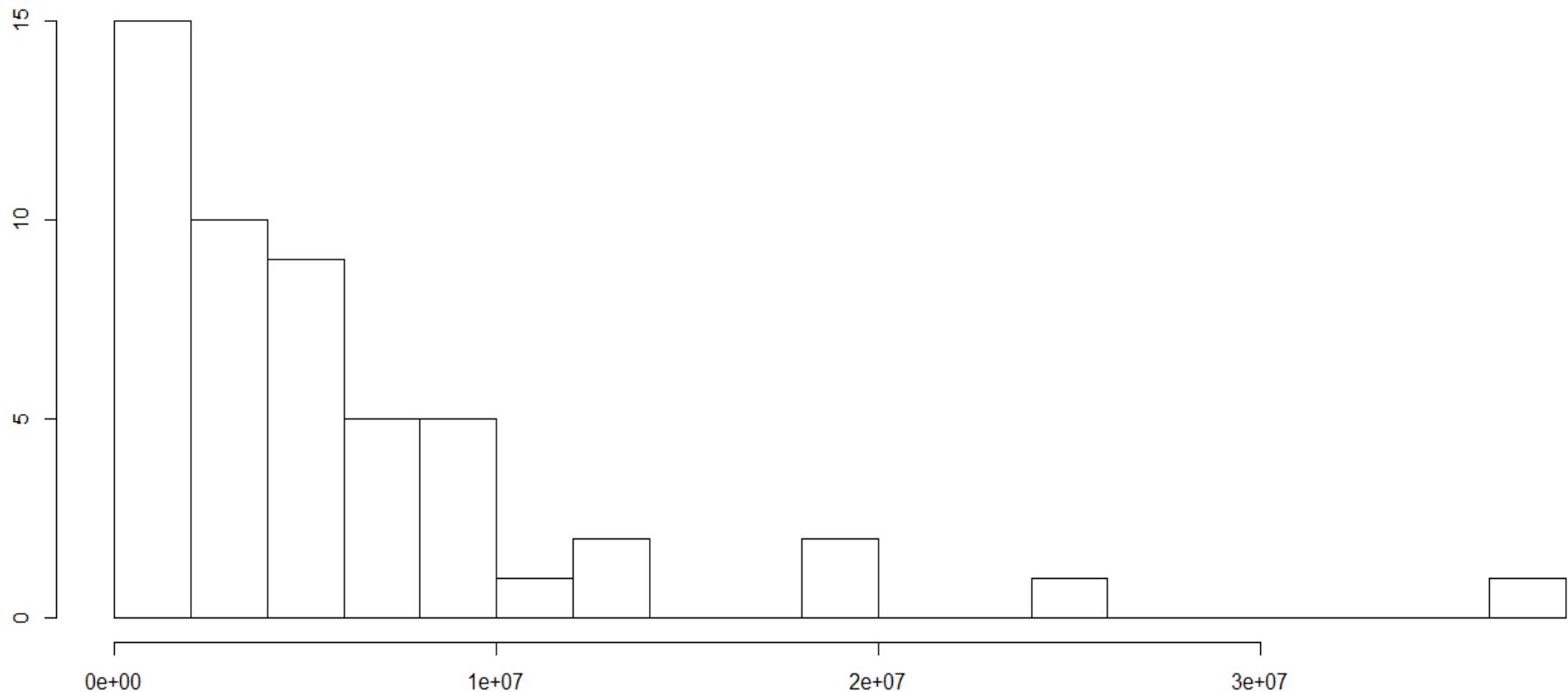
`hist(USstatePops$april10census)`

- Designed to show frequencies
- Interpretation
 - x and y axis
 - Two bars per tick mark
 - 30 states with populations under 5 million
 - Another 10 states with population under 10 million
 - Small number of states with populations greater than 10 million
- Observation: 40 states clustered into the first couple of bars



Example Histogram – More “breaks”

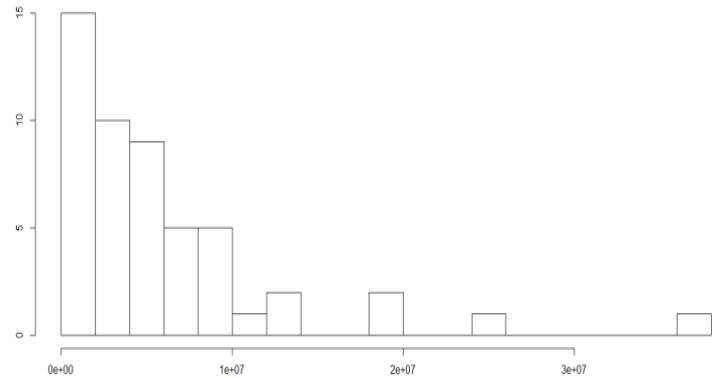
```
hist(USstatePops$april10census, breaks=20)
```



Example Histogram - Explained

```
hist(USstatePops$april10census, breaks=20)
```

- Designed to show more bars
- Interpretation
 - Five bars per tick mark
 - 15 states with populations under 2 million
 - Distribution pattern
 - Reverse-j distribution pattern



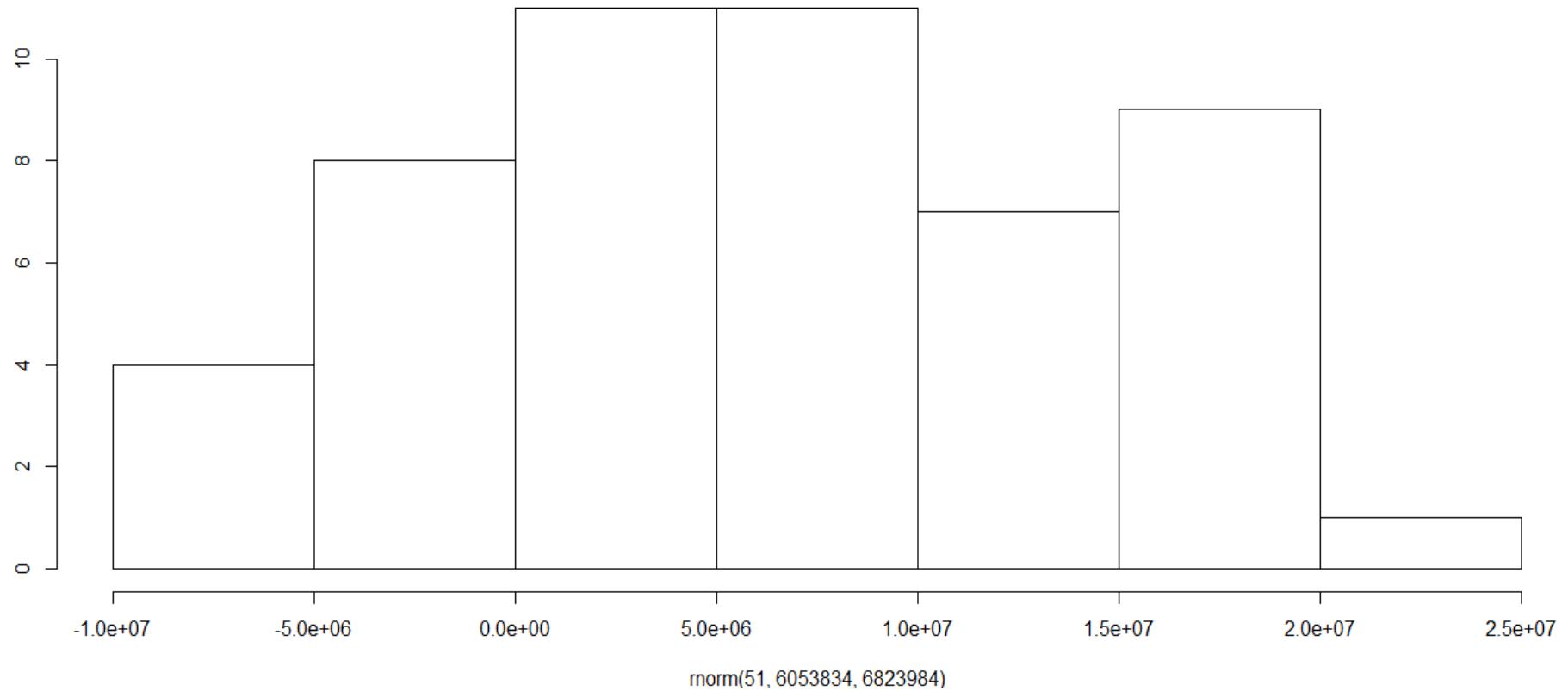
Distributions

Other distributions shapes

- The “bell” or normal distribution
- Meant to represent the most typical (normal) distribution
- Normal shape is that of a bell

Example Normal Distribution

```
Hist(rnorm(51, 6053834, 6823984))
```



Generating Normal Distributions

R components to render the normal distribution of the state population data set

- `sd(USstatePops$april10census)`
6823984
- `mean(USstatePops$april10census)`
605384
- 51 data points
- `hist(rnorm(51, 6043834, 6823984))`
- Function within a function

Generating Normal Distributions

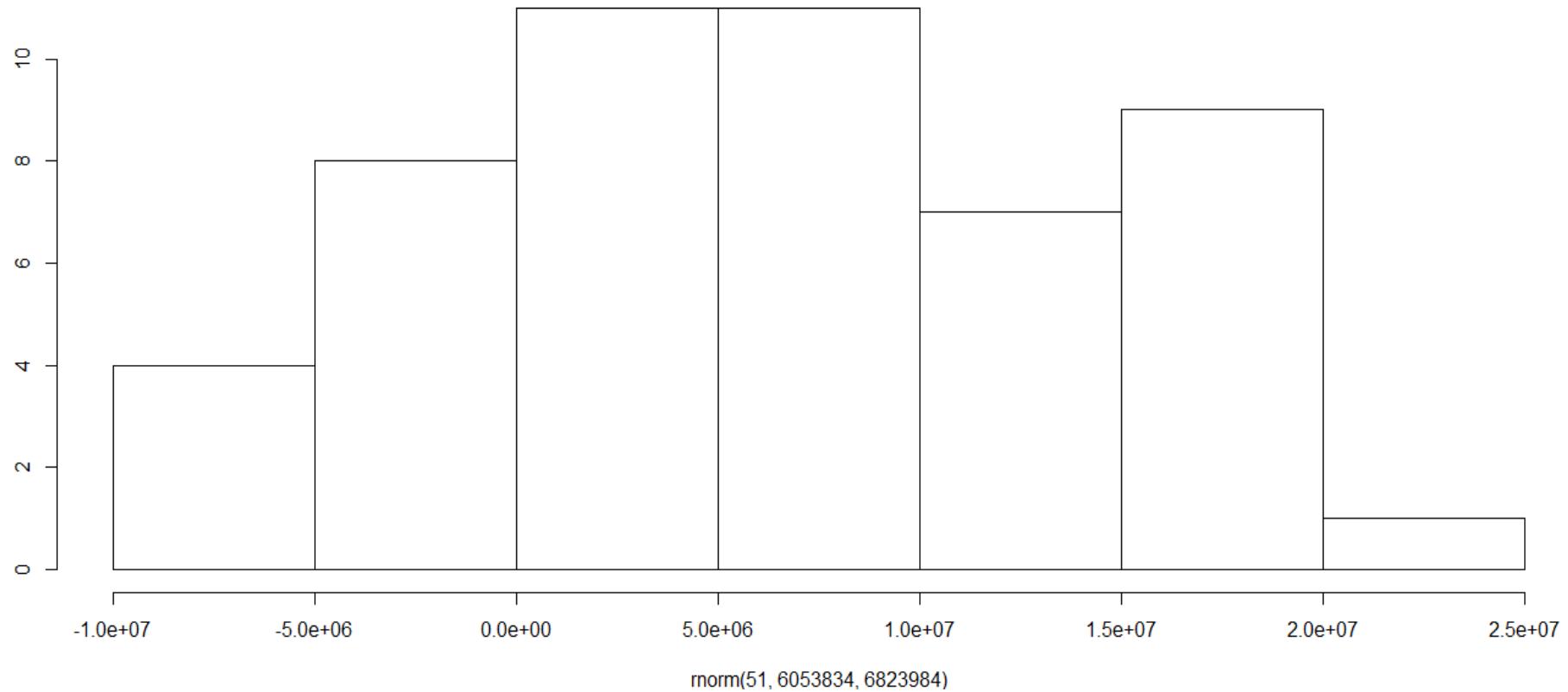
`rnorm(51, 6053834, 6823984)`

```
> rnorm(51, 6053834, 6823984)
[1] 2140927.6 8321903.6 7778823.9 1975788.9 9596056.1 9687365.8 -1796469.9 1731564.3 -600726.4 210103.8 7394635.1 16401844.3 -440114.7
[14] 8094555.1 8925681.5 10091285.1 10068318.1 1436260.6 14582483.8 11291498.0 14076210.0 -2962166.8 -9855689.9 1168526.9 1826509.8 13127648.3
[27] -3398999.6 3643236.9 3841644.3 23247836.8 -3381465.9 8236009.4 10619751.7 12571853.5 2011040.6 9655436.7 10854206.4 5199112.7 10984438.5
[40] -4681428.8 1457661.1 3361793.7 12720280.5 265577.1 -3793762.3 7048391.6 -10050883.7 931311.3 18782698.5 6231469.0 9752514.4
> |
```

`hist(rnorm(51, 6053834, 6823984))`

Generating Normal Distributions

`Hist(rnorm(51, 6053834, 6823984))`



Reviewing Distributions

R takeaways

- Normal distribution is used extensively through applied statistics as a tool for making comparisons
- Key pieces of information to enable comparisons
 - The distribution had a characteristic shape
 - The distribution had a center point, mean
 - A “spread” (variability) which was the standard deviation

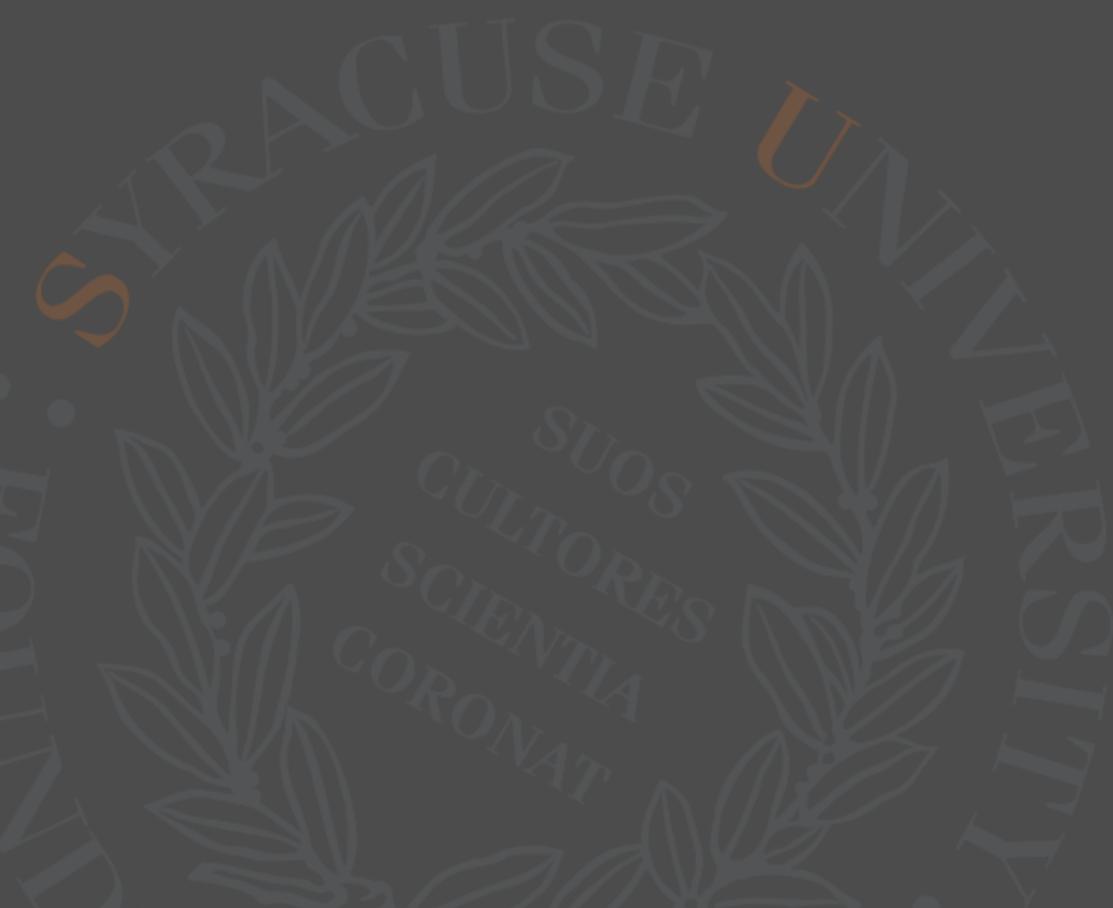
Real World Examples

Data Science Examples

Data Science in the real world

- Data Science is used in many Industries
- Data Science is used in many situations

Provide some examples of
“Data Science in the real world”



School of Information Studies
SYRACUSE UNIVERSITY

Introduction to Sampling

Data Science

Sample in a Jar



Sampling distributions are the conceptual key to statistical inference. Many approaches to understanding sampling distributions use examples of drawing marbles or gumballs from a large jar to illustrate the influence of randomness on sampling. Using the list of U.S. states illustrates how a non-normal distribution nonetheless has a normal sampling distribution of means.

Data Science

- Red gum balls, blue gum balls
- Same ratio of each in a jar
 - Draw a sample of eight (one draw)
 - What mix of red and blue gum balls will we get with one draw? → Really don't know.
- Forces of “randomness” driving uncertainty
- “Long run test” → multiple draws

DRAW	# RED
1	5
2	3
3	6
4	2

Data Science

- “Drawing” process
 - Population (gum balls)
 - Sample
 - Distribution
- Use R to draw samples
 - `sample(USstatePops$april10census, size=16, replace=TRUE)`
 - 12702379 19378102 8791894 19378102 9535483 6346105
4533372 5029196 25145561 6392017 19378102 6483802
8001024 8001024 12830632 814180
- Use R to calculate mean of the sample
 - `mean(sample(USstatePops$april10census, size=16, replace=TRUE))`
 - 5513472

DRAW	# RED
1	5
2	3
3	6
4	2

Data Science

Question:

- Why is sampling from a population important?
- What are some key things to think about when sampling?

Replicating Samples

Data Science

- Not interested in one sample but what happens over time, i.e., many samples
- Replication
 - `replicate(4,mean(sample(USstatePops$april10census,size=16,replace=TRUE)),simplify=TRUE)`
 - 5234752 5978035 5876217 4222350

Data Science

- `mean(replicate(400, mean(sample(USstatePops$april10census, size=16, replace=TRUE)), simplify=TRUE))`

> 6014258

- Interpretation
 - Draw 400 samples of size 16 from our state population.
 - Calculate the mean from each sample and keep it in a list.
 - Calculate the mean of the 400 sample means.
 - Calculated mean of means is off by 39,577.
 - $6,053,835 \text{ (mean of 51 states)} - 6,014,258 \text{ (mean of means)} = 39,577$
 - $39,577 / 6,053,835 = .65\% \text{ error}$

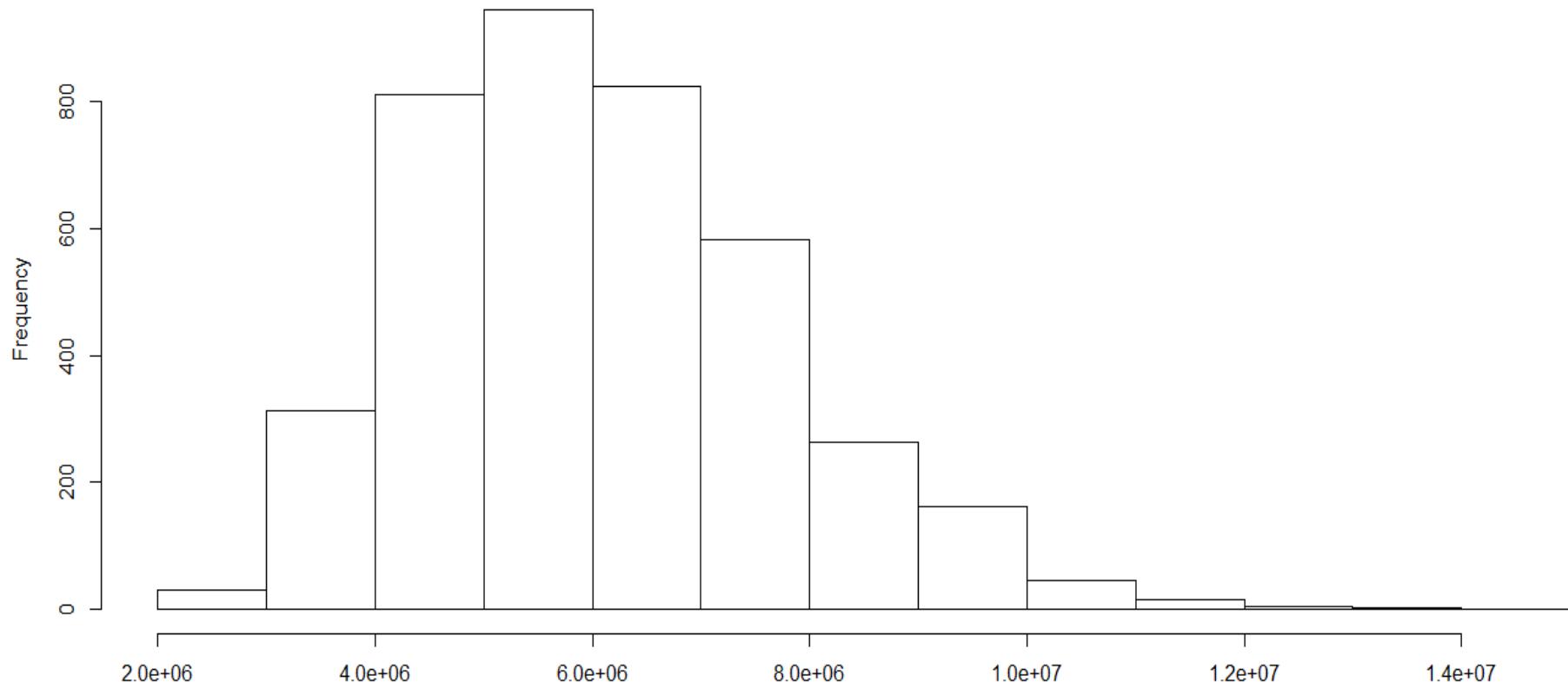
Data Science

- `mean(replicate(4000,mean(sample(USstatePops$april10census,size=16,replace=TRUE)), simplify=TRUE))`
 - 6053534

6,053,835 (mean of 51 states) – 6,053,534 (mean of means) = 301
- Display distribution of 4000 means via a histogram as frequencies
 - `hist(replicate(4000,mean(sample(USstatePops$april10census,size=16,replace=TRUE)),simplify=TRUE))`

Data Science

Histogram of replicate(4000, mean(sample(USstatePops\$V1, size = 16, replace = TRUE)), simplify = TRUE)



Data Science

- Law of large numbers
 - If you run a statistical process a large number of times, it will converge on a stable result.
- Central limit theorem
 - When we look at sample means and take into account the “law of large numbers,” the distribution of sampling means starts to create a bell-shaped or normal distribution, and the center of that distribution, the mean of the sample means, gets close to the population mean.

Data Science

- Sampling distribution

- Save one distribution sample

```
SampleMeans <- replicate(10000, mean(sample(USstatePops$april10census,  
size=120, replace=TRUE)), simplify=TRUE)
```

- Length(SampleMeans)

- 10000

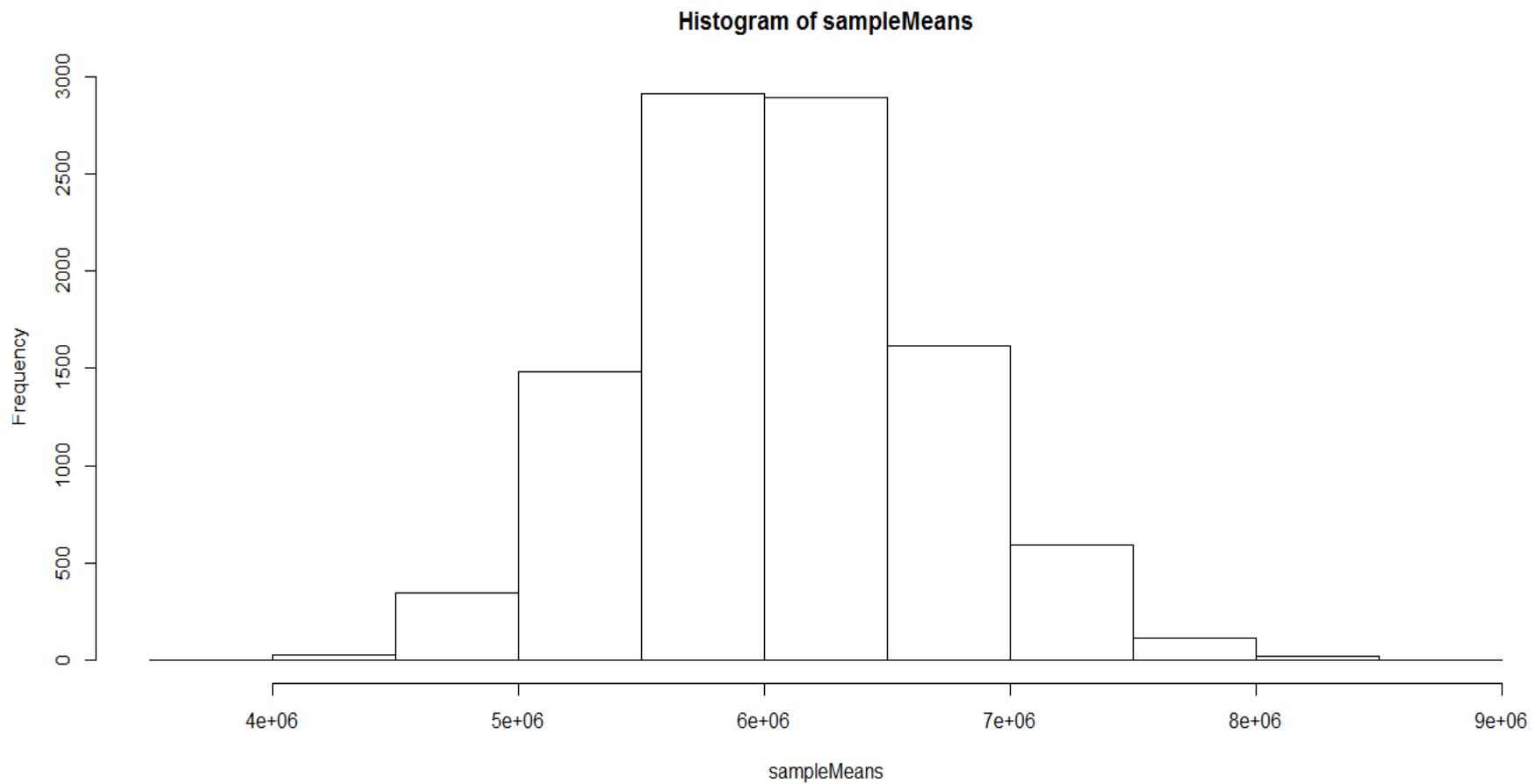
- mean(SampleMeans)

- 6058734

- Histogram

- Hist(SampleMeans)

Data Science



Data Science

- Sampling distribution
 - Summary(SampleMeans)

```
> summary(sampleMeans)
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
3922000	5625000	6032000	6059000	6461000	8745000

- Interpretation
 - Min vs. mean
 - Max vs. mean
 - Median vs. mean
 - Quartiles

Data Science

- Sampling distribution
 - Quantile function, similar to summary function

```
> quantile(sampleMeans, prob=c(0.25, 0.50, 0.75))
   25%      50%      75%
5625329 6031972 6461389
> quantile(sampleMeans, prob=c(0.05, 0.95))
   5%      95%
5078829 7149099
> quantile(sampleMeans, prob=c(0.025, 0.975))
  2.5%    97.5%
4917638 7351685
> quantile(sampleMeans, prob=c(0.01, 0.99))
   1%      99%
4714055 7574380
> quantile(sampleMeans, prob=c(0.005, 0.995))
  0.5%    99.5%
4604962 7761372
.
```

Mystery Samples

Data Science

- MysterySample

