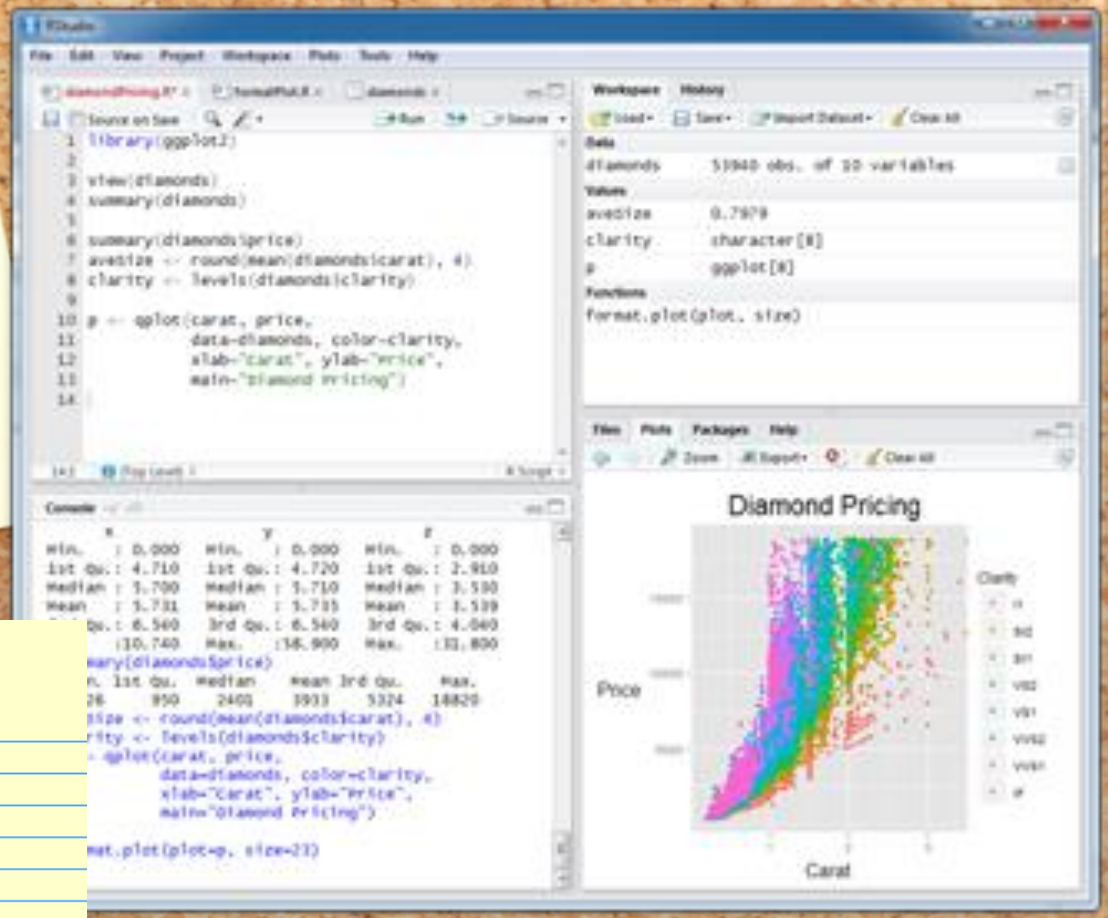
R

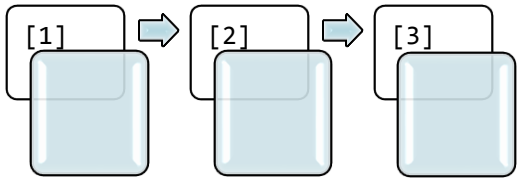


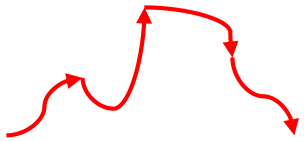
Learning by Examples

Michelle Darling
Fall 2013

R data structures

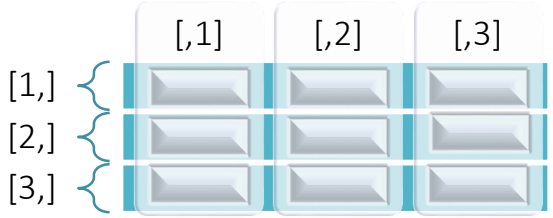
VECTOR





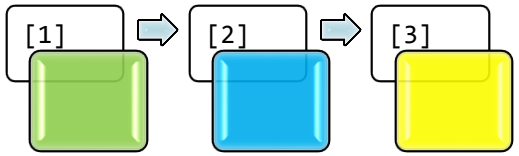
- 1 row, N columns.
- One data type only (numeric, character, date, OR logical).
- Uses: track changes in a single variable over time.
- Examples: stock prices, hurricane path, temp readings, disease spread, financial performance, sports scores.

