DATA 106 - Notes 1

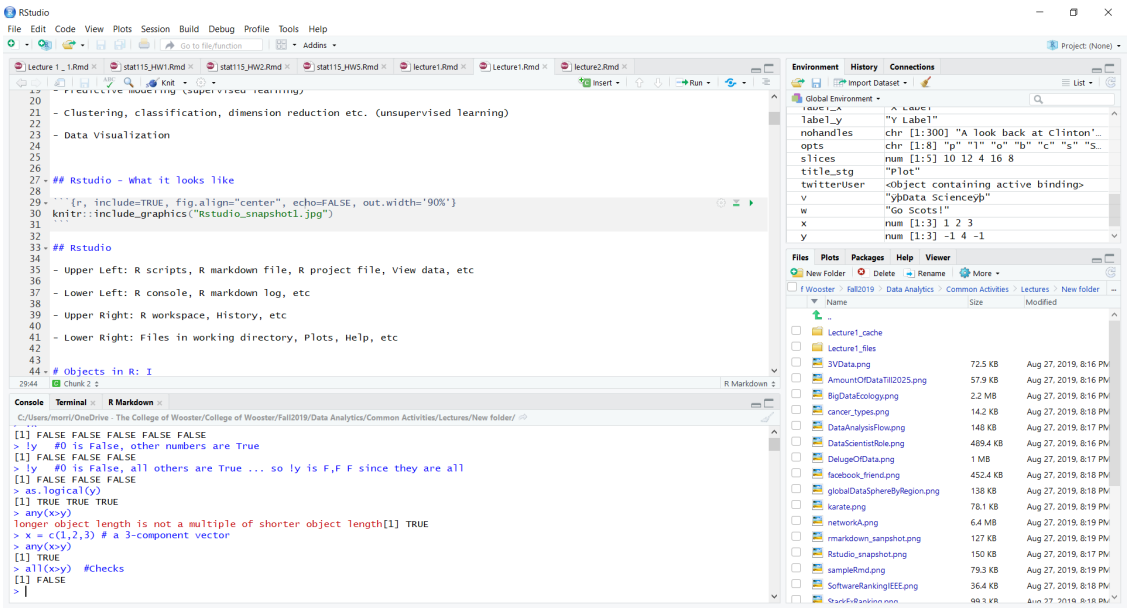
Jillian Morrison

August 28, 2019

Objectives for the course using R:

- Exploratory data analysis
- Predictive modeling (supervised learning)
- Clustering, classification, dimension reduction etc. (unsupervised learning)
- Data Visualization

Rstudio - What it looks like



Rstudio

- Upper Left: R scripts, R markdown file, R project file, View data, etc
- Lower Left: R console, R markdown log, etc
- Upper Right: R workspace, History, etc
- Lower Right: Files in working directory, Plots, Help, etc

Objects in R: I

Scalars in R

```
> x = 5 # assign value 3 to variable x
> y = 6
> x+y # addition
[1] 11
> x*y  # multiplication
[1] 30
> x/y # division
[1] 0.8333333
> x%%y # modulo
[1] 5
> x^y # exponentiation
[1] 15625
> x/0
[1] Inf
> 0/0 # undefifed
[1] NaN
```

Vectors I

```
> z = c(2,4,6) # a vector of 3 components
> v = c(8,10,12)
> z+v # vector addition
[1] 10 14 18
> z*v  # paired componentwise product
[1] 16 40 72
> z/v # paired componentwise division
[1] 0.25 0.40 0.50
> z%*%v # inner product
     [,1]
[1,]  128
> 2*z # scalar-vector multipication
[1]  4  8 12
```

Vectors II

```
> z = c(2,4,6)
> v = c(8,10,12)
> z[1] # access the 1st component of z
[1] 2
> t(v) # transpose of vector
     [,1] [,2] [,3]
[1,]    8   10   12
> z%*%t(v) # outer product
     [,1] [,2] [,3]
```

```
[1,]   16   20   24
[2,]   32   40   48
[3,]   48   60   72
```

Creating a sequence

```
> seq(from = 0, to = 1, by = ((to - from)/(length.out - 1)),
+     length.out = NULL, along.with = NULL, ...)
```

Usage:

```
> seq(0,1,by=0.1)
 [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

Matrices I - combining matrices