```
> MysterySample <- c(3706690, 159358, 106405,  
55519, 53883)
```

```
> mean(MysterySample)  
816731
```

Data Science

- MysterySample: sample of U.S. states or something else?
- Basis of comparison
 - Subsequent USstatePops\$april10census analysis
 - Sampling distribution of means: USstatePops\$april10census represented in vector SampleMeans
 - Compare MysterySample mean to SampleMeans:
“quantile analysis”
 - `mean(SampleMeans)`
 - `quantile(SampleMeans, probs=c(0.25, 0.50, 0.75))`
 - `quantile(SampleMeans, probs=c(0.05, .095))`
 - `quantile(SampleMeans, probs=c(0.01, .099))`
 - is MysterySample mean below 5% mark or above 95% mark
 - is MysterySample mean below 1% mark or above 99% mark

Data Science

- MysterySample: sample of U.S. states or something else?
 - 1% of all the SampleMeans are lower than 4,710,455.
 - Therefore, MysterySample mean of 816,731 would be a rare event.
 - We can infer, tentatively but based on good statistical evidence, that the MysterySample is not a sample of states.
 - Key takeaway
 - The mean of the MysterySample was sufficiently different from a known distribution of means such that we could make an inference that the sample was not drawn from the original population of data.

Data Science

- Basis for most all statistical inference
 - Construct a comparison distribution.
 - Identify a zone of extreme values.
 - Compare new sample of data to the distribution relative to the “extreme” zones.
 - If new sample does fall in the “extreme zone,” you can tentatively conclude that the new sample was obtained from some other source than what you used to create the comparison distribution.

Data Science

- Brief recap
 - Mean() of sampling distribution
 - Sampling distribution shape via hist()
- Need to quantify the distribution spread via sd()
 - sd(SampleMeans)
 - 621569

Data Science

- Standard deviation of the distribution of sampling means, also known as “standard error of the mean”

$sd(\text{population})/\sqrt{\# \text{ of samples}}$

- Alternative to `sd(SampleMeans)` relative to calculating the “standard error of the mean”
 - $sd(\text{USstatePops\$April10census})/\sqrt{120}$
 - 622941
 - Differences due to randomness of the distribution

Data Science

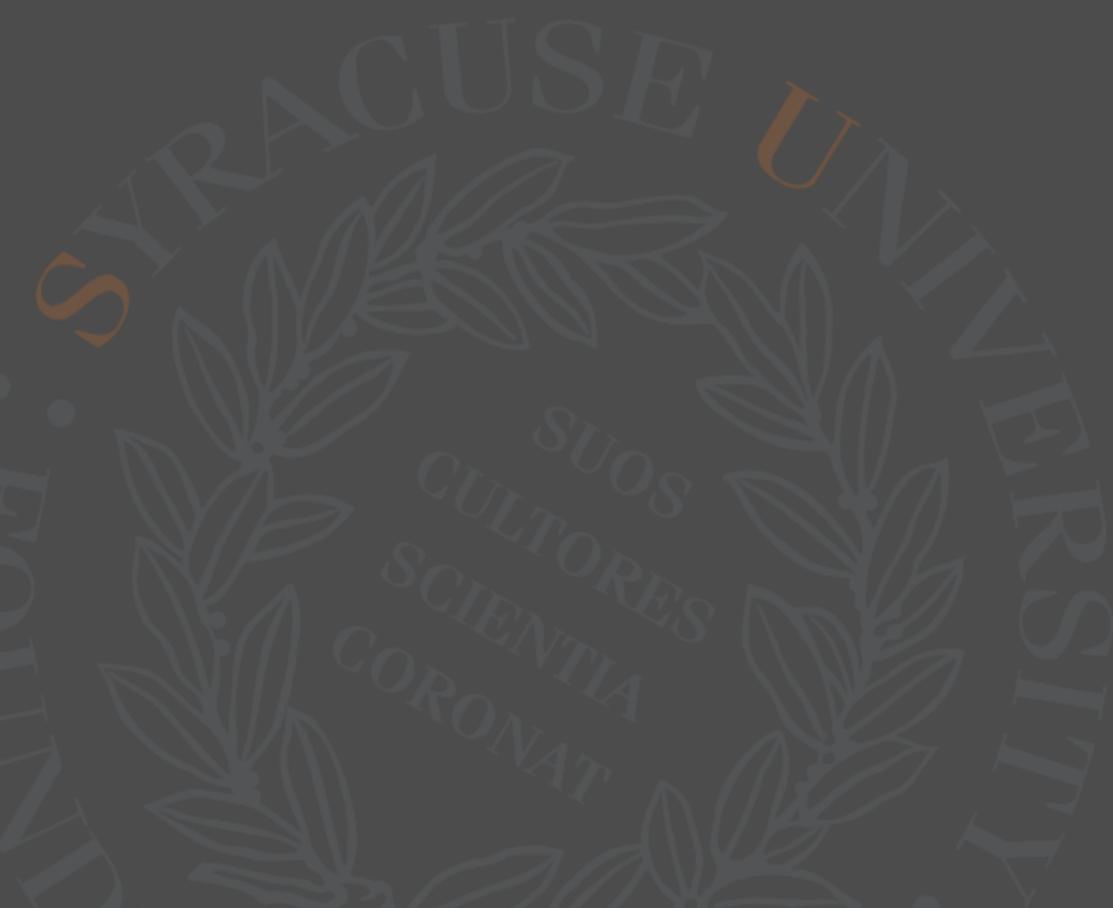
- Alternative to quantile() cut points
 - Use mean and standard error.
 - Two standard deviations down from the mean is the 5% cut point.
 - Two standard deviations up from the mean is the 95% cut point.
 - StdError <-sd(USstatePops\$April10census)/sqrt(120)
 - CutPoint5<-mean(USstatePops\$April10census)-(2 * StdError)
 - CutPoint95<-mean(USstatePops\$April10census)+(2 * StdError)
 - CutPoint5
 - 4807951
 - CutPoint95
 - 7299717

Data Science

- Summary
 - Data set with 51 data points, numbers of people in 51 states
 - Use R to construct distribution of sampling means
 - Highlighted the process of statistical inference
 - Construct a comparison distribution.
 - Identify a zone of extreme values.
 - Compare new sample of data to the distribution relative to the “extreme” zones.
 - If new sample falls in the extreme zone, you can tentatively conclude that the new sample was obtained from some other source than what you used to create the comparison distribution.

Data Science

- Question:
Why is it useful (or when is it useful) to compare two samples?



School of Information Studies
SYRACUSE UNIVERSITY

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

Acknowledgements

1 Introduction

1.1 Imports

1.1.1 Encodings

1.2 Export to text files

1.3 XML

2 Spreadsheet-like data

2.1 Variations on `read.table`

2.2 Fixed-width-format files

2.3 Data Interchange Format (DIF)

2.4 Using `scan` directly

2.5 Re-shaping data

2.6 Flat contingency tables

3 Importing from other statistical systems

3.1 EpiInfo, Minitab, S-PLUS, SAS, SPSS, Stata, Systat

3.2 Octave

4 Relational databases

4.1 Why use a database?

4.2 Overview of RDBMSs

4.2.1 SQL queries

4.2.2 Data types

4.3 R interface packages

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

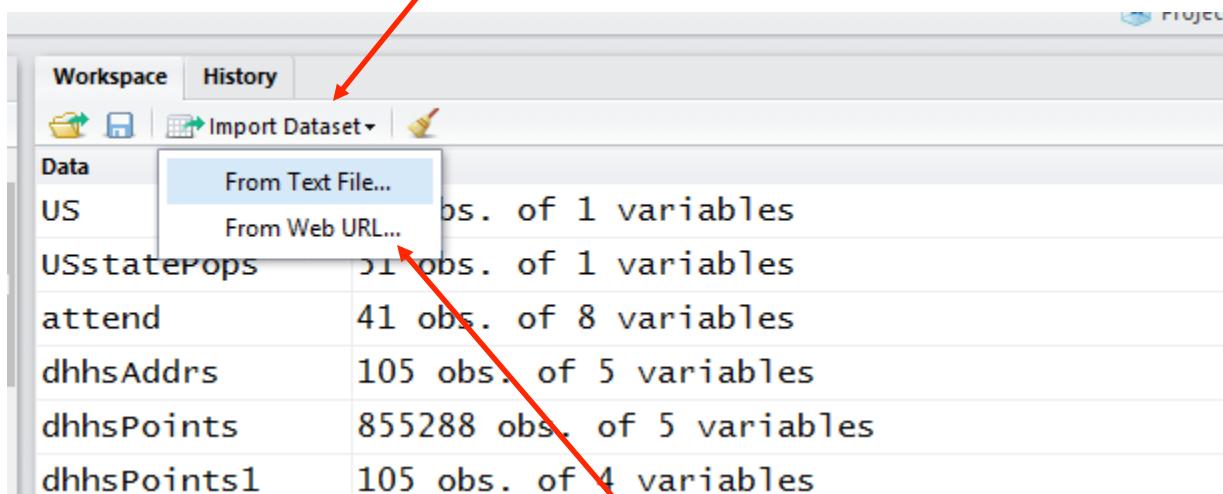
Bates, D.M. and Watts, D.G. (1988), *Nonlinear Regression Analysis and Its Applications*, Wiley, Appendix A1.4.

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

AirPassengers.txt
text file

Import Dataset

Name: AirPassengers

Heading: Yes No

Separator: Whitespace

Decimal: Period

Quote: Double quote ("")

Input File:

Year	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1949	112	118	132	129	121	135	148	148	136	119	104	118
1950	115	126	141	135	125	149	170	170	158	133	114	140
1951	145	150	178	163	172	178	199	199	184	162	146	166
1952	171	180	193	181	183	218	230	242	209	191	172	194
1953	196	236	235	229	243	264	272	237	211	180	201	
1954	204	188	235	227	234	264	302	293	259	229	203	229
1955	242	233	267	269	270	315	364	347	312	274	237	278
1956	284	277	317	313	318	374	413	405	355	306	271	306
1957	315	301	356	348	355	422	465	467	404	347	305	336
1958	340	318	362	348	363	435	491	505	404	359	310	337
1959	360	342	406	396	420	472	548	559	463	407	362	405
1960	417	391	419	461	472	535	622	606	508	461	390	432

AirPassengers.txt

Data Frame:

Year	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep
1949	112	118	132	129	121	135	148	148	136
1950	115	126	141	135	125	149	170	170	158
1951	145	150	178	163	172	178	199	199	184
1952	171	180	193	181	183	218	230	242	209
1953	196	196	236	235	229	243	264	272	237
1954	204	188	235	227	234	264	302	293	259
1955	242	233	267	269	270	315	364	347	312
1956	284	277	317	313	318	374	413	405	355
1957	315	301	356	348	355	422	465	467	404
1958	340	318	362	348	363	435	491	505	404
1959	360	342	406	396	420	472	548	559	463
1960	417	391	419	461	472	535	622	606	508

< >

Import Cancel

Import via Rstudio - Example

The screenshot illustrates the RStudio interface during the import of the 'AirPassengers' dataset. The interface is divided into several panes:

- File Explorer:** Shows the file structure with 'AirPassengers' selected.
- Console:** Displays the R code used to import the data:

```
> AirPassengers <- read.table("C:/Users/G/Desktop/Old  
Laptop/Syracuse_University/IST687 Fidelity/Chapter 14 Storage  
Wars/AirPassengers.txt", header=T, quote="")  
> View(AirPassengers)
```
- Data View:** Shows the 'AirPassengers' dataset as a table with 12 observations and 13 variables, spanning from 1949 to 1960.
- Environment:** Lists available datasets: 'AirPassengers', 'US', 'USstatePops', 'attend', 'dhhsAddrs', 'dhhsPoints', 'dhhsPoints1', 'myFamily', and 'pointData'. The 'AirPassengers' entry is highlighted with an orange circle.
- Packages:** Shows installed packages: 'bitops', 'boot', 'class', and 'cluster'. The 'cluster' package is highlighted with an orange circle.

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](http://www.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>

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

Viewing the testFrame

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

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

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

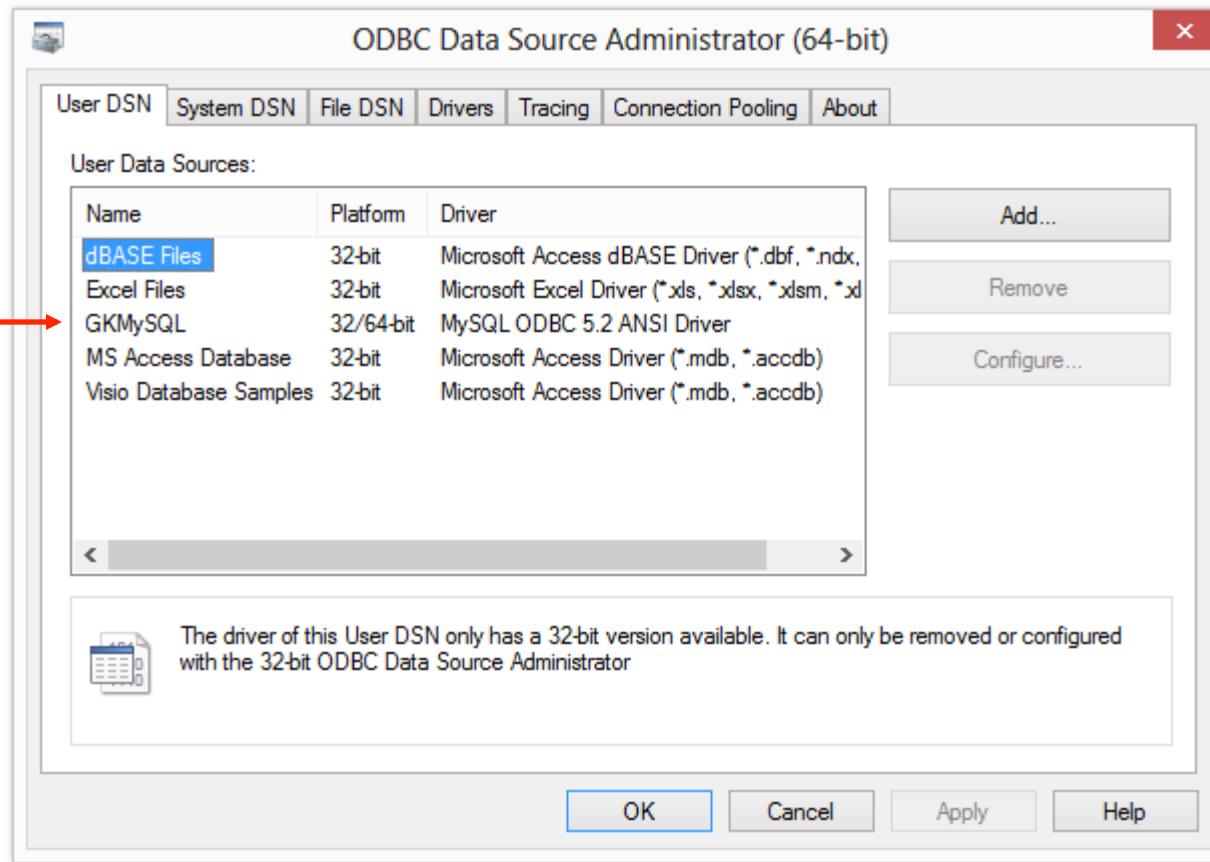
- Database connectivity packages
 - RMySQL
 - ROracle
 - RPostgresSQL
 - RSQLite
 - RMongo
 - RODBC
- Chapter examples
 - RODBC
 - MySQL
 - SQLServer 2012
 - Microsoft Access

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



Environment Prerequisites - MySQL

- MySQL R code

Note: RODBC package must be loaded.

 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
10	10	Tunnex	4	0	12 - 200 ml jars	21.00

	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	
	25	0	0	

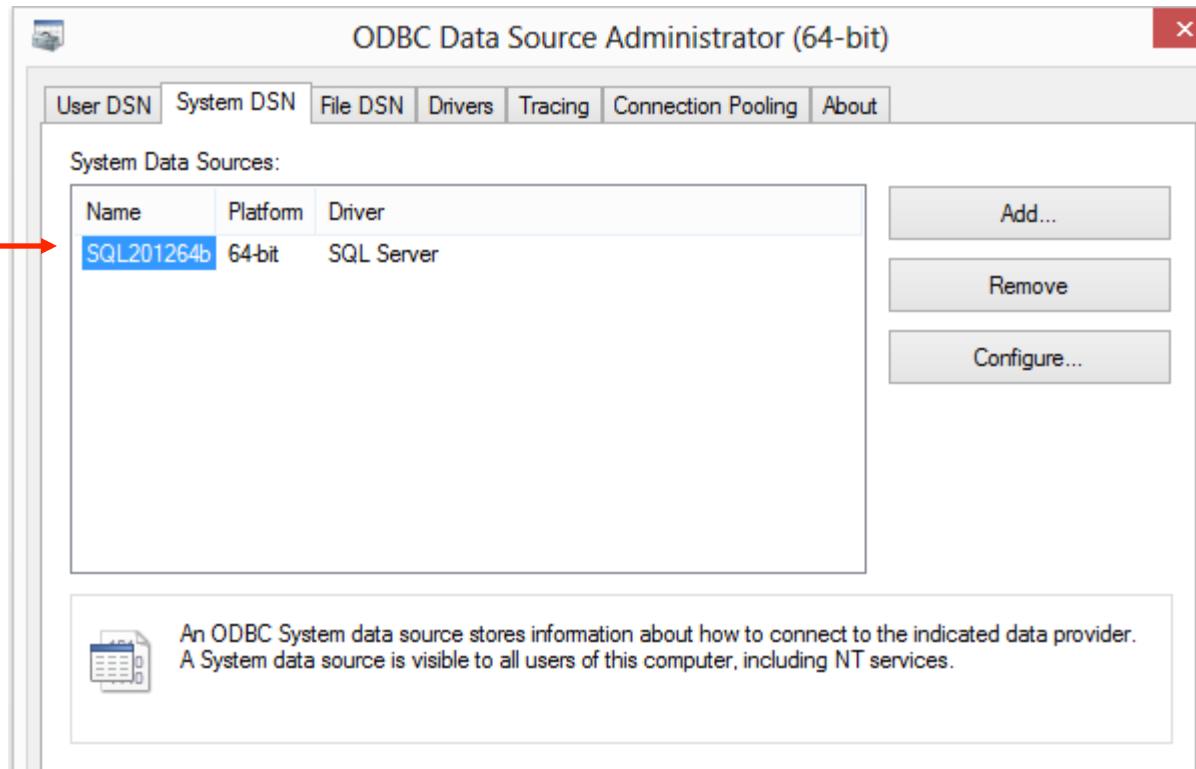
View (datamysql)

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

```
# establish R connection to SQL201264b
>conSQL2012 <- odbcConnect("SQL201264b")SQLServer ODBC name
# 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

School of Information Studies
SYRACUSE UNIVERSITY

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?

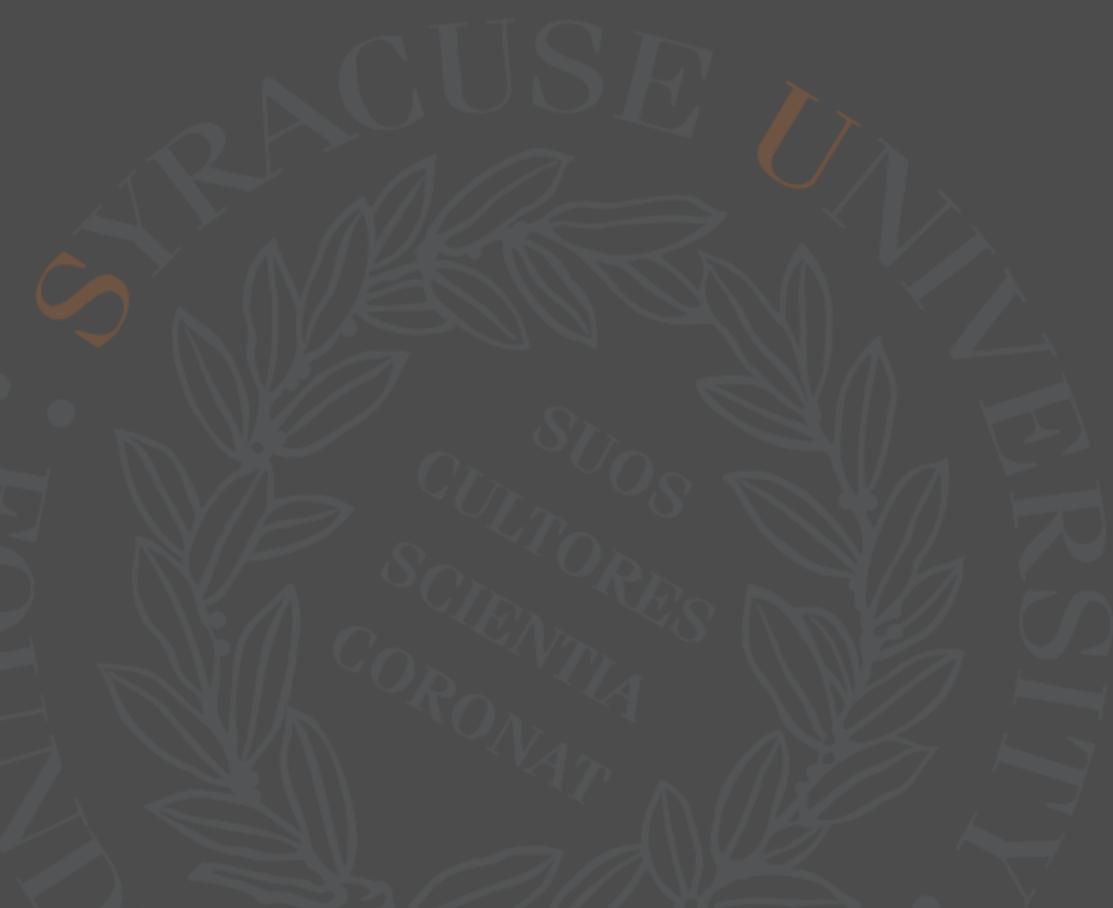
Why JSON?

→JSON

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

→Why make data available?

- ✓ Let others develop tools



School of Information Studies
SYRACUSE UNIVERSITY

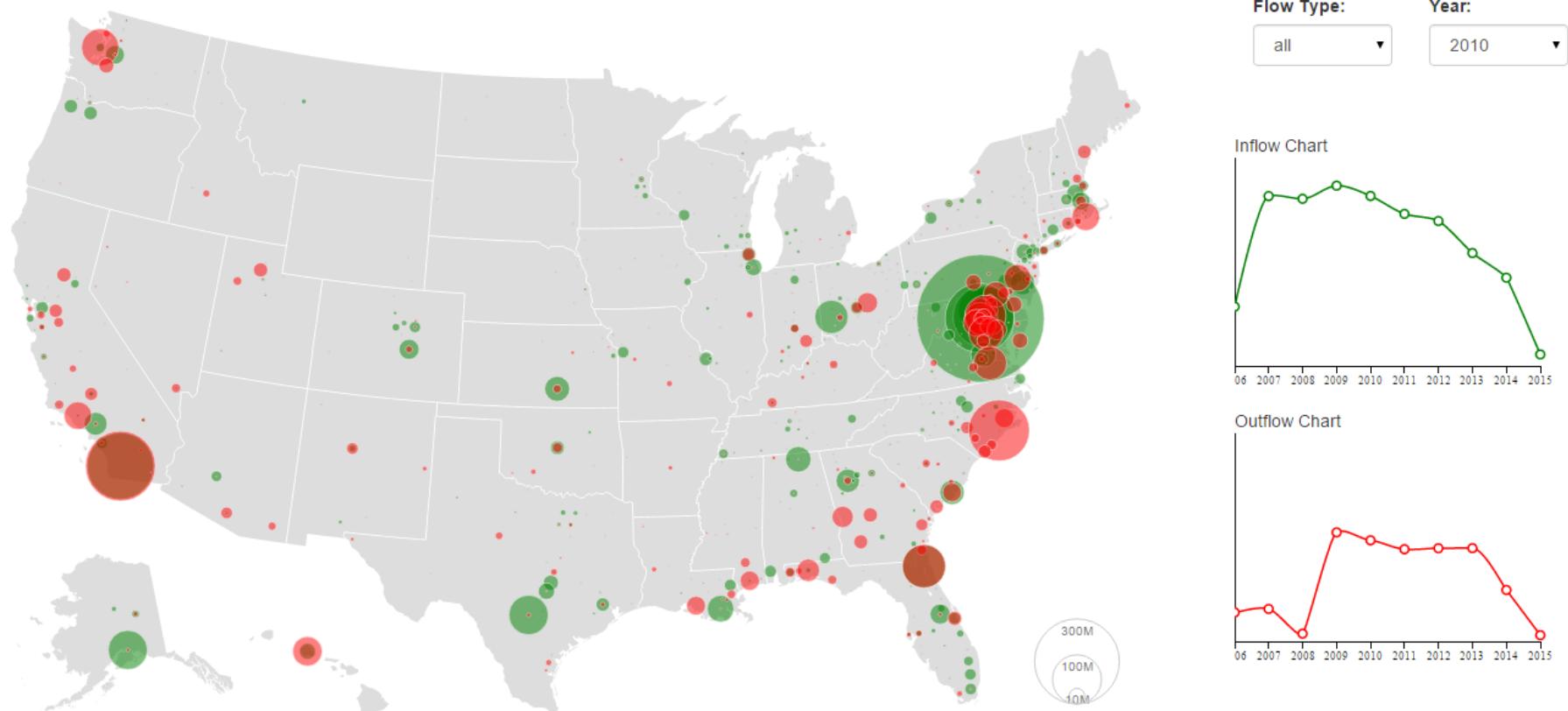
Example Visualizations

Professor Jeff Saltz

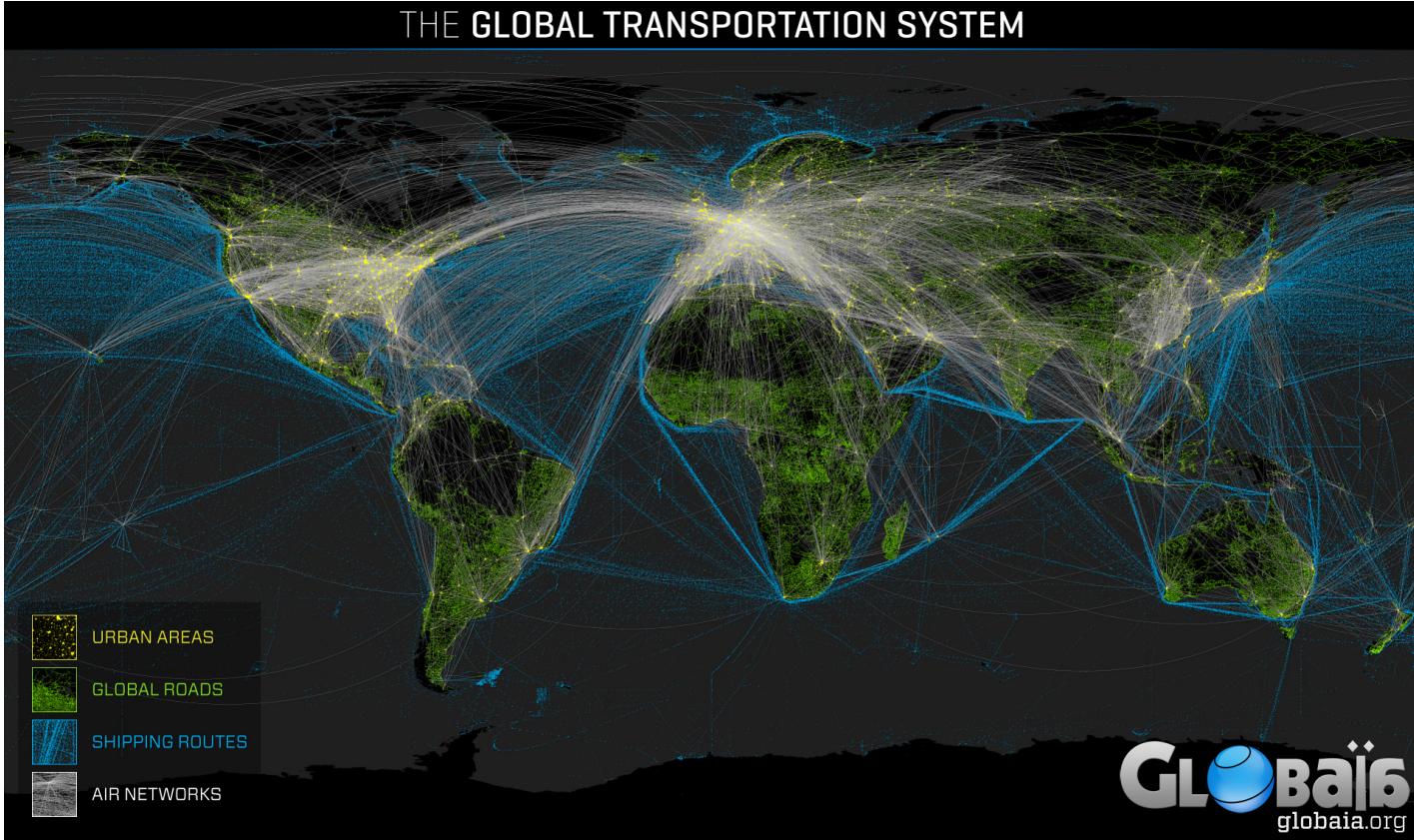
Learning Objectives

- Understand how visualization is used (via exploring some examples)
- Gain a basic understanding of how to represent numbers visually
- Gain knowledge of the principles of visualization
- Be able to generate basic visualizations using GGPlot2 in R

Web Science and Digital Libraries Research Group

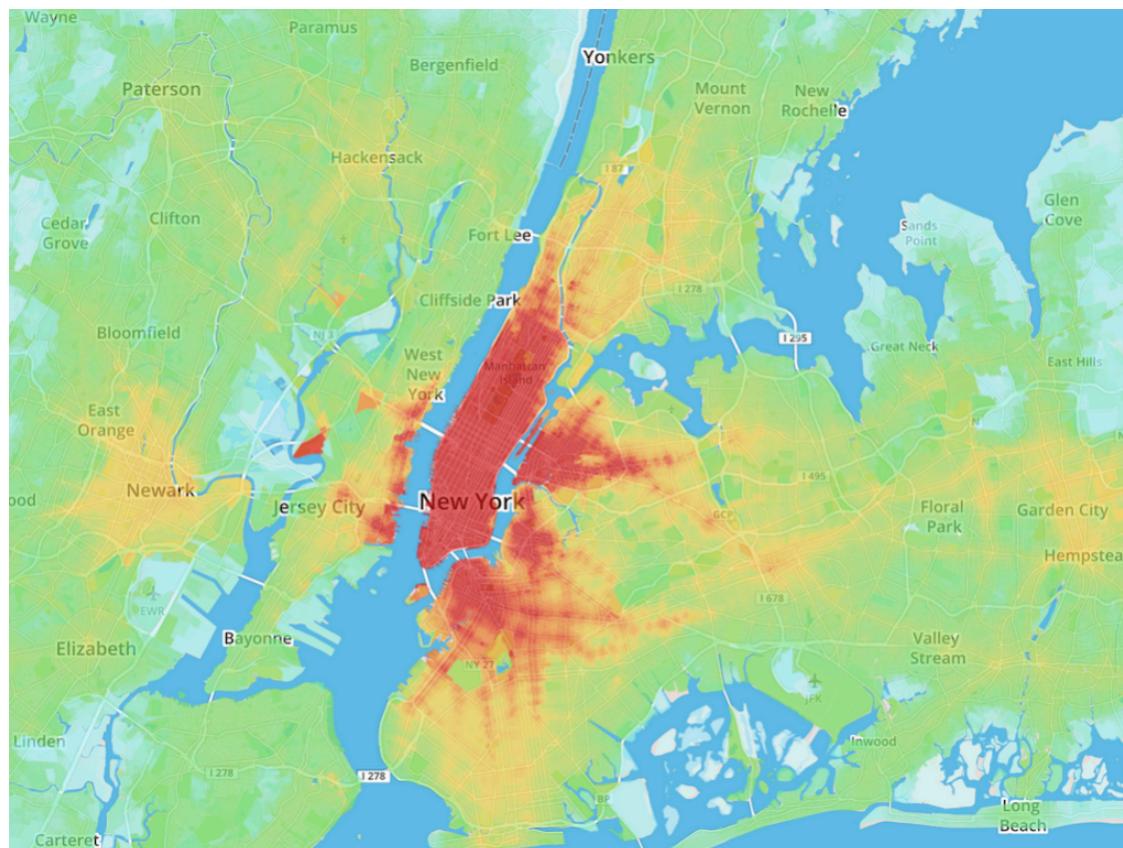


The Global Transportation System



<http://globaia.org/portfolio/cartography-of-the-anthropocene/>

America's 10 Best Cities for Commuting on Public Transit

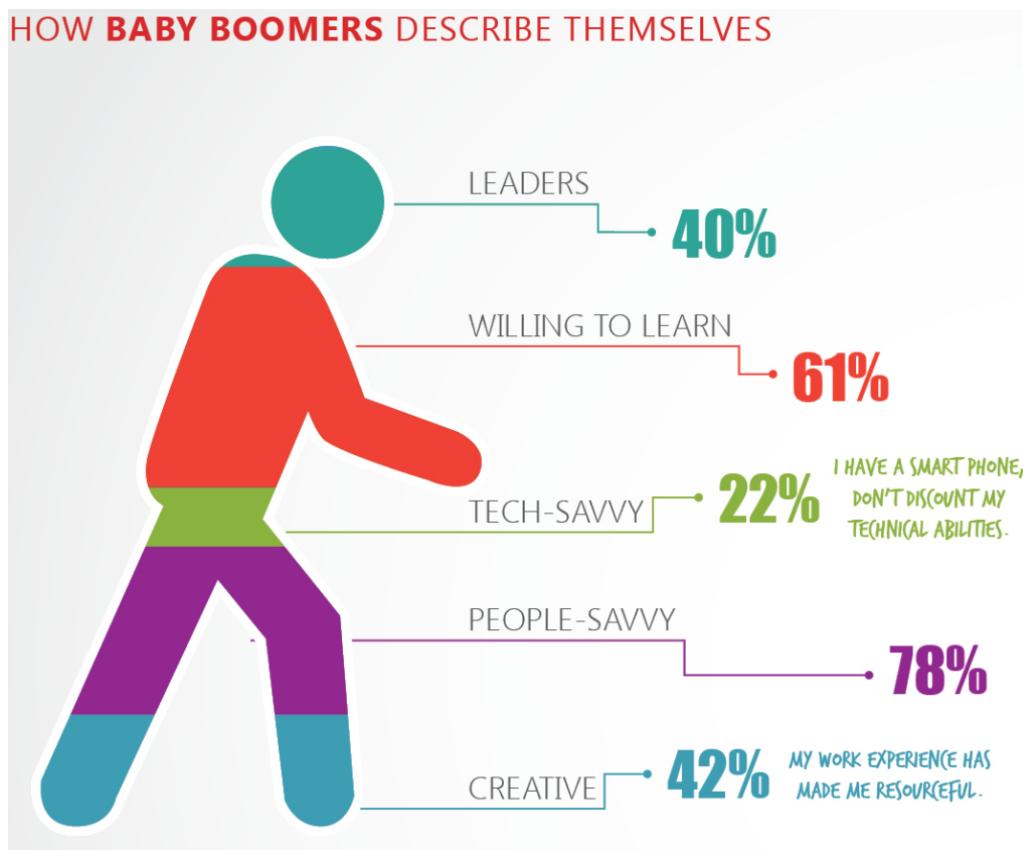


Jobs within 30 minutes
by transit, averaged 7 - 9 AM



Data Man

HOW BABY BOOMERS DESCRIBE THEMSELVES



Spiral Graph

Anatomy of a Winning TED Talk

● 1%

Sophisticated Visual Aids

We're not sure who puts the D in TED—most of the best presentations favor tepid PowerPoint slide shows (sorry, Brené Brown). Pictionary-quality drawings (really, Simon Sinek?), or no props at all.

● 5%

Opening Joke

Remember the one about the shoe salesmen who went to Africa in the 1900s? That's how Benjamin Zander opened his talk—which turned out to be about classical music.

● 5%

Spontaneous Moment

Don't overprepare. Tease the guy in the front row ("You could light up a village with this guy's eyes"). Command the stagehand who handles the human brain you brought.

● 5%

Statement of Utter Certainty

People come for answers—give 'em what they want, as Shawn Achor did: "By training your brain ... we can reverse the formula for happiness and success."

