Introduction to R at CDC Part 2

James Durant[[1]](#footnote-1)

December 05, 2017



# Learning Resources

## Additional Resources

* HHS Learning Portal
  + Introduction to R Programming (PG\_RPRG\_A01\_IT\_ENUS, 2.2)
  + Books 24x7 (under IT Skills -> Software Design and Development -> R).
* Massive online courses (MOC's) [Coursera Data Science Specialization by John Hopkins](https://www.coursera.org/specializations/jhu-data-science)
* Swirl (<http://swirlstats.com/>)
  + Interactive learning using R
* R users group (<http://rug.biotech.cdc.gov/presentations.html>)
* Online books (free)
  + R for Data Science by Garrett Grolemund and Hadley Wickham (<http://r4ds.had.co.nz/index.html>)

There are several resources available to help you learn R. At CDc. there are:

* online classes (free) available for learning R at HHS university.
* Books 24/7 through HHS university (has approx. 27 books and 30 short videos). Massive Online Courses
* (MOC's) such as coursera offer good instruction. We already have talked about Coursera.
* Swirl - swirl is an excellent way to provide interactive lessons that teach you R as you use R. (<http://rug.biotech.cdc.gov/slides/12.2_slides.pptx>)
* The RUG also has several of its past presentation materials and video's archived and available for users. There are a number of cheat sheets on the RUG website, such as <http://rug.biotech.cdc.gov/cheatsheets/base-r_baggott_2013-01.pdf> Which will be useful to have if you are learning R.
* Free online resources such as R for Data Science by Garrett Grolemund and Hadley Wickham are excellent intermediate/beginer texts.

# Basic R Syntax

## Data for Today

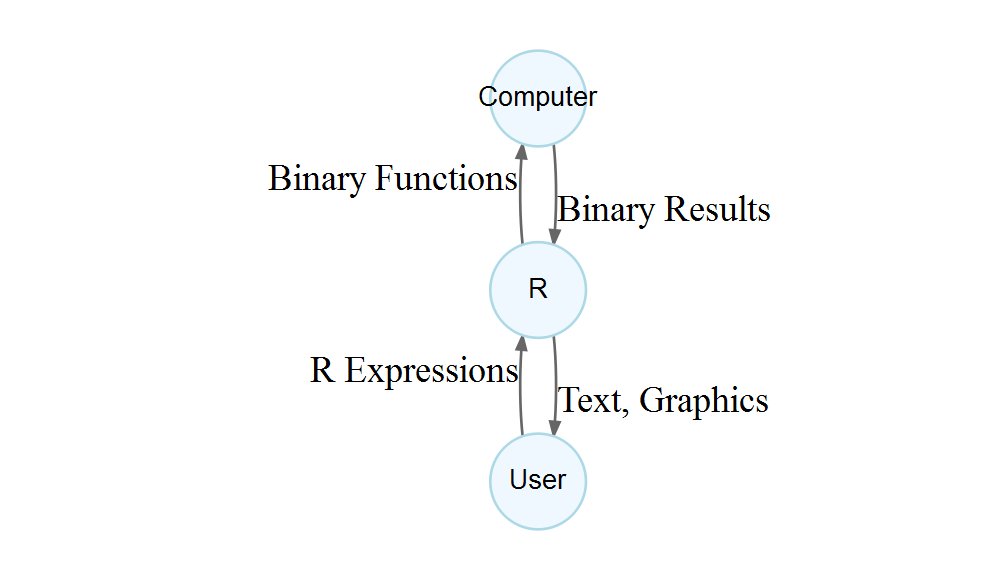
Available at <http://rug.cdc.gov/data/>

* <http://rug.cdc.gov/data/AGE_Surveillance_Log.xls>
* save in ../data/ directory
* also available as a csv file

## The R Console

Interactive Use

* R is an interpreted high level language
* Uses either command line interface or console to communicate with user
* Programs (sometimes called scripts) can either be passed through the console or interpreter.
* Results are returned through the interpreter to the user.



