Machine Learning for Business

Module 2: A brief introduction to R

Day 1, 13.00 – 16.00

Asst. Prof. Dr. Santitham Prom-on

Department of Computer Engineering, Faculty of Engineering
King Mongkut's University of Technology Thonburi

BIG DATA EXPERIENCE CENTER

Module 2 Overview

- R tutorial
- tidyr and dplyr
- ggplot2

BIG DATA EXPERIENCE

R Basics

Section 1

g·able

BIG DATA EXPERIENCE

What is R?

- R is a scripting language for data manipulation and analysis
- R is originated from S language at Bell Laboratories (AT&T)
  - R and S were created with purposes of providing an interactive environment without requiring programming skills
- Over the years, through a lot of academic and commercial contributions, numerous packages were developed with various purposes and better qualities.

g·able

BIG DATA EXPERIENCE

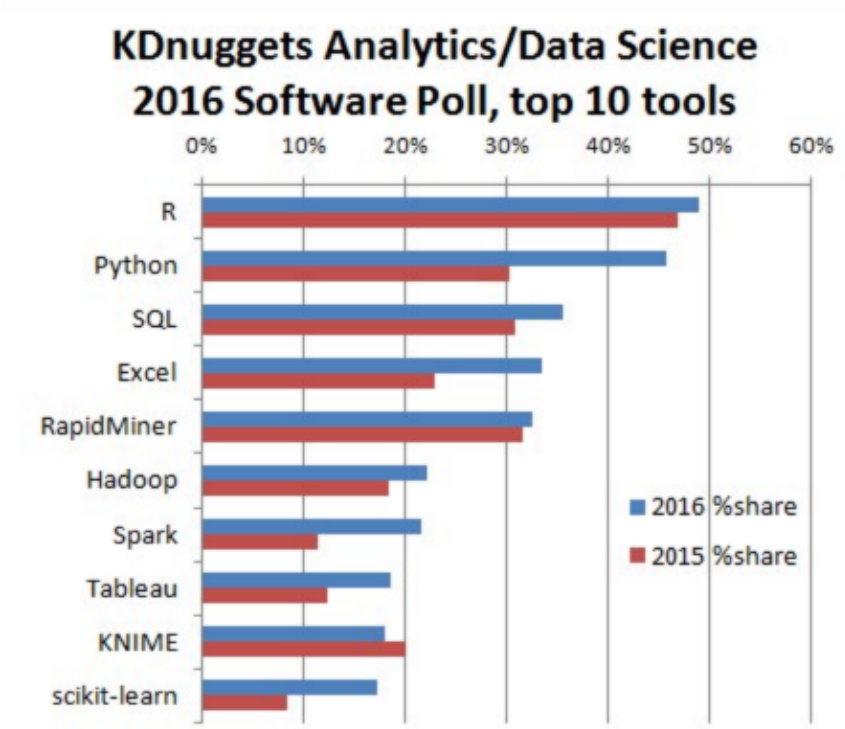
IEEE Ranking of Popular Programming Language 2016

Language Rank	Types	Spectrum Ranking
1. C		100.0
2. Java		98.1
3. Python		97.9
4. C++		95.8
5. R		87.7
6. C#		86.4
7. PHP		82.4
8. JavaScript		81.9
9. Ruby		73.8
10. Go		70.9

http://spectrum.ieee.org/static/interactive-the-top-programming-languages-2016

g·able

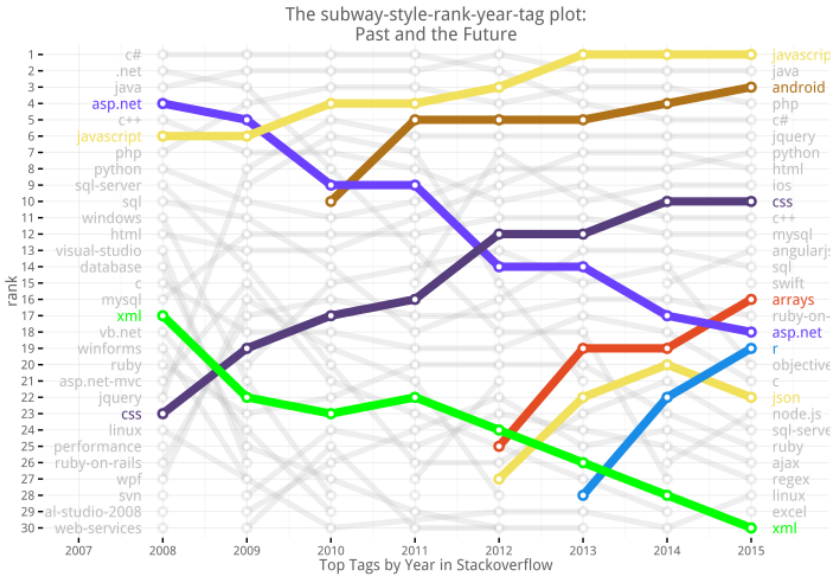
BIG DATA EXPERIENCE



g·able

BIG DATA EXPERIENCE

Stackoverflow Top Tags Ranking



BIG DATA EXPERIENCE

Why use R?

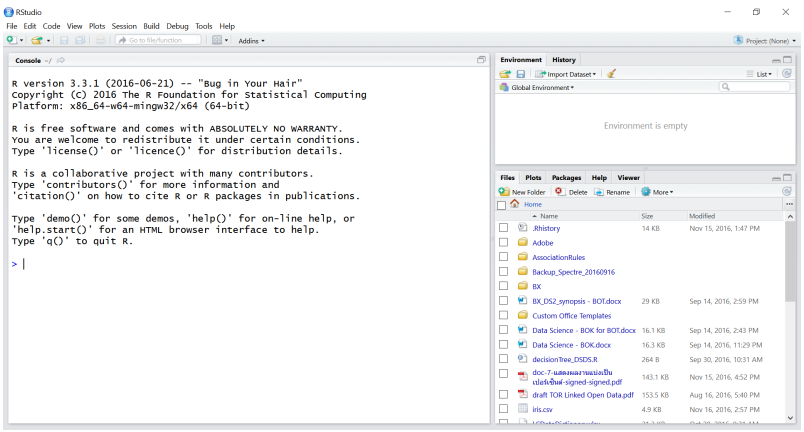
- Free
- Easy to get the job done
- Clear, more compact code
- Easier transition to distributed computing
- Great supporting community
- Object-oriented programming
- Functional programming