● 12%

Snappy Refrain

The TED equivalent of "I have a dream." Example: "People don't buy what you do; they buy why you do it." Repeat 7x.

● 23%

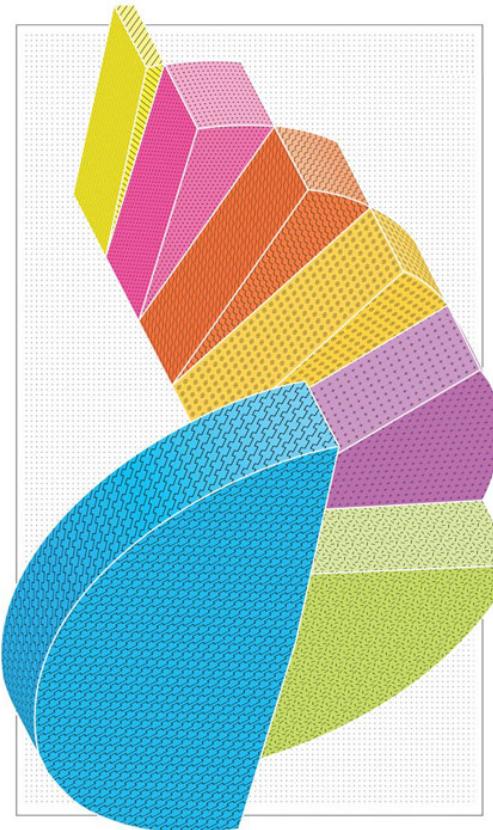
Personal Failure

Be relatable. We want to know about that nervous breakdown. Or at least the time you didn't fit in at summer camp.

● 49%

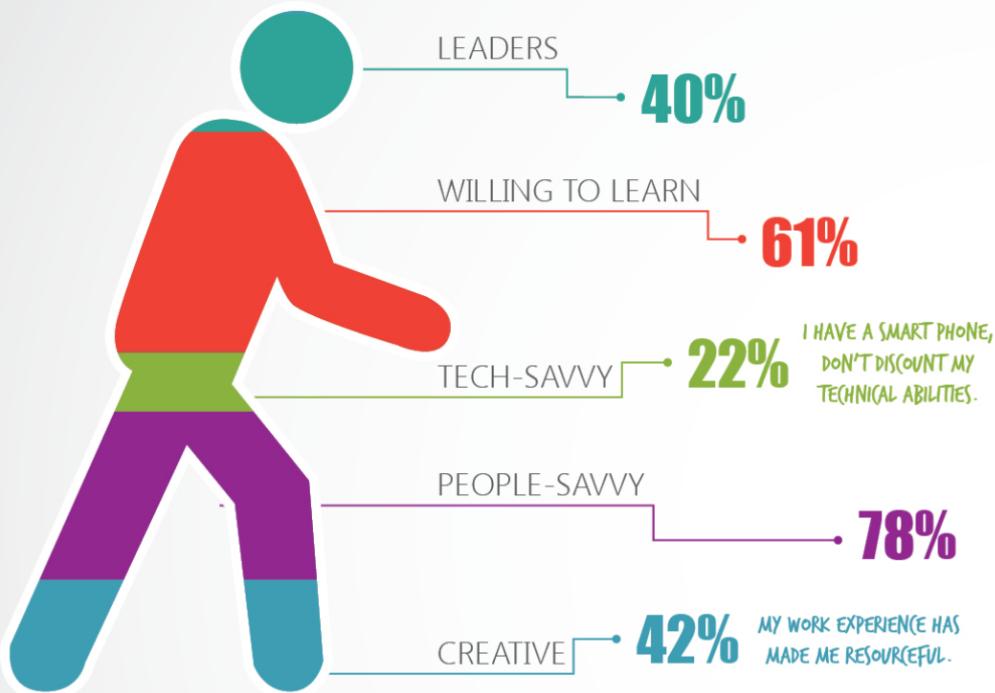
Contrarian Thesis

Wait a sec—we should be playing more videogames? The more choices we have, the worse off we are? TED is where conventional wisdom goes to die.



Question: What is Good or bad about these charts?

HOW BABY BOOMERS DESCRIBE THEMSELVES



Anatomy of a Winning TED Talk

● 1%

Sophisticated Visual Aids

We're not sure who puts the D in TED—most of the best presentations favor tepid PowerPoint slide shows (sorry, Brené Brown), Pictionary-quality drawings (really, Simon Sinek?), or no props at all.

● 5%

Opening Joke

Remember the one about the shoe salesmen who went to Africa in the 1900s? That's how Benjamin Zander opened his talk—which turned out to be about classical music.

● 5%

Spontaneous Moment

Don't overprepare. Tease the guy in the front row ("You could light up a village with this guy's eyes"). Command the stagehand who handles the human brain you brought.

● 5%

Statement of Utter Certainty

People come for answers—give 'em what they want, as Shawn Anchor did: "By training your brain ... we can reverse the formula for happiness and success."

● 12%

Snappy Refrain

The TED equivalent of "I have a dream." Example: "People don't buy what you do; they buy why you do it." Repeat 7x.

● 23%

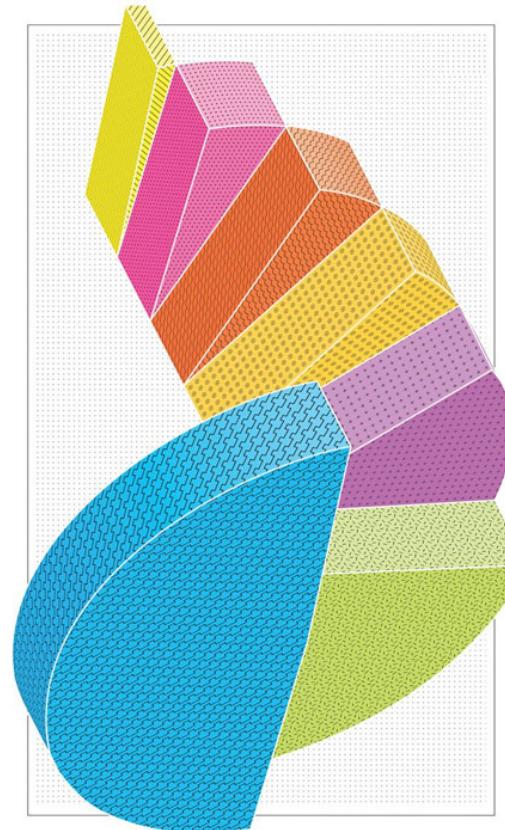
Personal Failure

Be relatable. We want to know about that nervous breakdown. Or at least the time you didn't fit in at summer camp.

● 49%

Contrarian Thesis

Wait a sec—we should be playing more videogames? The more choices we have, the worse off we are? TED is where conventional wisdom goes to die.



Analyzing Visualizations

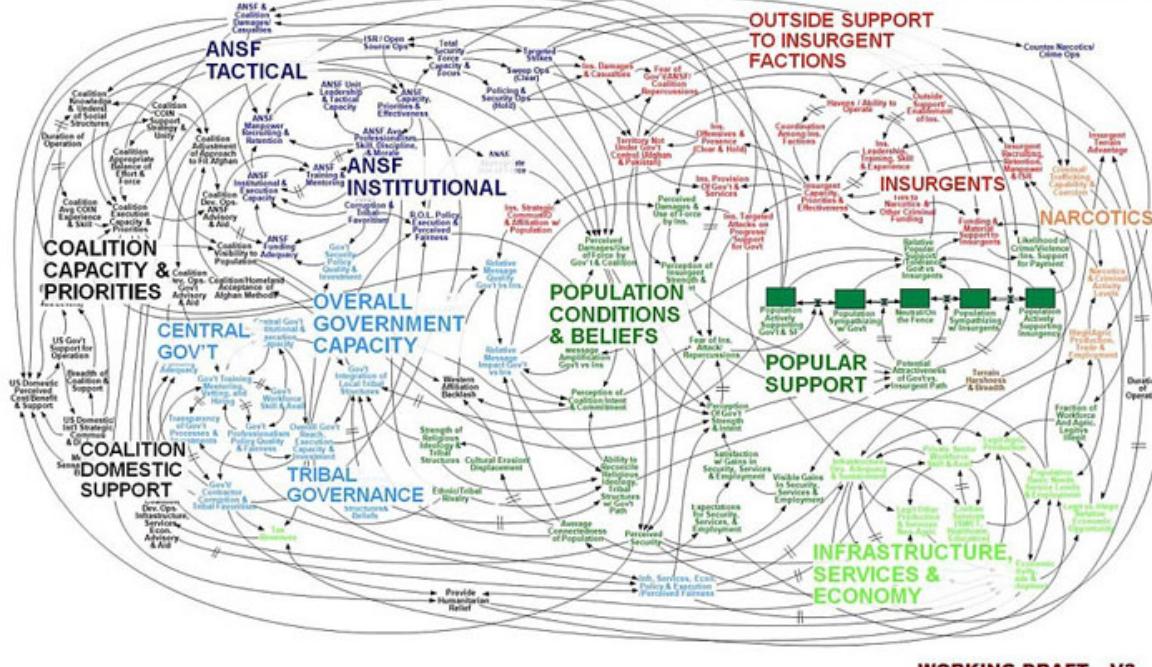
Professor Jeff Saltz

Too Much Information

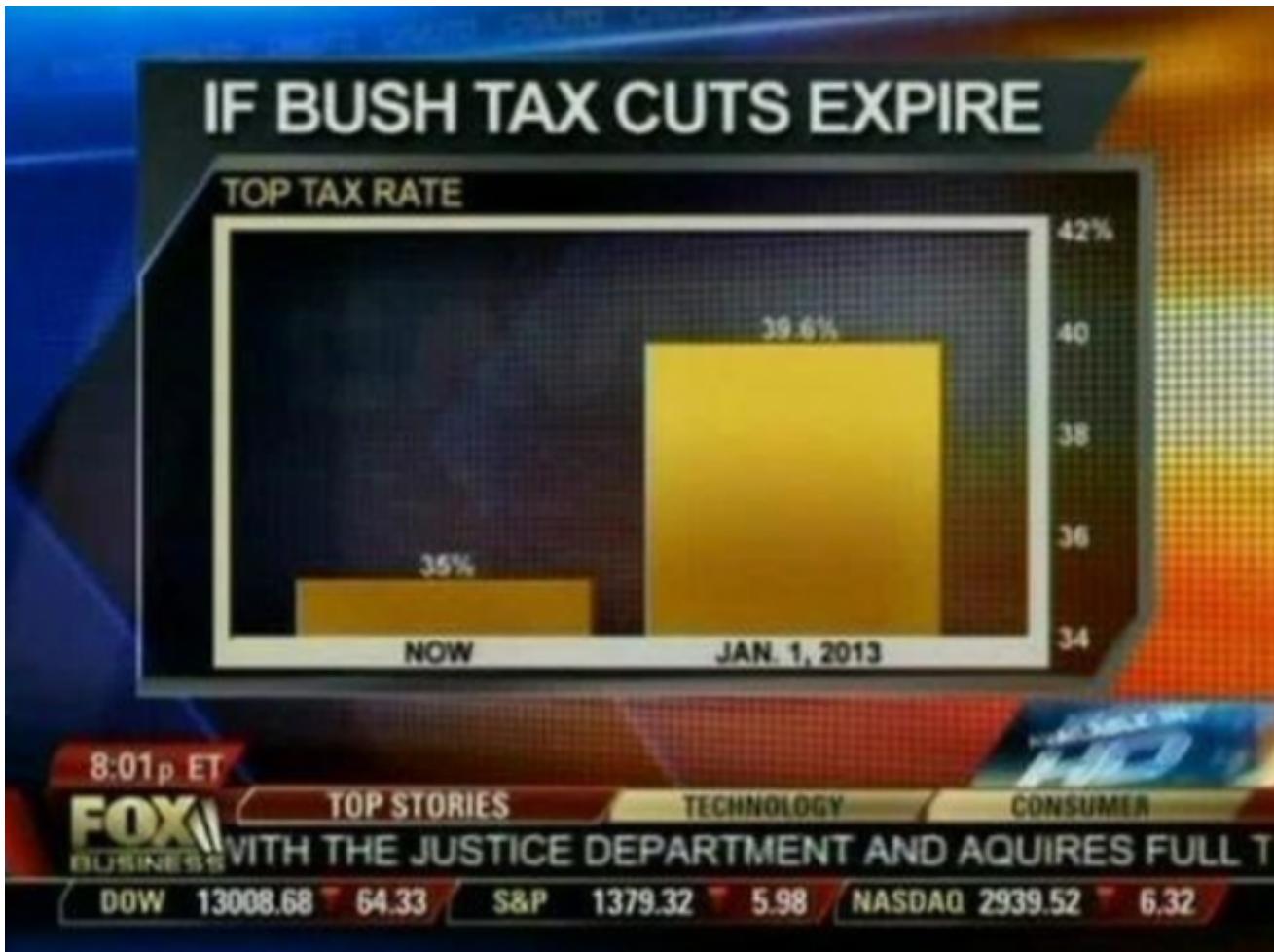
Afghanistan Stability / COIN Dynamics

 = Significant Delay

 Population/Popular Support
 Infrastructure, Economy, & Services
 Government
 Afghanistan Security Forces
 Insurgents
 Crime and Narcotics
 Coalition Forces & Actions
 Physical Environment



Simple Bar Chart



Oil Chart

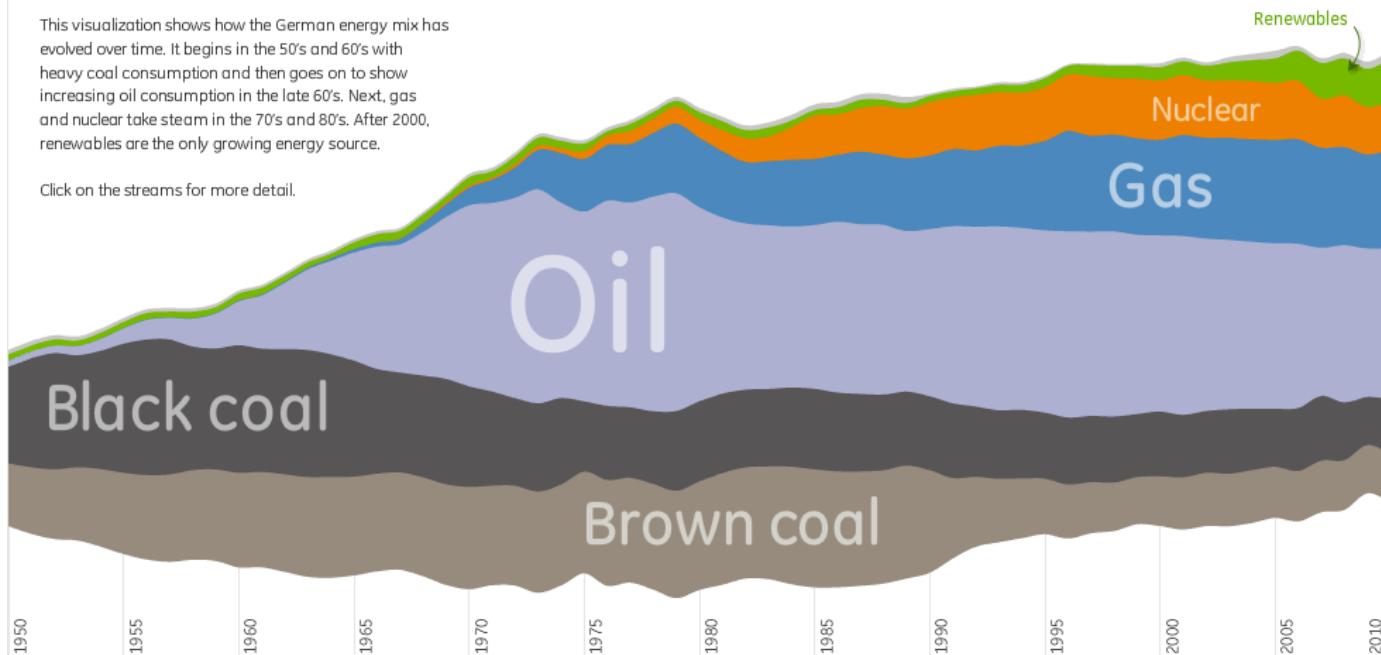
Historical Energy Mix

Energy Consumption By Source From 1950 to 2010

← 1 2 3

This visualization shows how the German energy mix has evolved over time. It begins in the 50's and 60's with heavy coal consumption and then goes on to show increasing oil consumption in the late 60's. Next, gas and nuclear take steam in the 70's and 80's. After 2000, renewables are the only growing energy source.

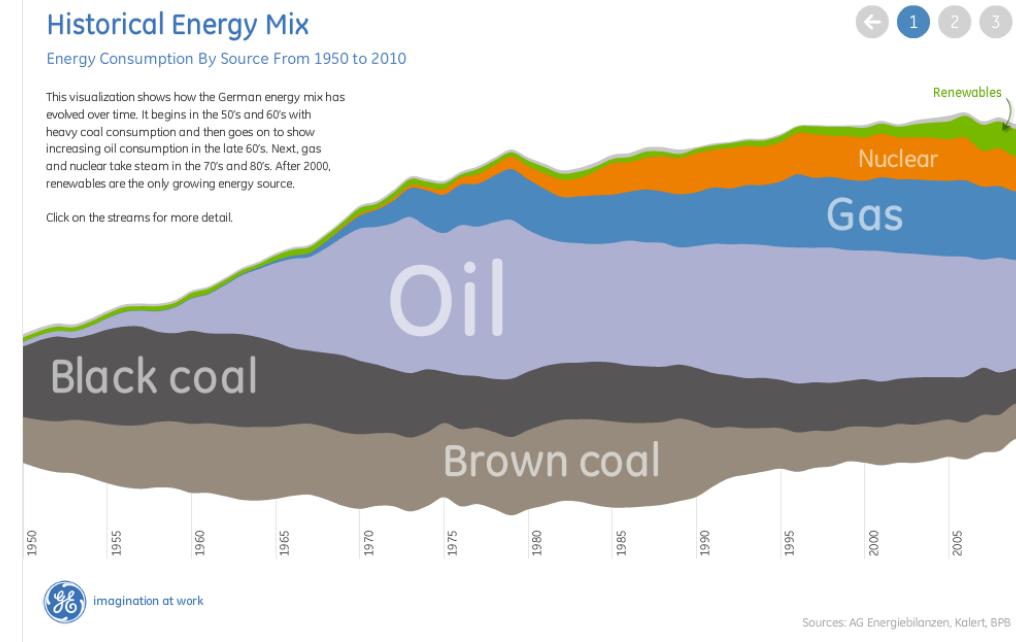
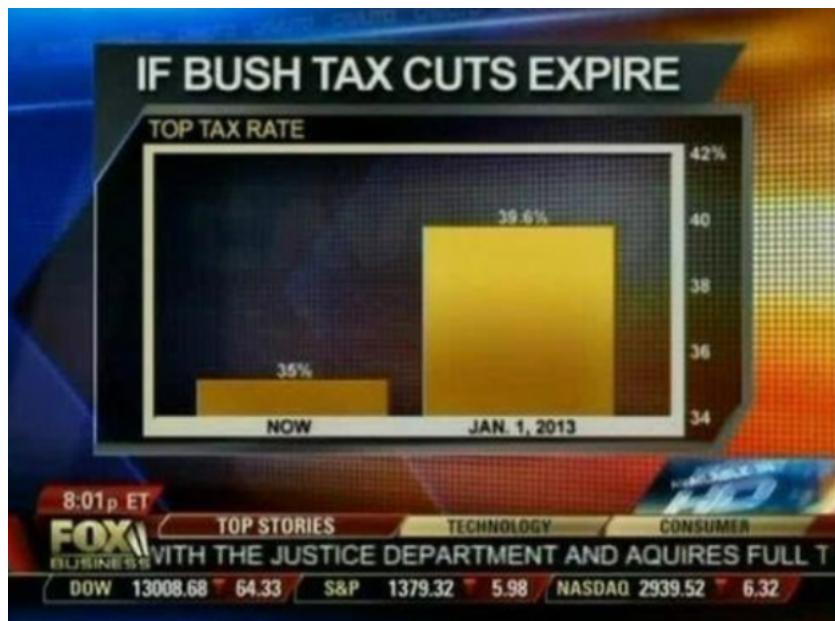
Click on the streams for more detail.



Sources: AG Energiebilanzen, Kalert, BPB

Question

What is misleading or confusing about these two visualizations?



Play With Visualizations

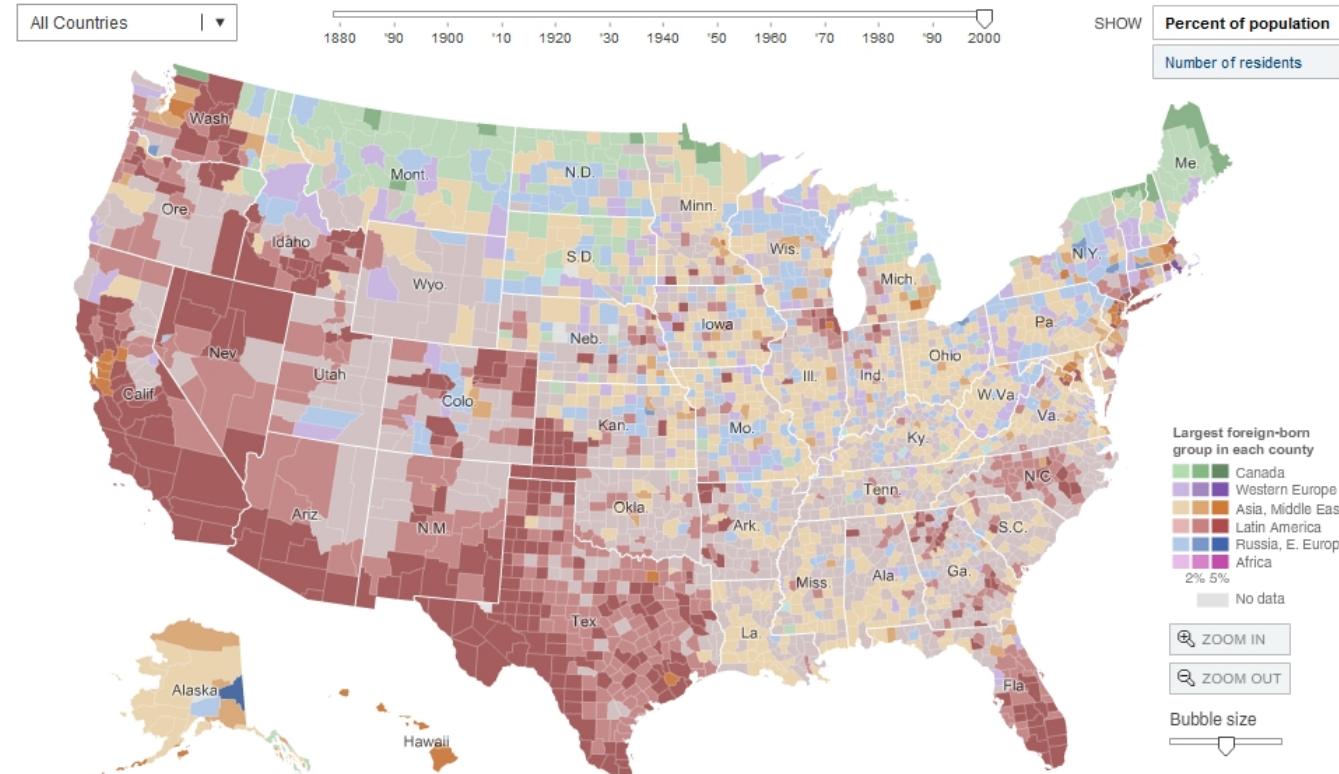
Professor Jeff Saltz

School of Information Studies
SYRACUSE UNIVERSITY

Examples: Geo Data Mapping

Immigration Explorer

Select a foreign-born group to see how they settled across the United States.



[Demo](#)

Note: Due to limitations in the Census data, foreign-born populations are not available in all areas for all years.

Sources: Social Explorer, www.socialexplorer.com; Minnesota Population Center; U.S. Census Bureau

Matthew Bloch and Robert Gebeloff/The New York Times

Examples: Map Your Moves

- Where New Yorkers move (10 years' data)

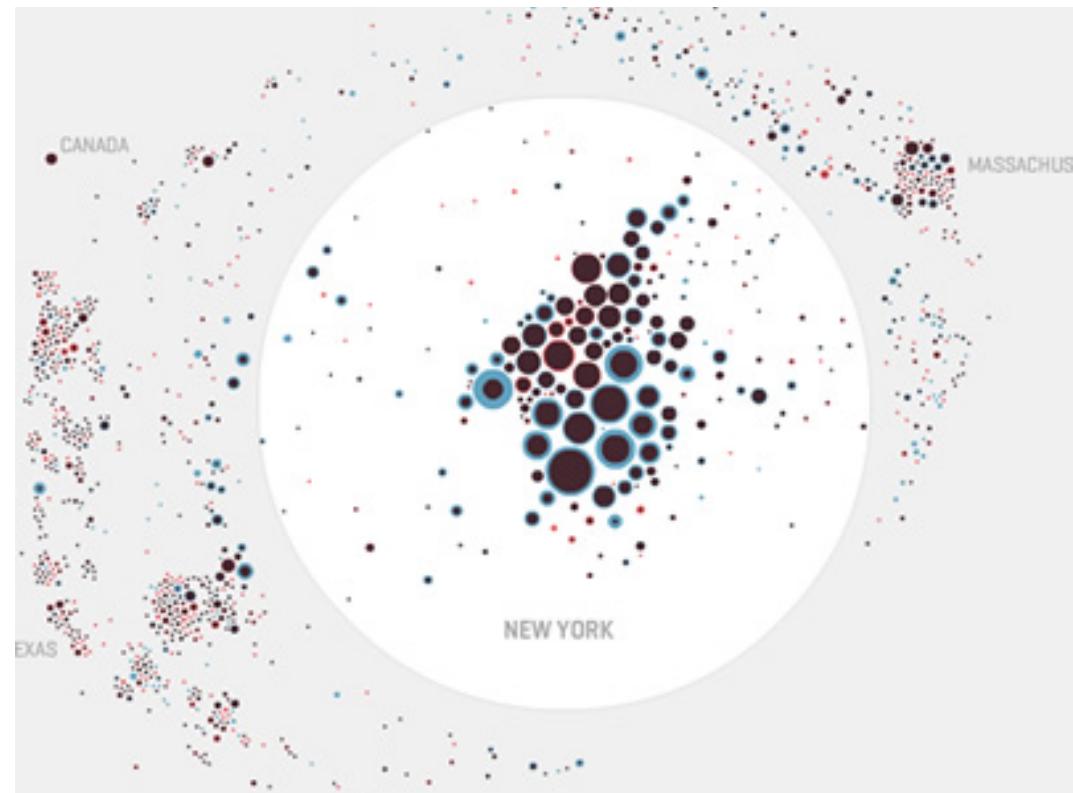
distorted map

circle = moves for
one zip code

red – out
blue – in

overlaid

[Demo](#)



Example: Circle Chart

Demo

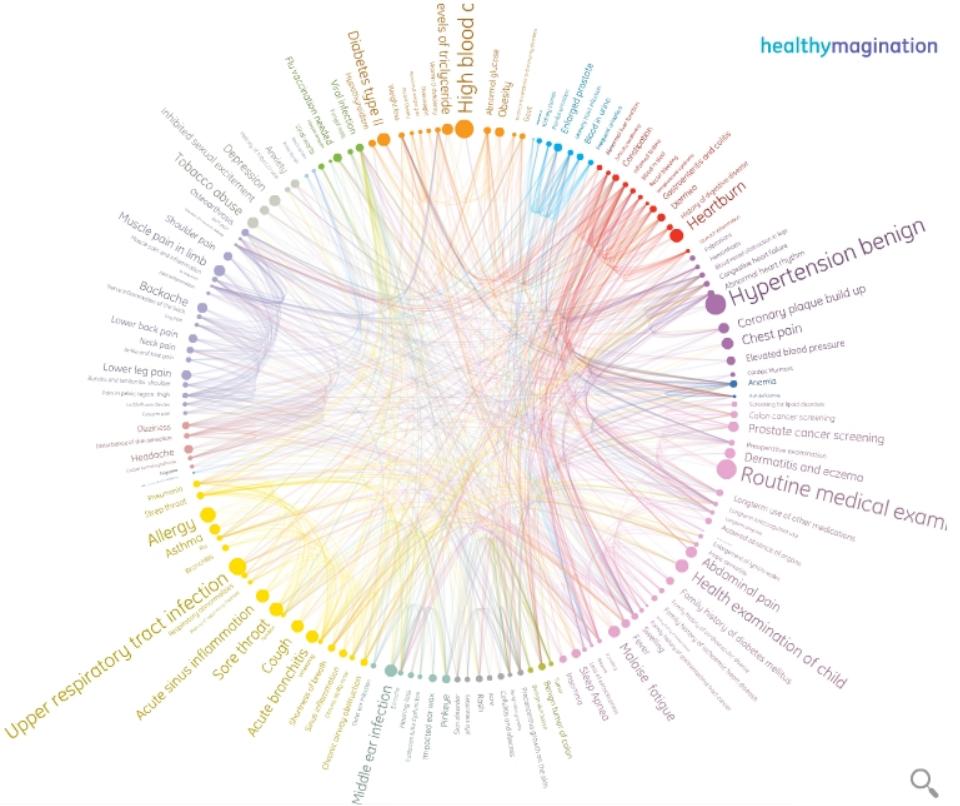


Layout Gender

Search

- Categories
- █ Blood Diseases
 - █ Circulatory System
 - █ Digestive System
 - █ Genitourinary System
 - █ Hormone Nutrition Immunity
 - █ Infections
 - █ Injury and Poisoning
 - █ Mental Health
 - █ Musculoskeletal System
 - █ Nervous System
 - █ Pregnancy Early Development
 - █ Respiratory System
 - █ Sensory Organs
 - █ Skin Conditions
 - █ Tumors
 - █ Unclassified

Source: GE's MQIC Database
More Info



Question: Which did you like best?

Why did you like it the best?

Could you suggest alternative visualizations?

Was the “interaction” helpful?

Introduction to Visualizations

Professor Jeff Saltz

What Is Visualization?

- “the communication of information using graphical representations” —Ward et al.
- “Transformation of the symbolic into the geometric” —McCormick et al.
- “...finding the artificial memory that best supports our natural means of perception”
—Bertin

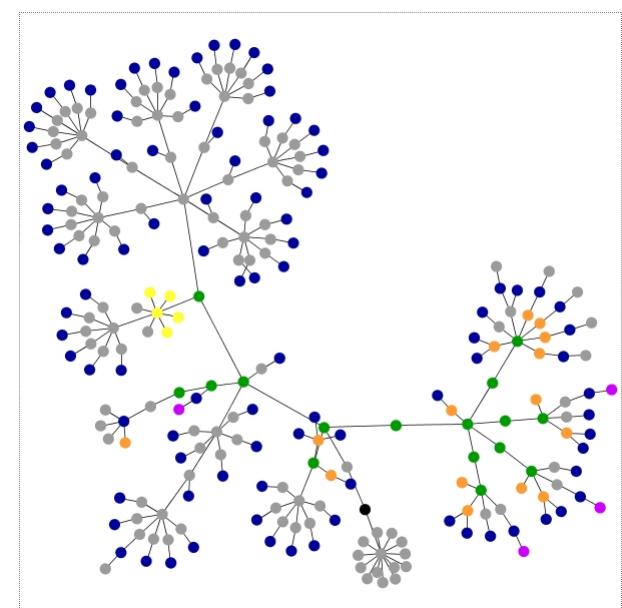
What Is Visualization? (cont.)

- *Information visualization* can be defined as the use of interactive visual representations of abstract data.
- Information visualization provides compact graphical presentations and user interfaces for interactively manipulating large numbers of items.