```
> matrix(1:6,nrow=2,ncol=3) # a 2-by-3 matrix
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> x = c(1,3,5) # a 3-component vector
> y = c(2,4,6)  #a 3-component vector
> # stack x and y as 2 rows to obtain a 2-by-3 matrix
> rbind(x,y)
  [,1] [,2] [,3]
x    1    3    5
y    2    4    6
> # stack x and y as 2 columns to obtain a 3-by-2 matrix
> cbind(x,y)
     x y
[1,] 1 2
[2,] 3 4
[3,] 5 6
```

Matrices II - retreiving elements and transpose

```
> x=matrix(1:6,nrow=2,ncol=3) #  2-by-3 matrix
> x
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> x[,1] # 1st column of x
[1] 1 2
> x[2,] # 2nd row of x
[1] 2 4 6
> x[1,2] # (1,2)-entry of x
[1] 3
```

```
> t(x) # transpose of x
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
```

Matrices III - creating/combining and multiplying matrices

```
> x=matrix(1:6,nrow=2,ncol=3) #  2-by-3 matrix
> x
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> y = rbind(c(0,1,0),c(1,1,1))
> y
     [,1] [,2] [,3]
[1,]    0    1    0
[2,]    1    1    1
> x %*%t(y) # matrix multiplication
     [,1] [,2]
[1,]    3    9
[2,]    4   12
```

Matrices IV - Operations

```
> x=matrix(1:6,nrow=2,ncol=3) # a 2-by-3 matrix
> x
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> y = rbind(c(0,1,0),c(1,1,1))
> y
     [,1] [,2] [,3]
[1,]    0    1    0
[2,]    1    1    1
> x + y # matrix addition
     [,1] [,2] [,3]
[1,]    1    4    5
[2,]    3    5    7
> 2*x # scalar multiplication
     [,1] [,2] [,3]
[1,]    2    6   10
[2,]    4    8   12
```

Data frames I - Checking objects and retrieving elements

```
> x <- data.frame("ID" = 1:2, "Age" = c(21,15),
+                 "Name" = c("Jacob","Maya"))
> x
  ID Age  Name
1  1  21 Jacob
2  2  15  Maya
> x$ID #access ID
[1] 1 2
> x[,1] # access ID
[1] 1 2
> class(x$ID) # check object type for ID
[1] "integer"
> class(x$Name) # check object type for Name
[1] "factor"
```

Data frames II - More access

```
> x <- data.frame("ID" = 1:2, "Age" = c(21,15),
+                 "Name" = c("Jacob","Maya"))
> x
  ID Age  Name
1  1  21 Jacob
2  2  15  Maya
> x$ID[2] #access the 2nd entry of ID
[1] 2
> x[1,2] #access the 1st entry of Age
[1] 21
```

Caution: do not transpose a data.frame when it contains different types of objects

Data frames III - Importing dataframe

Import (malaria related death) data as data.frame:

```
> setwd("C:/Users/morri/OneDrive - The College of Wooster/College of Wooster/Fall2019/Data Analytics/Co
>
> Y = read.csv("dataMalyria.csv",header = TRUE,sep=",",
+              colClasses=c("country"=NA,"percent"="numeric",
+                           "labels"=NA))
> head(Y)  ##Retrieves the first 6 rows of dataframe
     country percent labels
1    Lesotho       0   <1%
2  Mauritius       0   <1%
3 Seychelles       0   <1%
4 Cabo Verde       0   <1%
5    Algeria       0   <1%
6      Egypt       0   <1%
```

Data frames IV - Looking more at the dataset

Import (malaria related death) data as data.frame:

```
> str(Y) # object structure of Y
'data.frame':   53 obs. of  3 variables:
 $ country: Factor w/ 53 levels "Algeria","Angola",..: 25 32 41 7 1 15 27 33 50 47 ...
 $ percent: num  0 0 0 0 0 0 0 0 0 0 ...
 $ labels : Factor w/ 5 levels " <1% "," 1-4% ",..: 1 1 1 1 1 1 1 1 1 1 ...
> dim(Y) # dimension of Y
[1] 53  3
> Y$id = 1:53 # append a column called "id" to Y. This column has a sequence from 1:53
> Y[1:3,] # display the first 3 rows of Y
     country percent labels id
1    Lesotho       0   <1%   1
2  Mauritius       0   <1%   2
3 Seychelles       0   <1%   3
```

Objects II - Character Vectors

Character vectors