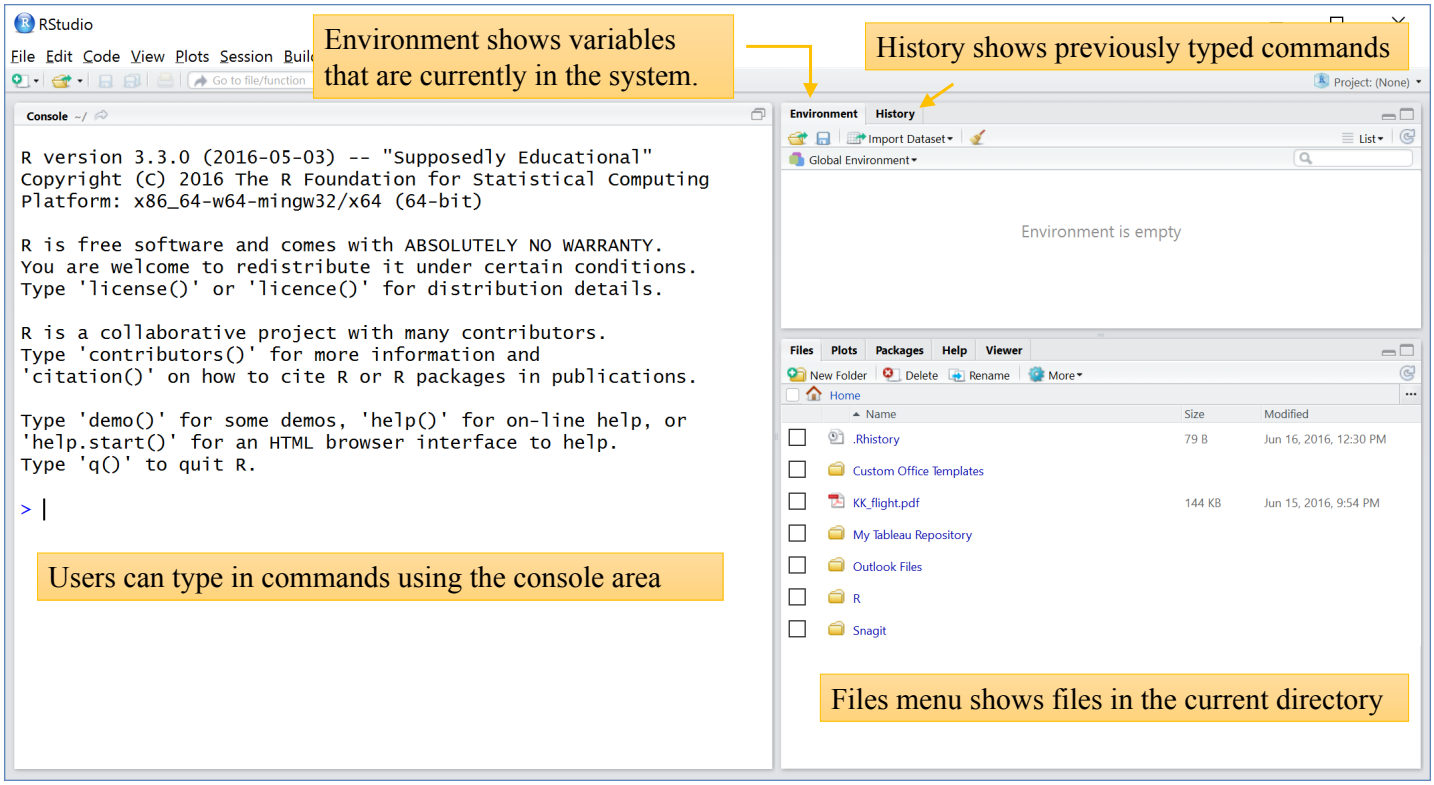
Learning R

- Learning R is not like learning C or Java languages
- R is a dynamic typing (no declaration required)
- R is functional (use function to run)
- R is data oriented (utilize data structure to do the work)
- R is learned through an interactive exploration of how to interact with the environment and data

Rstudio (via www.rstudio.com)

- RStudio is an IDE for R
- RStudio makes R easier to use
  - Console
  - Syntax highlight editor
  - Plotting
  - History
  - Environment
  - Help
  - Debugging





BIG DATA EXPERIENCE

Try this…

Type these commands and guess their functions

```
mean(runif(100))

1:10

2**10

abs(rnorm(10))
```

BIG DATA EXPERIENCE

R Session

```
x <- c(1,2,4)
```

- This function creates a vector of 1, 2 and 4
- There are no fixed types associated with variables
- `c()` is a function that concatenates three one-element vector
- The assignment operator in R are <- and =
- <- is a preferred assignment operator while = is used for parameter assignment
- The above code is the same as `x = c(1,2,4)`
- Try this `q <- c(x,x,8)`

g·able

BIG DATA EXPERIENCE

Variable printing and subsetting

```
> q
[1] 1 2 4 1 2 4 8
> q[1]
[1] 1
> q[1:4]
[1] 1 2 4 1
> q[-2]
[1] 1 4 1 2 4 8
> q[-1:-4]
[1] 2 4 8
> q[c(2,5)]
[1] 2 2
```

Print a variable `q`

Select an individual element of a vector, index 1

Subsetting range

Deselect

Deselect range

Selective subsetting

g·able

BIG DATA EXPERIENCE

Calling functions

```
y <- mean(x)
y1 <- sd(x)
```

- The above code calculate mean and standard deviation of `x` and store them to `y` and `y1` respectively

BIG DATA EXPERIENCE

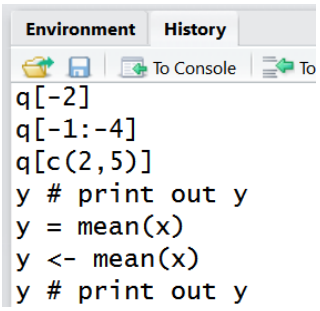
Comment

Try this

```
y # print out y
```

```
> y <- mean(x)
> y # print out y
[1] 2.333333
```

- Text after # is treated as comment
- They are not executed, but recorded in history



BIG DATA EXPERIENCE

Internal Dataset

- R is preloaded with the internal datasets
- To view the whole list type `data()`

R data sets

```
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
```

g·able

BIG DATA EXPERIENCE

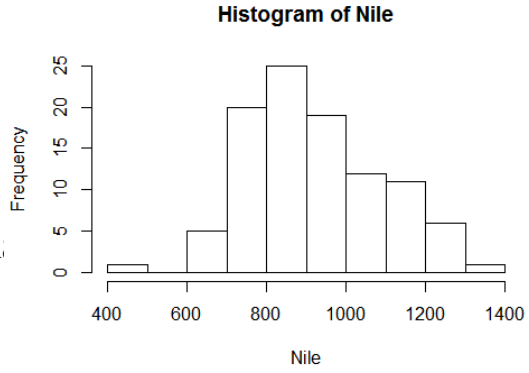
Internal Dataset

- You can load the internal dataset by typing the name
- Try the following

```
Nile
iris
mtcars
```