Information Visualization

- Information visualization: concerned with data that does not have a well-defined representation in 2D or 3D space (i.e., “abstract data”)

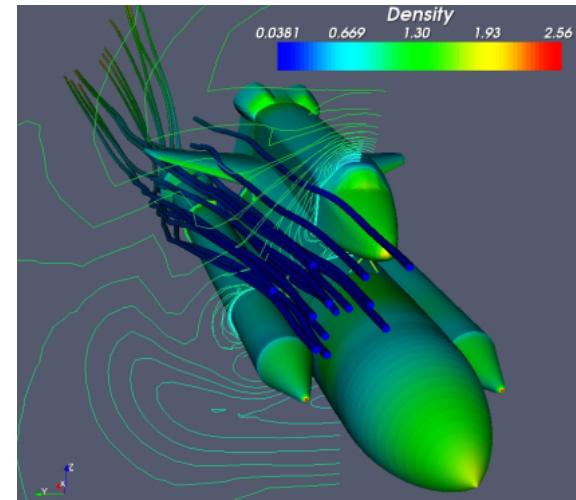
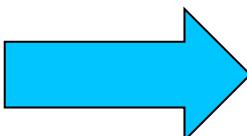
```
http://www.bu.edu/students/life/  
http://www.bu.edu/students/life/housing/  
http://www.bu.edu/students/life/dining-vending/  
http://www.bu.edu/students/life/phone/  
http://www.bu.edu/students/life/safety/  
http://www.bu.edu/students/life/transportation/  
http://www.bu.edu/students/life/activities/  
http://www.bu.edu/students/life/bu-global-orientation-registration/  
http://www.bu.edu/students/academics/  
http://www.bu.edu/students/academics/link/  
http://www.bu.edu/students/academics/admissions/  
http://www.bu.edu/students/academics/registration/  
http://www.bu.edu/students/academics/advising/  
http://www.bu.edu/students/academics/grades/  
http://www.bu.edu/students/academics/services/  
http://www.bu.edu/students/academics/support/  
http://www.bu.edu/students/health/  
http://www.bu.edu/students/health/services/  
http://www.bu.edu/students/health/counseling/  
http://www.bu.edu/students/health/facilities/  
http://www.bu.edu/students/health/clubsports/  
http://www.bu.edu/students/health/varsitysports/
```



“Sci Viz” vs. “Info Viz”

- Visualization: converting raw data to a form that is viewable and understandable to humans
- Scientific visualization: specifically concerned with data that has a well-defined representation in 2D or 3D space (e.g., from simulation mesh or scanner)

```
0265640 132304 133732 032051 037334 024721 015013 052226 001562  
0265660 025537 064663 054606 043244 074075 124153 135216 126614  
0265700 144210 056426 044700 042650 165230 137037 003655 006254  
0265720 134453 124327 176005 027034 107619 170774 073702 067274  
0265740 072451 007735 147620 061064 157435 113057 155356 114603  
0265760 107204 102316 171453 046040 120223 001774 030477 046673  
0266000 171317 116055 155117 134444 167210 041405 147127 050505  
0266020 004137 046472 124015 134360 173550 053517 044635 021135  
0266040 070176 047705 113734 175477 105532 076515 172366 056333  
0266060 041023 074017 127113 003214 037026 037640 065171 123424  
0266100 065701 037406 140000 165341 072410 100032 125455 056646  
0266120 006716 071402 055672 132571 105645 170073 050376 072117  
0266140 024451 007424 114200 077733 024434 012546 172404 102345  
0266160 040223 050170 055164 164634 047154 126525 112514 032315  
0266200 016041 176055 042766 025015 176314 017234 110060 014515  
0266220 117156 030746 154234 125001 151144 163706 136237 164376  
0266240 137055 062276 161755 115466 005322 132567 073216 002655  
0266260 171466 126161 117155 067683 016177 014460 112765 055527  
0266300 003767 175367 104754 036436 172172 150750 043643 145410  
0266320 072024 000007 040627 070652 173011 002151 125132 140214  
0266340 060115 014356 015164 067027 120201 070242 033065 131334  
0266360 170601 170106 040437 127277 124446 138631 041452 116321  
0266400 020243 005602 004146 121574 124651 005634 071331 102070  
0266420 157504 160307 166330 074251 024520 114433 167273 030635  
0266440 133614 106171 144160 010652 007365 026415 160716 100413  
0266460 026630 007210 000630 121224 076033 140764 000737 003276  
0266500 114060 042647 104475 110537 066718 104754 075447 112254  
0266520 030374 144251 077734 015157 002513 173526 035531 150003  
0266540 146207 015135 024445 130101 072457 040764 165513 156412  
0266560 166410 067251 156160 106406 136770 030515 064740 022032  
0266580 142156 123707 175121 071170 076357 037233 031136 015232  
0266620 075074 016744 044055 102230 110063 033395 052765 172463
```



*Adapted from The ParaView Tutorial, Moreland

Why Visualize?

- ❖ Vision is “highest bandwidth” sense
 - ❖ Fast and parallel
 - ❖ Preattentive processing
- ❖ Eye trained for pattern recognition
 - ❖ Scanning
 - ❖ Recognizing
 - ❖ Remembering images

Visualization Components

- ❖ Color
- ❖ Size
- ❖ Texture
- ❖ Proximity
- ❖ Annotation
- ❖ Interactivity
 - ❖ Selection/filtering
 - ❖ Zoom
 - ❖ Animation

GGPLOT

Professor Jeff Saltz

School of Information Studies
SYRACUSE UNIVERSITY

GGPLOT2

ggplot2 divides plot into three different fundamental parts:

Plot = Data + Aesthetics + Geometry

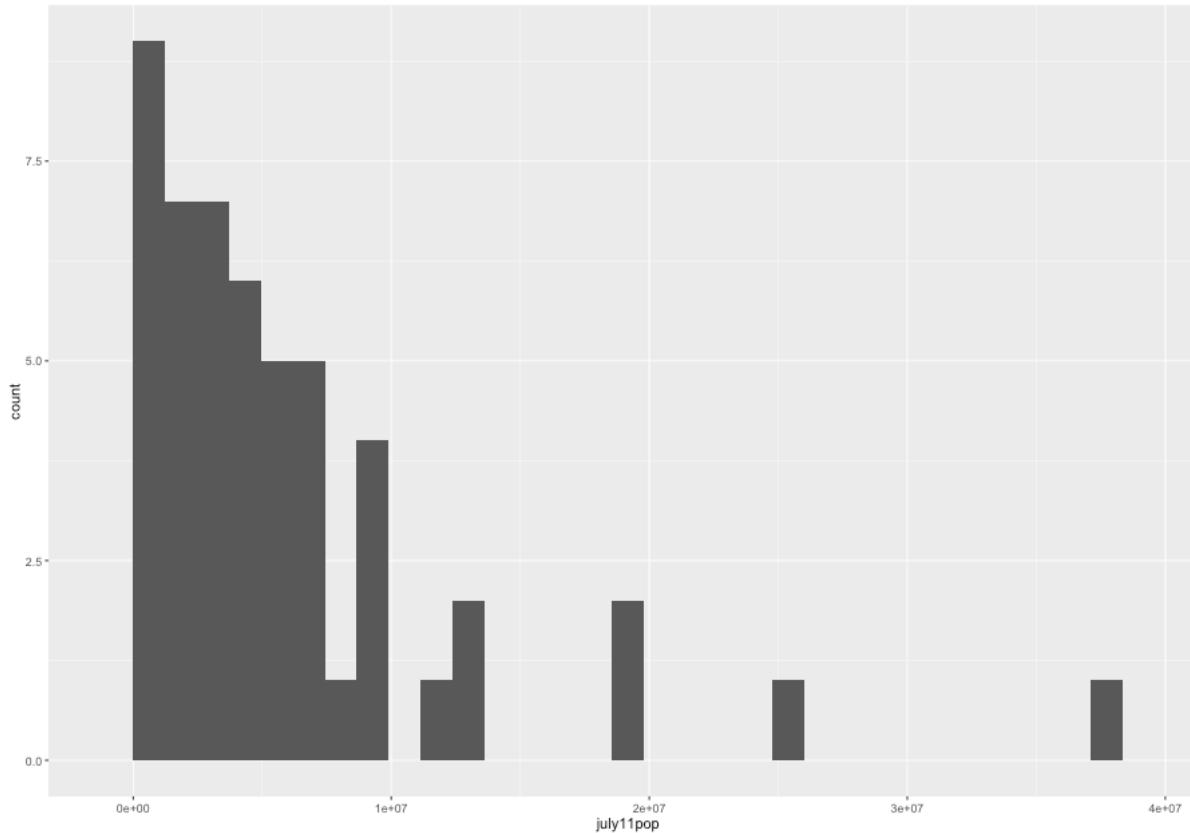
The principal components of every plot can be defined as follows:

- **Data** is a data frame.
- **Aesthetics** is used to indicate x and y variables. It can also be used to control the color, the size or the shape of points, the height of bars, etc.
- **Geometry** defines the type of graphics (histogram, box plot, line plot, density plot, dot plot, etc.)

GG stands for “grammar of graphics.”

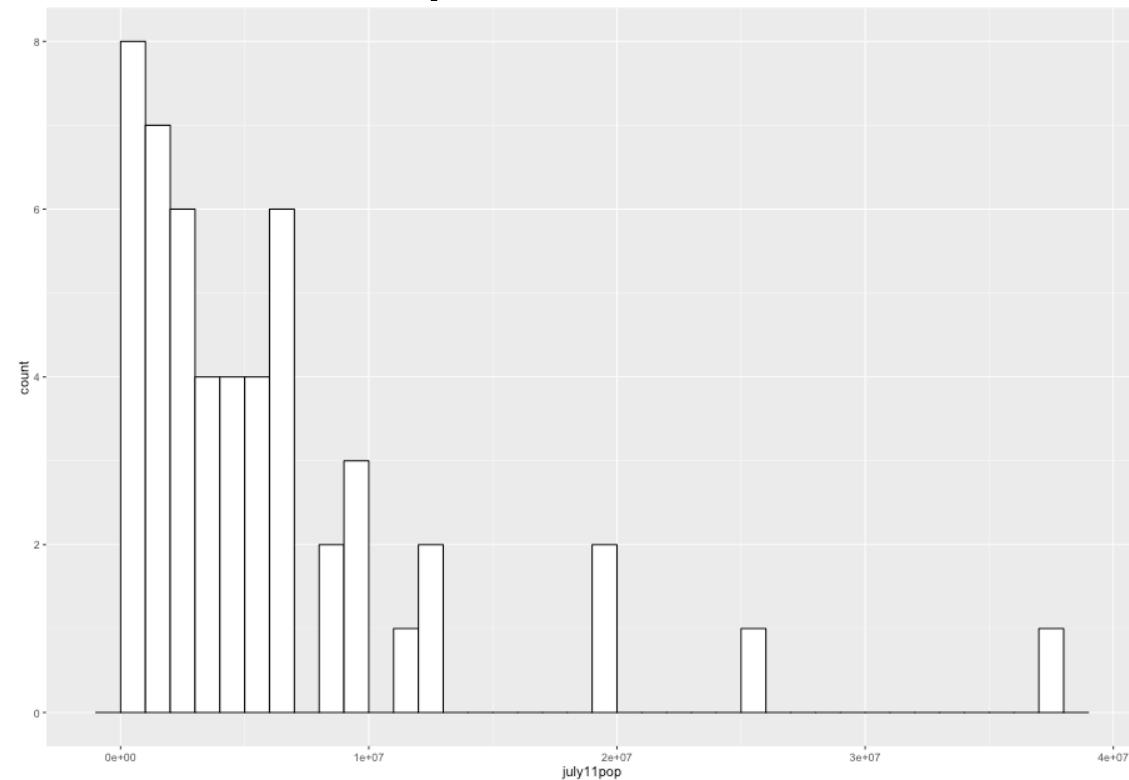
GGPLOT2: Histogram

```
ggplot(dfStates, aes(x=july11pop)) +  
  geom_histogram(bins=30)
```



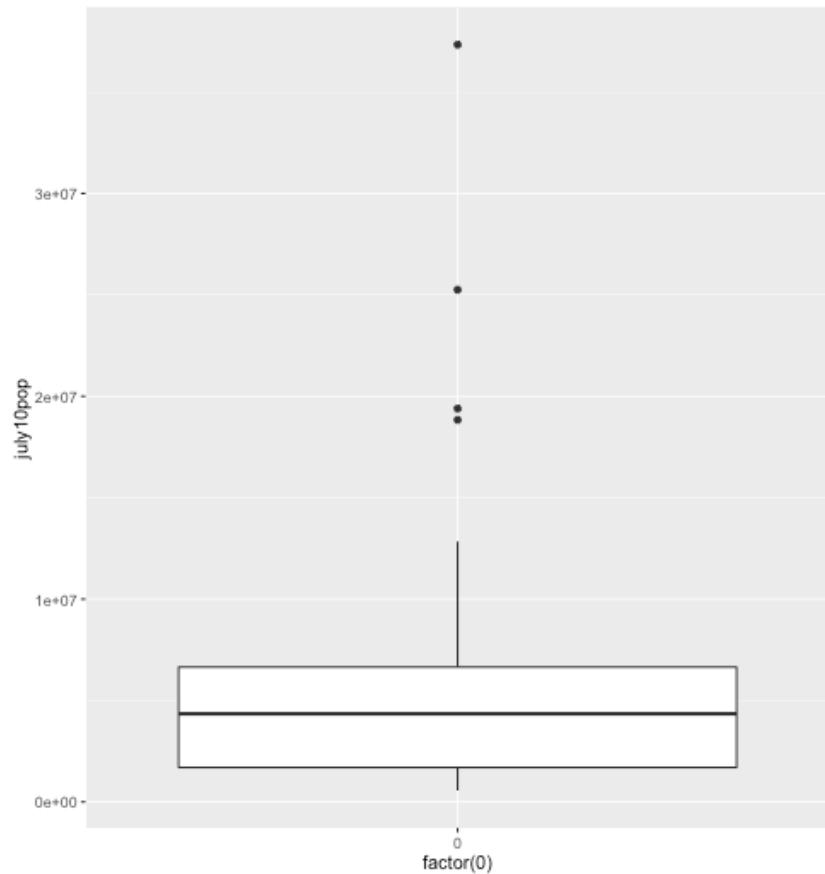
GGPLOT2: Histogram

```
ggplot(dfStates, aes(x=july11pop)) +  
  geom_histogram(binwidth=1000000,  
  color="black", fill="white")
```



GGPLOT2: Boxplot

```
ggplot(dfStates,aes(x=factor(0),july10pop)) +  
  geom_boxplot()
```



GGPLOT2: Timed Task

	dayOfWeek	time	week
1	mon	4.0	1
2	tues	4.5	1
3	wed	3.5	1
4	thurs	5.0	1
5	fri	4.0	1
6	sat	4.2	1
7	mon	4.5	2
8	tues	5.0	2
9	wed	3.8	2
10	thurs	5.2	2
11	fri	4.6	2
12	sat	4.3	2

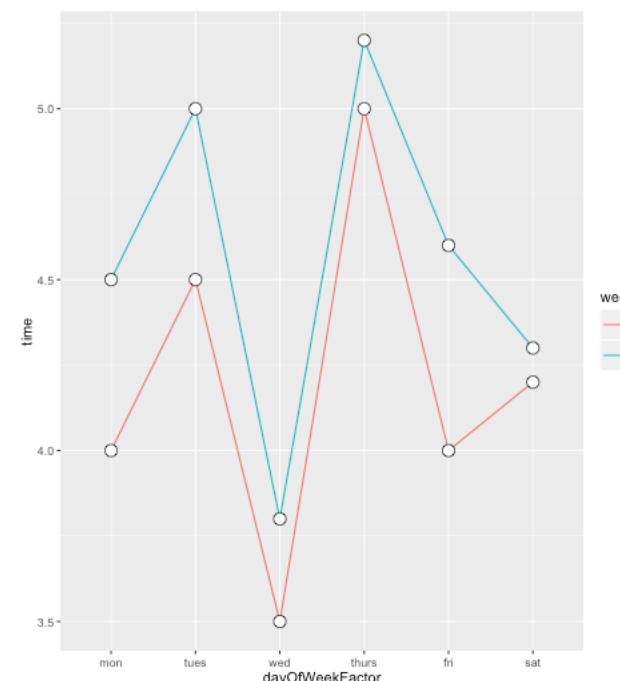
GGPLOT2: Lines

```
g <- ggplot(travel.df, aes(x=dayOfWeek,  
    group=week, color=week))  
+ geom_line(aes(y=time))
```

```
g <- g + geom_point(y=time, colour="black",  
size=4, shape=21, fill="white")
```

```
g <- g + ylab("time to NYC (in hours)") +  
ggttitle("compare weekly times")
```

```
g
```

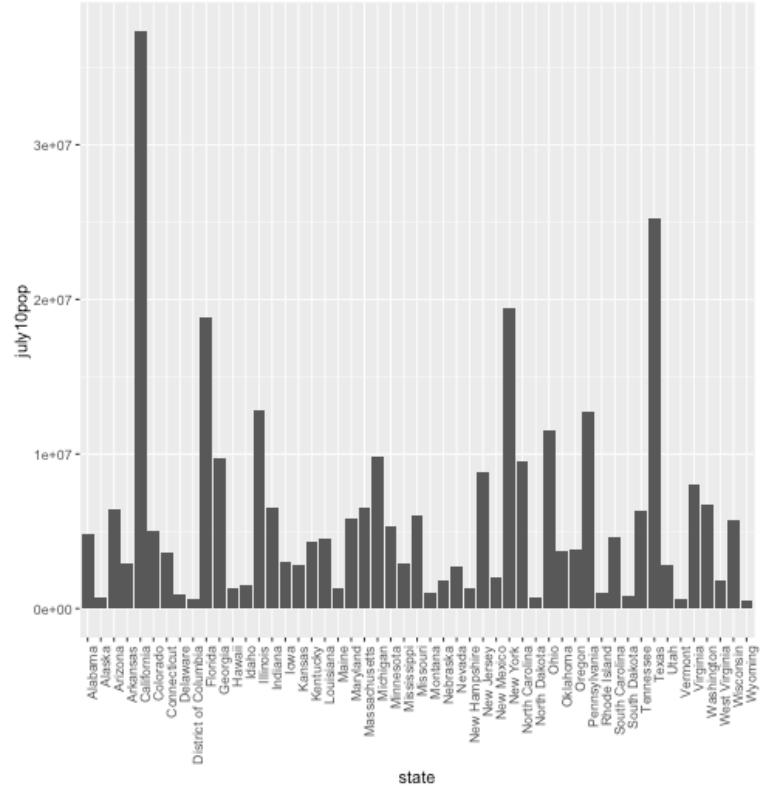


GGPLOT2: Bars

```
g <- ggplot(dfStates,  
aes(x=state, y=july10pop))  
+ geom_bar(stat="identity")
```

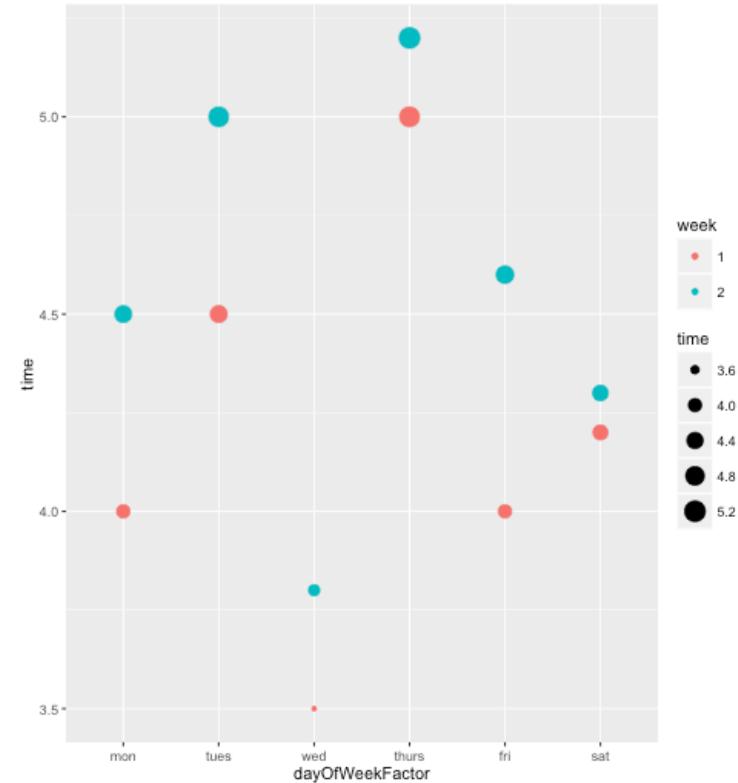
```
g <- g + theme(axis.text.x =  
element_text(angle = 90, hjust = 1))
```

```
g
```



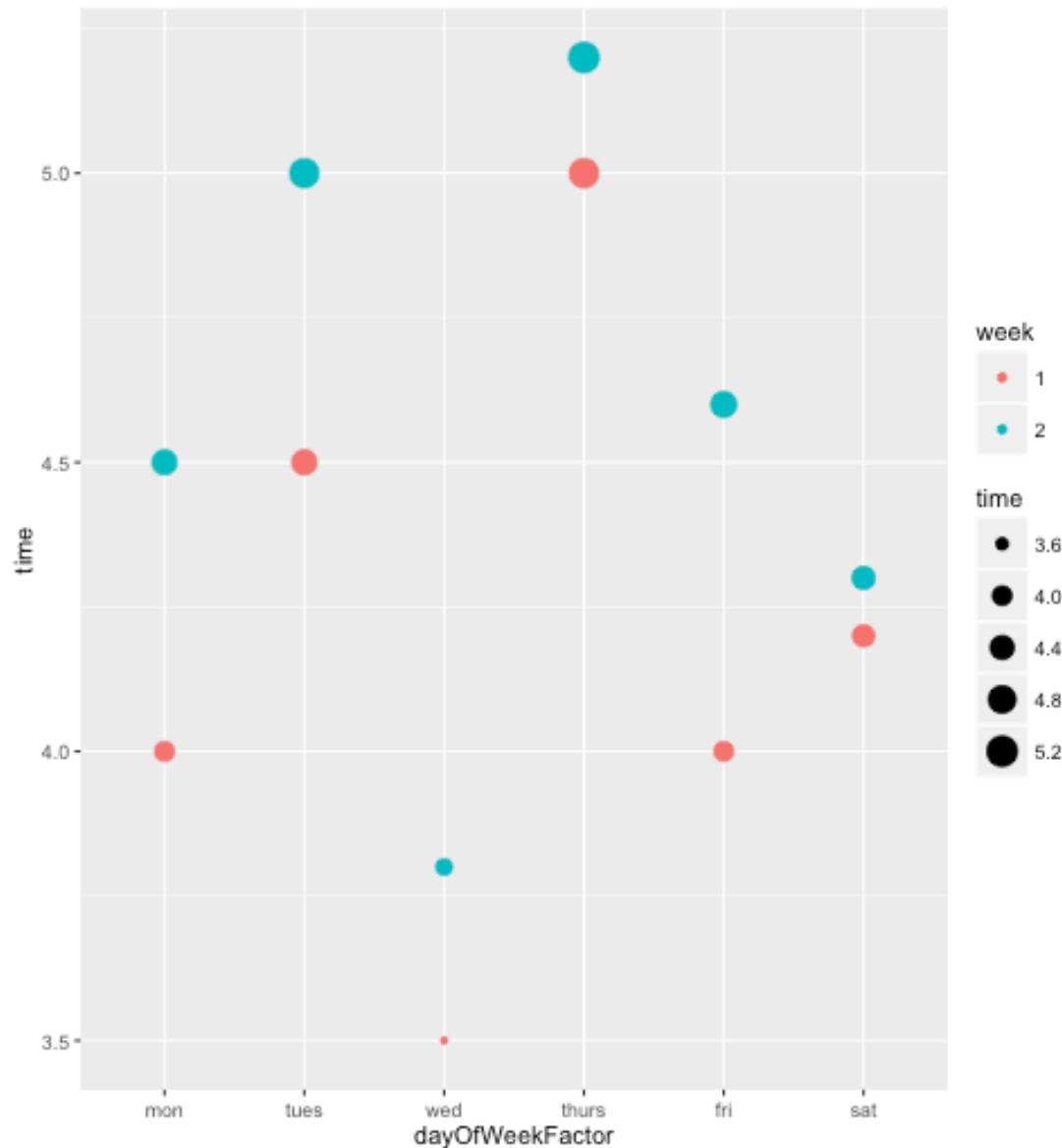
GGPLOT2: Scatter

```
ggplot(travel.df,  
       aes(x=dayOfWeek, y=time))  
+ geom_point(  
             aes(size=time,  
                  color=week))
```



GGPLOT2: Scatter

```
ggplot(travel.df, aes(x=dayOfWeek, y=tim  
+ geom_point(aes(size=time, color=week))
```



Ten Principles in Visualization

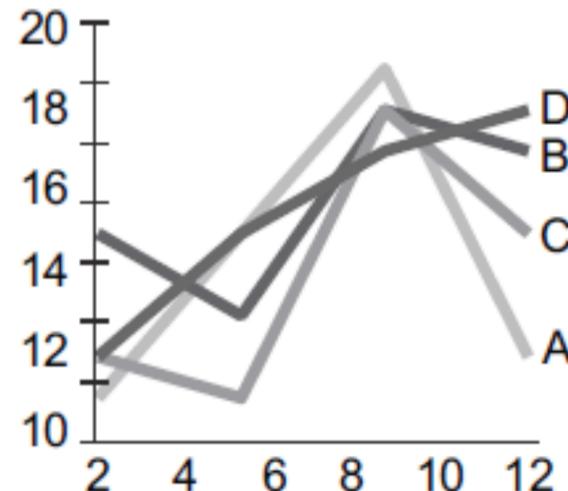
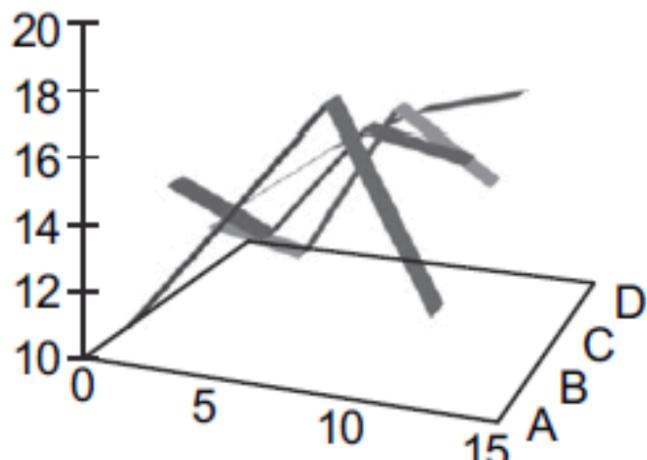
Professor Jeff Saltz

Ten Principles in Visualization

- Principles are fairly well known
- Adapted from Kelleher and Wagener (2011)

1. Simplicity

- Create the simplest graph that conveys the information you want to convey (Tufte).



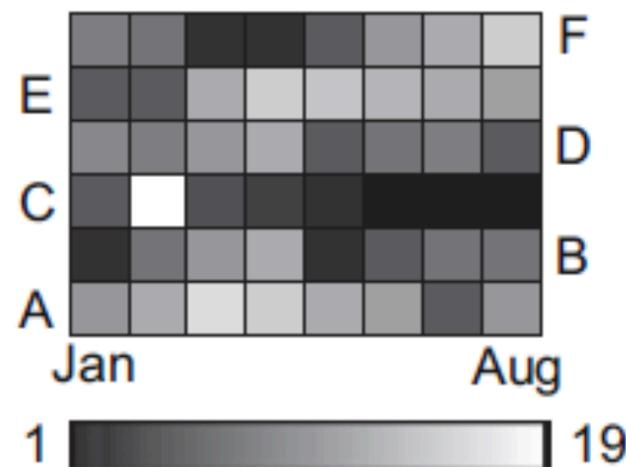
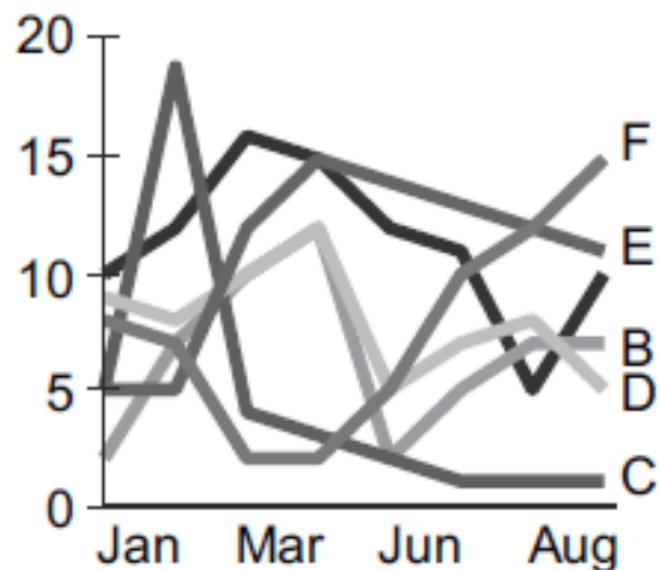
2. Encoding

- Consider the type of encoding object and attribute used to create a lot (Chambers, 1983; Cleveland & McGill, 1984).

Value encoding attribute									
Form	Length	Width	Orientation	Color	Hue	Intensity	Transparency	Density	
			/		● ● ●	● ● ●	● ● ●		
	Size	Shape	Curvature		● ● ●	● ● ●	● ● ●		
	●●●●				● ● ●	● ● ●	● ● ●		
Spatial Position									
Enclosure			2-D Position			Spatial Grouping		Density	
Blur			● ● ●			● ● ●		○○○	
Motion			Direction			Pathway		○○○	

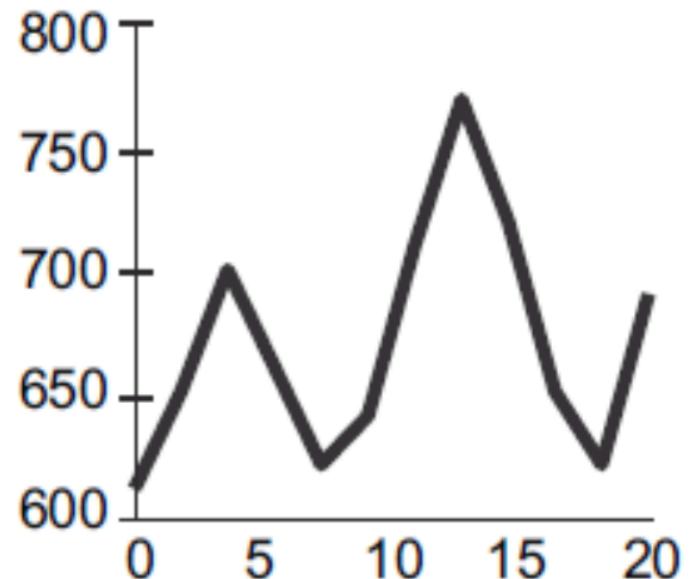
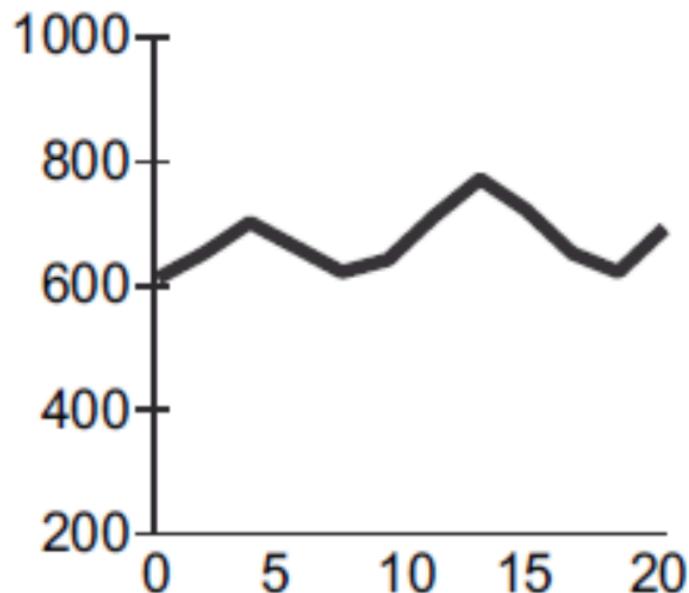
3. Patterns vs. Details

- Focus on visualizing patterns or on visualizing details (Few, 2004; Kosslyn & Chabris, 1992).