MATRIX



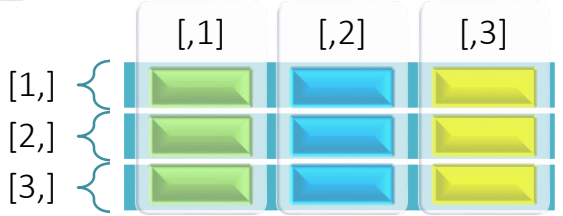
- N row, N columns.
- One data type only (any combination of numeric, character, date, logical).
- Basically, a collection of vectors.

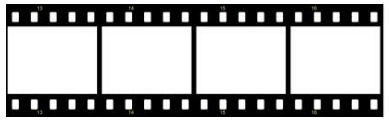
LIST



- 1 row, N columns. Multiple data types.
- Uses: ist detailed information for a person/place/thing/concept.
- Examples: Listing for real estate, book, movie, contact, country, stock, company, etc. Or, a "snapshot" or observation of an event or phenomenon such as stock market, or scientific experiment.

DATA FRAME





- N rows, N columns.
- Multiple data types.
- Basically, a collection of lists or snapshots which when assembled together provide a "bigger picture."

Other important R concepts

FACTORS

Stores each distinct value only once, and the data itself is stored as a vector of integers. When a factor is first created, all of its levels are stored along with the factor.

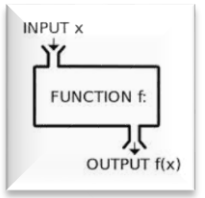
```
> weekdays=c("Monday","Tuesday","Wednesday","Thursday","Friday")
> wf <- factor(weekdays)
[1] Monday    Tuesday   Wednesday Thursday  Friday
Levels: Friday Monday Thursday Tuesday Wednesday
```

Used to group and summarize data:

```
WeekDaySales <- (DailySalesVector, wf, sum)
# Sum daily sales figures by M,T,W,Th,F
```

USER-DEFINED FUNCTIONS

```
> f <- function(a) { a^2 }
> f(2)
[1] 4
```



- Functions can be passed as arguments to other functions.
- Function behavior is defined inside the curly brackets { }.
- Functions can be nested, so that you can define a function inside another.
- The return value of a function is the last expression evaluated.

PACKAGES, FUNCTIONS, DATASETS

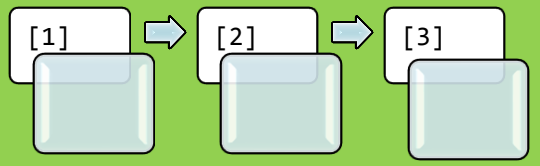
```
> search() # Search for installed packages & datasets
 [1] ".GlobalEnv"        "mtcars"            "tools:rstudio"
 [4] "package:stats"     "package:graphics"  "package:grDevices"

> library(ggplot2) # load package ggplot2
Attaching package: 'ggplot2'

> data()  # List available datasets

> attach(iris) # Attach dataset "iris"
```

SPECIAL VALUES

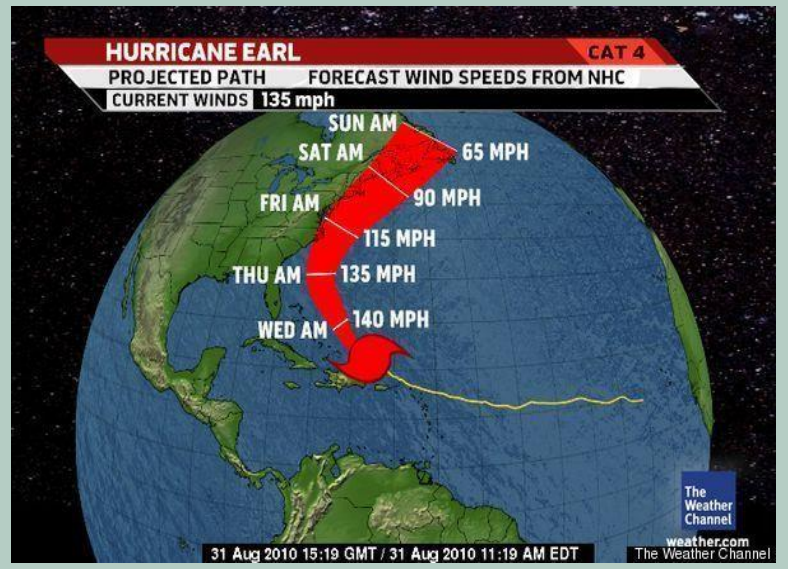
- **`pi=3.141593.`** Use lowercase "pi"; "Pi" or "PI" won't work
- **`inf=1/0`** (Infinity)
- **NA=Not Available.** A logical constant of length 1 that means neither TRUE nor FALSE. Causes functions to barf.
  - Tell function to ignore NAs: `function(args, na.rm=TRUE)`
  - Check for NA values: `is.na(x)`
- **NULL=Empty Value.** Not allowed in vectors or matrixes.
  - Check for NULL values: `is.null(x)`
- **NaN=Not a Number.** Numeric data type value for undefined (e.g., 0/0).