- To plot a histogram of Nile d

```
hist(Nile)
```



g·able

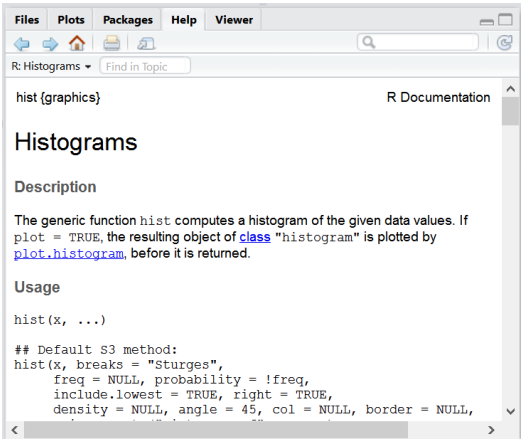
Help

- A skilled programmer always look for documentation
- To access the function, data or package documentations, try

```
help("hist")
```

or

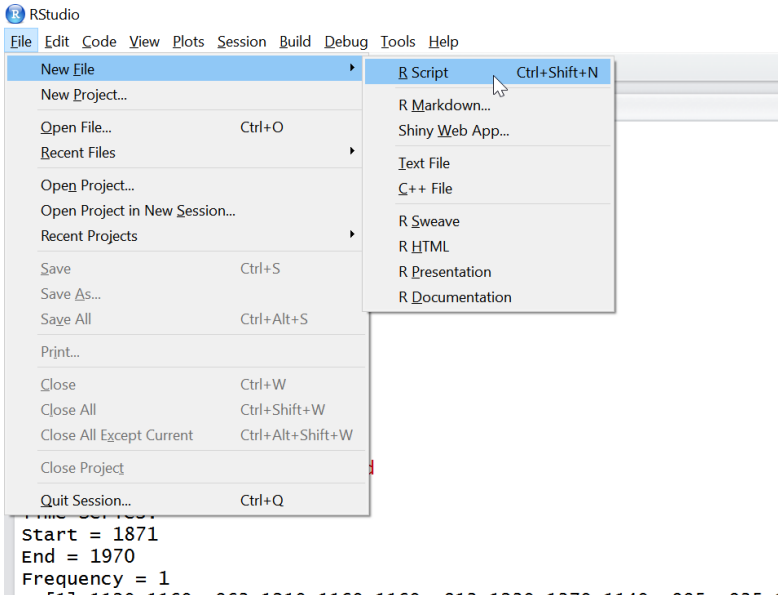
```
?hist
?Nile
```



Introduction to function

- As in most programming languages, the heart of R programming consists of writing *functions*.
- A function is a group of instructions that takes inputs, uses them to compute other values, and returns a result.
- As a simple introduction, let's define a function named oddcount(), whose purpose is to count the odd numbers in a vector of integers.
- We will write a function in RStudio editor, save it to file and load it to the workspace.

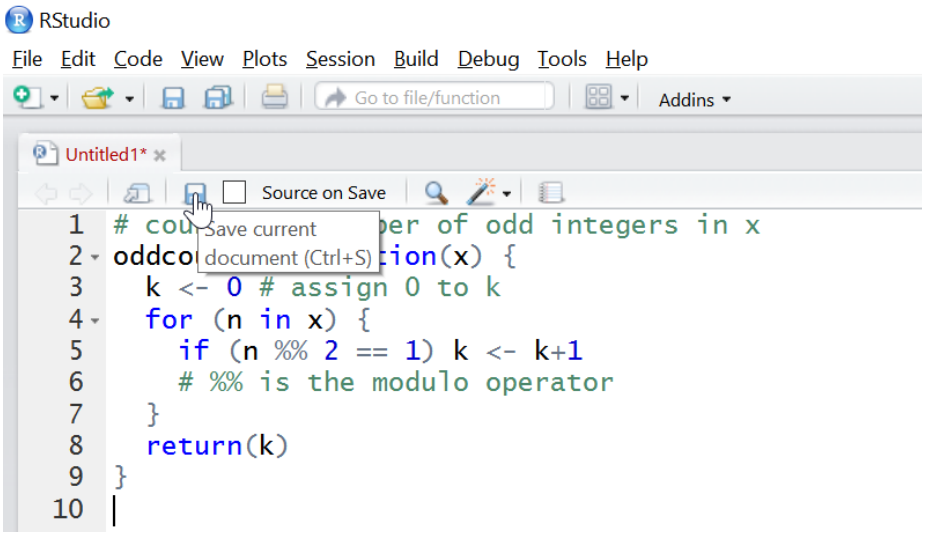
1. Create a new R script file



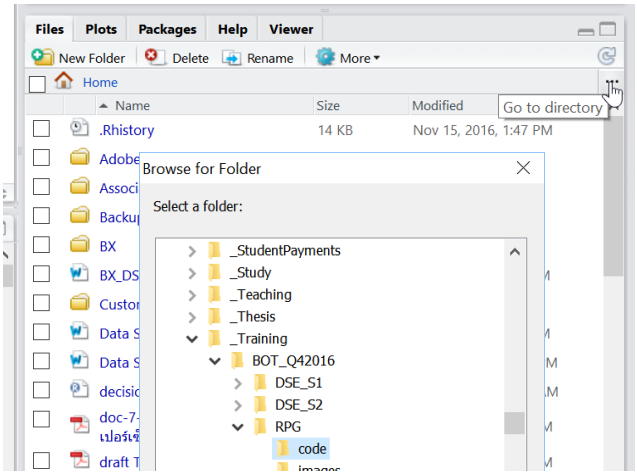
2. Write the oddcount function

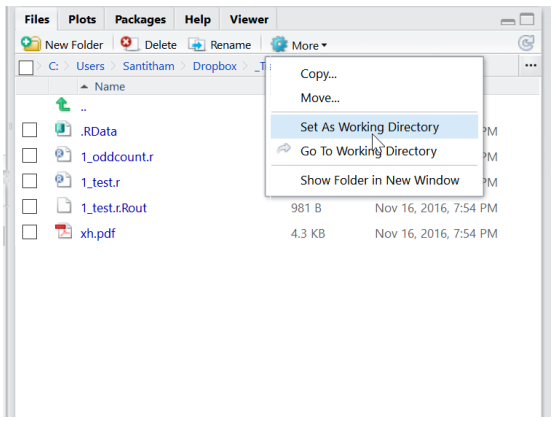
```
# counts the number of odd integers in x
oddcount <- function(x) {
    k <- 0 # assign 0 to k
    for (n in x) {
        if (n %% 2 == 1) k <- k+1
        # %% is the modulo operator
    }
    return(k)
}
```