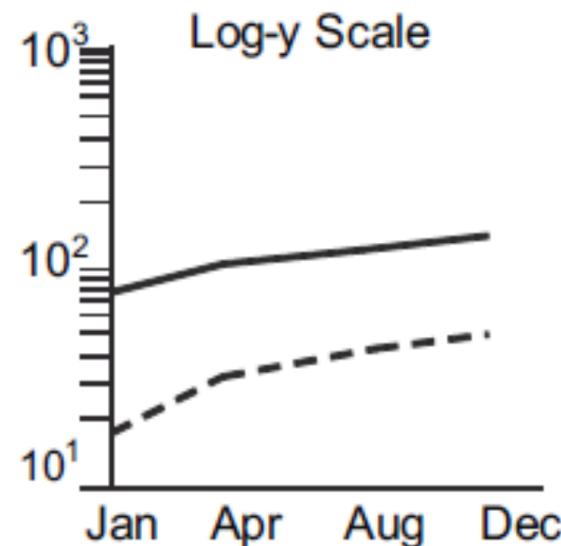
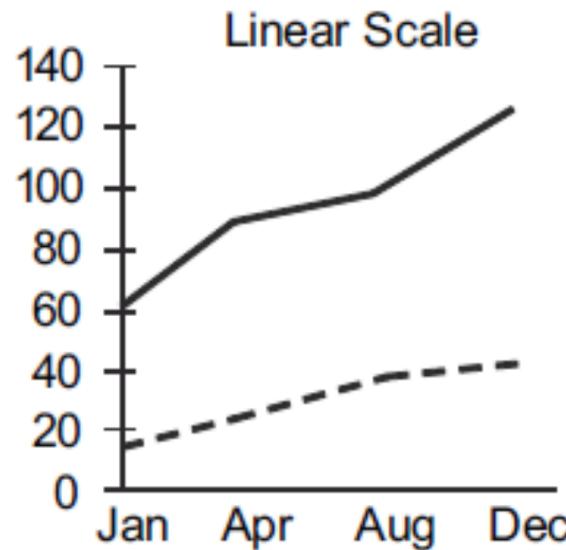
4. Ranges

- Select meaningful axis ranges
(Robbins, 2005; Tufte, 2006; Strange, 2007)



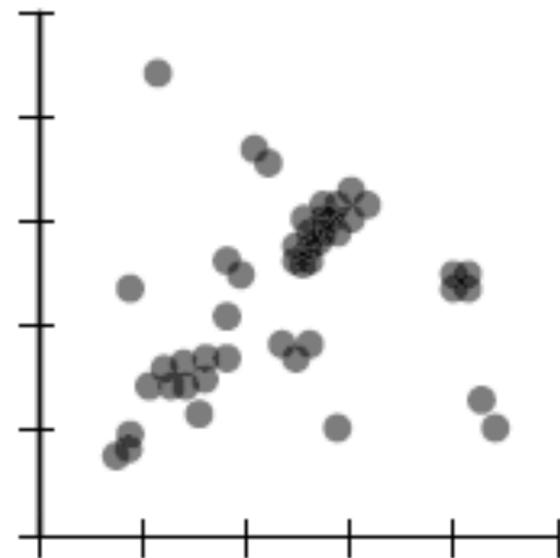
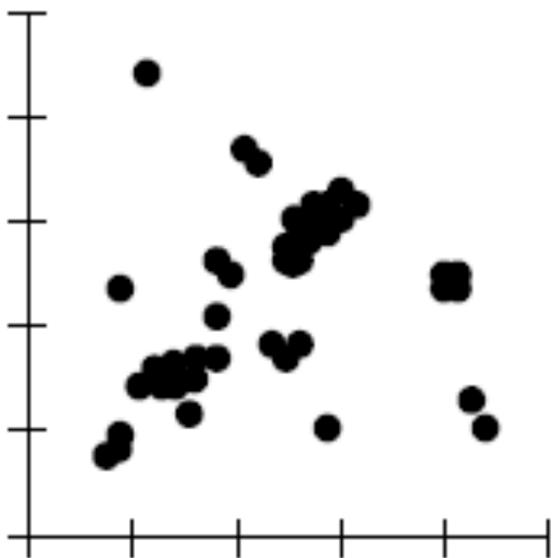
5. Transformations

- Data transformations and carefully chosen aspect ratios can be used to emphasize rates of change for time-series data (Cleveland, 1994 [p. 66, 95, 103]).



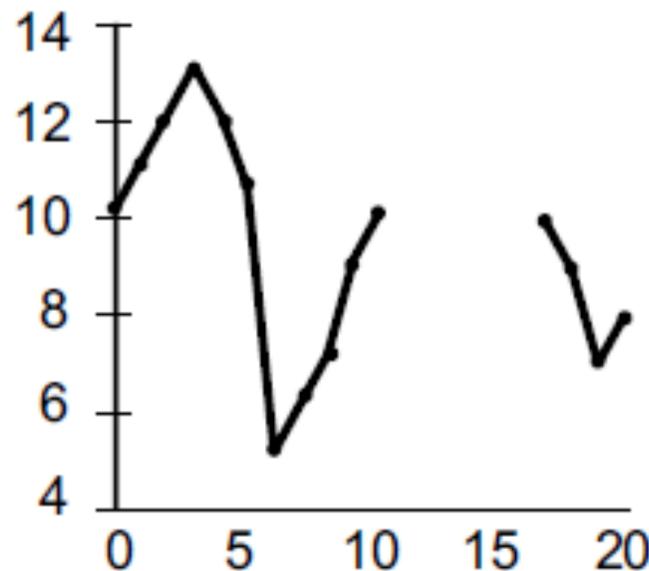
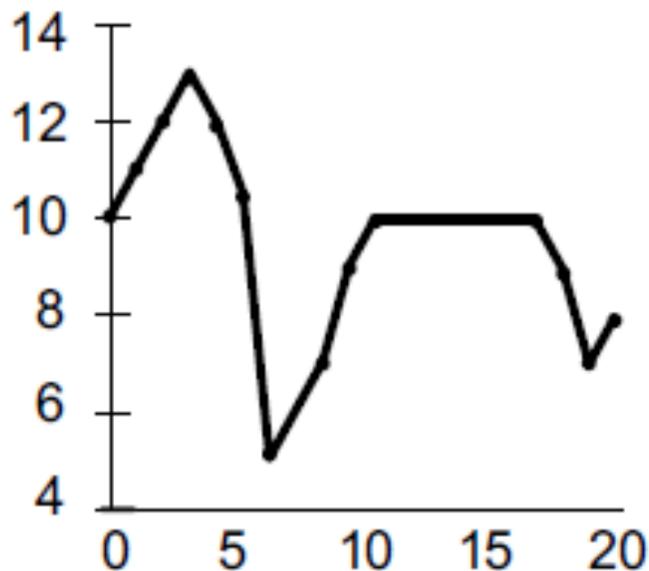
6. Show Density

- Plot overlapping points in a way that **density difference become apparent** (Few, 2009; Cleveland, 1994).



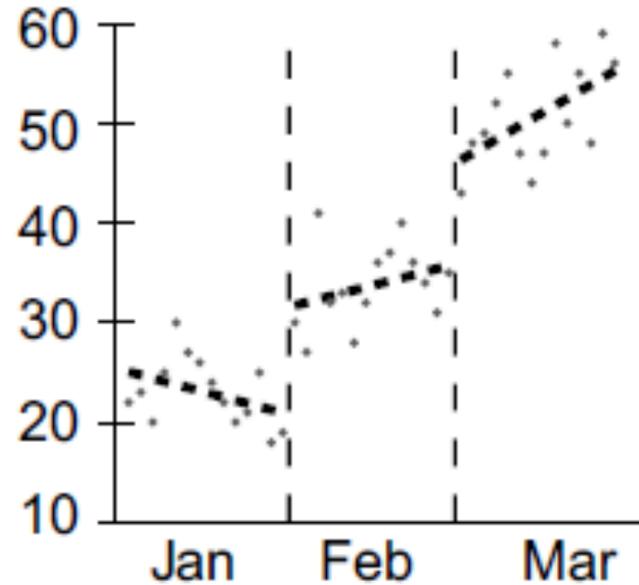
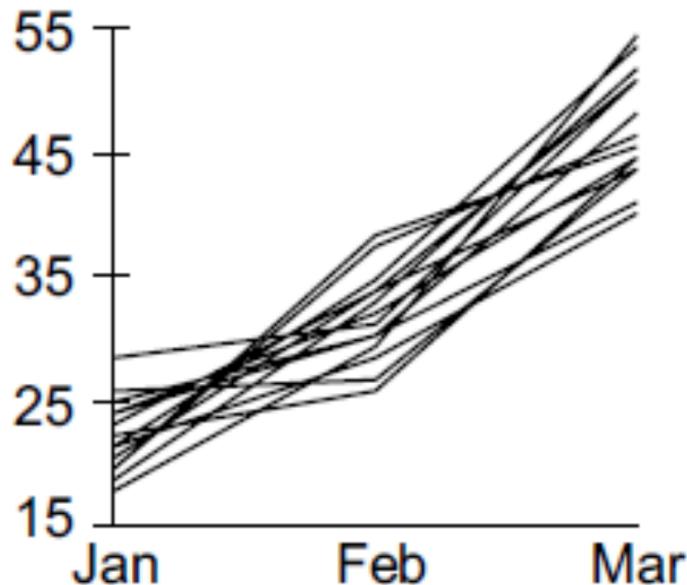
7. Connections

- Use lines when connecting sequential data in time-series plots (Strange, 2007).



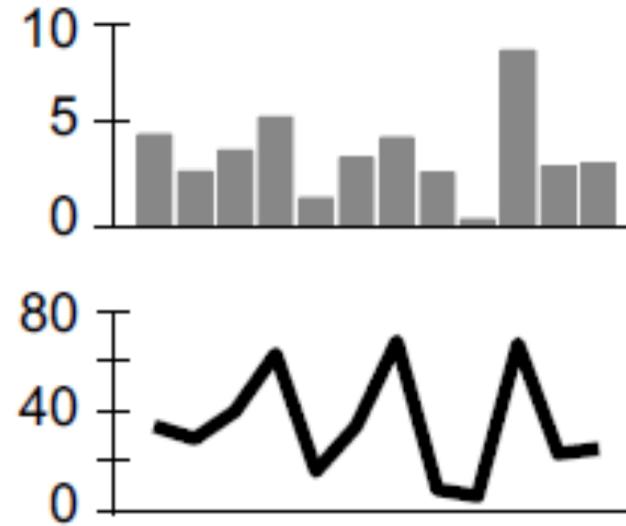
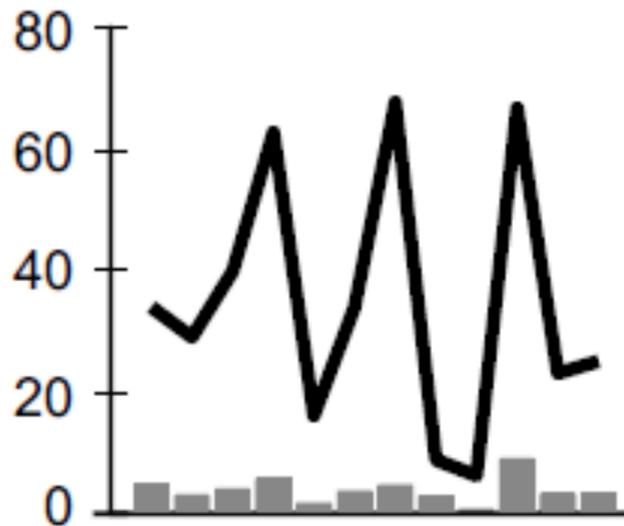
8. Aggregates

- Aggregate larger data sets in meaningful ways
(Cleveland & Devlin, 1980; Chambers, 1983)



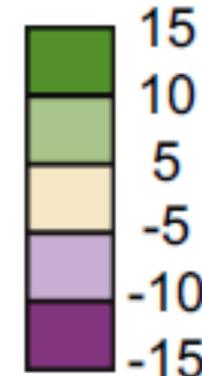
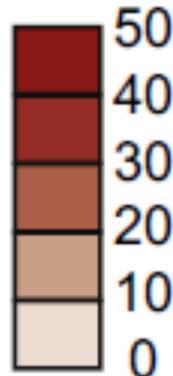
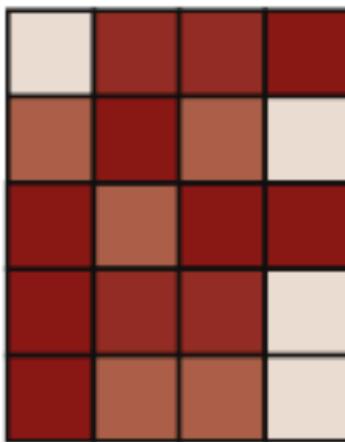
9. Comparison

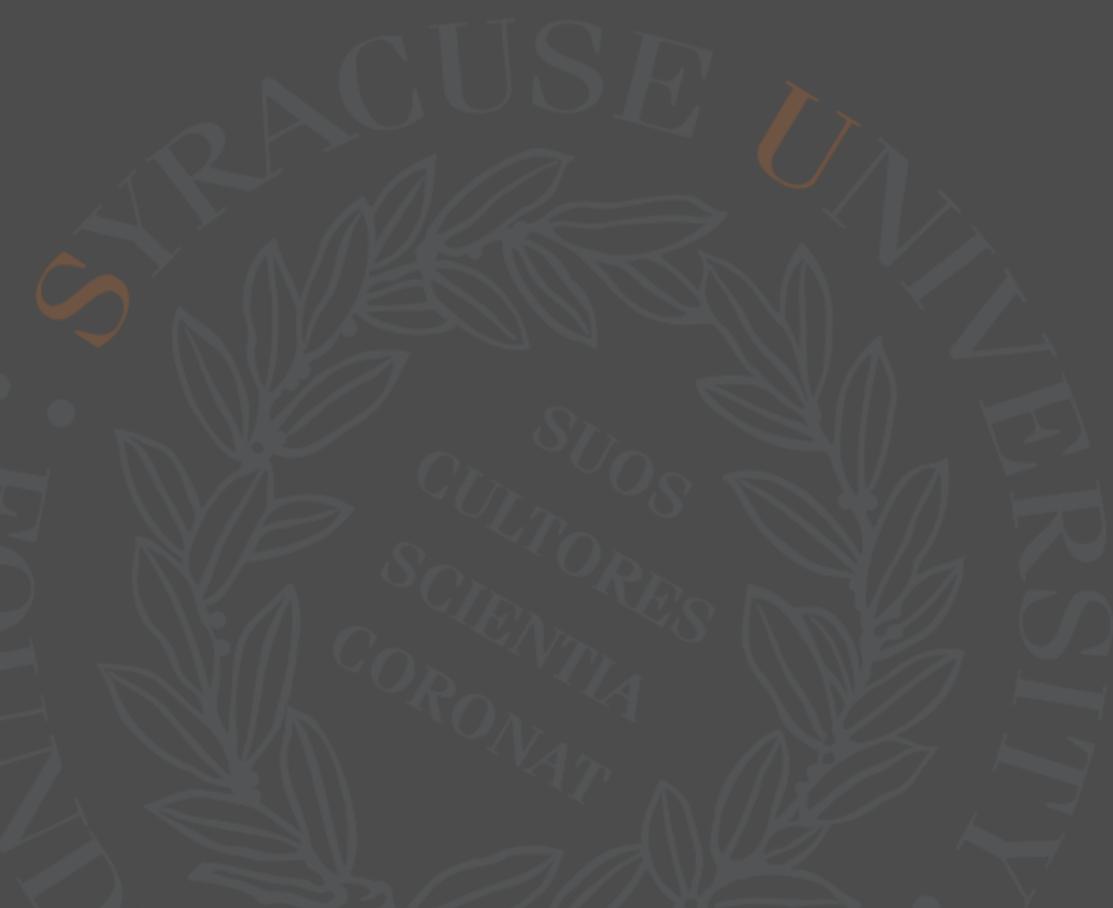
- **Keep axis ranges as similar as possible**
→ to compare variables
(Cleveland, 1994; Few, 2009).



10. Colors

- Select an appropriate color scheme based on the type of data (Brewer, 1994; Harrower and Brewer, 2003).





School of Information Studies
SYRACUSE UNIVERSITY

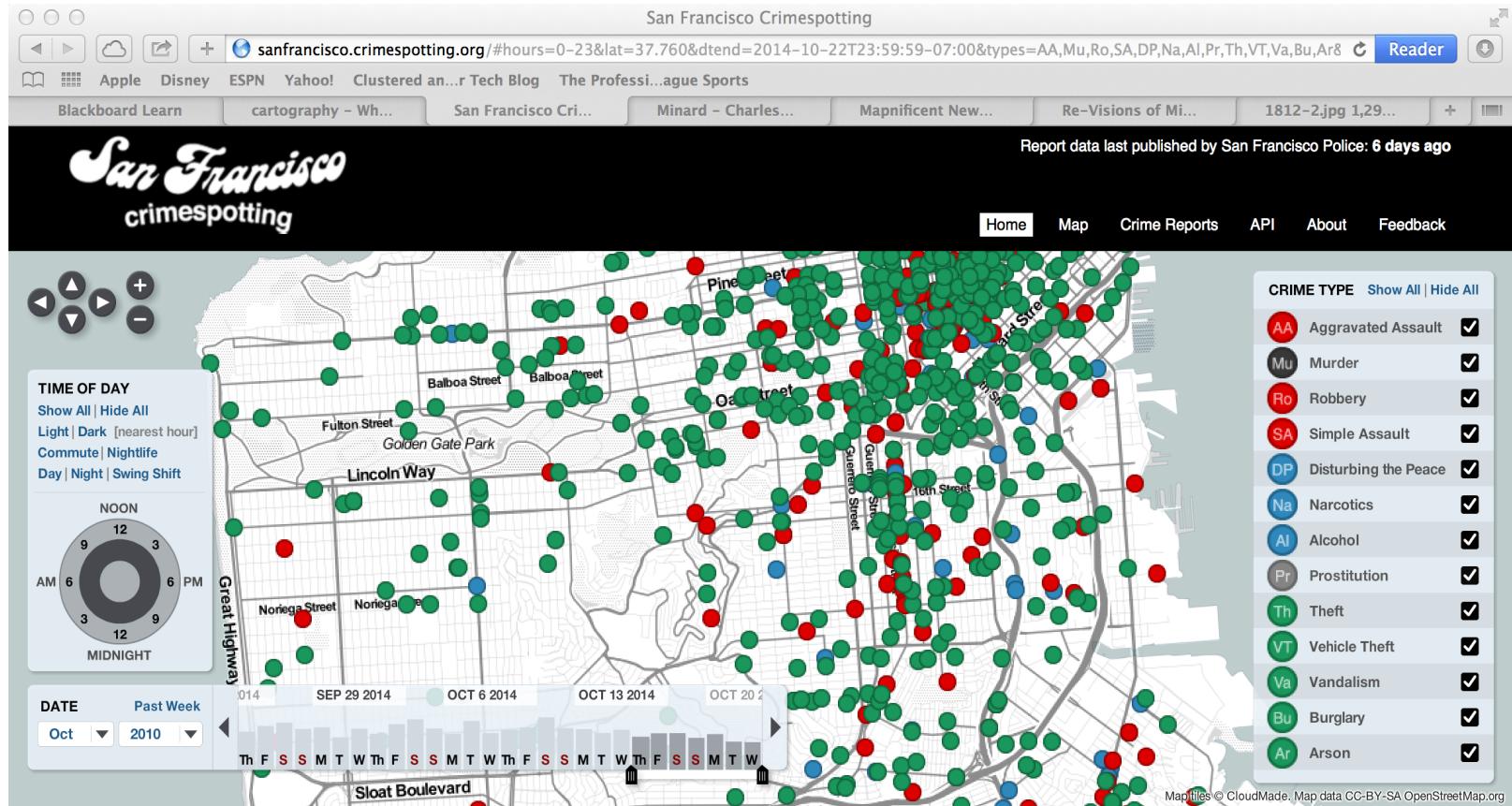
Example Map Visualizations

Mashup

- Anything that brings together disparate influences or elements
- Relative to application development
 - Bringing together various sources of data to create a new product with unique value

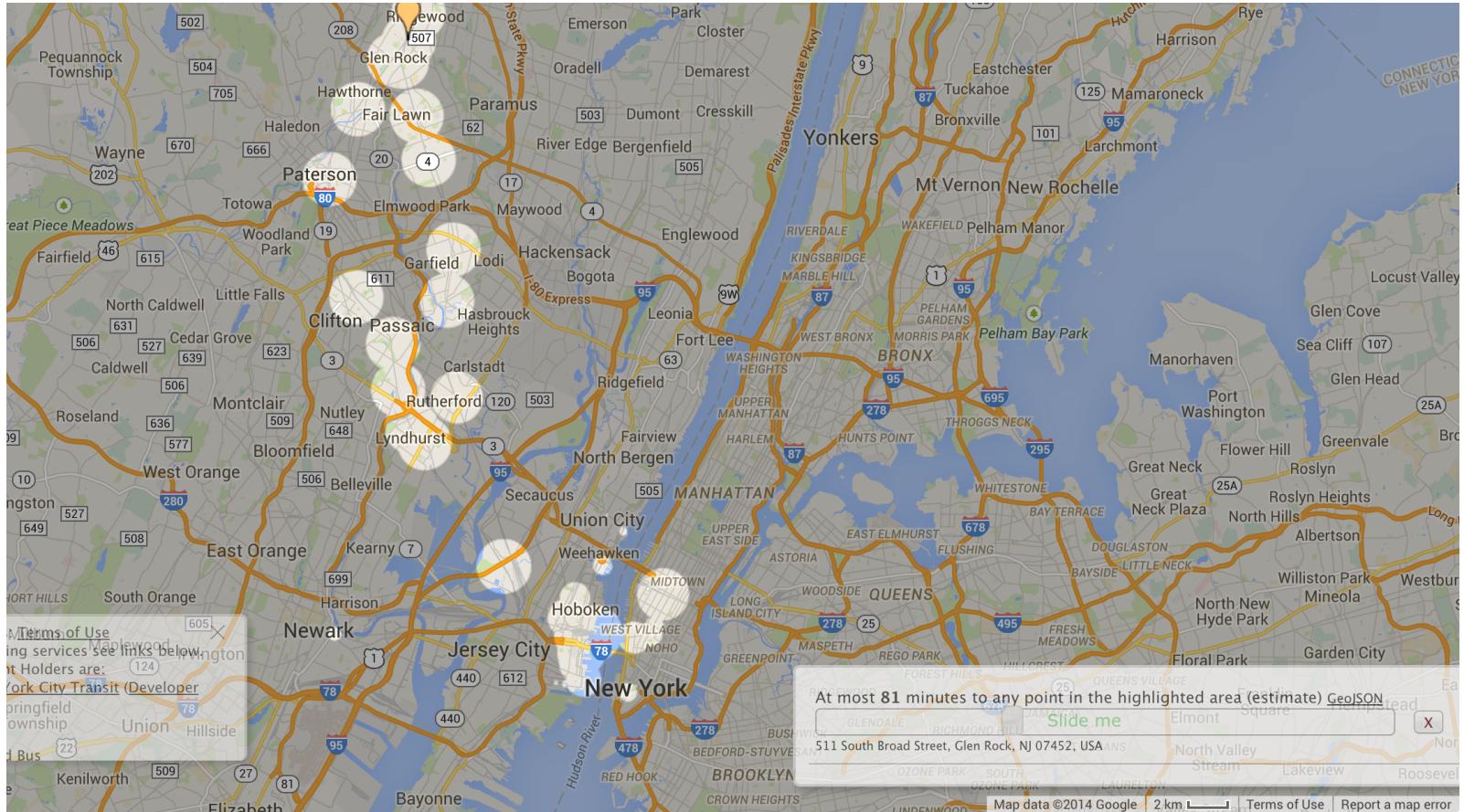
Mashup Examples

<http://sanfrancisco.crimespotting.org>



Mashup Examples

<http://www.mapnificent.net>



Question

- Explore visualizations: “Play” with the apps
- Question:
Is the visualization **useful**? If so, **how**?

GGPLOT Map Introduction

GGplot - Mapping Key Concepts

- `map_data`
Get the data on the region to be mapped.
- `geom_map`
How to render the map (e.g., colors, heatmaps).
- `coord_map`
Make sure the map is not “stretched.”

GGplot - Mapping Key Concepts

- map_id is part of the AES
 - Logical way to describe data (e.g., “new york”)
 - *Note that GGplot requires all lower case.*
- Longitude and latitude:
 - Can also add a specific physical location using geom_point; coordinates can be:
 - Hard coded
 - Via a conversion from logical to physical (e.g., use JSON to access google api)

GGPlot R Code

```
> dummyDF <- data.frame(state.name,  
                         stringsAsFactors=FALSE)
```

```
> dummyDF$state <-  
    tolower(dummyDF$state.name)
```

→ Note state.name is data available within R.
(There is additional info in the data set.)

```
> us <- map_data("state")
```

Creating a Simple Map

```
> map.simple <- ggplot(dummyDF, aes(map_id = state))

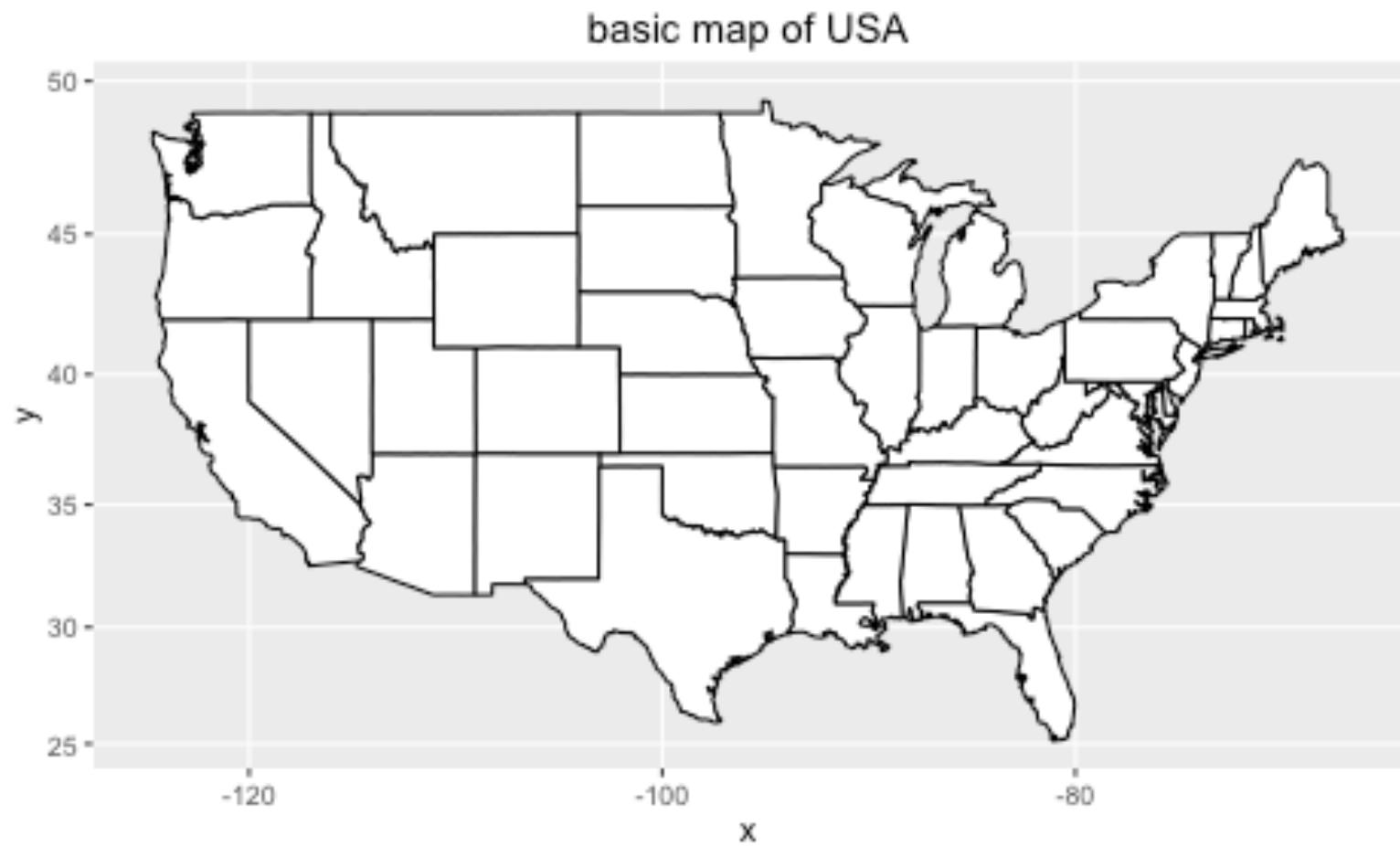
> map.simple <- map.simple +
  geom_map(map = us, fill="white", + color="black")

> map.simple <- map.simple +
  expand_limits(x = us$long, y = us$lat)

> map.simple <- map.simple +
  coord_map() + ggtitle("basic map of USA")

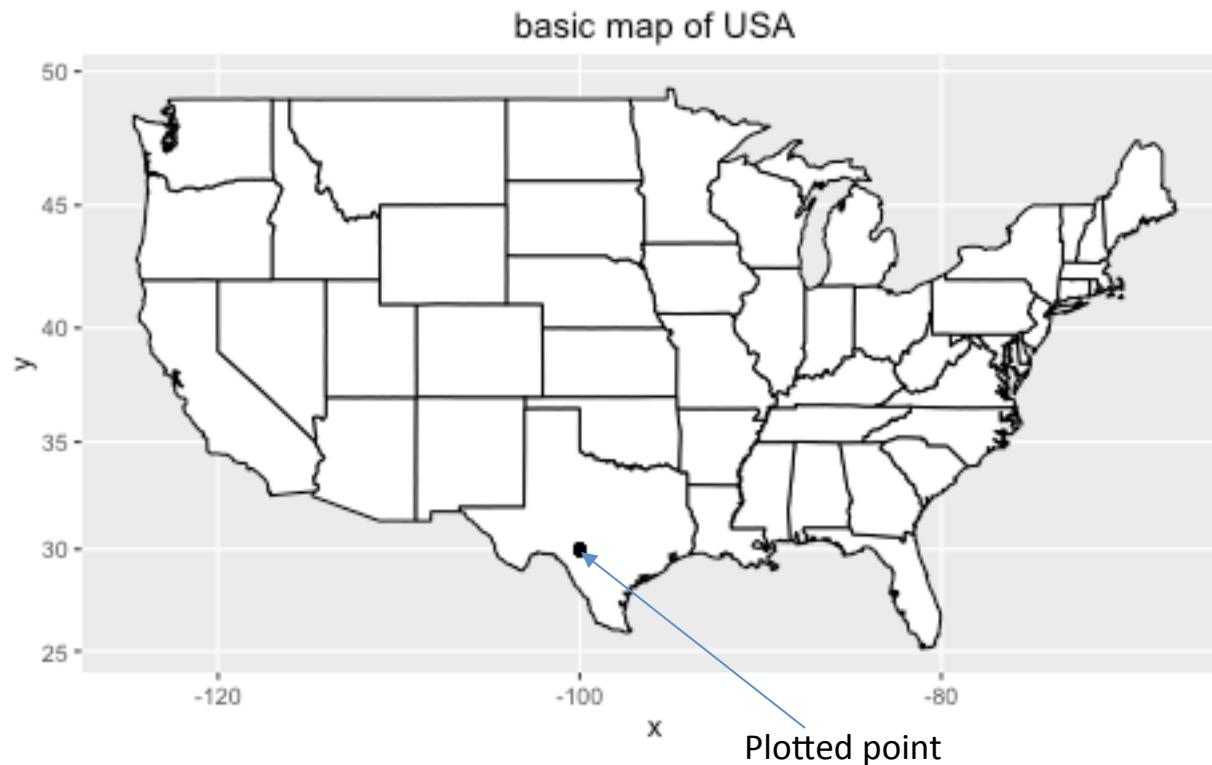
> map.simple
```

Creating a Simple Map



Add a Point to the Map

```
map.simple + geom_point(aes(x = -100, y = 30))
```



Practice

Create a map and put a mark
somewhere on the map.