See this for NA vs. NULL explanation.

VECTOR: Examples

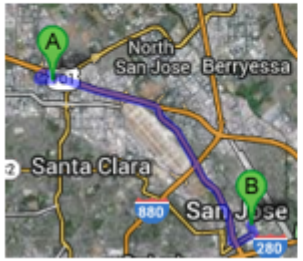



```
VECTOR: D_url
> D_url <-"www.data.gov"
> print(D_url)
[1] "www.data.gov"
> mode(D_url)
[1] "character"
> dim(D_url)
NULL
D_url is a 1X1 vector; data type CHARACTER.
dim() returns NULL because D_url has no row indexes.
```



VECTOR: Trip_A_to_B

[1]	[2]	[3]	[4]
37.382392	-121.971920	37.333718	-121.889410

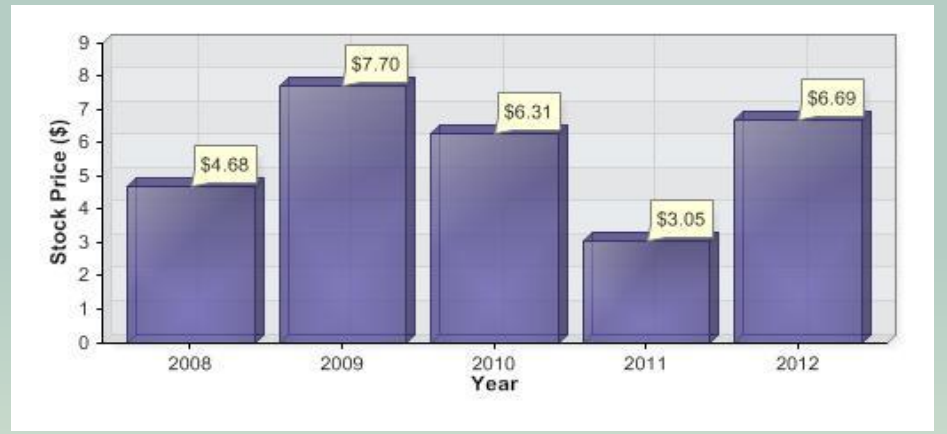


A UCSC Extension, Augustine Drive, Santa Clar

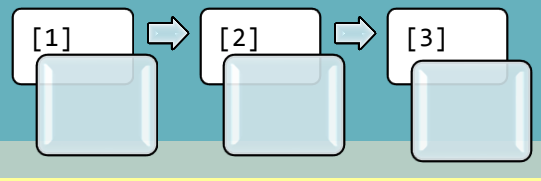
B Downtown San Jose, San Jose, CA

[1] Latitude of Point A
[2] Longitude of Point A
[3] Latitude of Point B
[4] Longitude of Point B

DATA TYPE: NUMERIC only.



VECTORS



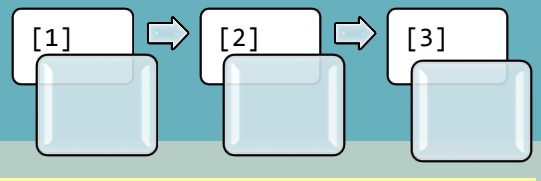

```
# 1xN array of same data type
> v<-c(1:3); v
[1] 1 2 3
> mode(v)   # displays data type
[1] "numeric"
> v <-c("one", "two", "three"); v
[1] "one"   "two"   "three"
> mode(v)
[1] "character"
> v <-c(TRUE,FALSE,TRUE); v
[1]  TRUE FALSE  TRUE
> mode(v)
[1] "logical"
> v<-c(pi, 2*pi, 3*pi); v
[1] 3.141593 6.283185 9.424778
> mode(v)
[1] "numeric"

# Numeric values coerced into character mode
> v<-c(1,2,3,"one", "two", "three"); v
[1] "1"     "2"     "3"     "one"   "two"
"three"
> mode(v)
[1] "character"
```