3. Save file with name “1_oddcount.r”



4. Change working directory to where you save file






```
> setwd("C:/Users/Santitham/Dropbox/_Training/BOT_Q42016/RPG/code")
> |
```

BIG DATA EXPERIENCE

5. Load the function to use

To load the function

```
source("1_oddcount.r")
```

To use the function

```
oddcount(c(1,2,3,5,7,10))
```

```
> oddcount(c(1,2,3,5,7,10))
[1] 4
```

g·able

BIG DATA EXPERIENCE

R objects – basic classes

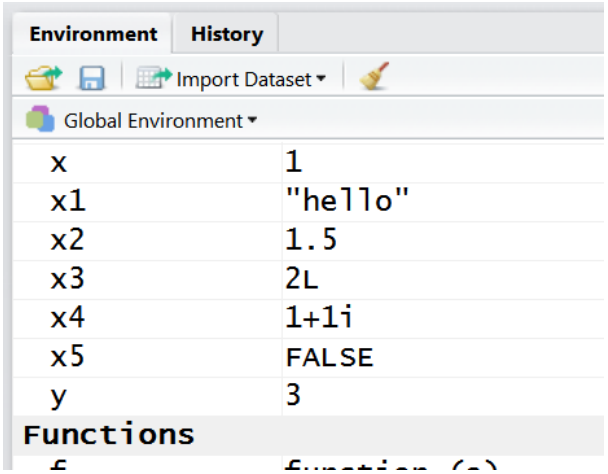
- R has five basic classes of objects
  - character
  - numeric (real numbers)
  - integer
  - complex
  - logical (True/False)
- Special values
  - Inf. Infinity occurs when the number is divided by zero
  - NaN. Not-a-number indicates undefined value, e.g. 0/0. It can also be thought as a missing value.

g·able

BIG DATA EXPERIENCE

Try this

```
x1 <- 'hello'
x2 <- 1.5
x3 <- as.integer(2)
x4 <- 1+1i
x5 <- FALSE
```



g·able

BIG DATA EXPERIENCE

Compound data structure

- Vector
- Matrices and Arrays
- Lists
- Data Frames
- Factors and Tables

g·able

BIG DATA EXPERIENCE

Vectors

- Vector is a basic foundation of other data structure

```
x <- 1
```

- This generates one-element vector

```
x <- c(2,4,6)
```

- This generates three-elements vector

BIG DATA EXPERIENCE

Adding or deleting vector elements

- Vectors are stored contiguously, thus you cannot insert or delete elements
- The size of a vector is determined at its creation, so if you wish to add or delete elements, you'll need to reassign the vector.

```
> x <- c(88,5,12,13)
> x <- c(x[1:3],168,x[4])
> x
[1]  88   5  12 168  13
```

BIG DATA EXPERIENCE

Obtaining length

- You can obtain the length of a vector by using the `length()` function:

```
> x <- c(88,5,12,13)
> x <- c(x[1:3],168,x[4])
> x
[1]  88   5  12 168  13
> length(x)
[1] 5
```

g·able

BIG DATA EXPERIENCE

Matrices and arrays as vectors

- Arrays and matrices (and even lists, in a sense) are actually vectors too
- They merely have extra class attributes.
- For example, matrices have the number of rows and columns.

```
> m = matrix(1:4,nrow=2)
> m
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> m + 10:13
     [,1] [,2]
[1,]   11   15
[2,]   13   17
```

g·able

BIG DATA EXPERIENCE

Common vector operation Arithmetic operator

- Adding/subtracting

```
x <- c(1,2,4)
x + c(5,0,1)
x - 1
x + c(2,1)
```

- (Element-wise) multiplication

```
x * c(5, 0, 4)
```

- (Element-wise) division

```
x / c(5, 4, -1)
```

- (Element-wise) modulus

```
x %% c(5,4,-1)
```

g·able

BIG DATA EXPERIENCE

Other ways to generate a vector

- With operator :

```
> 5:9
[1] 5 6 7 8 9
> -1:-5
[1] -1 -2 -3 -4 -5
```

- With function seq()

```
> seq(12,30,3)
[1] 12 15 18 21 24 27 30
```

- Repeating constants rep()

```
> rep(5,3)
[1] 5 5 5
> rep(c(1,2,3),3)
[1] 1 2 3 1 2 3 1 2 3
> rep(c(1,-1),each = 2)
[1]  1  1 -1 -1
```

g·able

BIG DATA EXPERIENCE

NA and NULL

- NA: missing value
- NULL: unknown value

```
> x <- c(88,NA,12,168,13)
> x
[1]  88  NA  12 168  13
> mean(x)
[1] NA
> mean(x,na.rm = T)
[1] 70.25
> x <- c(88,NULL,12,168,13)
> mean(x)
[1] 70.25
```

- NULL can be used as an initial empty variable

```
> z <- NULL
> for (i in 1:10) if (i%%2 == 0) z <- c(z,i)
> z
[1]  2  4  6  8 10
```

g·able

BIG DATA EXPERIENCE

Filtering

- Filtering allows us to extract a vector's elements that satisfy certain conditions.

```
> z <- c(5,2,-3,8)
> w <- z[z*z > 8]
> w
[1]  5 -3  8
```

- This is because the index vector become a logical vector

```
> z*z > 8
[1]  TRUE FALSE  TRUE  TRUE
```

- We can assign value to the filtered position

```
> x <- c(1,3,8,2,10)
> x[x > 3] <- 0
> x
[1] 1 3 0 2 0
```

g·able

BIG DATA EXPERIENCE

Selection function which()

- As you've seen, filtering consists of extracting elements of a vector z that satisfy a certain condition.
- In some cases, though, we may just want to find the positions within a vector at which the condition occurs.
- We can do this using `which()`, as follows:

```
> x <- c(1,3,8,2,10)
> which(x > 3)
[1] 3 5
```


BIG DATA EXPERIENCE

Testing vector equality

- To test vector equality, we need to combine == operator with all()

```
> x <- c(1,3,5)
> y <- x
> z <- c(1,3,2)
> all(x == y)
[1] TRUE
> all(x == z)
[1] FALSE
```

- Or using function `identical()`

```
> identical(x,y)
[1] TRUE
> identical(x,z)
[1] FALSE
```

BIG DATA EXPERIENCE

Creating matrix

```
> m <- matrix(1:4,nrow=2,ncol=2)
> m
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> y <- matrix(1:6,nrow=2)
> y
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> y <- matrix(1:6,nrow=2,byrow = T)
> y
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

g·able

BIG DATA EXPERIENCE

Matrix operation

Matrix Multiplication by Scalar