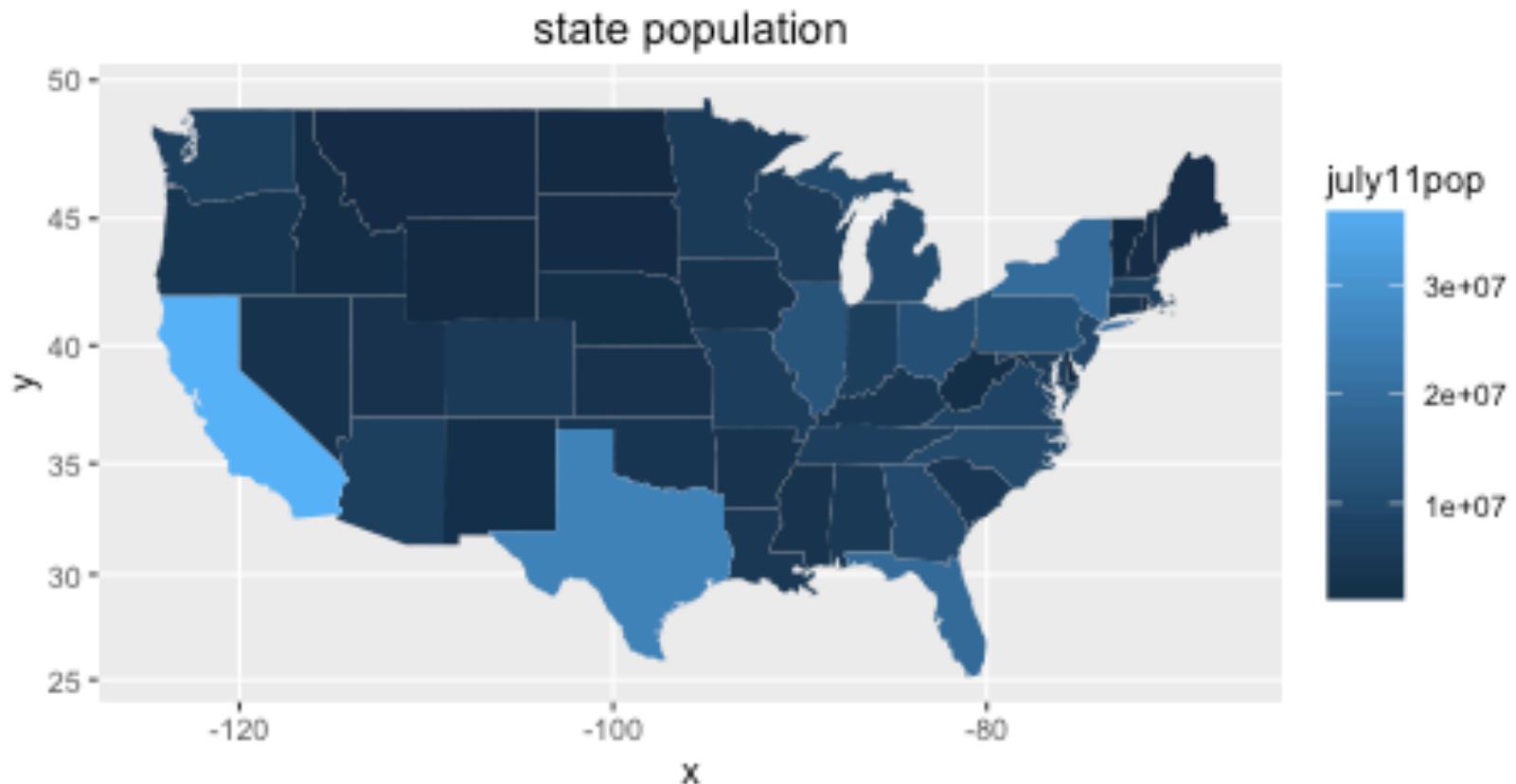
Maps and Data

A Real Example in R

Read in and Plot Census Data

```
#Read in the states and make lower case  
> dfStates <- readCensus()  
> dfStates$state <- tolower(dfStates$region)  
  
> map.popColor <- ggplot(dfStates, aes(map_id = state))  
> map.popColor <- map.popColor +  
    geom_map(map = us, aes(fill=july11pop))  
  
> map.popColor <- map.popColor +  
    expand_limits(x = us$long, y = us$lat)  
> map.popColor <- map.popColor +  
    coord_map() + ggttitle("state population")  
> map.popColor
```

Viewing Census Data



Getting a point to Map

```
> latlon <- geocode("syracuse university,  
                      syracuse, ny")
```

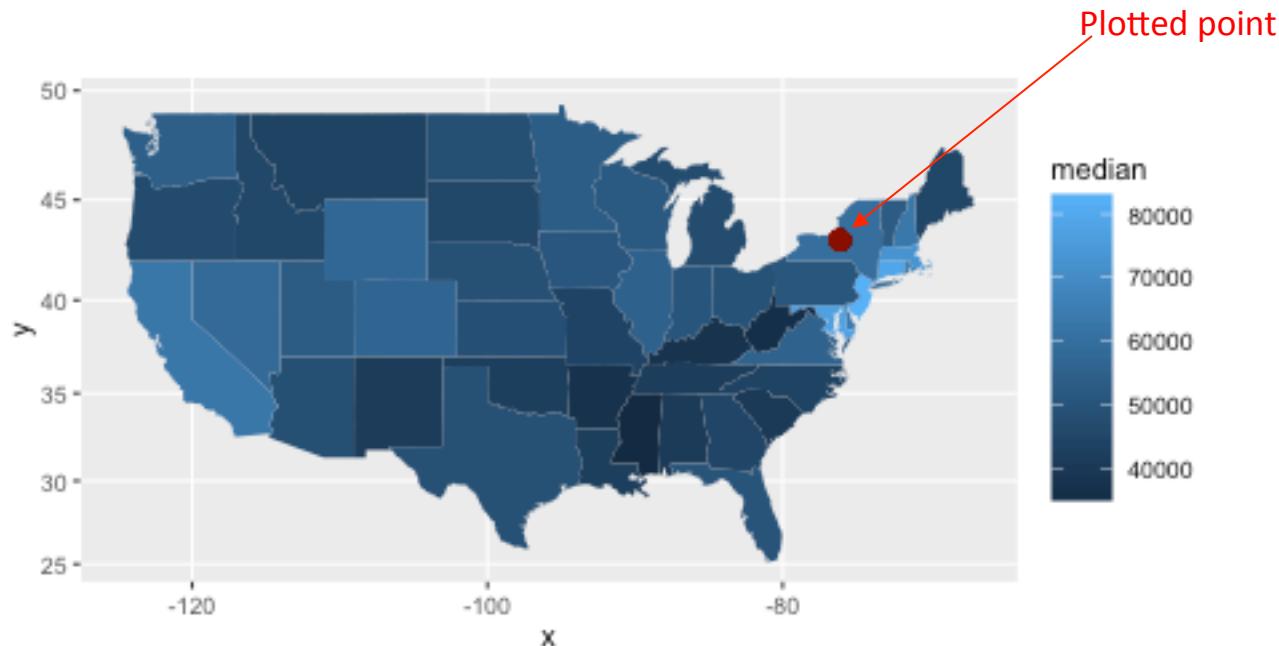
Information from URL : <http://maps.googleapis.com/maps/api/geocode/json?address=syracuse%20university,%20syracuse,%20ny&sensor=false>

```
> latlon  
    lon      lat  
1 -76.13452 43.03857
```

→ Could have used addr2latlon function we previously created!

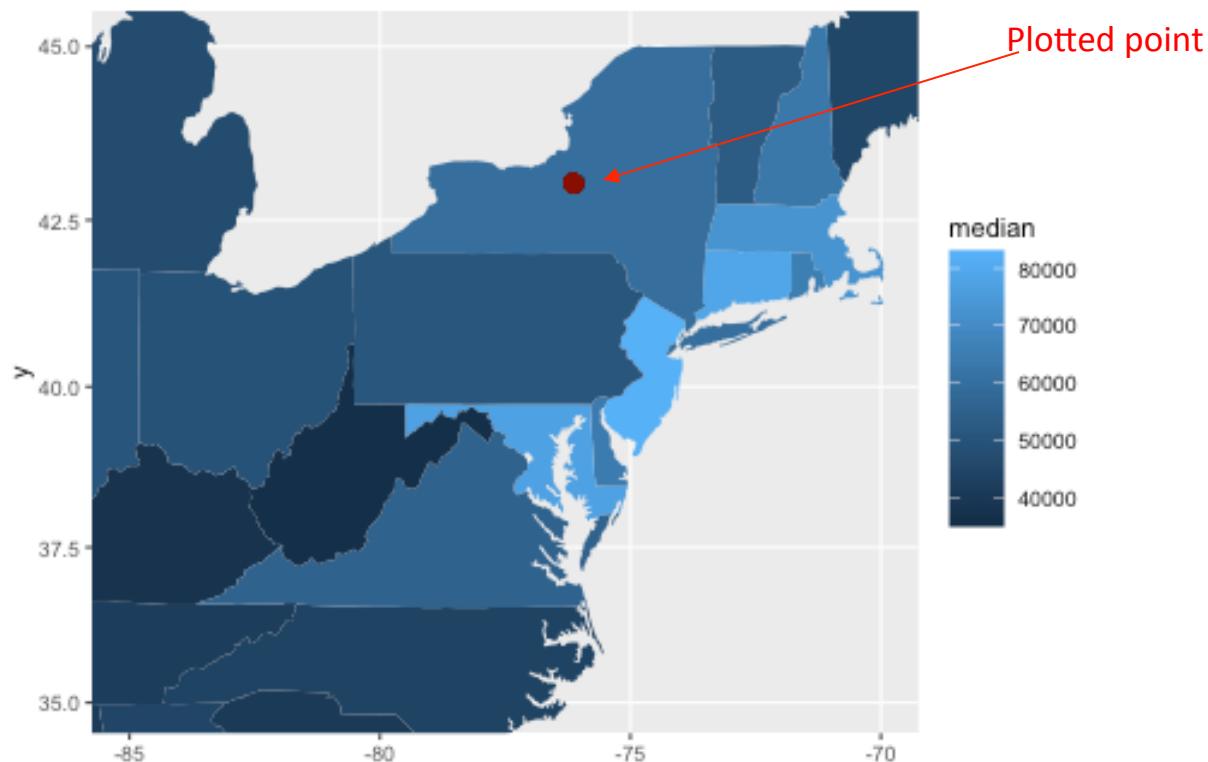
Plotting the Point on the Map

```
>map.popColor + geom_point(aes(x = latlon$lon,  
y = latlon$lat), color="darkred", size = 3)
```



Zooming

```
> g <- map.popColor + geom_point( aes(x = latlon$lon,  
y = latlon$lat), color="darkred", size = 3)  
> g + xlim(-85,-70) + ylim(35,45) + coord_map()
```



Activity

Create a map of the United States,
and put a mark in Miami, Florida.

Maps and Data

Example - Map and Data

Data on 10 Cities data set includes:

- Name (City, State)
- Business rating (1–10)
- Weather rating (1–10)
- Cost-of-living rating (1–10)

→ We want to map this information.

Example - Map and Data

City	State	Business Rating	Weather Rating	Living Rating
Manhattan	NY	10	5	7
Boston	MA	7	3	6
Philadelphia	PA	6	6	6
Tampa	FL	5	7	7
Chicago	IL	7	3	5
Boise	ID	3	6	4
San Francisco	CA	10	10	6
Seattle	WA	7	7	8
Houston	TX	5	2	2

Loading Data Into R

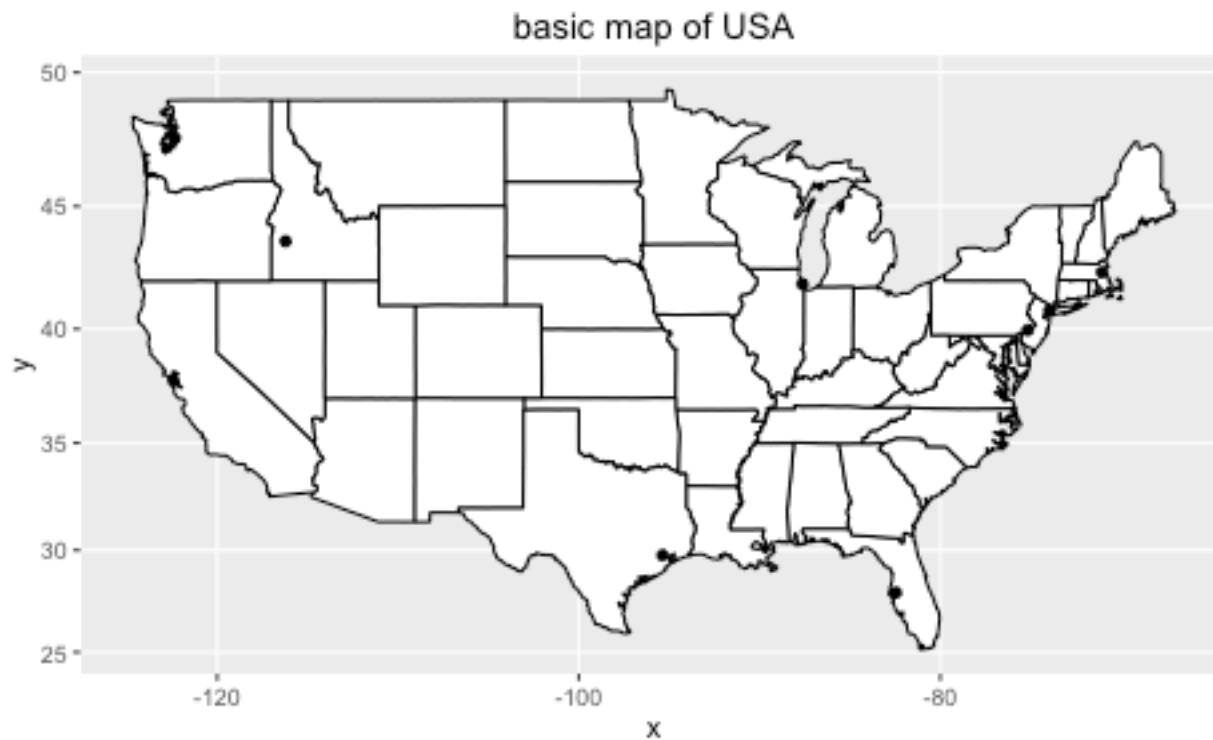
```
> cities <- c("Manhattan, NY", "Boston, MA",
  "Philadelphia, PA", "Tampa, FL", "Chicago, IL", "Boise, ID",
  "San Francisco, CA", "Seattle, WA", "Houston, TX")

> bus <- c(10,7,6,5,7,3,10,7,5)
> weather <- c(5,3,6,7,3,6,10,7,2)
> living <- c(7,6,6,7,5,4,6,8,2)

> city.df <- data.frame(cities, bus, weather, living)
> city.df$state <- statesFake
> city.df$geoCode <- geocode(cities)
```

Putting Points on a Map

```
> map.simple + geom_point(data=cities.df,  
aes(x = geoCode$lon, y = geoCode$lat))
```



Creating an Enhanced Map

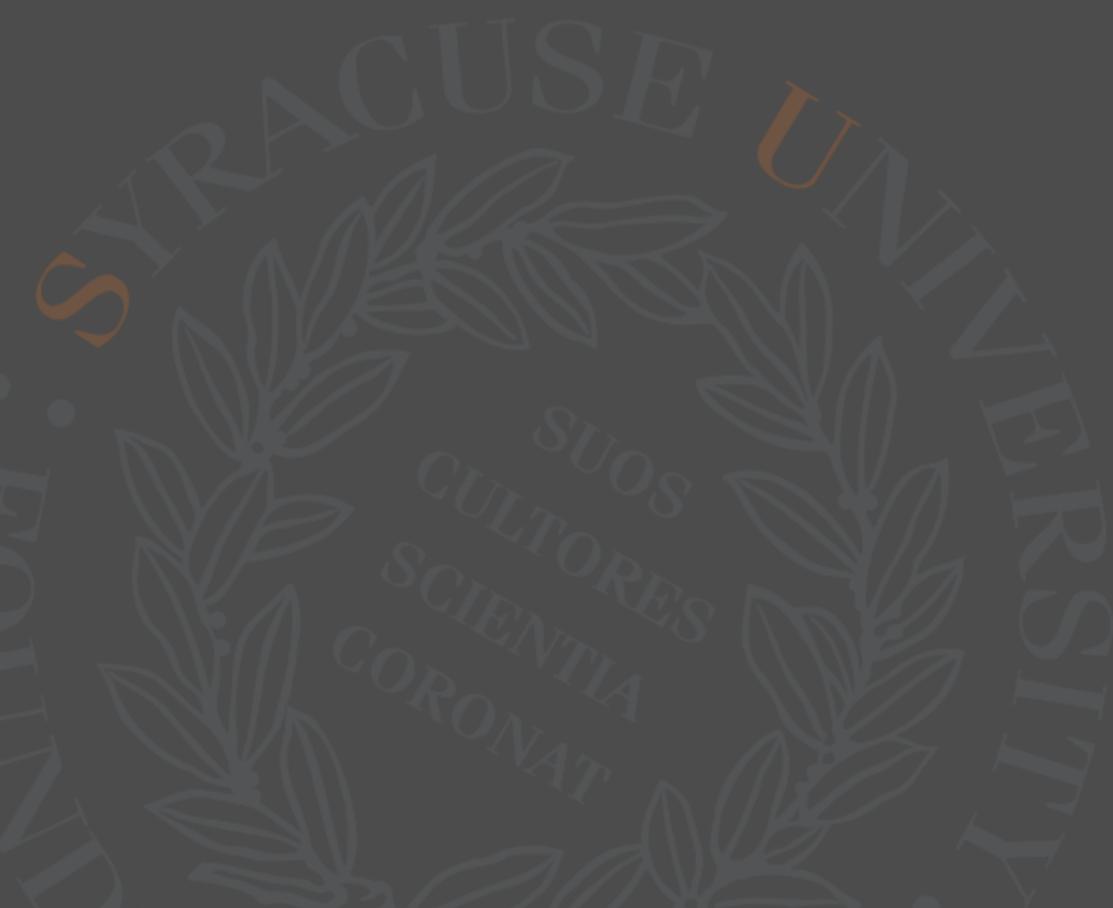
```
> map.simple + geom_point(data=city.df, aes(x =  
geoCode$lon, y = geoCode$lat, size=bus, color=weather))
```



Marker size represent business ranking
Color represent weather

Question

Where might maps and data be useful?



School of Information Studies
SYRACUSE UNIVERSITY

Model Overview

So Many Models

- Someone who appears in magazines?
- A small car or railroad?
- A data model (such as an ERD)?
- An “object” created in R that we can use for data understanding and data prediction

Why Use a Linear Model?

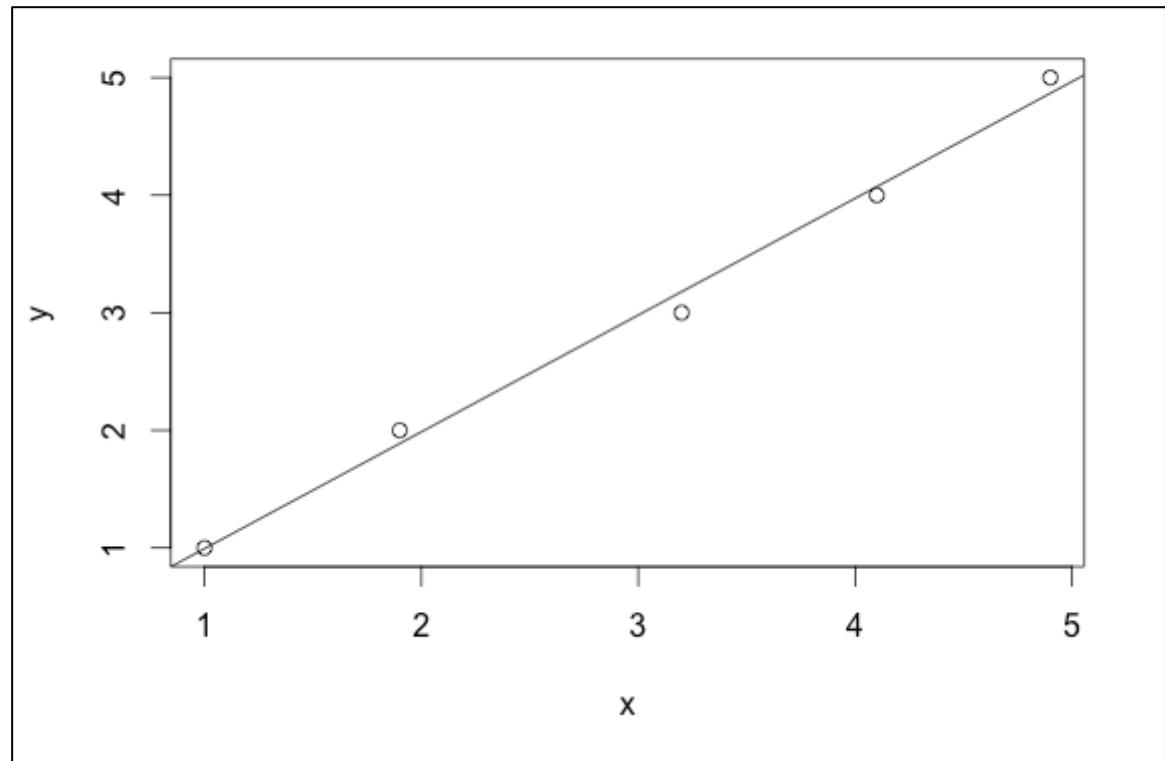
A linear model

—used for prediction (kind of like extrapolation)

Note that:

- It's not "perfect"
- Minimize "distance"
from points to the line

$$Y = MX + B$$



An Example

- Car maintenance (how often to change the oil)
 - We manage a “fleet” of cars
 - Cars get replaced every three years
 - Have information on:
 - Past repairs
 - Miles driven
 - # of oil changes during past three years
- Can we build a model to predict the cost of repairs?

Data for the Analysis

	oilChanges	repairs	miles
1	3	300	20100
2	5	300	23200
3	2	500	19200
4	3	400	22100
5	1	700	18400
6	4	420	23400
7	6	100	17900
8	4	290	19900
9	3	475	20100
10	2	620	24100
11	0	600	18200
12	10	0	19600
13	7	200	20800
14	8	50	19700

Question

Which are independent and which are dependent variables?

Why?

	oilChanges	repairs	miles
1	3	300	20100
2	5	300	23200
3	2	500	19200
4	3	400	22100
5	1	700	18400
6	4	420	23400
7	6	100	17900
8	4	290	19900
9	3	475	20100
10	2	620	24100
11	0	600	18200
12	10	0	19600
13	7	200	20800
14	8	50	19700

Working Through an Example

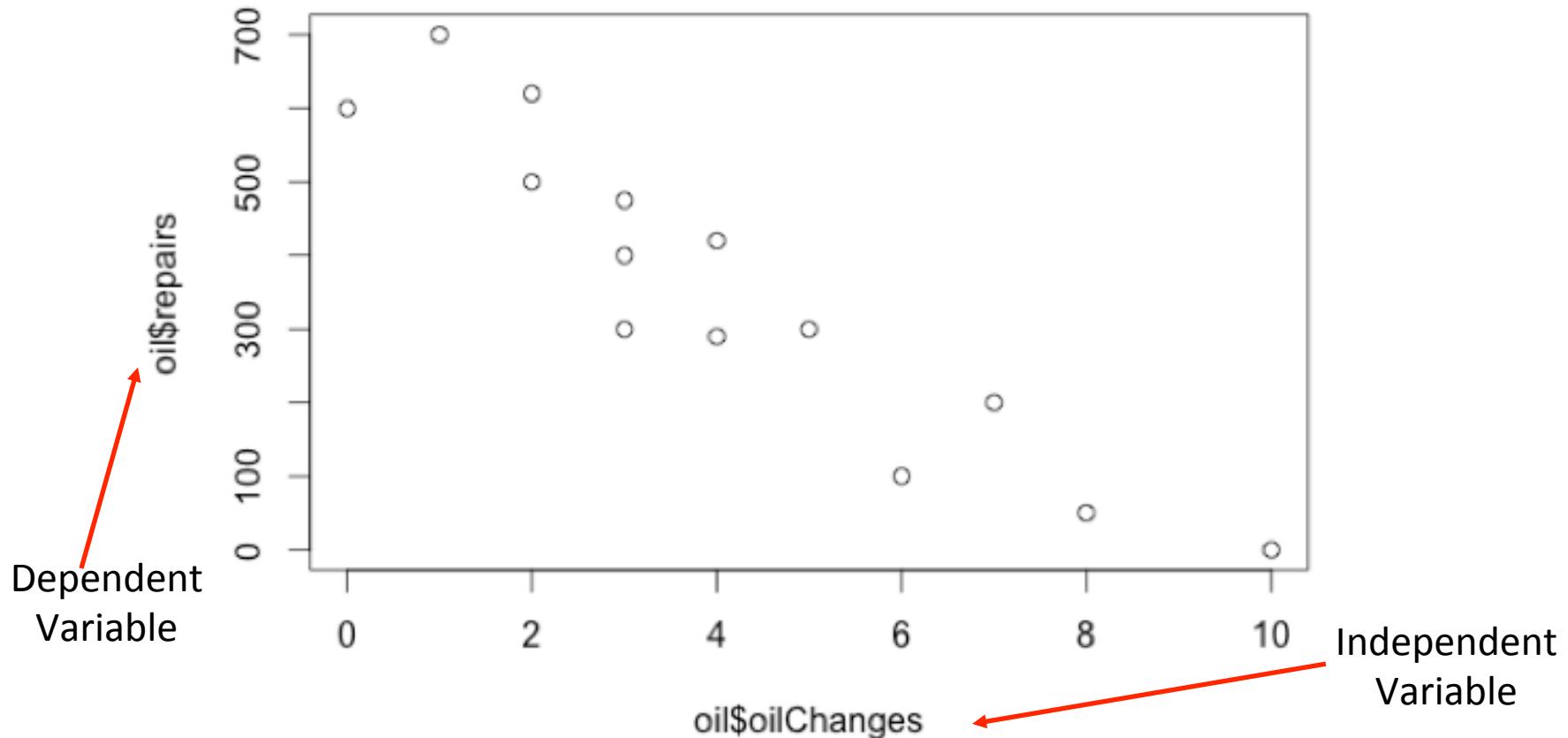
Back to our Data

Let's use this data to build a model

	oilChanges	repairs	miles
1	3	300	20100
2	5	300	23200
3	2	500	19200
4	3	400	22100
5	1	700	18400
6	4	420	23400
7	6	100	17900
8	4	290	19900
9	3	475	20100
10	2	620	24100
11	0	600	18200
12	10	0	19600
13	7	200	20800
14	8	50	19700

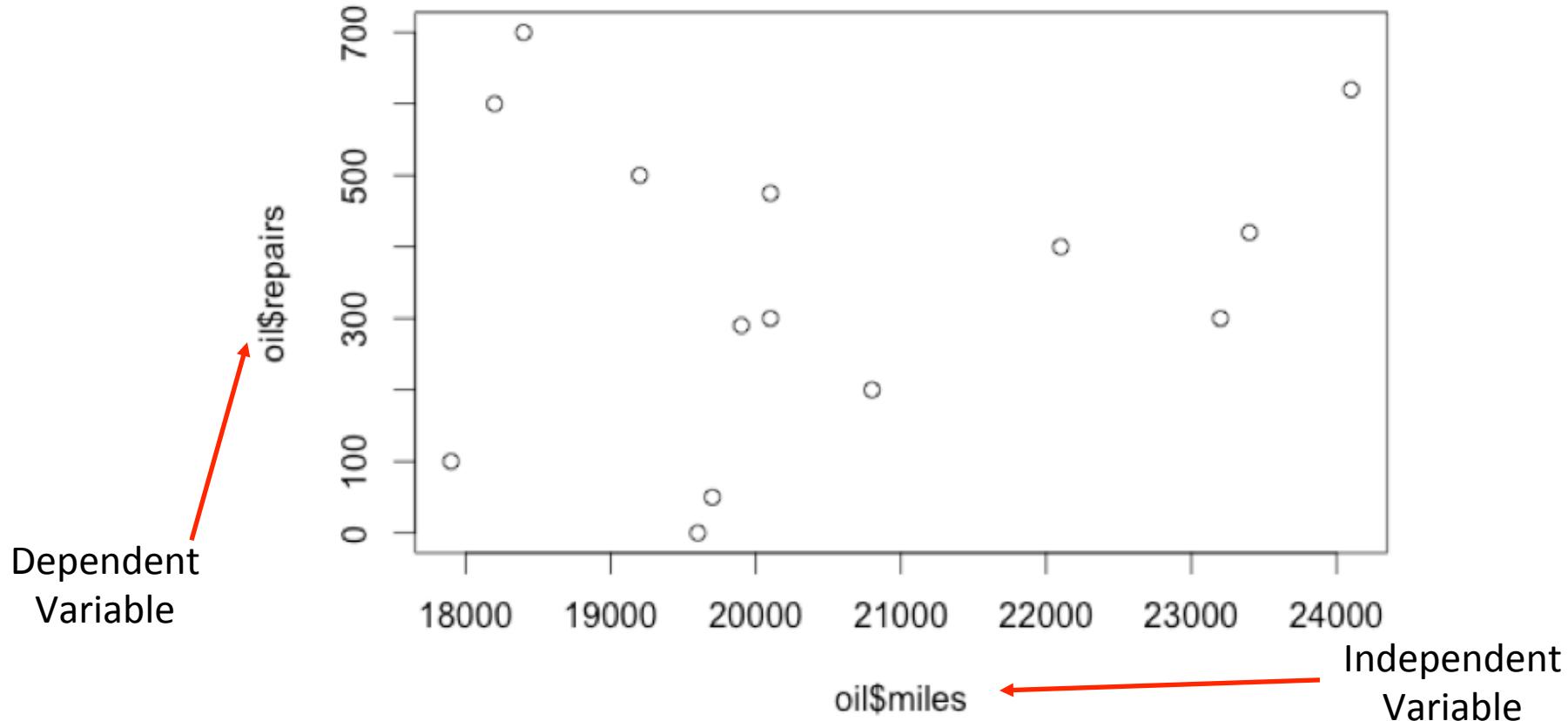
Exploring the Data

```
> plot(oil$oilChanges, oil$repairs)
```



Exploring the Data

```
> plot(oil$miles, oil$repairs)
```



Generating the First Model

```
> model1 <- lm(formula=repairs ~ oilChanges, data=oil)
```

```
> summary(model1)
```

Call:

```
lm(formula = repairs ~ oilChanges, data = oil)
```

Residuals:

Min	1Q	Median	3Q	Max
-136.208	-48.195	-0.211	54.782	119.803

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	652.191	40.537	16.089	1.74e-09 ***
oilChanges	-71.994	8.202	-8.778	1.44e-06 ***

Signif. codes: 0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 82.72 on 12 degrees of freedom

Multiple R-squared: 0.8653, Adjusted R-squared: 0.854

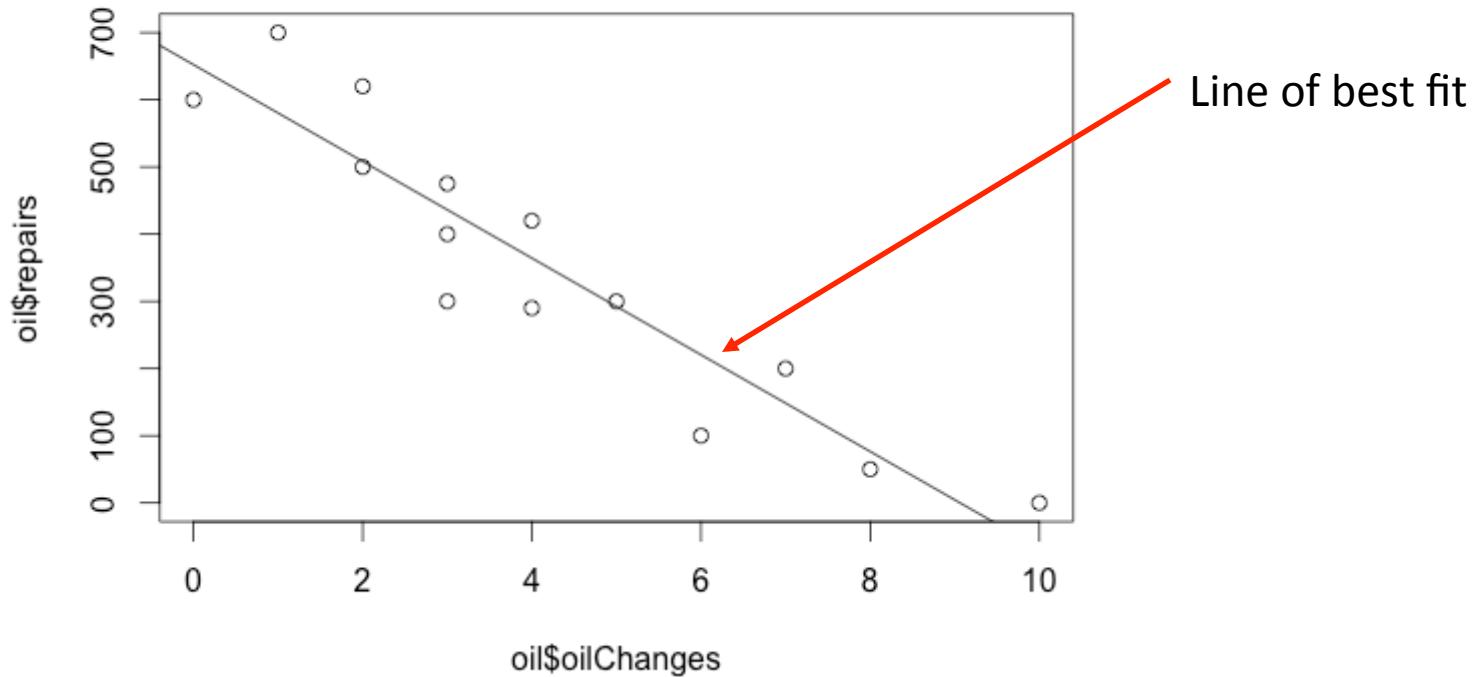
F-statistic: 77.05 on 1 and 12 DF, p-value: 1.436e-06

Interpreting the Model

- R-squared value 0.8653.
- Known as the coefficient of determination
- The proportion of the variation that is accounted for in the dependent variable by the whole set of independent variables.
- The closer to 1.0 , the greater the influence the independent variable has on predicting the value of the dependent variable.
- The R-squared value of 0.8653 indicates that the oil changes accounts for 86.53% of the cost of repairs.