BASIC OPERATIONS

```
# Addition
> v1<-1:3
> v2 <- c(10,10,10)
> mode(v1)
[1] "numeric"
> mode(v2)
[1] "numeric"
> v1+v2
[1] 11 12 13

# Multiplication &
Division
> v1 * v2
[1] 10 20 30
> v1 / v2
[1] 0.1 0.2 0.3
> v2 / v1
[1] 10.000000  5.000000
3.333333
```

```
#Subtraction
> v1-v2
[1] -9 -8 -7
> v2-v1
[1] 9 8 7

# Logical Comparison
> v1==v2
[1] FALSE FALSE FALSE
> v1 != v2
[1] TRUE TRUE TRUE
> v1 > v2
[1] FALSE FALSE FALSE
> v1 < v2
[1] TRUE TRUE TRUE
```

VECTORS

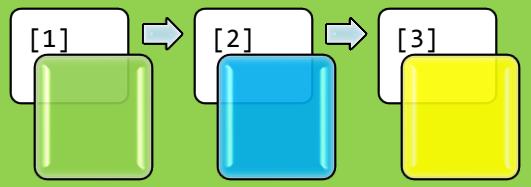



```
# By default, column numbers are used as indexes
> v3[1]
[1] 1

# But columns can be given meaningful names…
> names(v3) # What are current column names?
NULL
> names(v3)<- c("1st","2nd","3rd","4th","5th",
"6th") # Rename column names.
> names(v3) [1] "1st" "2nd" "3rd" "4th" "5th"
"6th"
> v3
1st 2nd 3rd 4th 5th 6th
  1   2   3  10  10  10
# Now we can use names as indexes:
> v3["6th"] # same as v3[6]
6th
 10
> v3[c("1st","6th")] # same as v3[c(1,6)]
1st 6th
  1  10
> v3[-1]  # Can exclude columns using (-)
2nd 3rd 4th 5th 6th
  2   3  10  10  10
```

INDEXING, SELECTING & SUBSETTING

```
> v3[v3==10] # Select values equal to 10
[1] 10 10 10

> v3[v3!=10] # Select values NOT equal to 10
[1] 1 2 3

> median(v3)
[1] 6.5
> v3[v3<median(v3)] # Select values < median
[1] 1 2 3
> v3[v3>median(v3)] # Select values > median
[1] 10 10 10
> v3 < median(v3)  # Test if value < median?
[1]  TRUE  TRUE  TRUE FALSE FALSE FALSE

> v3 %% 2==0 # Test if value is an even number?
[1] FALSE  TRUE FALSE  TRUE  TRUE  TRUE

> v3 %% 2==1 # Test if value is an odd number?
[1]  TRUE FALSE  TRUE FALSE FALSE FALSE
```

LIST: Examples



LIST: Forecast_Today

[1]	"Mostly Sunny"
[2]	67.00
[3]	NW
[4]	10
[5]	60.00
[6]	06:39
[7]	08:59
[8]	"Waxing Crescent"

Data Types: NUMERIC, DATE, CHARACTER

Mostly Sunny

67°F
High

Chance of Rain:	0%
Wind:	NW at 10 mph
Humidity:	60%
UV Index:	4 - Moderate
Sunrise:	6:39 am
Moonrise:	8:59 am
Moonphase:	Waxing Crescent

Philippines

Country

The Philippines, officially known as the Republic of the Philippines, is a sovereign island country in Southeast Asia situated in the western Pacific Ocean. Wikipedia

Capital: Manila

Dialing code: 63

Currency: Philippine peso

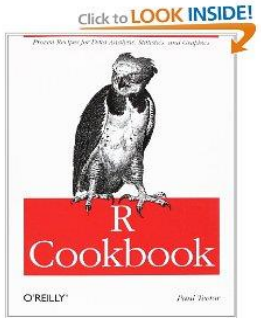
Population: 96.71 million (2012) World Bank

President: Benigno "Noynoy" Aquino III

Official languages: English Language, Filipino language

Product Details

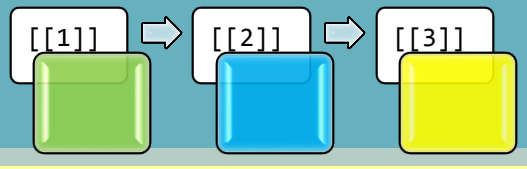
- **Series:** O'Reilly Cookbooks
- **Paperback:** 438 pages
- **Publisher:** O'Reilly Media; 1 edition (March 22, 2011)
- **Language:** English
- **ISBN-10:** 0596809158
- **ISBN-13:** 978-0596809157
- **Product Dimensions:** 0.9 x 7 x 9.2 inches
- **Shipping Weight:** 1.6 pounds