R in interactive use

R is an interpretted high level langauge - which means that it is supposed to be easier for people to understand than a computer. Computers speak "binary" - basically one's and zeros.

There are 10 types of persons in the world, those that speak binary and those that don't.

## Basic Example Operations in Console

* Typing the following after the > and pressing enter will cause R to evaluate the expression.
* R will return either a value or an error. If the statement is incomplete, it will display a continuation prompt (+).
* Any characters on a line with # will be disregarded as a comment.
* Multiple expressions on one line can be seperated by ;

1 + 1 #Add 1 plus 1  
sqrt(5000) # square root 5000  
3/3  
4\*2   
2\*pi  
pi%%pi   
5 %/% 2

## Arthmetic Operators

These operators perform basic arithmatic in R

|  |  |
| --- | --- |
| Operator | Operation |
| + | addition |
| - | subtraction |
|  | multiplication |
| / | division |
| ^ | exponentiation |
|  | exponentiation |
| %% | modulus |
| %/% | integer division |

## Scientific Notation

Scientific Notation consists of: - A floating point number - "e", a + or - and - Number indicating the direction and amount to shift the decimal point - By default, R will convert small scientific numbers to fixed notation, see ?options

1.3e3 ; 1.3e-3; 1e4 ; 1e-4

## [1] 1300

## [1] 0.0013

## [1] 10000

## [1] 1e-04

## Logs, Roots

* To obtain natural log, use log() function
* Inverse of log() is exp()
* log10() is base 10 logarithms, log2() is base 2 logarithms
* other logs can be calculated log() with base argument e.g.

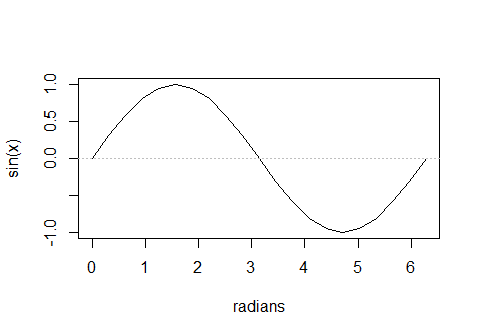
log(9, base = 3)

## [1] 2

## Trigonometry Functions

* sin(), cos(), tan(), asin(), acos(), atan()
* For help on trigonometry functions see ?Trig
* But input is radians!

plot(x = seq(0, 2, 0.1) \* pi, y = sin(seq(0, 2, 0.1) \* pi),   
 type="l", xlab = "radians", ylab = "sin(x)")  
abline(h=0, lty = 3, col = "grey")



# Objects, class and Type

## Assignments

Variables are stored in R by *assigning* them. You can see what variables are in R's memory environment by using the function objects()or ls(). You can use <- or = for assignment.

* Note that R is case sensitive!
* Object names cannot start with a number!
* (Shortcut key: Alt + -) in Rstudio for <-

x = 7 # same as:  
x <- 7   
print(x)

## [1] 7

print(X) # error!

## Error in print(X): object 'X' not found

y <- 10 - 9

The up and down arrow keys can be useful to scroll through your command history. UP will give you the previous commands, and down will move foward in your command history. That way, if you make a typo, you can correct your previous commands easily. You can also use the command history() to see all the past commands you have run.

## Objects and Clearing Environment

To see what objects are stored in memory:

objects()

To clear all objects from memory:

rm(list=ls())

## Logic Examples

To use logical comparisons, use the symbols <, >, >=, <=, ==, !=

x <- 1  
x > 2

## [1] FALSE

x < 0

## [1] FALSE

x >= 4

## [1] FALSE

x <= 5

## [1] TRUE

x == 1

## [1] TRUE

x != 1

## [1] FALSE

x <= 1 & x < 2

## [1] TRUE

x == 1 | x < 0

## [1] TRUE

## Logical Comparisons

|  |  |
| --- | --- |
| Operator | Operation |
| < | less than |
| <= | less than or equal |
| > | greater than |
| >= | greater than or equal |
| == | equal |
| != | not equal |
| ! | NOT |
| | | OR |
| & | AND |
| || | ALL OR |
| && | ALL AND |

## Atomic structures

Objects can have classes. Atomic structures are the lowest level of class.