Looking at the “abline”

abline(model1)



The model suggests that we should do as many oil changes as possible.

→ it predicts very low (almost 0) repairs if we do 9 or more oil changes, but about \$680 if we do no oil changes.

Question

What if we factor in the cost to change the oil?

- How “model” the cost?
- What might be some ranges of the cost?

Working Through a Refined Example

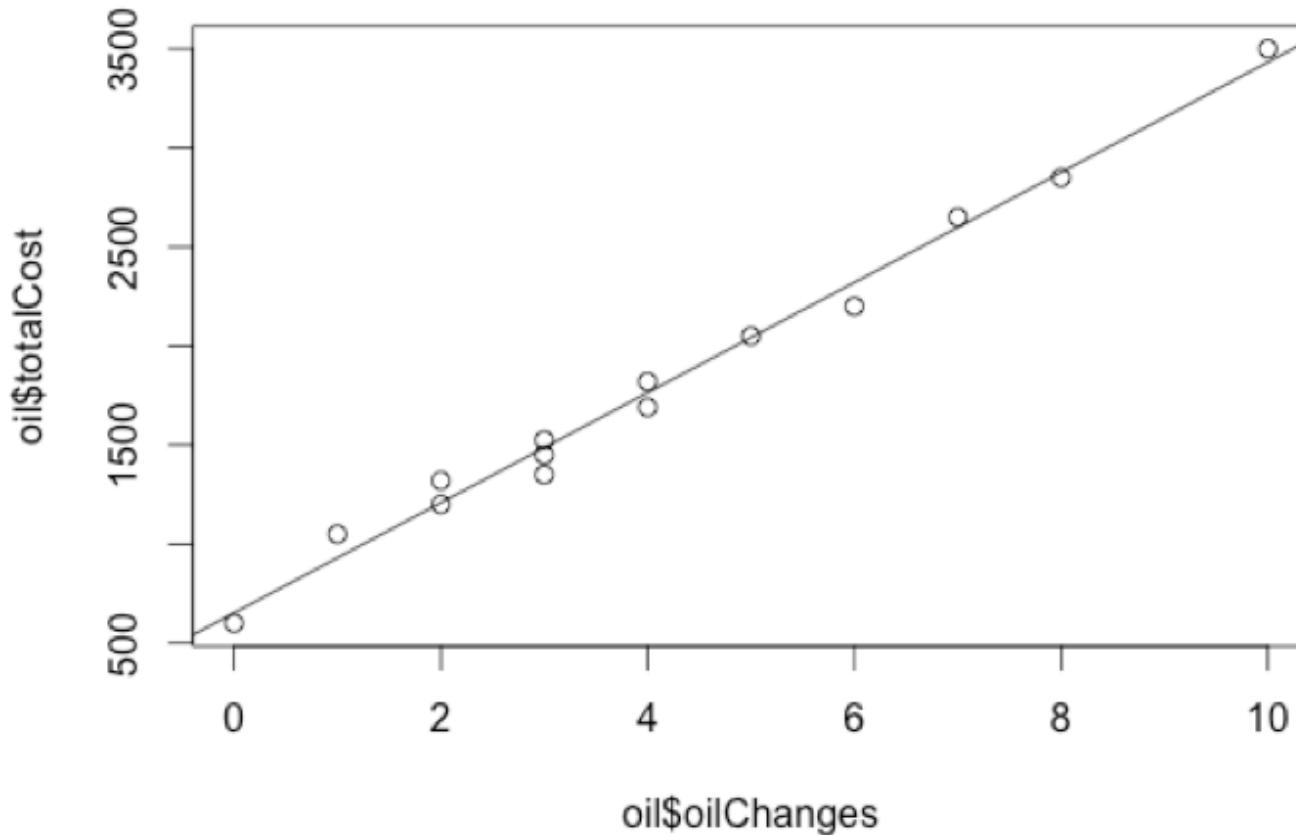
Include the Cost of an Oil Change

- What if oil changes cost \$350 each?

```
> oil$oilChangeCost <- oil$oilChanges * 350  
> oil$totalCost <- oil$oilChangeCost +  
                           oil$repairs  
> m <- lm(formula=totalCost ~ oilChanges,  
                           data=oil)  
> plot(oil$oilChanges, oil$totalCost)  
> abline(m)
```

Viewing the Data

- What if oil changes cost \$350 each?



Using the Model to Predict

- Prediction equation

```
> test = data.frame(oilChanges=0)
```

```
> predict(m,test, type="response")
```

652.191

```
> test = data.frame(oilChanges=5)
```

```
> predict(m,test, type="response")
```

2042.219

```
> test = data.frame(oilChanges=10)
```

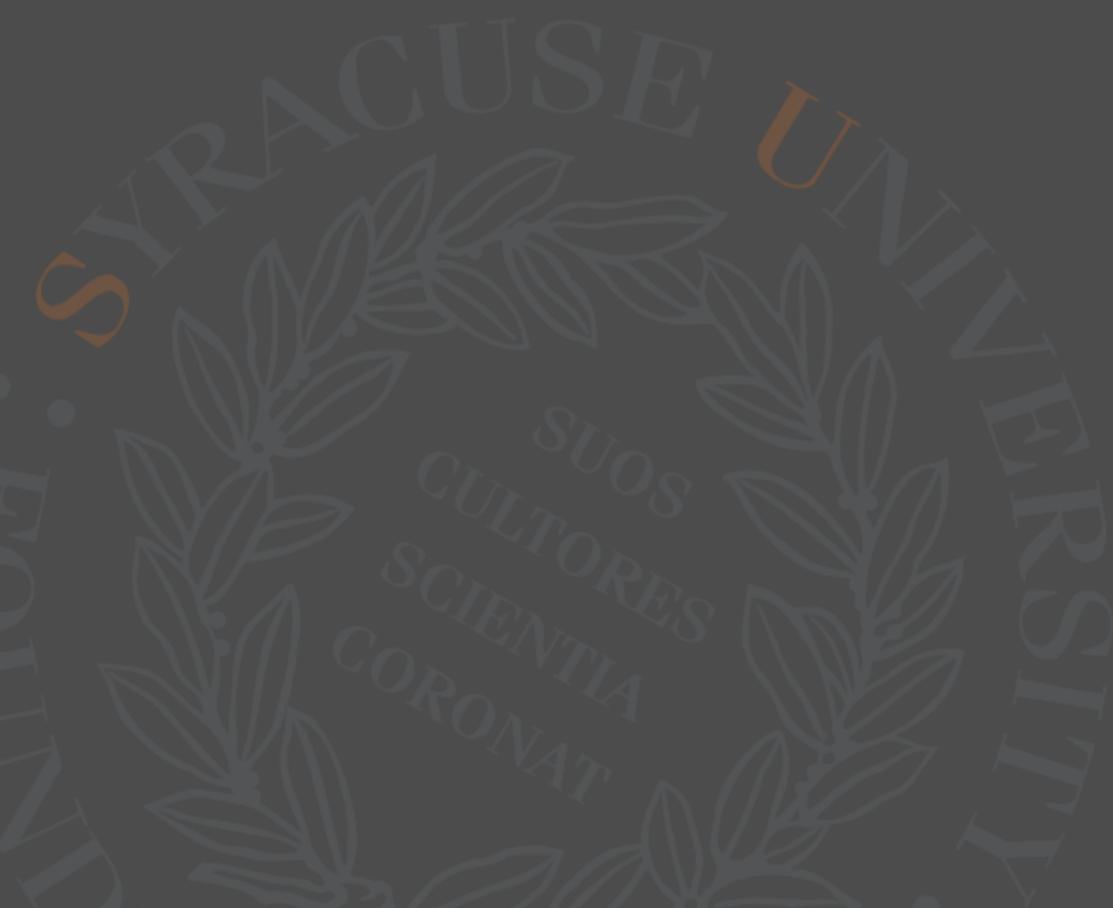
```
> predict(m,test, type="response")
```

3432.247

Question

How accurate is the model?

- Did we have all the facts?
- Did we have all the data?



School of Information Studies
SYRACUSE UNIVERSITY

Data Mining Overview

What is Data Mining?

- Use of algorithms and computers to discover novel and interesting patterns within data
- Diapers and beer
- Machine learning

Data Mining: Four Processes

1. Data preparation
 2. Exploratory data analysis
 3. Model development
 4. Interpretation of results
- Iterative process vs. linear

Data Mining: Four Processes

1. Data preparation
 - a) Most time-consuming
 - b) Data organization
 - c) Data completeness
 - d) Data accuracy
 - e) Data transformation
2. Exploratory data analysis
3. Model development
4. Interpretation of results

Data Mining: Four Processes

1. Data preparation
2. Exploratory data analysis
 - a) Preliminary data context assessment
 - b) Visualization analysis
 - c) Key values for parameters
3. Model development
4. Interpretation of results

Data Mining: Four Processes

1. Data preparation
2. Exploratory data analysis
3. Model development
 - a) Most complex/interesting
 - b) Test selection of data mining techniques
 - c) Example: “association rules mining”
4. Interpretation of results

Data Mining: Four Processes

1. Data preparation
2. Exploratory data analysis
3. Model development
4. Interpretation of results
 - a) Making sense of data mining output
 - b) Determine actionable conclusions

Data Mining: Four Processes

1. Data preparation
2. Exploratory data analysis
3. Model development
4. Interpretation of results

Question

→ How is this different from the overall data science process?

Associative Rule Mining

First Look at Data Mining

CHAPTER 17

Hi Ho, Hi Ho - Data Mining We Go



Data mining is an area of research and practice that is focused on discovering novel patterns in data. As usual, R has lots of possibilities for data mining. In this lecture we will begin experimentation with essential data mining techniques by trying out one of the easiest methods to understand: association rules mining. More beer and diapers, please!

What is Association Rules Mining

Association rules:

- Association rules are if/then statements.
- The rules help uncover relationships between seemingly unrelated data.

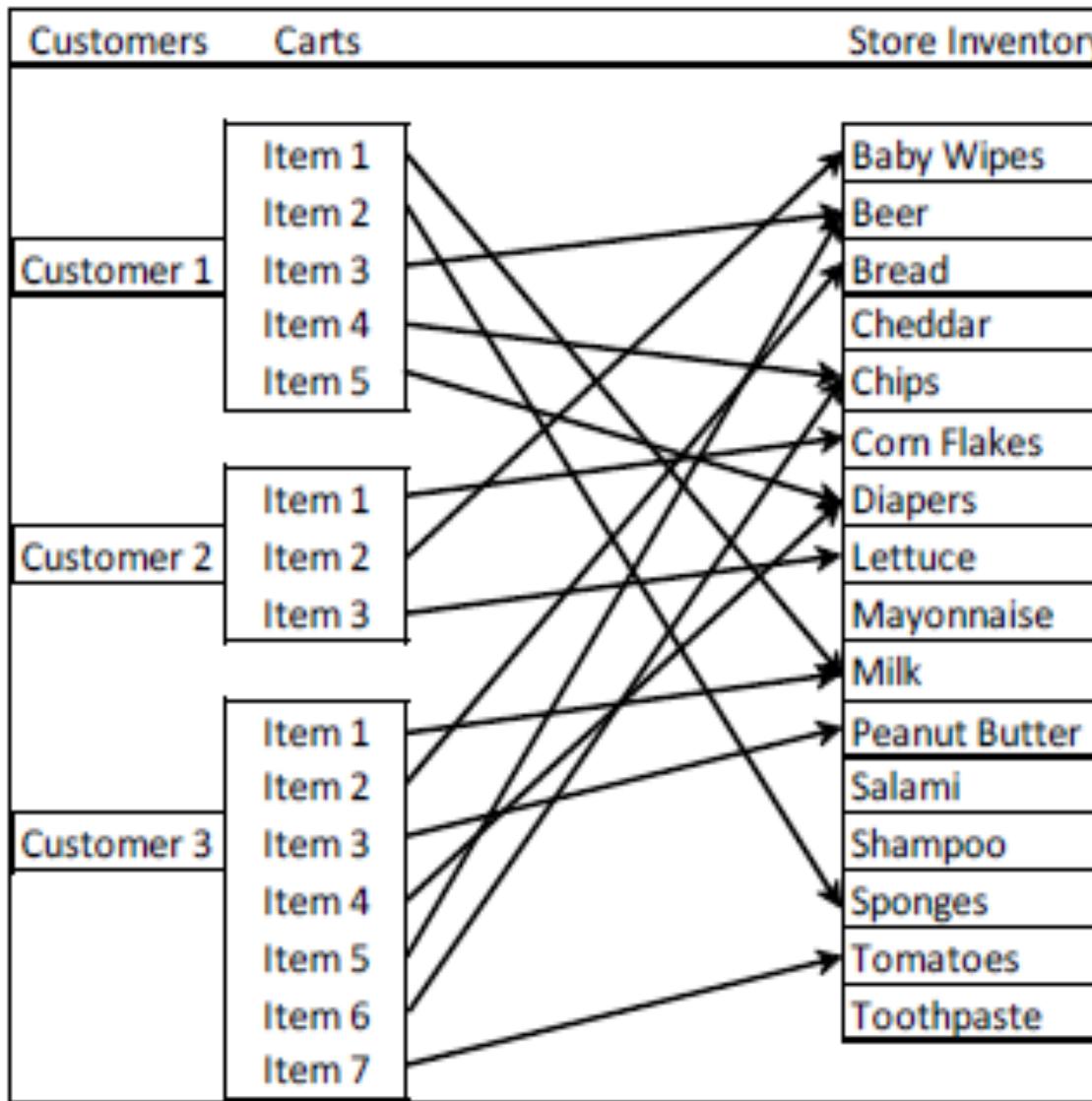
→ An example

“If a customer buys a dozen eggs, that person is 80% likely to also purchase milk.”

Example Dataset

- A row is a “basket” of items from one purchase
 - citrus fruit, semi-finished bread, margarine ,ready soups
 - tropical fruit, yogurt, coffee
 - whole milk
 - pip fruit, yogurt, cream cheese, meat spreads
 - other vegetables, whole milk, condensed milk
 - whole milk, butter, yogurt, rice, abrasive cleaner
 - rolls/buns

Use case: supermarket grocery cart



Support / Confidence / Lift

- **Support rule**
 - Proportion that a paring occurs across all baskets
→ # of rows having both A AND B / Total # of rows
- **Confidence quantity**
 - How frequently a particular pair occurs among all the items when the first item is present
→ # of rows having both A AND B / # of rows with A
- **Lift: Confidence / Probability of second item**
 - Confidence / Expected confidence

Expected confidence = # of rows with B / Total # of rows

An Example: Rule for Diapers → Beer

- Support value: 0.67
 - Diapers and beer occurred in two out of three carts
- Confidence value: 0.5
 - Beer occurred 50% of the time diapers were in the cart
- Lift value: 2.5 (probability of beer in cart: 20%)
[0.5/0.2]

Questions:

What are some **examples** of association rules mining “**in the real world**”?

What might be some **possible issues** with using this algorithm?

Associative Rule Mining in R

The DM Process – Data Prep

- **Data Mining Process 1: Data Preparation**

```
install.packages("arules")
```

```
library("arules")
```

- Groceries data set
- Ready to be analyzed

The DM Process – Explore

- **Data Mining Process 2: Exploratory Data Analysis**

`data(Groceries)`

`summary(Groceries)`

The DM Process – Explore/View

> Summary(Groceries)

transactions as itemMatrix in sparse format with
9835 rows (elements/itemsets/transactions) and
169 columns (items) and a density of 0.02609146

most frequent items:

whole milk	other vegetables	rolls/buns	soda	yogurt	(other)
2513	1903	1809	1715	1372	34055

element (itemset/transaction) length distribution:

sizes

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
2159	1643	1299	1005	855	645	545	438	350	246	182	117	78	77	55	46	29	14	14	9	11	4	6
24	26	27	28	29	32																	
1	1	1	1	3	1																	

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
1.000	2.000	3.000	4.409	6.000	32.000

The DM Process – Data Summary

- **Process 2: Exploratory Data Analysis**
 - Groceries data set → context
 - itemMatrix, sparse format
 - 9,835 rows, 169 columns
 - Row = basket/grocery cart
 - Column = grocery item/product
 - Item in grocery basket indication -> 1, 0
 - Items occurring most frequently

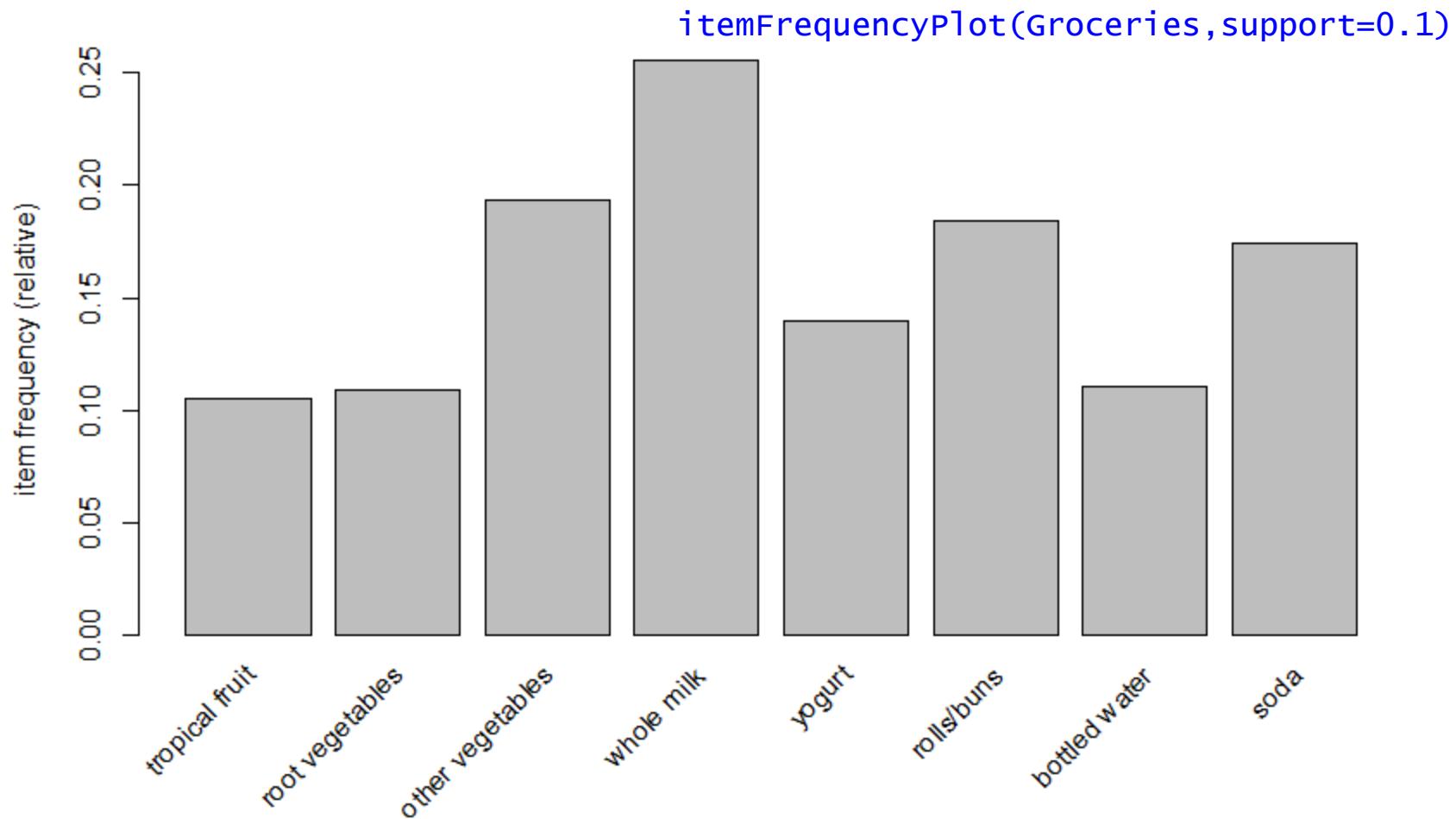
The DM Process – itemFrequencyPlot

- **Process 2: Exploratory Data Analysis**
 - Groceries data set → context
 - itemFrequencyPlot() function
 - Need to specify support parameter/argument

Example:

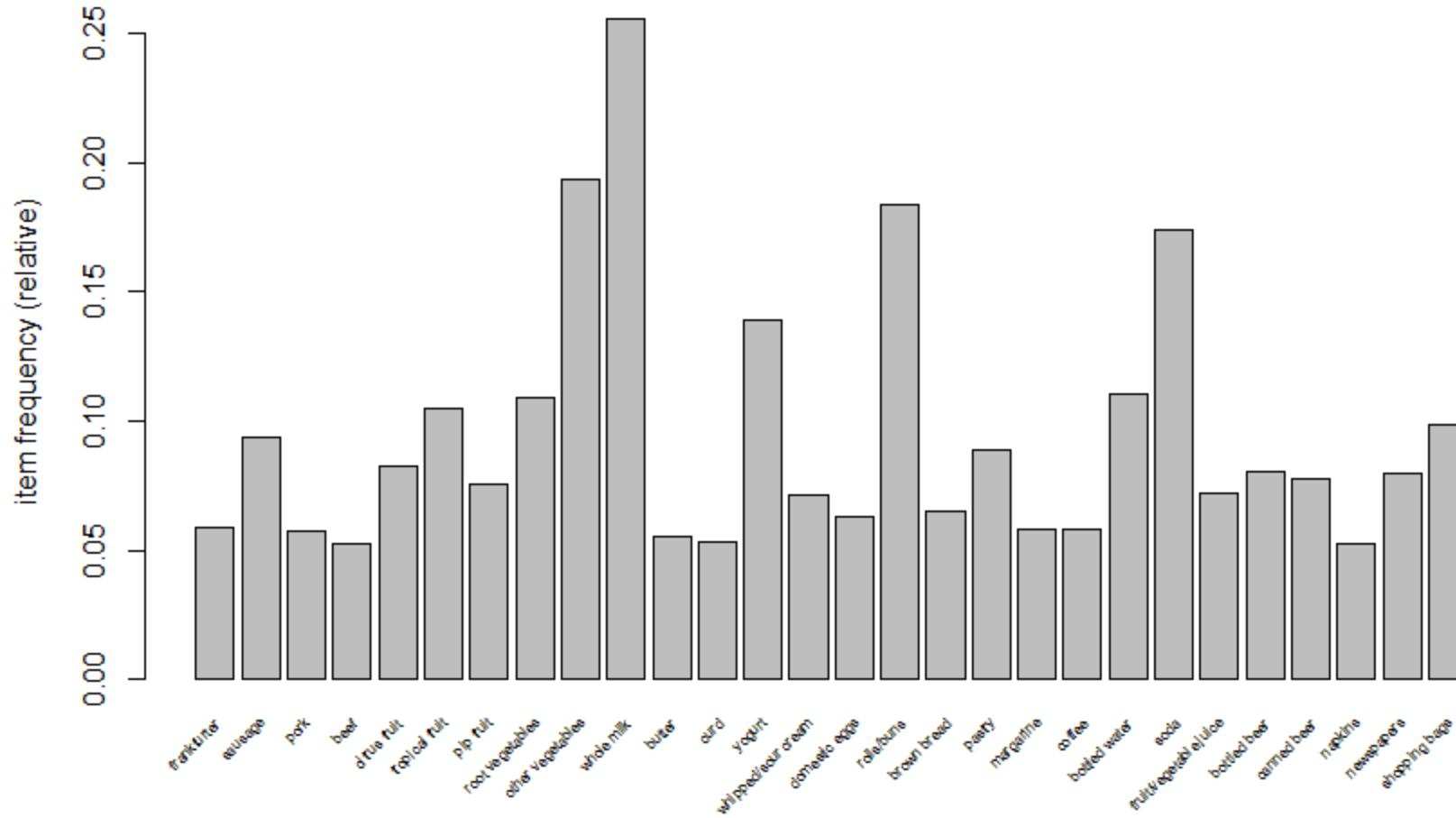
```
itemFrequencyPlot(Groceries, support=0.1)
```

The DM Process – itemFrequencyPlot



The DM Process – itemFrequencyPlot

itemFrequencyPlot(Groceries, support=0.05, cex.names=0.5)



The DM Process – Model Development

- Generate rules
 - Values of support
 - Focus on items with meaningful frequency
 - `apriori()` command
 - Finding rules in transaction data
 - Rules format: Equation context: LHS (lefthand side), RHS (righthand side)
 - RHS/LHS
 - RHS: one item
 - LHS: multiple items

The DM Process – Support/Lift/Confidence

- Support for a rule
 - Frequency of co-occurrence: LHS & RHS together
 - Ex: milk & butter occur together in 10% of carts
- Confidence of a rule
 - Proportion of time that LHS & RHS occur together vs. the total # of appearances of LHS
 - If milk by itself, occurs in 25% of the carts, then milk/butter confidence is .40 (.10/.25)
- Lift
 - Confidence / the probability of the RHS occurring

Using Apriori

```
apriori(Groceries, parameter=list(support=0.005, confidence=0.5))
```

parameter specification:

confidence	minval	smax	arem	aval	originalSupport	support	minlen	maxlen	target	ext
0.5	0.1	1	none	FALSE	TRUE	0.005	1	10	rules	FALSE

algorithmic control:

filter	tree	heap	memopt	load	sort	verbose
0.1	TRUE	TRUE	FALSE	TRUE	2	TRUE

apriori - find association rules with the apriori algorithm
version 4.21 (2004.05.09) (c) 1996-2004 Christian Borgelt
set item appearances ... [0 item(s)] done [0.00s].
set transactions ... [169 item(s), 9835 transaction(s)] done [0.00s].
sorting and recoding items ... [120 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 3 4 done [0.00s].
writing ... [120 rule(s)] done [0.00s].
creating S4 object ... done [0.00s].
set of 120 rules

Iterating Using Apriori

- Change support value.
- Rerun apriori.
- Store resulting rules.
- Review output via summary function.

Looking at the Rules

```
ruleset <- apriori(Groceries,  
parameter=list(support=0.01,confidence=0.5))  
summary(ruleset)
```

set of 15 rules

rule length distribution (lhs + rhs):sizes

3
15

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
3	3	3	3	3	3	3

summary of quality measures:

	support	confidence	lift
Min.	:0.01007	:0.5000	Min. :1.984
1st Qu.	:0.01174	:0.5151	1st Qu.:2.036
Median	:0.01230	:0.5245	Median :2.203
Mean	:0.01316	:0.5411	Mean :2.299
3rd Qu.	:0.01403	:0.5718	3rd Qu.:2.432
Max.	:0.02227	:0.5862	Max. :3.030

mining info:

	data	ntransactions	support	confidence
Groceries		9835	0.01	0.5

Looking at the Rules

	lhs	rhs	support	confidence	lift
1	{curd, yogurt}	=> {whole milk}	0.01006609	0.5823529	2.279125
2	{other vegetables, butter}	=> {whole milk}	0.01148958	0.5736041	2.244885
3	{other vegetables, domestic eggs}	=> {whole milk}	0.01230300	0.5525114	2.162336
4	{yogurt, whipped/sour cream}	=> {whole milk}	0.01087951	0.5245098	2.052747
5	{other vegetables, whipped/sour cream}	=> {whole milk}	0.01464159	0.5070423	1.984385
6	{pip fruit, other vegetables}	=> {whole milk}	0.01352313	0.5175097	2.025351
7	{citrus fruit, root vegetables}	=> {other vegetables}	0.01037112	0.5862069	3.029608
8	{tropical fruit, root vegetables}	=> {other vegetables}	0.01230300	0.5845411	3.020999
9	{tropical fruit, root vegetables}	=> {whole milk}	0.01199797	0.5700483	2.230969

Question:

Why do you think there are so many rules focused on milk?

Associative Rule Mining in R

Data Science

```
#  
>install.packages("arulesViz")  
>library(arulesViz)  
#  
>ruleset <-  
apriori(Groceries,parameter=list(support=0.005,confidence=0.35))  
>plot(ruleset)  
#
```

Data Science

```
> ruleset <- apriori(Groceries, parameter=list(support=0.005, confidence=0.35))

parameter specification:
  confidence minval smax arem aval originalSupport support minlen maxlen
    0.35      0.1     1 none FALSE           TRUE   0.005       1      10
  target  ext
  rules FALSE

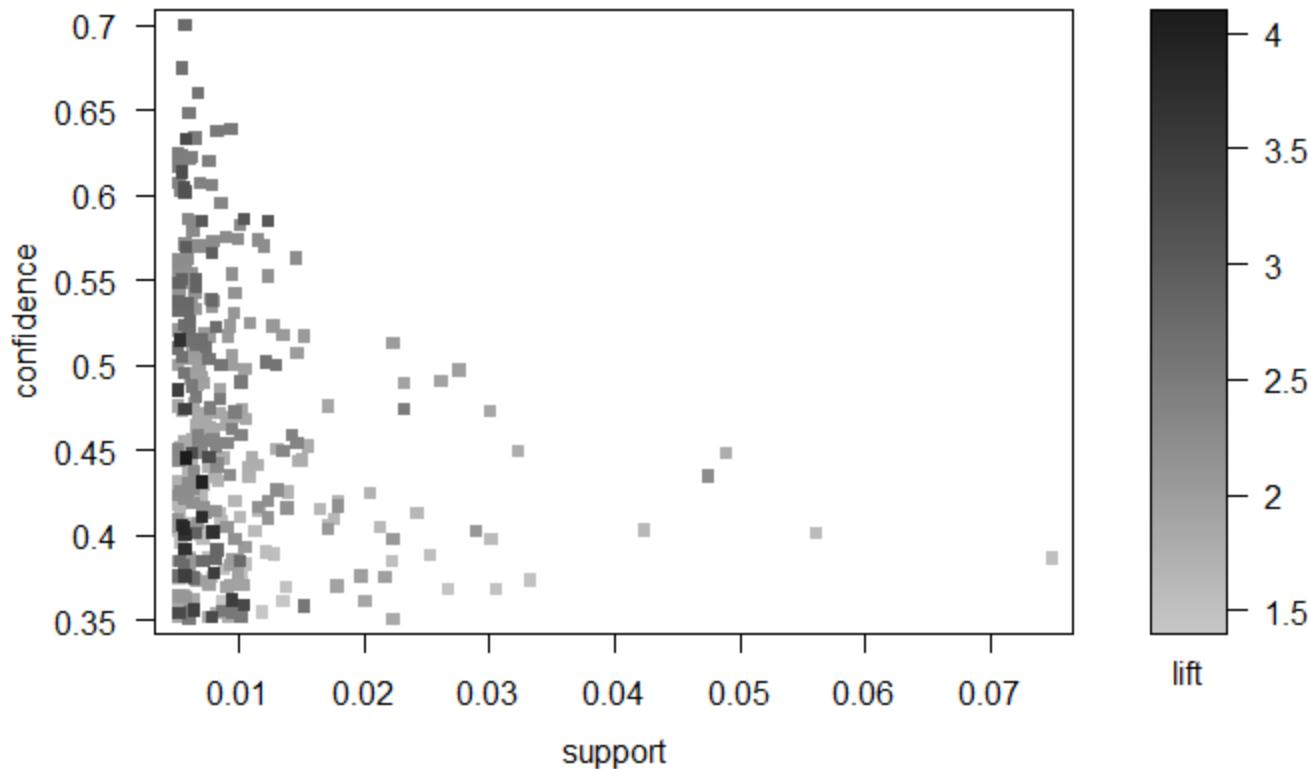
algorithmic control:
  filter tree heap memopt load sort verbose
    0.1 TRUE TRUE FALSE TRUE     2      TRUE

apriori - find association rules with the apriori algorithm
version 4.21 (2004.05.09)          (c) 1996-2004 Christian Borgelt
set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
sorting and recoding items ... [120 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 3 4 done [0.00s].
writing ., [357 rule(s)] done [0.00s].
creating S4 object ... done [0.00s].
```

Data Science

`plot(ruleset)`

Scatter plot for 357 rules



Data Science

Data Mining Process 4: Interpreting Results

```
> goodrules <- ruleset[quality(ruleset)$lift > 3.5]
> inspect(goodrules)
   lhs                      rhs          support confidence      lift
1 {herbs}                => {root vegetables} 0.007015760  0.4312500 3.956477
2 {onions,
  other vegetables} => {root vegetables} 0.005693950  0.4000000 3.669776
3 {beef,
  other vegetables} => {root vegetables} 0.007930859  0.4020619 3.688692
4 {tropical fruit,
  curd}                 => {yogurt}        0.005287239  0.5148515 3.690645
5 {citrus fruit,
  pip fruit}            => {tropical fruit} 0.005592272  0.4044118 3.854060
6 {pip fruit,
  other vegetables,
  whole milk}          => {root vegetables} 0.005490595  0.4060150 3.724961
7 {citrus fruit,
  other vegetables,
  whole milk}          => {root vegetables} 0.005795628  0.4453125 4.085493
8 {root vegetables,
  whole milk,
  yogurt}              => {tropical fruit} 0.005693950  0.3916084 3.732043
9 {tropical fruit,
  other vegetables,
  whole milk}          => {root vegetables} 0.007015760  0.4107143 3.768074
.
```

Data Science

- **Data Mining Process 4: Interpreting Results**
 - Shoppers purchasing in particular combinations that might be recipe oriented
 - Soup
 - Fruit platter with dip
 - Actionable recommendations to management might include:
 - Publish recipes along with coupons.
 - Publish visuals of the finished product along with the appropriate marketing verbiage.
 - Place recipes and visuals coincident with item/product locations.