Ender's Game new!
PG-13 , 1 hr 54 min
Click Showtime to Buy Tickets

11:20a | 2:00p | 4:50p | **7:30p** | **10:10p**

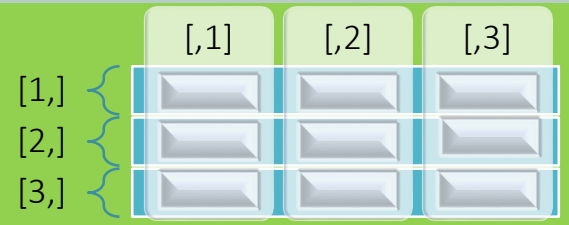
LISTS




```
# 1xN array of multiple data types/modes
> c1 <-c("A", "B", "C")
> n1 <-c(1:3)
> l2 <- list(c1,n1,Sys.Date(),TRUE);l2
[[1]]
[1] "A" "B" "C"
[[2]]
[1] 1 2 3
[[3]]
[1] "2013-11-03"
[[4]]
[1] TRUE
> str(l2)
List of 4
 $ : chr [1:3] "A" "B" "C"
 $ : int [1:3] 1 2 3
 $ : Date[1:1], format: "2013-11-03"
 $ : logi TRUE
> l2[[4]]
[1] TRUE
> l2[[1]]
[1] "A" "B" "C"
--------
>fix('l2')
list(c("A", "B", "C"), 1:3, structure(16012, class =
"Date"),TRUE)
```

```
# Append to a list; the results get trippy
> l2 <- list(l2,pi); l2
[[1]]
[[1]][[1]]
[1] "A" "B" "C"
[[1]][[2]]
[1] 1 2 3
[[1]][[3]]
[1] "2013-11-03"
[[1]][[4]]
[1] TRUE
[[2]]
[1] 3.141593
# Basically, a new () gets added each time the list is
appended
list(list(c("A", "B", "C"), 1:3, structure(16012, class =
"Date"), TRUE), 3.14159265358979)
# [[1]] is not the same as [1]
> mode(l3[[1]])
[1] "numeric"
> mode(l3[1])
[1] "list"
# To avoid confusion, use names
> l3 = list(x=1,y=2,z=3); l3
$x
[1] 1
$y
[1] 2
$z
[1] 3
> l3$x # this is the same as l3[[1]]
[1] 1
```

MATRIX: Examples

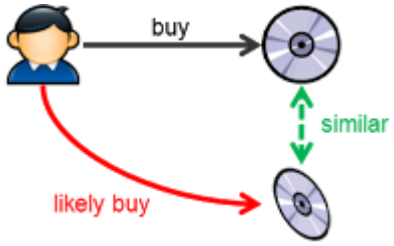


	Ben	Tom	John	Fred
Season 1	5	5	0	5
Season 2	5	0	3	4
Season 3	3	4	0	3
Season 4	0	0	5	3
Season 5	5	4	4	5
Season 6	5	4	5	5

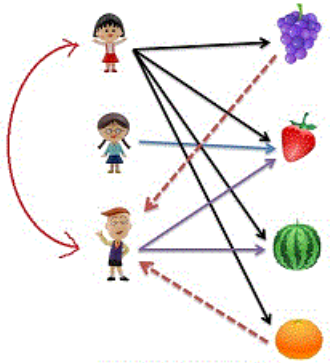
	Avg. APR	Last Week	6 Months
Low Interest	10.46%	10.46%	10.29%
Balance Transfer	12.49%	12.49%	12.59%
Business	12.98%	12.98%	12.98%
Student	13.27%	13.27%	13.16%
Airline	14.51%	14.51%	14.63%
Cash Back	14.62%	14.62%	14.13%
Reward	14.87%	14.87%	14.72%
Bad Credit	23.48%	23.48%	23.64%
Instant Approval	28.00%	28.00%	15.49%

Source:CreditCards.com

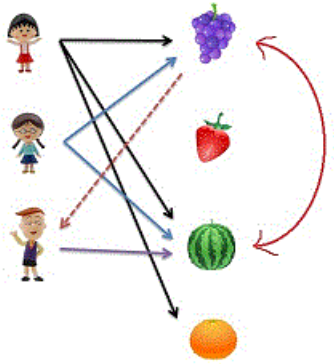
Recommendation Engine Matrices



	item1	item2	item3	item4	item5	item6
user1	bought	bought	bought			
user2						
user3	bought	likely buy	likely buy			
user4						