```
> m
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> m %*% m
     [,1] [,2]
[1,]    7   15
[2,]   10   22
```

Matrix Multiplication

```
> 3*m
     [,1] [,2]
[1,]    3    9
[2,]    6   12
> m + m
     [,1] [,2]
[1,]    2    6
[2,]    4    8
```

Matrix Addition

g·able

BIG DATA EXPERIENCE

Matrix indexing

```
> z <- matrix(1:16, nrow = 4)
> z
     [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    2    6   10   14
[3,]    3    7   11   15
[4,]    4    8   12   16
> z[,2:3]
     [,1] [,2]
[1,]    5    9
[2,]    6   10
[3,]    7   11
[4,]    8   12
```

Selecting only 2nd and 3rd columns

g·able

BIG DATA EXPERIENCE

Adding or deleting matrix rows and columns

- Just like vector, matrices are fixed length and dimension
- However, they can be reassigned
- cbind() bind the column together

```
> z
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
> cbind(z,rep(1,3))
     [,1] [,2] [,3]
[1,]    1    4    1
[2,]    2    5    1
[3,]    3    6    1
```

g·able

BIG DATA EXPERIENCE

Object coercion

If more than one class are introduced, the vector will be coerced.

Implicit

```
> y <- c(1.7, "a")
## character
> y <- c(TRUE, 2)
## numeric
> y <- c("a", TRUE)
## character
```

Explicit

```
> x <- 0:6
> class(x)
 [1] "integer"
> as.numeric(x)
 [1] 0 1 2 3 4 5 6
> as.logical(x)
 [1] FALSE TRUE TRUE TRUE TRUE TRUE TRUE
> as.character(x)
 [1] "0" "1" "2" "3" "4" "5" "6"
```

g·able

BIG DATA EXPERIENCE

List

- In contrast to a vector, in which all elements must be of the same mode, R's list structure can combine objects of different types.
- Ordinary vectors are termed *atomic* vectors, since their components cannot be broken down into smaller components.
- In contrast, lists are referred to as *recursive* vectors.

g·able

BIG DATA EXPERIENCE

Creating a list

- Let us consider an employee database.
- For each employee, we wish to store the name, salary, and a Boolean indicating union membership.

```
> j <- list(name="Joe", salary=55000, union=T)
> j
$name
[1] "Joe"

$salary
[1] 55000

$union
[1] TRUE
```

g·able

BIG DATA EXPERIENCE

List indexing

- You can access a list component in several different ways:

```
> j$salary
[1] 55000
> j[["salary"]]
[1] 55000
> j[[2]]
[1] 55000
```

g·able

BIG DATA EXPERIENCE

Adding list elements

```
> z = list(a='abc',b=12)
> z
$a
[1] "abc"

$b
[1] 12
```

```
> z$c = TRUE
> z
$a
[1] "abc"

$b
[1] 12

$c
[1] TRUE
```

Element can be added directory

BIG DATA EXPERIENCE

Deleting list elements

```
> z
$a
[1] "abc"

$b
[1] 12

$c
[1] TRUE

[[4]]
[1] 1+1i

[[5]]
[1] FALSE

[[6]]
[1] TRUE

[[7]]
[1] TRUE
```

```
> z[4:7] <- NULL
> z
$a
[1] "abc"

$b
[1] 12

$c
[1] TRUE
```

BIG DATA EXPERIENCE

Getting list size

```
> z
$a
[1] "abc"

$b
[1] 12

$c
[1] TRUE

> length(z)
[1] 3
```

g·able

BIG DATA EXPERIENCE

Accessing list components and values

- If the components in a list do have tags, as is the case with name, salary, and union for `j`, you can obtain them via names():

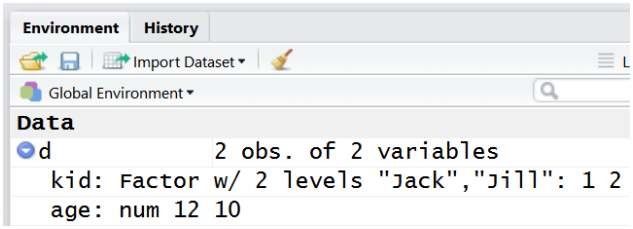
```
> names(j)
[1] "name"   "salary" "union"
> uj <- unlist(j)
> class(uj)
[1] "character"
> uj
   name  salary   union
  "Joe" "55000"  "TRUE"
```

g·able

Data Frames

- On an intuitive level, a *data frame* is like a matrix, with a two-dimensional rows and columns structure.
- On a technical level, a data frame is a list, with the components of that list being equal-length vectors.

```
> kid <- c("Jack","Jill")
> age <- c(12,10)
> d <- data.frame(kid,age)
> d
   kid age
1 Jack  12
2 Jill  10
```



Accessing data frame

```
> d[[1]]
[1] Jack Jill
Levels: Jack Jill
> d$kid
[1] Jack Jill
Levels: Jack Jill
> d[,1]
[1] Jack Jill
Levels: Jack Jill
```

BIG DATA EXPERIENCE

Object internal structure

```
> str(d)
'data.frame':   2 obs. of  2 variables:
 $ kid: Factor w/ 2 levels "Jack","Jill": 1 2
 $ age: num  12 10
```

g·able

BIG DATA EXPERIENCE

Example: mtcars

- You can view the top rows of the data frame by using head()

```
> head(mtcars)
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

g·able

BIG DATA EXPERIENCE

Adding column to data frame

```
> mtcars$ratio = mtcars$hp / mtcars$cyl
> head(mtcars)
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
                     ratio
Mazda RX4         18.33333
Mazda RX4 Wag     18.33333
Datsun 710        23.25000
Hornet 4 Drive    18.33333
Hornet Sportabout 21.87500
Valiant           17.50000
```

g·able

BIG DATA EXPERIENCE

Delete column from data frame

```
> mtcars$ratio <- NULL
> head(mtcars)
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

g·able

Adding row to data frame