```
> w = c("a","b","c") # a vector of 3 character components
> w[2] # access the 2nd component
[1] "b"
> # 1st 10 upper case letters in the alphabet
> LETTERS[seq( from = 1, to = 10 )]
 [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J"
> # 1st 10 lower case letters in the alphabet
> letters[seq( from = 1, to = 10 )]
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"
> Q = c("Go","COW","Scots","!")
> Q
[1] "Go"    "COW"   "Scots" "!"
> # concatenate two character vectors
> c(w,Q)
[1] "a"     "b"     "c"     "Go"    "COW"   "Scots" "!"
```

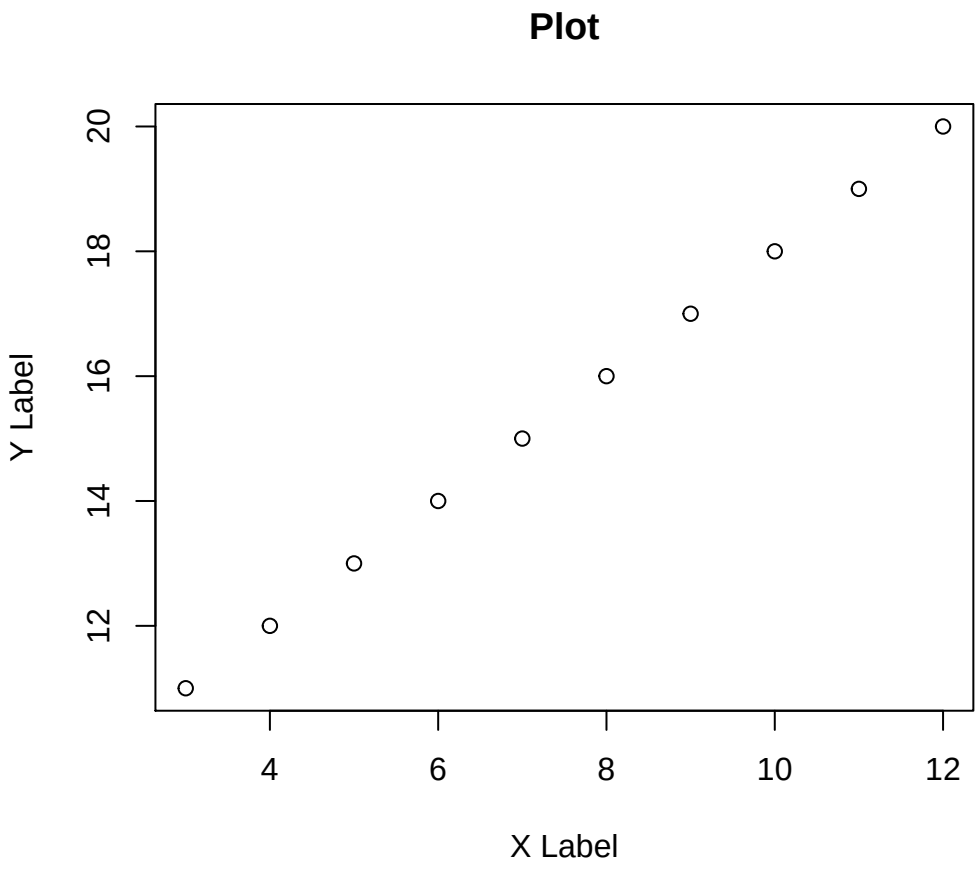
Strings

```
> w = "Go Scots!"
> w
[1] "Go Scots!"
>
> v = "Data Science"
> v
[1] "Data Science"
>
> # concatenate two strings
```