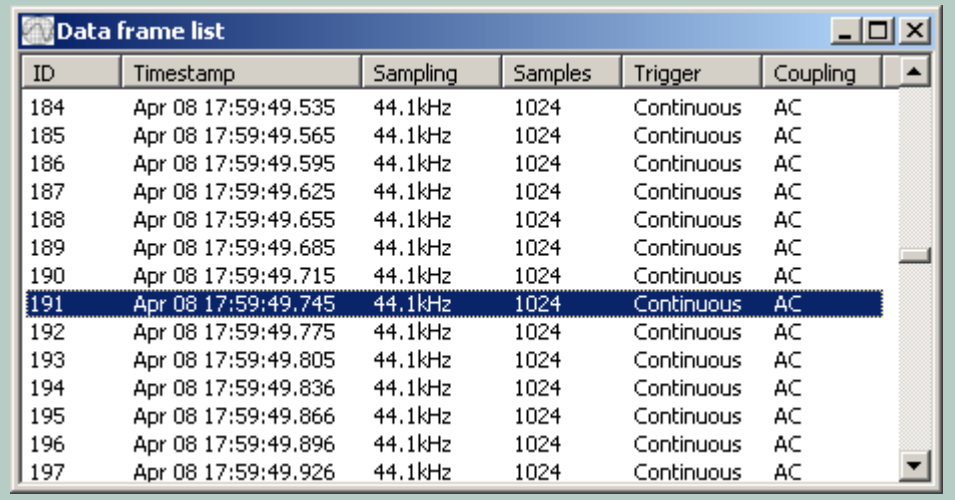
User-based filtering



Item-based filtering

DATA FRAME: Examples

	Country.Name	Capital	Currency Code	Currency. Name	Telephone .Code	Country. Code.TL
2	Afghanistan	Kabul	AFN	Afghani	93	.af
3	Albania	Tirana	ALL	Lek	355	.al
4	Algeria	Algiers	DZD	Dinar	213	.dz
5	Andorra	Andorra la	EUR	Euro	376	.ad
6	Angola	Luanda	AOA	Kwanza	244	.ao
7	Antigua and Barbuda	Saint John	XCD	Dollar	-267	.ag
8	Argentina	Buenos Ai	ARS	Peso	54	.ar
9	Armenia	Yerevan	AMD	Dram	374	.am
10	Australia	Canberra	AUD	Dollar	61	.au
11	Austria	Vienna	EUR	Euro	43	.at
12	Azerbaijan	Baku	AZN	Manat	994	.az
13	Bahamas, The	Nassau	BSD	Dollar	-241	.bs
14	Bahrain	Manama	BHD	Dinar	973	.bh
15	Bangladesh	Dhaka	BDT	Taka	880	.bd
16	Barbados	Bridgetow	BBD	Dollar	-245	.bb
17	Belarus	Minsk	BYR	Ruble	375	.by
18	Belgium	Brussels	EUR	Euro	32	.be
19	Belize	Belmopan	BZD	Dollar	501	.bz
20	Benin	Porto-Nov	XOF	Franc	229	.bj



Data Frames: Most frequently used structure for storing and manipulating data sets. Similar to:

- A database table
- A spreadsheet

Like the above, DFs have rows x columns, but terminology is different:

- Observations = rows
- Variables = Columns

R Table vs. Data Frame: KISS and stick to data frames for now.

#Convert table to data frame:

```
> HEC <- data.frame(HairEyeColor)
> str(HEC)
'data.frame':32 obs. of  4 variables:
 $ Hair: Factor w/ 4 levels "Black","Brown",..: 1 2 3 4 1 2 3 4 1 2 ...
 $ Eye : Factor w/ 4 levels "Brown","Blue",..: 1 1 1 1 2 2 2 2 3 3 ...
 $ Sex : Factor w/ 2 levels "Male","Female": 1 1 1 1 1 1 1 1 1 1 ...
 $ Freq: num  32 53 10 3 11 50 10 30 10 25 ...
```

DATA FRAMES

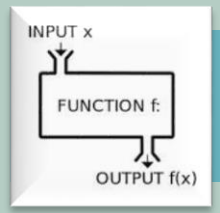

```
# HEC[1,] returns a row
> HEC[1,]
   Hair   Eye  Sex Freq
1 Black Brown Male   32

# Subsetting made easier
> HEC6 <-subset(HEC,select=Hair); str(HEC6)
'data.frame':         32 obs. of  1 variable:
 $ Hair: Factor w/ 4 levels "Black","Brown",..: 1 2 3 4

> HEC7 <-subset(HEC,select= c(Hair,Eye)); str(HEC7)
'data.frame':         32 obs. of  2 variables:
 $ Hair: Factor w/ 4 levels "Black","Brown"...
 $ Eye : Factor w/ 4 levels "Brown","Blue"...

> HEC8 <-subset(HEC, subset=(Hair == "Black" & Eye ==
"Brown")); HEC8
    Hair   Eye    Sex Freq
1  Black Brown   Male   32
17 Black Brown Female   36
```