#Variable classes  
x <- 1.7  
class(x)

## [1] "numeric"

typeof(x)

## [1] "double"

y <- 'Hello World!'  
class(y)

## [1] "character"

z <- TRUE  
class(z)

## [1] "logical"

## Complex and Raw class

In addition to the classes above, there are complex number classes and the raw class. Raw classes are used to hold raw bytes. For the most part, beginers will probably not need these classes, but for advanced programming applications, they can be quite useful.

a\_complex\_number <- 1+0i  
class(a\_complex\_number)

## [1] "complex"

xx <- charToRaw("I am not raw")  
xx

## [1] 49 20 61 6d 20 6e 6f 74 20 72 61 77

class(xx)

## [1] "raw"

rawToChar(xx)

## [1] "I am not raw"

## Precision

* Computers are not perfectly perfect
* Uses IEEE 754 double precision floating-point numbers
* Generally accurate to about 16 significant digits
* The RUG hosted a talk on precision by Seth Simms (<http://rug.biotech.cdc.gov/slides/13_slides.pptx>)

1e-324 == 0 ; 1e-323 == 0

## [1] TRUE

## [1] FALSE

1 - 1e-16 == 1 ;1 - 1e-17 == 1

## [1] FALSE

## [1] TRUE

a <- sqrt(2) ; a^2 == 2

## [1] FALSE

Note that there are some problems using R and any computer for math. For instance: 0.3/3 == 0.1 returns FALSE(!). Some other resources for floating point issues - Floating point arithmatic and issues are tricky to grasp. Chapter 1 of the R Inferno by Burns has a good discussion of the pitfalls <http://www.burns-stat.com/pages/Tutor/R_inferno.pdf> - "Lies my Computer and Calculator Told Me"" by James Stewart <http://www.stewartcalculus.com/data/default/upfiles/LiesCalcAndCompTold.pdf> - see ?.Machine for numerical limits on precision.

## Significant Digits

* You can use the signif() function to set significant digits
* You can use print() with digits = to see more digits
* By default R will print a set number of digits (getOption("digits"))

# Vectors

## Numeric Vectors

* Data structures hold data.
* There are different classes of data structures.
* Simpliest data structure is a vector.
* All items in vector must have same atomic class.

The vector is the building block of more complex structures, so understanding their basic operations is essential.

X <- c(10.4, 5.6, 3.1, 6.4, 21.7) # creates a numeric vector  
1/X

## [1] 0.09615385 0.17857143 0.32258065 0.15625000 0.04608295

Y <- 1/X  
X\*\*2

## [1] 108.16 31.36 9.61 40.96 470.89

summary(Y)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 0.04608 0.09615 0.15625 0.15993 0.17857 0.32258

The c function is concatnate. It combines individual datum into a vector that contains data. Also note that for the mathmatical functions, the operations are performed on each datum. Some functions (like summary) perform an operaton on the entire vector.

## Regular and Random Sequences

* R can be used to create regular sequencies of numbers using seq.
* Random numbers at created using random number generating functions (e.g. rnorm).
* set.seed can be used to ensure reproducability.