```
> paste(w,v,sep = " ") #Join w and v with a space between
[1] "Go Scots! Data Science"
```

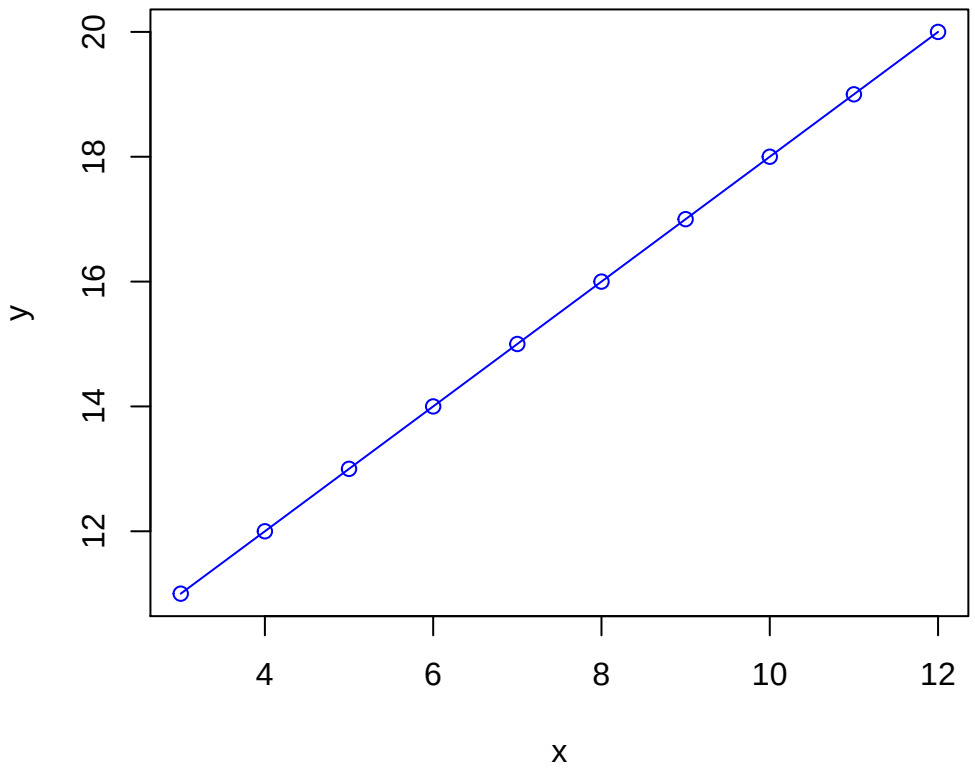
Example: create a scatter plot

```
> x = seq(3,12,by=1) # generate vector
> y = seq(11,20,by=1) # generate vector
> title_stg = "Plot" # generate string
> label_x="X Label"
> label_y="Y Label"
> plot(x,y,main = title_stg, xlab = label_x, ylab=label_y) # scatter plot
```



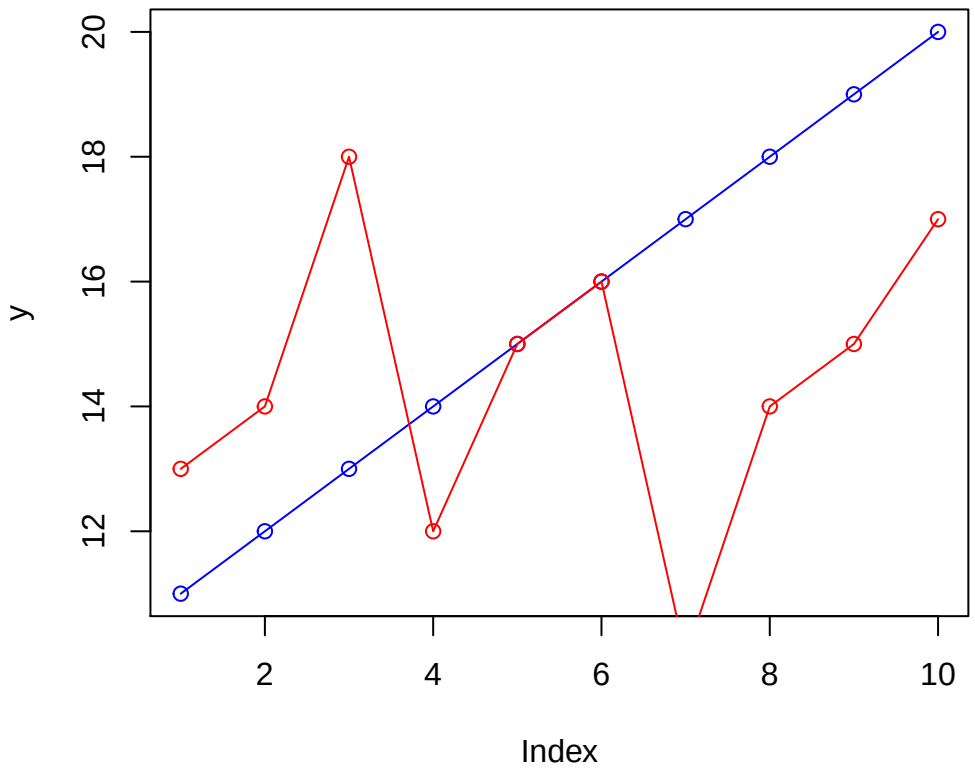
More plot