INDEXING, SELECTING & SUBSETTING

```
# HEC[[1]], HEC[,"Hair"], HEC$Hair return column
> HEC1 <-HEC[[1]]; HEC1
> str(HEC1)
 Factor w/ 4 levels "Black","Brown",..: 1 2 3 4 1 2 3 4 1 2 ...

# HEC[1] and HEC["Hair"] return column dframe
> HEC2 <-HEC[1]; HEC2
> str(HEC2)
'data.frame':32 obs. of  1 variable:
 $ Hair: Factor w/ 4 levels "Black","Brown",..: 1 2 3 4 1 2 3 4 1 2
> HEC4 <-HEC["Hair"]
> HEC2 == HEC4
      Hair
 [1,] TRUE
 [2,] TRUE  etc.

# Returning multiple columns in a data frame
# This is the same as HEC[,c(1, 4)]
> HEC5 <-HEC[,c("Hair", "Freq")]
> str(HEC5)
'data.frame':           32 obs. of  2 variables:
$ Hair: Factor w/ 4 levels "Black","Brown",..: 1 2 3 4 1 2 3 4 1
$ Freq: num  32 53 10 3 11 50 10 30 10 25 ...
```

DATA FRAMES

```
# Combine 2 DFs columnwise
> echo <- cbind(HEC2, HEC4)
> echo
    Hair  Hair
1  Black Black
2  Brown Brown
3  Red   Red  etc

# Stack 2 DFs (UNION)
> rbind(HEC8, HEC8)
     Hair   Eye    Sex Freq
1   Black Brown   Male   32
17  Black Brown Female   36
11  Black Brown   Male   32
171 Black Brown Female   36

# Skip having to specify the DF for col names
> f <- sum(HEC$Freq)  # Instead of this

> attach(HEC)
> f <- sum(Freq) # Use this
```



FUNCTIONS

SEQUENCING

seq(*from*,*to*,*by*)	generate a sequence
	`indices <- seq(1,10,2)`
	`#indices is c(1, 3, 5, 7, 9)`

rep(*x*,*ntimes*)	repeat *x* *n* times
	`y <- rep(1:3, 2)`
	`# y is c(1, 2, 3, 1, 2, 3)`

cut(*x*,*n*)	divide continuous variable in factor with *n* levels `y <- cut(x, 5)`

DATE PROCESSING

Sys.Date()	generate today's date
	`> Sys.Date()`
	`[1] "2013-11-03`

as.date()	Convert string to date format
	`> to=as.Date('2006-1-10')`
	`> mode(to)`
	`[1] "numeric"`
	`> class(to)`
	`[1] "Date"`

CHARACTER PROCESSING

CHARACTER PROCESSING	Description
substr(*x*, start=*n1*, stop=*n2*)	Extract or replace substrings in a character vector. x <- "abcdef" substr(x, 2, 4) is "bcd" substr(x, 2, 4) <- "22222" is "a222ef"
grep(*pattern*, *x* , ignore.case=FALSE, fixed=FALSE)	Search for *pattern* in *x*. If fixed =FALSE then *pattern* is a regular expression. If fixed=TRUE then *pattern* is a text string. Returns matching indices. grep("A", c("b","A","c"), fixed=TRUE) returns 2
sub(*pattern*, *replacement*,*x*, ignore.case =FALSE, fixed=FALSE)	Find *pattern* in *x* and replace with *replacement* text. If fixed=FALSE then *pattern* is a regular expression. If fixed = T then *pattern* is a text string. sub("\\s",".","Hello There") returns "Hello.There"
strsplit(*x*, *split*)	Split the elements of character vector *x* at *split*. strsplit("abc", "") returns 3 element vector "a","b","c"
paste(..., sep="")	Concatenate strings after using *sep* string to seperate them. paste("x",1:3,sep="") returns c("x1","x2" "x3") paste("x",1:3,sep="M") returns c("xM1","xM2" "xM3") paste("Today is", date())
toupper(*x*)	Uppercase
tolower(*x*)	Lowercase

TYPE CONVERSION

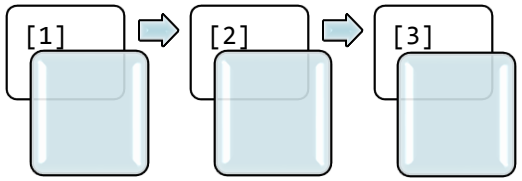
```
as.character(x)
as.complex(x)
as.numeric(x)
as.logical(x)
```

STRUCTURE CONVERSION

```
as.data.frame(x)
as.list(x)
as.matrix(x)
as.vector(x)
```

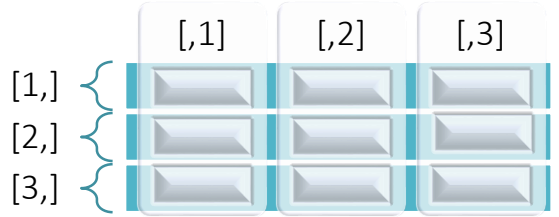
DATA TRANSFORMATIONS

VECTOR

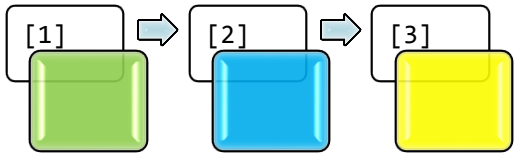



```
# s=simplify into a vector
# sapply returns a vector
l <- sapply(lst,function)

# lapply returns a list
v <- lapply(lst,function)
```