```
> tail(rbind(mtcars,list(99,10,200,999,9,9,99,1,1,9,9)))
                  mpg cyl  disp  hp drat    wt qsec vs am gear carb
Lotus Europa     30.4   4  95.1 113 3.77 1.513 16.9  1  1    5    2
Ford Pantera L   15.8   8 351.0 264 4.22 3.170 14.5  0  1    5    4
Ferrari Dino     19.7   6 145.0 175 3.62 2.770 15.5  0  1    5    6
Maserati Bora    15.0   8 301.0 335 3.54 3.570 14.6  0  1    5    8
Volvo 142E       21.4   4 121.0 109 4.11 2.780 18.6  1  1    4    2
33               99.0  10 200.0 999 9.00 9.000 99.0  1  1    9    9
```

rbind() produces a new data frame with additional row
tail() function can be used to view the end of data frame

Factors

- Factors form the basis for many of R's powerful operations, including many of those performed on tabular data.
- The motivation for factors comes from the notioncof *nominal*, or *categorical*, variables in statistics.
- These values are nonnumerical in nature, corresponding to categories such as RENT, MORTGAGE, OWN, although they may be coded using numbers.
- Statistical model relies on factors (and levels) to indicate group of data to be analyzed

BIG DATA EXPERIENCE

g·able

Factors and levels

- An R *factor* might be viewed simply as a vector with a bit more information added (though, as seen below, it's different from this internally).
- That extra information consists of a record of the distinct values in that vector, called *levels*. Here's an example:

```
> x <- c(5,12,13,12)
> xf <- factor(x)
> xf
[1] 5  12 13 12
Levels: 5 12 13
>
> str(xf)
 Factor w/ 3 levels "5","12","13": 1 2 3 2
```

BIG DATA EXPERIENCE

g·able

Data Preparation

Section 2

BIG DATA EXPERIENCE

g·able

Two packages to help you work with the structure of data.

tidyr

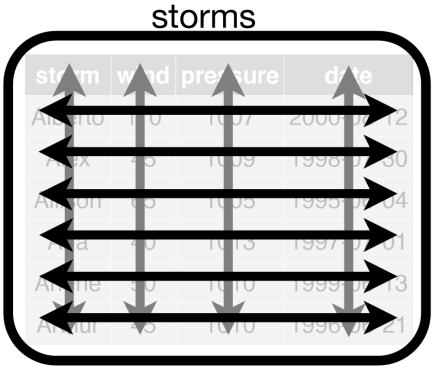
dplyr

BIG DATA EXPERIENCE

g·able

Tidy data

storms



1. Each **variable** is saved in its own **column**.
2. Each **observation** is saved in its own **row**.
3. Each "type" of observation stored in a **single table** (here, storms).

BIG DATA EXPERIENCE

Load data

Data	
cases	3 obs. of 4 variables
pollution	6 obs. of 3 variables
storms	6 obs. of 4 variables
tb	3800 obs. of 6 variables

```
load("2-DataPrep.RData")
```

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

cases

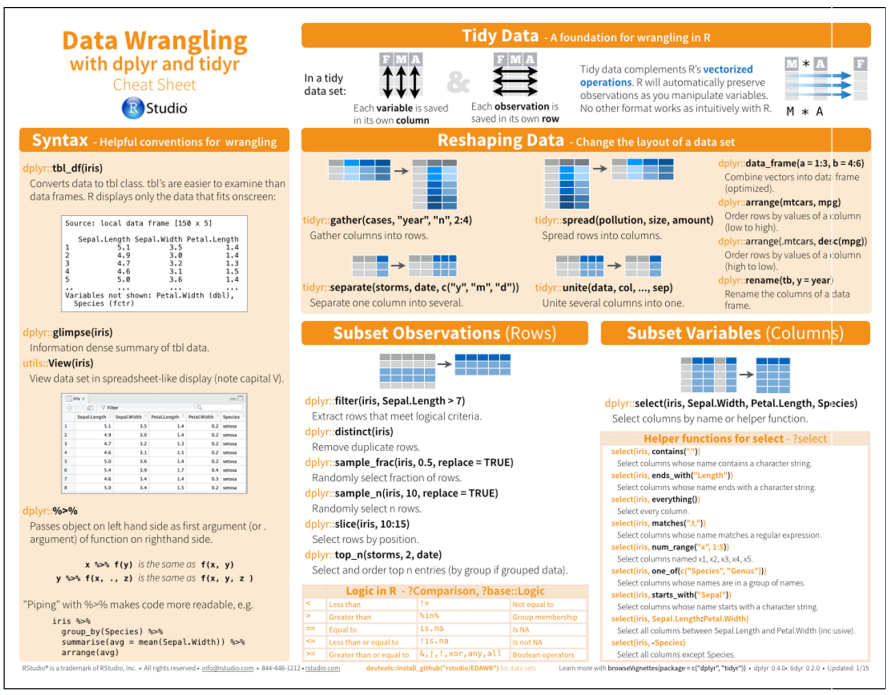
Country	2011	2012	2013
FR	7000	6900	7000
DE	5800	6000	6200
US	15000	14000	13000

pollution

city	particle size	amount ($\mu g/m^3$)
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

Tidy data

BIG DATA EXPERIENCE



http://www.rstudio.com/resources/cheatsheets/

BIG DATA EXPERIENCE

Data Preparation Tasks

- Reshaping table: gather, spread
- Reshaping column: unite, separate
- Subsetting rows: filter, distinct, sample_n, top_n
- Subsetting columns: select
- Summarise data: summarise
- Group data: group_by
- Make new variables: mutate
- Integration: left_join, union, bind_cols

g·able

BIG DATA EXPERIENCE

tidyr
Gather columns

```
##   country  2011  2012  2013
## 1      FR  7000  6900  7000
## 2      DE  5800  6000  6200
## 3      US 15000 14000 13000
```

```
##   country year     n
## 1      FR 2011  7000
## 2      DE 2011  5800
## 3      US 2011 15000
## 4      FR 2012  6900
## 5      DE 2012  6000
## 6      US 2012 14000
## 7      FR 2013  7000
## 8      DE 2013  6200
## 9      US 2013 13000
```

```
gather(cases, "year", "n", 2:4)
```

g·able

BIG DATA EXPERIENCE

tidyr
Spread columns

```
##       city  size amount
## 1 New York large     23
## 2 New York small     14
## 3   London large     22
## 4   London small     16
## 5  Beijing large    121
## 6  Beijing small     56
```

```
##       city large small
## 1  Beijing   121    56
## 2   London    22    16
## 3 New York    23    14
```

```
spread(pollution, size, amount)
```

g·able

BIG DATA EXPERIENCE

separate()

Separate splits a column by a character string separator.

```
separate(storms, date, c("year", "month", "day"), sep = "-")
```

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

storms2

storm	wind	pressure	year	month	day
Alberto	110	1007	2000	08	12
Alex	45	1009	1998	07	30
Allison	65	1005	1995	06	04
Ana	40	1013	1997	07	1
Arlene	50	1010	1999	06	13
Arthur	45	1010	1996	06	21

g·able

BIG DATA EXPERIENCE

unite()

Unite unites columns into a single column.