#regular sequences  
seq(from = -5, to = 5, by = 2)

## [1] -5 -3 -1 1 3 5

1:15

## [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

## Numeric Summaries

set.seed(123)  
randomNums <- rnorm(1000)  
summary(randomNums); mean(randomNums); sd(randomNums)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## -2.80978 -0.62832 0.00921 0.01613 0.66460 3.24104

## [1] 0.01612787

## [1] 0.991695

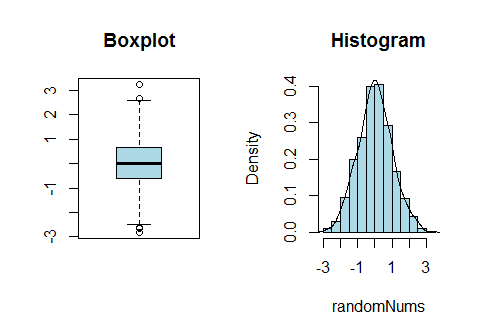
fivenum(randomNums)

## [1] -2.809774679 -0.628742409 0.009209639 0.664787870  
## [5] 3.241039935

You can find out which distributions are supported by typing ?distributions. You can control type of random number generation in R, and set random number generator seed numbers - see ?RNG.

## Numeric Vectors Summary and Visualizations

oldpar <- par(mfrow = c(1,2))  
boxplot(randomNums, main = "Boxplot", col="lightblue")  
hist(randomNums, main = "Histogram", col="lightblue", freq=FALSE)   
lines(density(randomNums), main = "Density")



par(oldpar)

## Character Vectors

Character vectors behave in a different manner than numeric vectors.

Stooges <- c("Moe", "Larry", "Curly")   
lastNames <- c("Howard", "Fine", "Howard")  
1/Stooges # Error

## Error in 1/Stooges: non-numeric argument to binary operator

ThreeStooges <- paste(Stooges, lastNames)   
ThreeStooges # prints

## [1] "Moe Howard" "Larry Fine" "Curly Howard"

plot(ThreeStooges)

## Warning in xy.coords(x, y, xlabel, ylabel, log): NAs  
## introduced by coercion

## Warning in min(x): no non-missing arguments to min;  
## returning Inf

## Warning in max(x): no non-missing arguments to max;  
## returning -Inf

## Error in plot.window(...): need finite 'ylim' values



table(ThreeStooges)

## ThreeStooges  
## Curly Howard Larry Fine Moe Howard   
## 1 1 1

The paste() function pastes the contents two vectors together item-by-item. If the vector lengths are different, R will issue a warning, and "recycle" to the top of the shorter vector.

There is not arithmatic, summary or plot methods for character vectors, but the table() function can provide a count of unquie character strings in the vector.

## Logical Vectors

Logical vectors can be created as well, either directly by assignment or evaluating a vector.

logicVector1 <- c(TRUE, FALSE, FALSE, TRUE)   
logicVector2 <- c(T,F,F,T) #T or F means TRUE or FALSE  
(logicVector3 <- X < 10)

## [1] FALSE TRUE TRUE TRUE FALSE

summary(logicVector3)

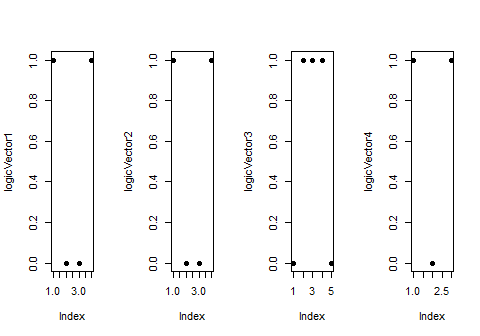
## Mode FALSE TRUE   
## logical 2 3

(logicVector4 <- lastNames == "Howard")

## [1] TRUE FALSE TRUE

## Plotting Logical Vectors

oldpar <- par(mfrow=c(1,4))  
plot(logicVector1, pch = 19)  
plot(logicVector2, pch = 19)  
plot(logicVector3, pch = 19)  
plot(logicVector4, pch = 19)



par(oldpar)

## Subsetting Vectors

Logical vectors are often used to subset other vectors. TRUE values are retained, while FALSE are discarded. Numeric values can also be used to subset vectors.