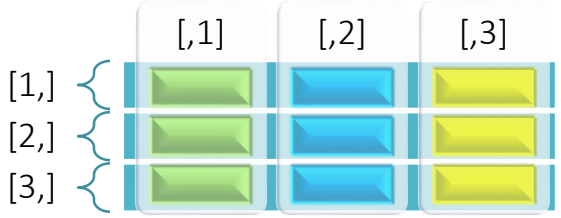
MATRIX



LIST



DATA FRAME



GET HELPFUL INFO

```
# Get help
>help.search("cat") # find info about "cat"
>?mean # get help about function
>example(mean) # get examples

# List objects in workspace
> ls()
[1] "tbl"   "w_day"

# List all available datasets
> data()

# Get structure
> str(HairEyeColor)
 table [1:4, 1:4, 1:2] 32 53 10 3 11 50 10 30 10 25
...
 - attr(*, "dimnames")=List of 3
  ..$ Hair: chr [1:4] "Black" "Brown" "Red" "Blond"
  ..$ Eye : chr [1:4] "Brown" "Blue" "Hazel" "Green"
  ..$ Sex : chr [1:2] "Male" "Female"
# Get Class (vector,list, dataframe, table, matrix,
numeric, function, factor,, et)
> class(HairEyeColor)
[1] "table"

# Use Google R style sheet
```

PRINTING

```
> print(matrix(c(1234),2,2))
     [,1] [,2]
[1,] 1234 1234
[2,] 1234 1234

> print(matrix(c(1,2,3,4),2,2))
     [,1] [,2]
[1,]    1    3
[2,]    2    4

> print ("print works on only");print("one
string or variable at a time"); print(pi)
[1] "print works on only"
[1] "one string or variable at a time"
[1] 3.141593

> num <-1:10
> print(num)
 [1]  1  2  3  4  5  6  7  8  9 10

# cat works only on strings and vectors
> cat("the first 10 numbers are:", num, "\n")
the first 10 numbers are: 1 2 3 4 5 6 7 8 9 10
```

INPUT / OUTPUT

```
Ctrl-R executes the selected line(s)

# Getting and setting the working directory
> getwd()
[1] "C:/Users/mdarling/Documents"
> setwd("DA/data")
[1] "C:/Users/mdarling/Documents/DA/data"

# Enter data using spreadsheet editor
w_day <- data.frame()
w_day <- edit(w_day)

# Read data from URL
> tbl <-
read.csv("http://www.andrewpatton.com/countrylist.csv")

# Write data to csv file
> write.csv(tbl, "countries.csv")
# Read data from HTML tables
> library(XML)
> url <-"http://www.andrewpatton.com/countrylist.html"
> tbls <- readHTMLTable(url)
```

MORE DATE PROCESSING

```
library(timeDate)
ymdhs <- "2012-03-04 05:06:07"
pd.sec <- as.POSIXlt(ymdhs)$sec
pd.hour <- as.POSIXlt(ymdhs)$hour
pd.min <- as.POSIXlt(ymdhs)$min
pd.mday <- as.POSIXlt(ymdhs)$mday
pd.mon <- ((as.POSIXlt(ymdhs)$mon)+1)
pd.year <- ((as.POSIXlt(ymdhs)$year) + 1900)
```

PLOTTING

```
Plotting in R
-base
-ggplot2, ggmap, map

Types of Graphs
-chloropleth
-heat map

# Base plots
plot(faithful, type = 'l') #line graph
plot(faithful, type = 'p') #point graph
hist(faithful$waiting)     #histogram of column waiting

# Quickly plot a matrix of scatterplots
# This plots each column vs. all the other ones
names(iris)
[1] "Sepal.Length" "Sepal.Width"  "Petal.Length"
"Petal.Width"  "Species"
pairs(iris[,-5])
pairs(iris[,1:2])

# Plot x vs. y using 2 df columnns and geom_point()
ggplot(movies, aes(x=year, y=budget)) + geom_point()

# Plot histogram using 1 column, Note: geom_bar()
ggplot(movies, aes(x=year)) + geom_bar()

# plot all rows vs. mpaa column
plot(movies[, "mpaa"]) # plot has lots of nulls
mpaa.movies <- subset(movies, mpaa != "")! # exclude nulls
plot(mpaa.movies[, "mpaa"])
# Or use na.rm
```