```
unite(storms2, "date", year, month, day, sep = "-")
```

storms2

storm	wind	pressure	year	month	day
Alberto	110	1007	2000	08	12
Alex	45	1009	1998	07	30
Allison	65	1005	1995	06	04
Ana	40	1013	1997	07	1
Arlene	50	1010	1999	06	13
Arthur	45	1010	1996	06	21

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

KM UNI g·able

BIG DATA EXPERIENCE

Ways to access information

1. **Extract** existing variables. — **select()**
2. **Extract** existing observations. — **filter()**
3. **Derive** new variables (from existing variables) — **mutate()**
4. **Change** the unit of analysis — **summarise()**

KM UNI g·able

BIG DATA EXPERIENCE

select()

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

storm	pressure
Alberto	1007
Alex	1009
Allison	1005
Ana	1013
Arlene	1010
Arthur	1010

```
select(storms, storm, pressure)
```

g·able

BIG DATA EXPERIENCE

select()

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

wind	pressure	date
110	1007	2000-08-12
45	1009	1998-07-30
65	1005	1995-06-04
40	1013	1997-07-01
50	1010	1999-06-13
45	1010	1996-06-21

```
select(storms, -storm)
# see ?select for more
```

g·able

BIG DATA EXPERIENCE

select()

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

wind	pressure	date
110	1007	2000-08-12
45	1009	1998-07-30
65	1005	1995-06-04
40	1013	1997-07-01
50	1010	1999-06-13
45	1010	1996-06-21

```
select(storms, wind:date)
# see ?select for more
```

g·able

BIG DATA EXPERIENCE

Useful select functions

* Blue functions come in dplyr

-	Select everything but
:	Select range
contains()	Select columns whose name contains a character string
ends_with()	Select columns whose name ends with a string
everything()	Select every column
matches()	Select columns whose name matches a regular expression
num_range()	Select columns named x1, x2, x3, x4, x5
one_of()	Select columns whose names are in a group of names
starts_with()	Select columns whose name starts with a character string

g·able

BIG DATA EXPERIENCE

filter()

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Allison	65	1005	1995-06-04
Arlene	50	1010	1999-06-13

```
filter(storms, wind >= 50)
```

g·able

BIG DATA EXPERIENCE

filter()

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Allison	65	1005	1995-06-04

```
filter(storms, wind >= 50,
  storm %in% c("Alberto", "Alex", "Allison"))
```

g·able

BIG DATA EXPERIENCE

logical tests in R

`?Comparison`

<	Less than
>	Greater than
==	Equal to
<=	Less than or equal to
>=	Greater than or equal to
!=	Not equal to
%in%	Group membership
is.na	Is NA
!is.na	Is not NA

`?base::Logic`

&	boolean and
|	boolean or
xor	exactly or
!	not
any	any true
all	all true

g·able

BIG DATA EXPERIENCE

mutate()

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

→

storm	wind	pressure	date	ratio	inverse
Alberto	110	1007	2000-08-12	9.15	0.11
Alex	45	1009	1998-07-30	22.42	0.04
Allison	65	1005	1995-06-04	15.46	0.06
Ana	40	1013	1997-07-01	25.32	0.04
Arlene	50	1010	1999-06-13	20.20	0.05
Arthur	45	1010	1996-06-21	22.44	0.04

```
mutate(storms, ratio = pressure / wind, inverse = ratio^-1)
```

g·able

BIG DATA EXPERIENCE

Useful mutate functions

* All take a vector of values and return a vector of values
** Blue functions come in dplyr

pmin(), pmax()	Element-wise min and max
cummin(), cummax()	Cumulative min and max
cumsum(), cumprod()	Cumulative sum and product
between()	Are values between a and b?
cume_dist()	Cumulative distribution of values
cumall(), cumany()	Cumulative all and any
cummean()	Cumulative mean
lead(), lag()	Copy with values one position
ntile()	Bin vector into n buckets
dense_rank(), min_rank(), percent_rank(), row_number()	Various ranking methods

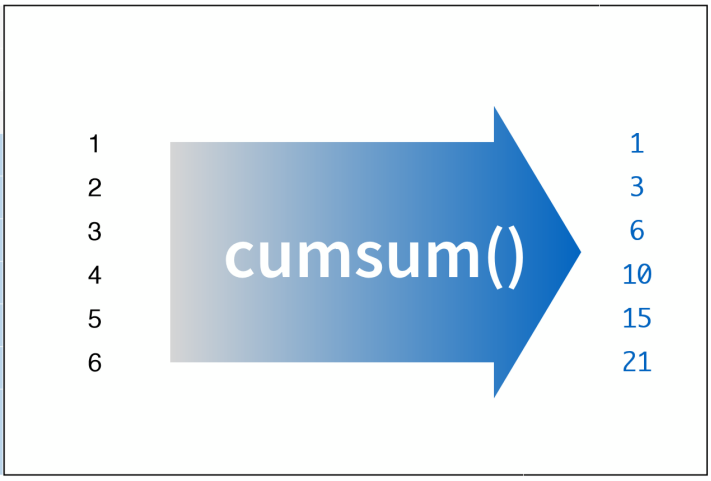
g·able

BIG DATA EXPERIENCE

"Window" functions

* All take a vector of values and return a vector of values

pmin(), pmax()
cummin(), cummax()
cumsum(), cumprod()
between()
cume_dist()
cumall(), cumany()
cummean()
lead(), lag()
ntile()
dense_rank(), min_rank(), percent_rank(), row_number()



g·able

BIG DATA EXPERIENCE

summarise()

city	particle size	amount ($\mu g/m^3$)
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

median	variance
22.5	1731.6

```
pollution %>% summarise(median = median(amount), variance = var(amount))
```

BIG DATA EXPERIENCE

summarise()

city	particle size	amount ($\mu g/m^3$)
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

mean	sum	n
42	252	6