```
> x = seq(3,12,by=1) # generate vector
> y = seq(11,20,by=1) # generate vector
> plot(x,y, col="blue", type="o")  #Adding color and lines
```



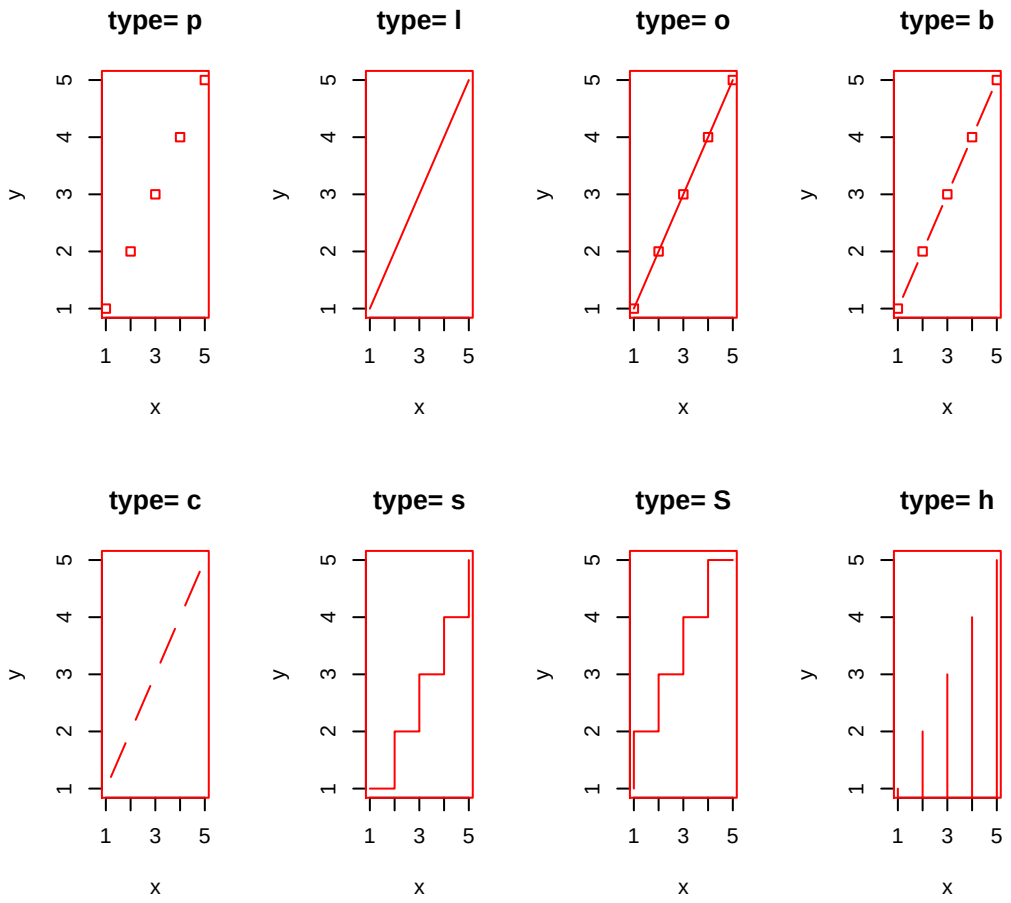
More plots... Adding multiple lines to a plot

```
> y = seq(11,20,by=1) # generate vector
> z = c(13,14,18,12,15,16,10,14,15,17)  #generate another vector
> plot(y, col="blue", type="o")  #Adding color and li
> lines(z, col="red",type="o")
```



More plots... different types (this uses a loop...)