V <- 1:10  
V[V<5]

## [1] 1 2 3 4

lastNames[1]

## [1] "Howard"

lastNames[c(1,3)]

## [1] "Howard" "Howard"

logicVector1[logicVector1]

## [1] TRUE TRUE

## Factors

Factor variables are used to indicate discrete classification, or grouping.

factorStates <- as.factor(sample(sample(state.abb, 10), 100,   
 replace = TRUE))   
factorStates

## [1] AL WI NV MD GA WY AZ WI MO DE GA MN GA WY WY WI MD NV  
## [19] MN AZ NV NV NV WI WY AZ MD GA GA AZ MN WY MN AZ GA GA  
## [37] AL WY DE MN NV DE GA WY GA WY MD AZ AZ MD WY AL MD AL  
## [55] WY DE MO WI AL GA WY GA MD MN MD AZ NV MO GA DE WY MD  
## [73] AZ WY MN MD MO GA MD MO WY MO DE DE MO AZ GA MN NV DE  
## [91] MD MO MN GA MD AZ WY WI AL AL  
## Levels: AL AZ DE GA MD MN MO NV WI WY

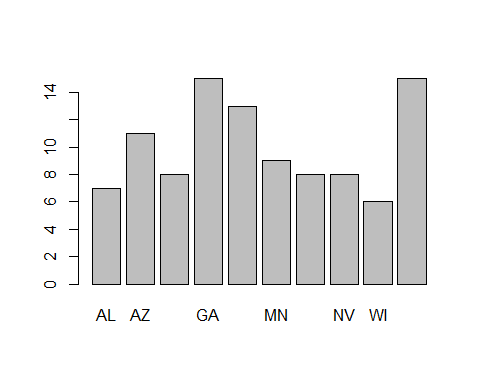
summary(factorStates)

## AL AZ DE GA MD MN MO NV WI WY   
## 7 11 8 15 13 9 8 8 6 15

# table(factorStates)

## Factors Plotting

plot(factorStates)



## Coercion

* Numbers concatnated together will create a numeric vector
* Charcter strings will create a character vector
* Numeric data and character data concatnation will make a character
* conversion of one type of data into another will:
  + result in either NA (character to number - unless the string is a number)
  + result in a factor level number (if factor)

as.numeric(factorStates)

## [1] 1 9 8 5 4 10 2 9 7 3 4 6 4 10 10 9 5 8  
## [19] 6 2 8 8 8 9 10 2 5 4 4 2 6 10 6 2 4 4  
## [37] 1 10 3 6 8 3 4 10 4 10 5 2 2 5 10 1 5 1  
## [55] 10 3 7 9 1 4 10 4 5 6 5 2 8 7 4 3 10 5  
## [73] 2 10 6 5 7 4 5 7 10 7 3 3 7 2 4 6 8 3  
## [91] 5 7 6 4 5 2 10 9 1 1

class(c("One", 2, 3, "four"))

## [1] "character"

(dontdothis <-as.numeric(factor(c(10,20,30))))

## [1] 1 2 3

(lookout <- as.numeric(c("One", 2, 3, "four")))

## Warning: NAs introduced by coercion

## [1] NA 2 3 NA

(dothis <- as.numeric(as.character((factor(c(10,20,30))))))

## [1] 10 20 30

## Dates and Time

* Date class contains dates without time
* POSIXct is dates and times (including timezones). These are stored as a numeric value (seconds since 01/01/1970) and displayed as a string.
* POSIXlt is also dates and times (including timezones) but these are stored in a more complicated list format
* as.Date() creates a date from a string
* strptime() create POSIXlt

## Creating a Date

(adate <- as.Date("11/11/2011", format = "%m/%d/%Y"))

## [1] "2011-11-11"

class(adate)

## [1] "Date"

(bdate <- strptime("11-17-2014 13:30",   
 format = "%m-%d-%Y %H:%M", tz = "US/Eastern"))

## [1] "2014-11-17 13:30:00 EST"

class(bdate) ; bdate$wday; format(bdate, "%a")

## [1] "POSIXlt" "POSIXt"

## [1] 1

## [1] "Mon"

Notes on dates:

* The Date class is actually the number of days since January 1, 1970. It will print accoring to the Gregorian calendar, even on dates when that calandar was not in use!
* The POSIXct is number of seconds since start of January 1, 1970.
* POSIXlt is a list of vectors that contains representing parts of the date (e.g. hour, day of month, month, years since 1990).
* to see your timezone on your machine use: Sys.timezone(location = TRUE)
* for available timezones on your system use: OlsonNames()

Note that except where replaced, the operation of time zones is an OS service, and even where replaced a third-party database is used and can be updated (see the section on â€˜Time zone namesâ€™). Incorrect results will never be an R issue, so please ensure that you have the courtesy not to blame R for them.

## Vector Manipulations

* You can index a vector and use the assignment to change its value:

ThreeStooges[2] <- "James Durant"  
ThreeStooges

## [1] "Moe Howard" "James Durant" "Curly Howard"

X[X<5] <- 0  
X

## [1] 10.4 5.6 0.0 6.4 21.7

X[X<5] <- rnorm(100) #warning

## Warning in X[X < 5] <- rnorm(100): number of items to  
## replace is not a multiple of replacement length

X[X>50] <- rnorm(100)

Note the replacement vector must be either 1 - which replaces all values, or of equal length to the replacement. If the index results in a zero length vector, R will not substitute the values (and not issue any warning either!)

## Missing Values

* Missing, not available or unknown values are coded with NA.
* is.na(x) gives logical vector of length x with value of TRUE for NA values in vector x.

x <- c(1:3,NA)  
is.na(x)

## [1] FALSE FALSE FALSE TRUE

x == NA

## [1] NA NA NA NA

=

## Setting vector elements to NA

If you have a missing code in your data (e.g. -999), then you will need to set that value to NA. To do that, you must pass is.na a logical vector.

mydata <- c( 1, 2.4, pi, 40, -999)  
mean(mydata)

## [1] -190.4917

is.na(mydata) <- mydata == -999  
mean(mydata)

## [1] NA

mean(mydata, na.rm=TRUE)

## [1] 11.6354

R by default will assume you do *NOT* want to drop NA values.

## Not a Number Values

* Results of math computations that are not nubmers are coded as NaN.
* is.nan(x) gives logical vector of length x with value of TRUE for NaN values.

x[5] <- 0/0  
x[6] <- Inf - Inf  
is.na(x)

## [1] FALSE FALSE FALSE TRUE TRUE TRUE

is.nan(x)

## [1] FALSE FALSE FALSE FALSE TRUE TRUE

## Vector Calculations and Concatnation

* You can calculate values using numeric vectors:

X1 <- c(1,2,3,4,5)  
X2 <-c(1,2,3)  
X1 + X2

## Warning in X1 + X2: longer object length is not a multiple  
## of shorter object length

## [1] 2 4 6 5 7

X2 <- c(X2, 4, 5)  
X1 + X2

## [1] 2 4 6 8 10

Note the fact that when vector lengths are not the same, R will recycle the shorter vector. It will also issue a warning if the longer vector is not divisible by the shorter vector.

# Regular Expressions

## XKCD



xkcd

(<https://xkcd.com/208/>)

* These are used in commands such as grep, grepl, gsub etc. and are very powerful tools for the programmer. They are common in many languages, including R.

## What are Regular Expressions?

Represent language using:

* literals: are the words/text matched literally.
* matacharacters: "special meaning" characters. Together they find classes of words or patterns in strings.