```
pollution %>% summarise(mean = mean(amount), sum = sum(amount), n = n())
```

BIG DATA EXPERIENCE

Useful summary functions

* All take a vector of values and return a single value
** Blue functions come in dplyr

min(), max()	Minimum and maximum values
mean()	Mean value
median()	Median value
sum()	Sum of values
var, sd()	Variance and standard deviation of a vector
first()	First value in a vector
last()	Last value in a vector
nth()	Nth value in a vector
n()	The number of values in a vector
n_distinct()	The number of distinct values in a vector

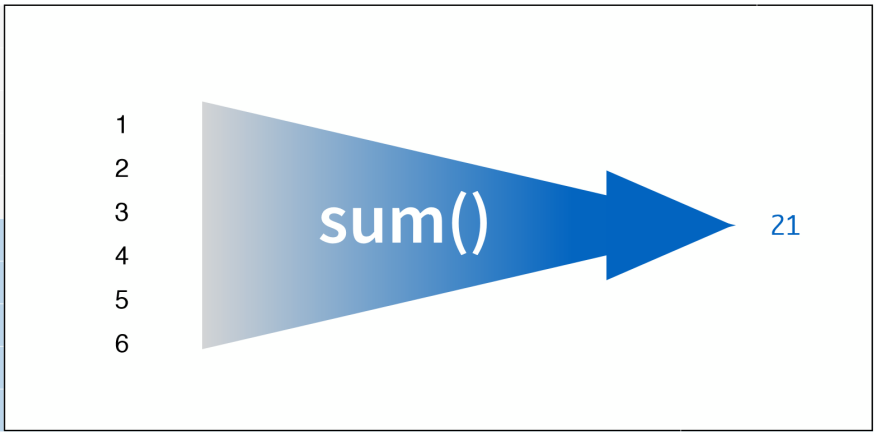
g·able

BIG DATA EXPERIENCE

"Summary" functions

* All take a vector of values and return a single value

min(), max()
mean()
median()
sum()
var, sd()
first()
last()
nth()
n()
n_distinct()



g·able

BIG DATA EXPERIENCE

arrange()

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

storm	wind	pressure	date
Ana	40	1013	1997-07-01
Alex	45	1009	1998-07-30
Arthur	45	1010	1996-06-21
Arlene	50	1010	1999-06-13
Allison	65	1005	1995-06-04
Alberto	110	1007	2000-08-12

```
arrange(storms, wind)
```

g·able

BIG DATA EXPERIENCE

arrange()

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

storm	wind	pressure	date
Ana	40	1013	1997-07-01
Alex	45	1009	1998-07-30
Arthur	45	1010	1996-06-21
Arlene	50	1010	1999-06-13
Allison	65	1005	1995-06-04
Alberto	110	1007	2000-08-12

```
arrange(storms, wind)
```

g·able

BIG DATA EXPERIENCE

arrange()

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Allison	65	1005	1995-06-04
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21
Alex	45	1009	1998-07-30
Ana	40	1013	1997-07-01

```
arrange(storms, desc(wind))
```

g·able

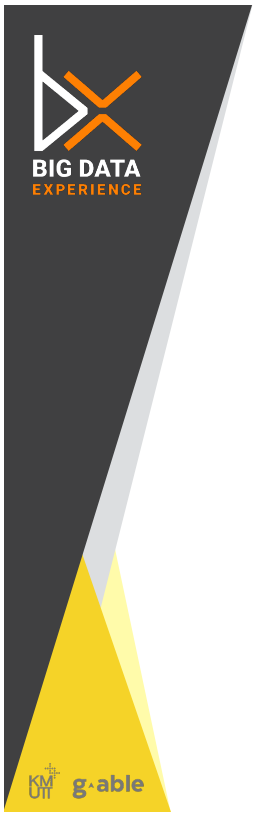
BIG DATA EXPERIENCE

The pipe operator %>%

```
library(dplyr)
select(tb, child:elderly)
tb %>% select(child:elderly)
```


```
tb          select( _____, child:elderly)
```

g·able



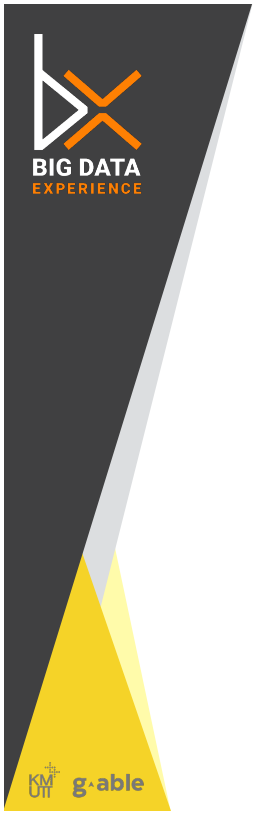
select()

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

storm	pressure
Alberto	1007
Alex	1009
Allison	1005
Ana	1013
Arlene	1010
Arthur	1010

```
select(storms, storm, pressure)
```



select()

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

storm	pressure
Alberto	1007
Alex	1009
Allison	1005
Ana	1013
Arlene	1010
Arthur	1010

```
storms %>% select(storm, pressure)
```

BIG DATA EXPERIENCE

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

storm	pressure
Alberto	1007
Allison	1005
Arlene	1010

```
storms %>%
  filter(wind >= 50) %>%
  select(storm, pressure)
```

g·able

BIG DATA EXPERIENCE

mutate()

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21


```
storms %>%
  mutate(ratio = pressure / wind) %>%
  select(storm, ratio)
```

g·able

BIG DATA EXPERIENCE

mutate()

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

storm	ratio
Alberto	9.15
Alex	22.42
Allison	15.46
Ana	25.32
Arlene	20.20
Arthur	22.44

```
storms %>%
  mutate(ratio = pressure / wind) %>%
  select(storm, ratio)
```


BIG DATA EXPERIENCE

dplyr::**bind_cols()**

y

x1	x2
A	1
B	2
C	3

+

z

x1	x2
B	2
C	3
D	4

=

x1	x2	x1	x2
A	1	B	2
B	2	C	3
C	3	D	4

```
bind_cols(y, z)
```

BIG DATA EXPERIENCE

dplyr::**bind_rows()**

y

x1	x2
A	1
B	2
C	3

+

z

x1	x2
B	2
C	3
D	4

=

x1	x2
A	1
B	2
C	3
B	2
C	3
D	4

```
bind_rows(y, z)
```

g·able

BIG DATA EXPERIENCE

city	particle size	amount (μg/m³)
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

→

mean	sum	n
18.5	37	2
19.0	38	2
88.5	177	2

group_by() + summarise()

g·able

BIG DATA EXPERIENCE

city	size	amount
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

city	size	amount
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

city	mean
New York	18.5
London	19.0
Beijing	88.5

```
pollution %>% group_by(city) %>%  summarise(mean = mean(amount))
```

g·able

BIG DATA EXPERIENCE

dplyr::**left_join()**

songs

song	name
Across the Universe	John
Come Together	John
Hello, Goodbye	Paul
Peggy Sue	Buddy

+

artists

name	plays
George	sitar
John	guitar
Paul	bass
Ringo	drums

=

song	name	plays
Across the Universe	John	guitar
Come Together	John	guitar
Hello, Goodbye	Paul	bass
Peggy Sue	Buddy	<NA>

```
left_join(songs, artists, by = "name")
```

g·able

BIG DATA EXPERIENCE

Activity & Assignment 1

- Load a data frame "2-bank-data.csv"
  - Hint: `read.csv`
- Write a function that
  - receive two numbers (x_1,x_2) and
  - return data frame with only customers that $x_1 < age < x_2$
- Use the function `head(data-frame)` to view the top of the table

g·able

BIG DATA EXPERIENCE

Thank you

Question?

g·able