SVM Overview

Intro to SVM



In this lecture, we will examine a set of supervised learning approaches known as support vector machines (SVMs). SVMs are flexible algorithms that excel at addressing classification problems.

Supervised Learning

- “Supervised learning” Data Mining
 - Train algorithm on an initial set of data.
 - Test algorithm on a new set of data.
 - Validate “trained” algorithm predicted the right outcome.

Supervised Learning Example

- Train an algorithm to predict the weather
- Collect weather data over a period of time
 - Sunny, cloudy
 - Temperature
 - Barometer
 - Wind speed and direction
- Train an algorithm with these variables
- Collect more weather data and then:
predict the weather via our trained algorithm,
and validate the prediction

Supervised Learning Strategy

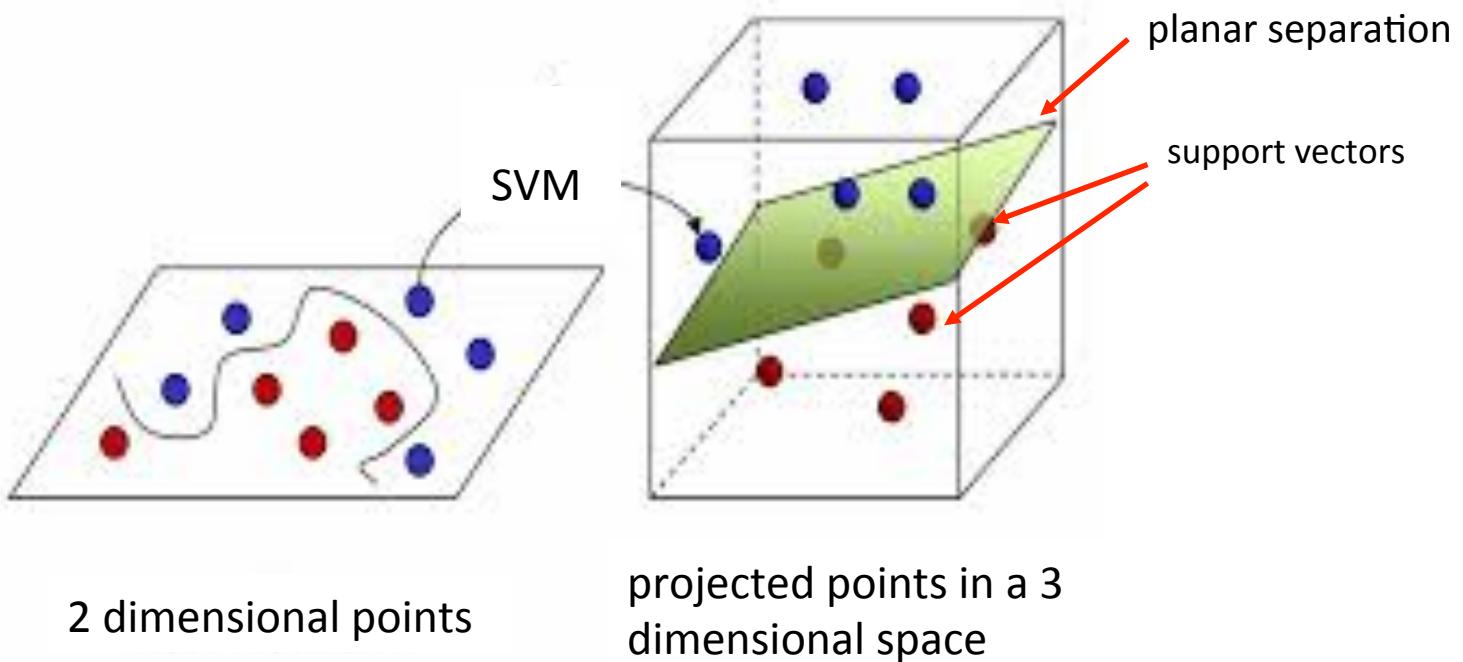
- **Substantial number of training cases** that the algorithm can use to discover and mimic the underlying pattern
- Use the results of this process on a test data set to **determine how well algorithm performed**, i.e., “cross validate”
- **Cross validate**: the process of verifying that the trained algorithm can carry out its prediction or classification task accurately on novel data

Chapter Use Case

- Kernel: mapping algorithm
 - Input data
(independent variables from a given case)
 - Kernel:
Formula run on each case's input data
 - Output data:
Position of that case in a multidimensional space

SVM Illustration

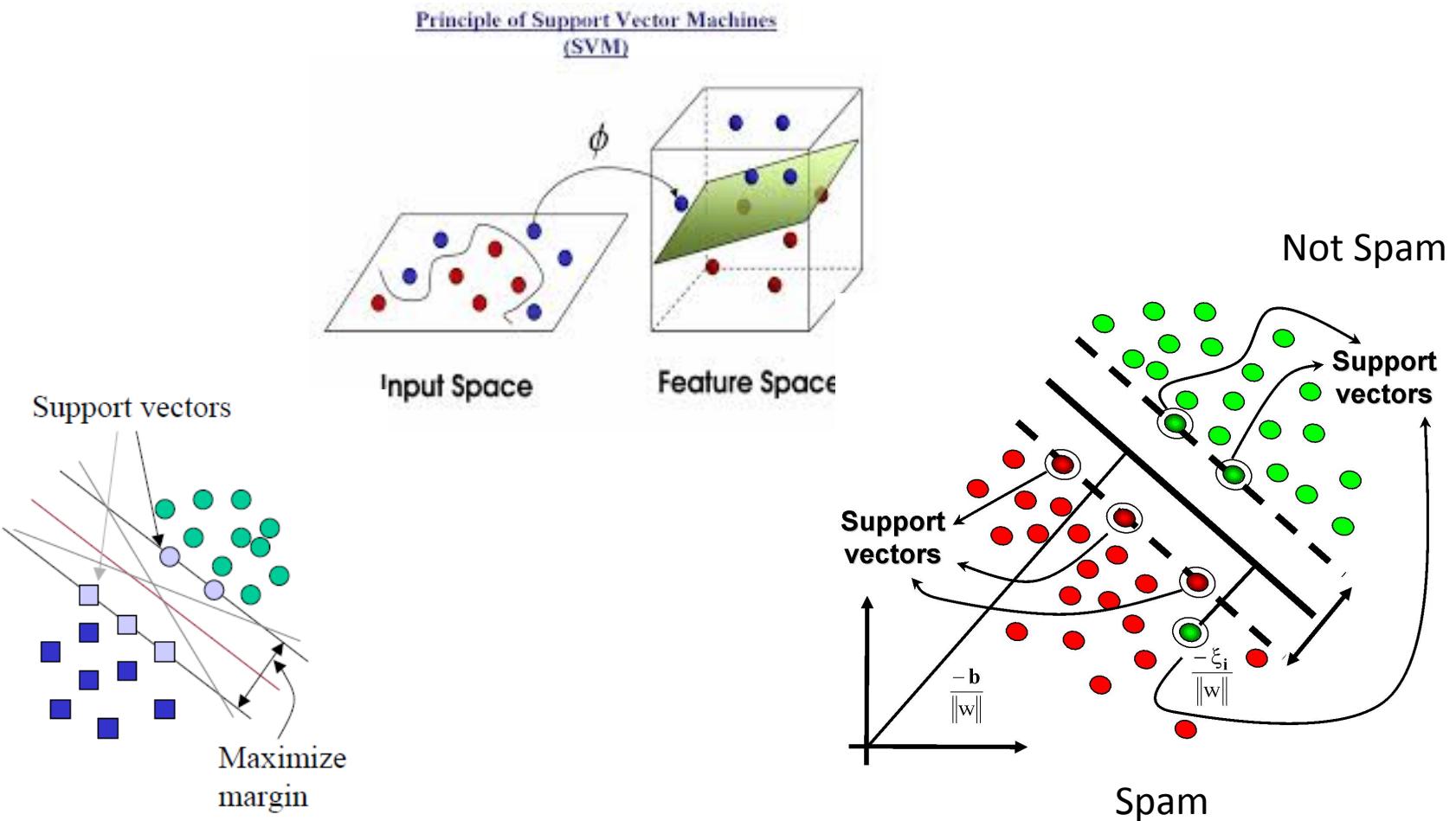
Principle of Support Vector Machines (SVM)



Another Metaphor

- Snow-capped mountain
- 2D/3D perspective
- Linear (planar) separation: snow cap/tree line
- Support vector machine analysis
 - Mathematical description of the position and orientation of the planar separation
- Novel data point to be mapped via a SVM into higher-dimensional space and indicate whether the point was above or below the planar separation

SVM Illustration (continued)



Question

What are some examples where something like SVM could be used?

SVM Example

SVM Setup

- R kernlab package
- Spam data set
<http://archive.ics.uci.edu/ml/datasets/Spambase>

An Example – Spam Email

- Develop a support vector machine (SVM) to classify e-mails as **spam or not spam**
- SVM:
Map a low-dimensional problem into a higher-dimensional space
- Goal:
Being able to describe geometric boundaries between different regions

Looking at Spam in R

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

# load R supplied data set
data(spam)

# structure, Inspect contents
str(spam)

# dimension, overview
dim(spam)

# delineates spam, nonspam counts
table(spam$type)
```

Looking at the Structure of Spam Data

```
> str(spam)
'data.frame': 4601 obs. of 58 variables:
 $ make           : num  0 0.21 0.06 0 0 0 0 0 0.15 0.06 ...
 $ address        : num  0.64 0.28 0 0 0 0 0 0 0.12 ...
 $ all            : num  0.64 0.5 0.71 0 0 0 0 0 0.46 0.77 ...
 $ num3d          : num  0 0 0 0 0 0 0 0 0 ...
 $ our             : num  0.32 0.14 1.23 0.63 0.63 1.85 1.92 1.88 0.61 0.19 ...
 $ over            : num  0 0.28 0.19 0 0 0 0 0 0 0.32 ...
 $ remove          : num  0 0.21 0.19 0.31 0.31 0 0 0 0.3 0.38 ...
 $ internet        : num  0 0.07 0.12 0.63 0.63 1.85 0 1.88 0 0 ...
 
 $ capitalAve     : num  3.76 5.11 9.82 3.54 3.54 ...
 $ capitalLong    : num  61 101 485 40 40 15 4 11 445 43 ...
 $ capitalTotal   : num  278 1028 2259 191 191 ...
 $ type            : Factor w/ 2 levels "nonspam","spam": 2 2 2 2 2 2 2 2 2 2
 |

> dim(spam)
[1] 4601 58
> table(spam$type)

nonspam    spam
 2788    1813
> |
```

SVM Setup

Divide into a “training” and “test” data set:

- Suggested delineation
 - 2/3 → Training
 - 1/3 → Test
- Create a data set of 4601 random indexes utilizing sample function
- Create 2/3 cut point
- Create **trainData** (using 2/3 cut point) and **testData** (using the rest of the data) and the previously created random indexes

R Code for SVM Setup

```
> randIndex <- sample(1:dim(spam)[1])      # Create list/vector variable-random index
> summary(randIndex)                      # verify indicies in randIndex
   Min. 1st Qu. Median     Mean 3rd Qu. Max.
   1       1151    2301    2301    3451    4601
> length(randIndex)                      # verify indicies in randIndex
[1] 4601
> head(randIndex)                        # look at first few cases
[1] 2272 2974 4153 3142 2280 2491
> cutPoint2_3 <- floor(2 * dim(spam)[1]/3) # create 2/3 cutpoint
> cutPoint2_3                            # verify 2/3 cutpoint
[1] 3067
> trainData <-spam[randIndex[1:cutPoint2_3],]  #create training data set
> testData <-spam[randIndex[(cutPoint2_3+1):dim(spam)[1]],] # create test data set
```

trainData & TestData

`trainData`

3067 obs. of 58 variables

testData

1534 obs. of 58 variables

Creating the SVM Model

- Train support vector model
 - `svmOutput <- ksvm(type ~ .,
data=trainData,kernel="rbfdot",kpar="automatic",
C=5,cross=3,prob.model=TRUE)`
- type ~ specifies the model we want to test, i.e., want to have the **type** variable (spam, nonspam) as the output variable to predict.
- “.” uses all other variables in the data frame to predict **type**.
- Data specifies the data frame to use in the analysis, i.e., trainData.

Creating the SVM Model (continued)

- Train support vector model
 - `svmOutput <- ksvm(type ~ .,
data=trainData,kernel="rbfdot",kpar=
"automatic",C=5,cross=3,prob.model=TRUE)`
- `rbfdot`—kernel function that projects the low-dimensional problem into higher-dimensional space, i.e., getting the maximum separation of distance between spam and nonspam cases
- **kpar** refers to a variety of parameters that can be used to control the radial bias function kernel (`rbfdot`)

Creating the SVM Model (continued)

- Train support vector model
 - `svmOutput <- ksvm(type ~ .,
data=trainData,kernel="rbfdot",kpar="auto
matic",C=5,cross=3, prob.model=TRUE)`
- C argument refers to “cost of constraints”
 - High C value impact (See Next Slide)
 - Low C value impact (See Next Slide)
- cross refers to the cross-validation model that the algorithm uses -- “overfitting”

Cost Parameter ('C')

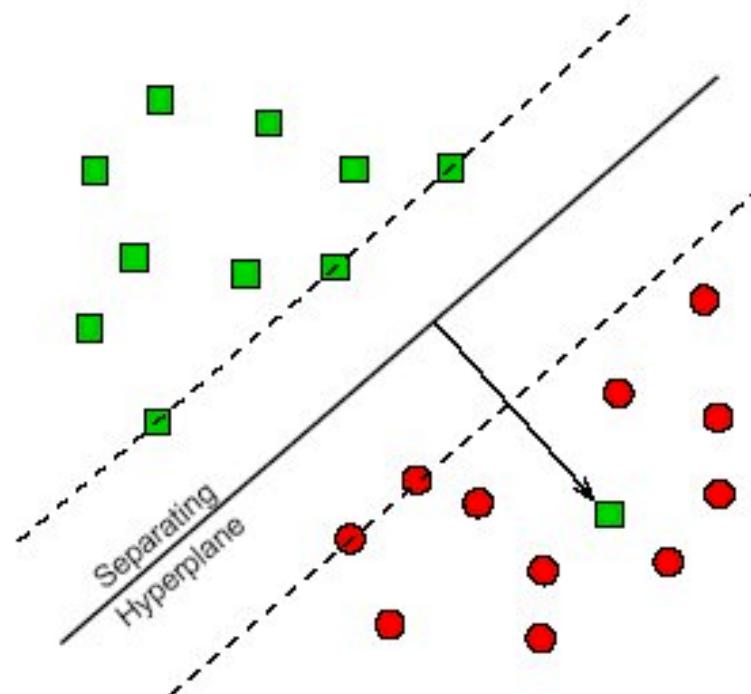
ksvm Cost Parameter Impact Summary

Higher Cost 'C' Value	Fewer Classification Mistakes Fewer Problem Points	Smaller Margin of Separation	Specialized Model	Higher Cross-Validation Error	Lower Training Error
Lower Cost 'C' Value	More Classification Mistakes More Problem Points	Higher Margin of Separation	Generalized Model	Lower Cross-Validation Error	Higher Training Error

Why Error Might Not Be Bad

Non-separable training sets

Use linear separation, but admit training errors.



Penalty of error: distance to hyperplane multiplied by *error cost C*.

Looking at the SVM Output

```
> svmOutput
Support Vector Machine object of class "ksvm"

SV type: C-svc (classification)
parameter : cost C = 5

Gaussian Radial Basis kernel function.
Hyperparameter : sigma = 0.0292958021260521

Number of Support Vectors : 948

Objective Function Value : -1739.444
Training error : 0.031953
Cross validation error : 0.074994
Probability model included.
> |
```

Changing the “Cost” Parameter

```
> svmOutput <- ksvm(type ~ .,  
  data=trainData,kernel="rbfdot",kpar="automatic" C=50,cross=3,prob.model=TRUE)  
Using automatic sigma estimation (sigest) for RBF or laplace kernel  
> svmOutput  
Support Vector Machine object of class "ksvm"  
  
SV type: C-svc (classification)  
parameter : cost C = 50  
  
Gaussian Radial Basis kernel function.  
Hyperparameter : sigma = 0.0290816646010172  
  
Number of Support Vectors : 847  
  
Objective Function value : -7525.442  
Training error : 0.011412  
Cross validation error : 0.091296  
Probability model included.
```

C=5

```
Objective Function Value : -1739.444  
Training error : 0.031953  
Cross validation error : 0.074994  
Probability model included.
```

> |

Predicting “ testData ” Results

- Predict “ testData ” where svmOutput had C=5

```
> svmPred <- predict(svmOutput, testData, type="votes")
> compTable <- data.frame(testData[,58],svmPred[1,])
> table(compTable)
```

	svmPred.1...	Interpretation
testData...58.	0 1	0 – spam 1 – not spam
nonspam	33 884	33 cases not spam but classified as spam
spam	551 66	884 cases not spam and classified as not spam
		551 cases spam and classified as spam
		66 cases classified as spam but not spam

0 – spam 1 – not spam

33 cases not spam but classified as spam

884 cases not spam and classified as not spam

551 cases spam and classified as spam

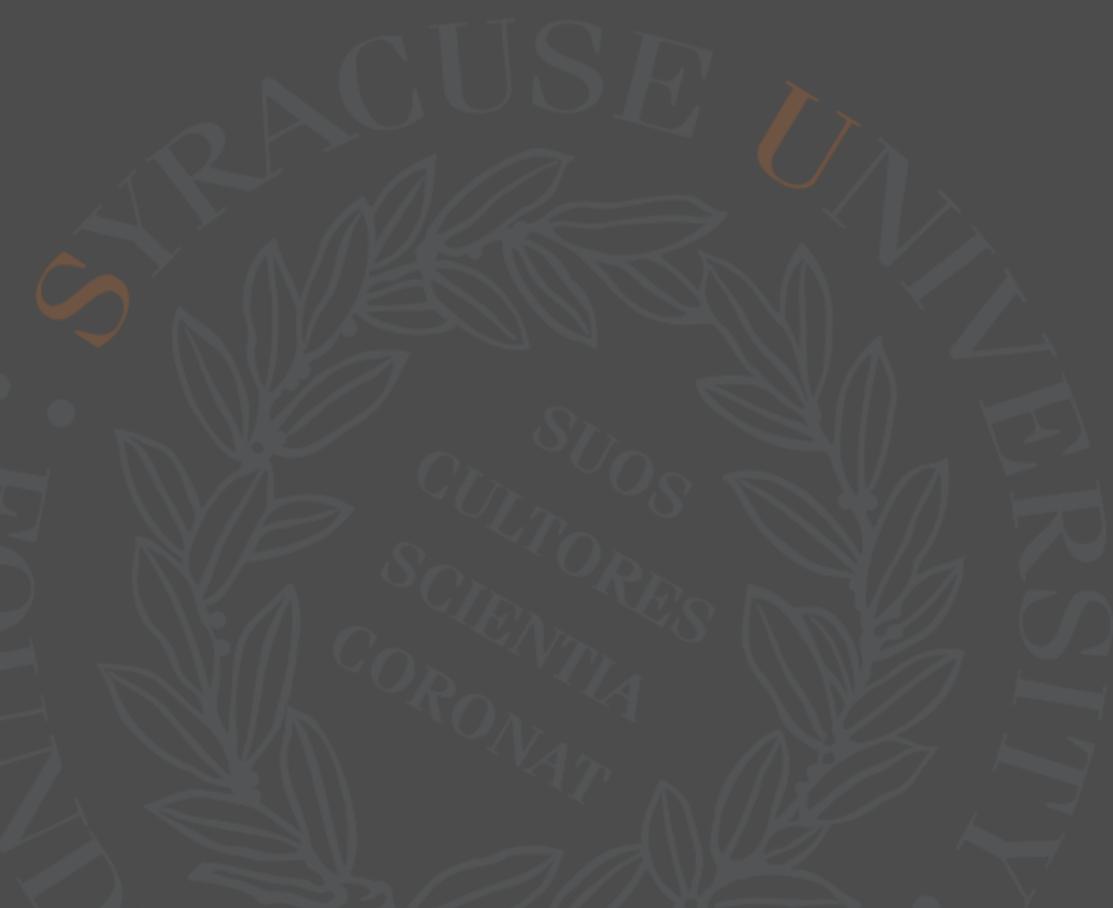
66 cases classified as spam but not spam

$33 + 66 = 99$ error cases

$99/1534 = .05867$ total error rate %

Cross Validation Error : .0749

Accuracy rate: 94.2%



School of Information Studies
SYRACUSE UNIVERSITY

Word Clouds

Building a Word Cloud

Use text mining to build a word cloud

- Extract text from a speech (or any other text)
- Build a term-document matrix (from text)
- Find frequent words and associations from the matrix.
- A word cloud is used to present frequently occurring words in documents.

Text Mining Packages

Text mining packages in R include:

- tm: provides functions for text mining
- wordcloud: visualizes results

Reading Text

One way to read in text in R

```
> sbaFile <-  
"http://www.historyplace.com/speeches/  
anthony.htm"
```

```
> sba <- readLines(sbaFile)
```

```
> str(sba)
```

```
chr [1:15] "Friends and fellow citizens: I stand before  
you tonight under indictment for the alleged crime of  
having voted at the last pres" | __truncated__ ...
```

Corpus = “Bag of Words”

We first need to build a corpus

- A corpus is a “bag of words.”
- We coerce our text file vector ('sba') into a custom "Class" provided by the tm package called a "Corpus".
- The Corpus Class defines the most fundamental object that text miners care about, a corpus containing a collection of documents.

Text Transformations

Four transformations

- Making all of the letters lowercase
- Removing the punctuation
- Removing numbers
- Taking out the "stop" words
 - Words such as *the*, *a*, and *at* appear in so many different parts of the text that they are useless for differentiating between documents.

Text Transformations in R

Four transformations:

```
> words.vec <- VectorSource(sba)  
> words.corpus <- Corpus(words.vec)  
> words.corpus  
<<VCorpus>>
```

Metadata: corpus specific: 0, document level (indexed): 0

Content: documents: 15

```
> words.corpus <- tm_map(words.corpus,  
content_transformer(tolower))  
> words.corpus <- tm_map(words.corpus, removePunctuation)  
> words.corpus <- tm_map(words.corpus, removeNumbers)  
> words.corpus <- tm_map(words.corpus, removeWords,  
stopwords("english"))
```

A Term-Document Matrix

- A rectangular data structure with terms (words) as the rows and documents as the columns
- A term may be a single word, for example, *biology*, or it could also be a compound word, such as *data analysis*.

A Term-Document Matrix (continued)

- If a term like *data* appears once in the first document, twice in the second, and not at all in the third document, then the column for the term *data* will contain 1, 2, 0.
- Most term document matrices are quite sparse—the overwhelming number of cells that contain zero—indicating that the term does not appear in a document.

Creating a TermDocumentMatrix

```
> tdm <- TermDocumentMatrix(words.corpus)  
> tdm  
<<TermDocumentMatrix (terms: 189,  
documents: 15)>>
```

Non-/sparse entries: 225/2610

Sparsity : 92%

Maximal term length: 20

Weighting : term frequency (tf)

The wordcloud Function

- Expects two vectors as input arguments:
 - The first a list of the terms
 - The second a list of the frequencies of occurrence of the terms
- The list of terms and frequencies must be sorted with the most frequent terms first.
 - We first have to coerce our text data back into a plain data matrix so that we can sort it by frequency.

Creating a Word Cloud In R

```
> m <- as.matrix(tdm)
> wordCounts <- rowSums(m)
> wordCounts <- sort(wordCounts, decreasing=TRUE)
> head(wordCounts)
    women  citizens oligarchy   people   states blessings
    7       6          5        5       5       4
> cloudFrame<-data.frame( +
  word=names(sortedMatrix),freq=sortedMatrix)
> wordcloud(cloudFrame$word,cloudFrame$freq)
```

Word Cloud Example

states
oligarchy
constitution
people
women
half united
liberty state
posteriority every
government law
sex **citizens**
blessings

Another Word Cloud Example

```
wordcloud(names(wordCounts), wordCounts, min.freq=2, +  
max.words=50, rot.per=0.35, colors=brewer.pal(8, "Dark2"))
```



Question:

How useful are word clouds?

When are they appropriate to use?

Sentiment Analysis

Conceptual Methodology

- Load Positive and Negative word Lists
- **Count** positive words and negative words
(in entire document or part of a document)
- **Compute the ratio** of positive to negative words

Example R Code

Get the Positive and Negative Word Files

Access to positive and negative words:

<https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html>

```
> pos <- "positive-words.txt"
> neg <- "negative-words.txt"

> #read the files
> p <- scan(pos, character(0),sep = "\n") # separate each word
Read 2040 items

> n <- scan(neg, character(0),sep = "\n") # separate each word
Read 4817 items
```

Clean Up the Word Files

```
#remove the first 34 lines (header info)
```

```
> p <- p[-1:-34]
```

```
> n <- n[-1:-34]
```

```
> head(p, 10)
```

```
[1] "a+"      "abound"   "abounds"   "abundance" "abundant"  "accessible"  
[7] "accessible" "acclaim"   "acclaimed"  "acclamation"
```

```
> head(n,10)
```

```
[1] "2-faced"  "2-faces"  "abnormal"  "abolish"   "abominable" "abominably"  
[7] "abominate" "abomination" "abort"     "aborted"
```

More R Code

Get the total number of words:

```
#calculate the total number of words  
> totalWords <- sum(wordCounts)
```

```
#have a vector that just has all the words  
words <- names(wordCounts)
```

```
> matched <- match(words, p, nomatch = 0)  
> head(matched,10)  
[1] 0 0 0 0 0 0 0 0 1083 0
```

More R Code

Explore the matched words:

```
> matched[9]  
[1] 1083
```

```
> p[1083]  
[1] "liberty"
```

```
> words[9]  
[1] "liberty"
```

Calculate the Positive Word Count

```
> mCounts <- wordCounts[which(matched != 0)]  
> length(mCounts)  
[1] 12  
  
> mWords <- names(mCounts)  
> nPos <- sum(mCounts)  
> nPos  
[1] 17
```

Calculate the Negative Word Count

```
> matched <- match(words, n, nomatch = 0)
> nCounts <- wordCounts[which(matched != 0)]
> nNeg <- sum(nCounts)
> nWords <- names(nCounts)
> nNeg
[1] 13
> length(nCounts)
[1] 11
```

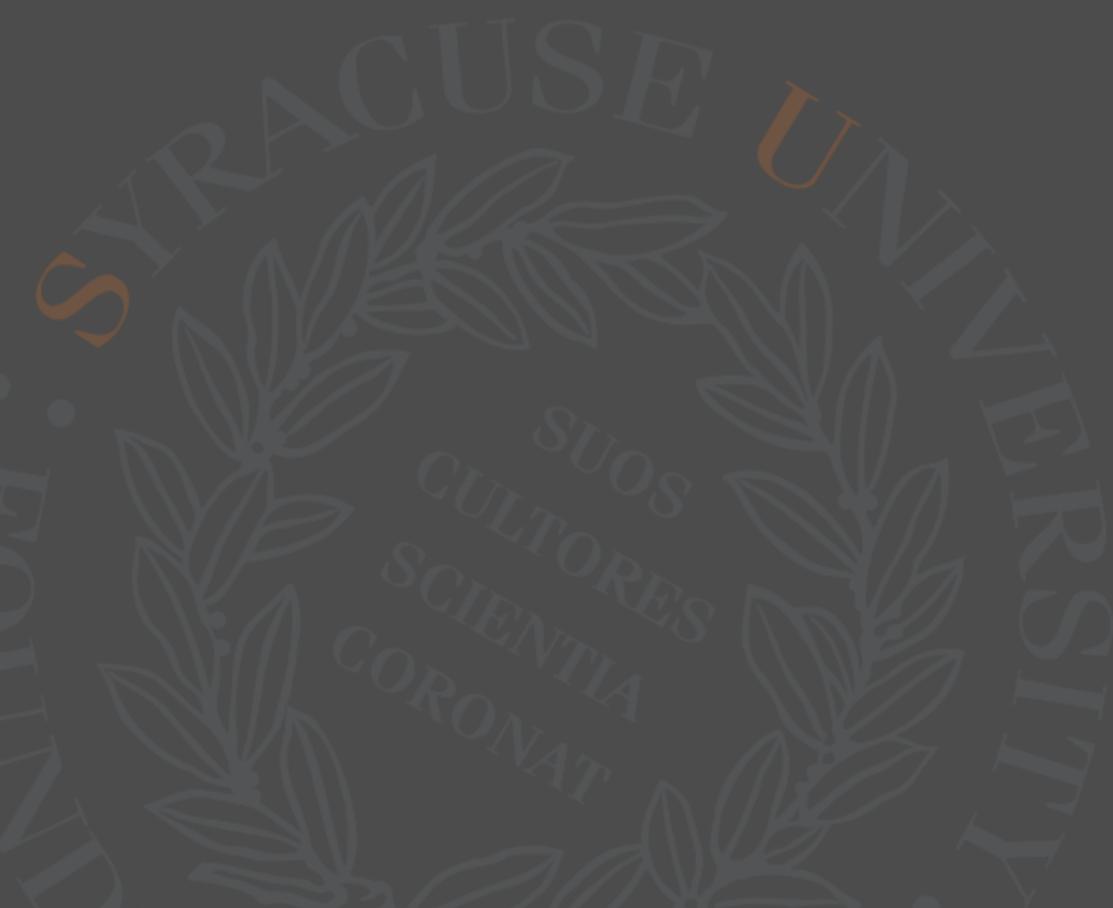
Calculate the Sentiment!

```
> #calculate the % of words that are positive or negative  
> totalWords <- length(words)  
> ratioPos <- nPos/totalWords  
> ratioPos  
[1] 0.08994709  
  
> ratioNeg <- nNeg/totalWords  
> ratioNeg  
[1] 0.06878307
```

Given this, we can see that Susan B. Anthony's speech was about **9% positive** and a little less than **7% negative**.

Question

- Does this truly measure sentiment?
- Where could it go wrong?



School of Information Studies
SYRACUSE UNIVERSITY