## Regular Expression Functions

grep , grepl, sub and gsub

* grep searches and returns matching string or index in vector
* grepl searches and returns a vector of TRUE/FALSE is matches occur
* sub substitutes first match for the second argument
* gsub substitutes all matches for the second argument
* regexpr and gregexpr seaches character vector and returns index and length of match

## Regular Expressions - More Training

documentation is challenging - 2 good videos to watch

* <https://www.youtube.com/watch?v=NvHjYOilOf8>
* <https://www.youtube.com/watch?v=q8SzNKib5-4>

## Regular Expression Examples

grep("A", state.name)

## [1] 1 2 3 4

grep("A", state.name, value=TRUE)

## [1] "Alabama" "Alaska" "Arizona" "Arkansas"

grep("A", state.name, ignore.case = TRUE)

## [1] 1 2 3 4 5 6 8 9 10 11 12 14 15 16 18 19 20 21  
## [19] 22 23 26 27 28 29 33 34 36 38 39 40 41 43 44 46 47 48

grep("A", state.name, ignore.case = TRUE,   
 invert = TRUE, value = TRUE )

## [1] "Connecticut" "Illinois" "Kentucky" "Mississippi"  
## [5] "Missouri" "New Jersey" "New Mexico" "New York"   
## [9] "Ohio" "Oregon" "Tennessee" "Vermont"   
## [13] "Wisconsin" "Wyoming"

grep("^A", state.name, value=TRUE)

## [1] "Alabama" "Alaska" "Arizona" "Arkansas"

grep("a$", state.name, value=TRUE)

## [1] "Alabama" "Alaska" "Arizona"   
## [4] "California" "Florida" "Georgia"   
## [7] "Indiana" "Iowa" "Louisiana"   
## [10] "Minnesota" "Montana" "Nebraska"   
## [13] "Nevada" "North Carolina" "North Dakota"   
## [16] "Oklahoma" "Pennsylvania" "South Carolina"  
## [19] "South Dakota" "Virginia" "West Virginia"

grep("^A|^V", state.name, value = TRUE)

## [1] "Alabama" "Alaska" "Arizona" "Arkansas" "Vermont"   
## [6] "Virginia"

(ThreeStooges <- gsub("James", "Jim", ThreeStooges))

## [1] "Moe Howard" "Jim Durant" "Curly Howard"

## Regular Expression Examples

grep("^A.+b", state.name, value = TRUE)

## [1] "Alabama"

grep("9.\*11", c("911", "9/11", "9/123116/1212"))

## [1] 1 2 3

grep("9.\*11$", c("911", "9/11", "9/123116/1212"))

## [1] 1 2

# Application

## Importing a data.frame

* Data.frame is just a set of vectors arranged in columns
* Line listing of cases of acute gastroenteritis (AGE) on cruise
* You can import using from Excel "Import Dataset" wizard in Rstudio

library(readxl)  
 Age\_sl <- read\_excel(  
 "../data/AGE\_Surveillance\_Log.xls",   
 skip = 7)  
names(Age\_sl)[  
 grepl(  
 "^X\_\_[0-9]|Y/N\_\_[0-9]|#|^.F$|^Y/N$",names(Age\_sl))  
 ] <-  
 c(  
 "Age", "Sex", "Pax\_or\_Crew", "Cabin", "Meal\_seat",  
 "Crew\_position", "Diarrhea\_n", "Vomiting\_n",  
 "Fever", "Temp", "Cramps", "Headache", "Myalgia",  
 "Spec\_req", "Spec\_rec", "AD\_meds", "Reportable",  
 "UI"  
 )

## Vectors in Data.Frames

* Vectors within data.frame objects can be addressed with a $.

summary(Age\_sl$Age)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 21.00 54.00 65.00 58.77 70.25 83.00

table(Age\_sl$Sex)

##   
## F M   
## 28 32

## Application Activities

* Using R, create a new vector Age\_sl$adj.temp that changes any aberent temperatures in cases to missing.
* Let's assume the aberant temperatures were recorded in centigrade, convert only those values in the Age\_sl$Temp to Fahrenheit scale. .
* Standardize the Age\_sl$Meal\_seat vector.
* Make a boxplot of ages of only male non-crew cases.
* Create Age\_sl$dt\_ill which captures the date-time of illness onset.

1. ATSDR Division of Community Investigations [↑](#footnote-ref-1)