```
> x <- c(1:5); y <- x # create some data
> par(pch=22, col="red") # plotting symbol and color
> par(mfrow=c(2,4)) # all plots on one page
> options = c("p","l","o","b","c","s","S","h")
> for(i in 1:length(options)){
+   heading = paste("type=",options[i])
+   plot(x, y, type="n", main=heading)
+   lines(x, y, type=options[i])
+ }
```



Factors I - Creating factor vectors

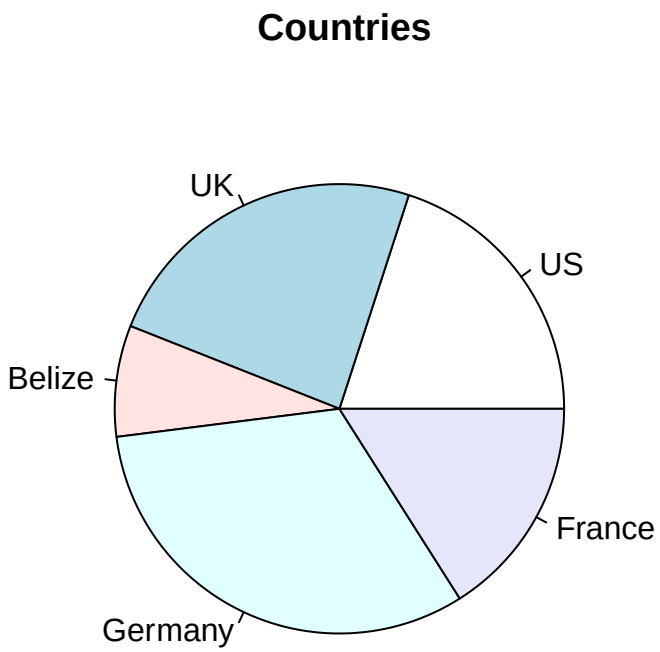
```
> grades = c("A","F","D","C","B") # character vector
> grades
[1] "A" "F" "D" "C" "B"
> class(grades) ##Checks the class - character..
[1] "character"
> gradesF = factor(grades) # gradesF is a now factor
> gradesF
[1] A F D C B
Levels: A B C D F
>
> class(gradesF) #Checks the class - now it is a factor
[1] "factor"
>
>
> levels(gradesF)   # levels of the factor "gradesF" and lists them alphabetically
[1] "A" "B" "C" "D" "F"
> unique(gradesF)   # levels of the factor "gradesF" and lists them alphabetically
[1] A F D C B
Levels: A B C D F
```

Factors II - More factor stuff

```
> x = c(1,3,2) # numeric vector
> b = factor(x) # change x into a factor
> b
[1] 1 3 2
Levels: 1 2 3
> levels(b) # levels are ordered from smallest to largest
[1] "1" "2" "3"
>
> d = factor(x,labels = c("3Level","1Level","2Level")) # relabel levels of b
> d
[1] 3Level 2Level 1Level
Levels: 3Level 1Level 2Level
```

Pie Chart

```
> # Simple Pie Chart
> slices <- c(10, 12,4, 16, 8)  ##Creating a vector
> lab <- c("US", "UK", "Belize", "Germany", "France")
> pie(slices, labels = lab, main=" Countries")
```



Logic operators I

```
> x = 0 # assign 0 to x
> x >0   #Checks if x>0
[1] FALSE
> x == 0 #Checks if x=0
[1] TRUE
>
> y = 1 #Assign 1 to y
> y >= 1 #Checks if y>=1
[1] TRUE
>
> x & y # "and"; return FALSE
[1] FALSE
> x | y # "or"; return TRUE
[1] TRUE
```

Logic operators II

```
> x = 1
> y = -1
> x >0 & y > 0 # "and"
[1] FALSE
> x > 0 | y > 0 # "or"
[1] TRUE
> x >0 & !(y>0)
[1] TRUE
```

Logic operators III

```
> x = c(1,2,3,4) # a vector with 4 elements
> x >0 # returns a logic vector with 4 elements
[1] TRUE TRUE TRUE TRUE
> x > 2 # returns a logic vector with 4 elements ... again
[1] FALSE FALSE  TRUE  TRUE
>
> which(x>2) # return indices of entries of x that are greater than 2
[1] 3 4
>
> x[x >=2] # take the subvector of x whose entries not smaller than 2
[1] 2 3 4
```

Logic operators IV

```
> x = c(1,2,3) # a 3-component vector
> y = c(-1,4,-1) # a 3-component vector
> # compare x and y entrywise; return a 3-component vector
> x > y
[1]  TRUE FALSE  TRUE
> as.logical(y) #0 is False, all others are True
[1] TRUE TRUE TRUE
> !y
[1] FALSE FALSE FALSE
> !x
[1] FALSE FALSE FALSE
> x == y  #Checks equality elementwise
[1] FALSE FALSE FALSE
> x >= y  #Checks relationship elementwise
[1]  TRUE FALSE  TRUE
> any(x>y) #Checks if there are any x>y
[1] TRUE
> all(x>y)  #Checks if all x>y
[1] FALSE
```

Set operations I - Checking for specific elements

```
> x = c(1,2,3) # a vector with 3 elements
> 1 %in% x  # check if 1 belongs to x
[1] TRUE
> c(2,3) %in% x  #Check if the vector of 2,3 is in x
[1] TRUE TRUE
>
> y = c("DATA","106","Notes")
> "DATA" %in% y  #Checks if DATA belongs to y
[1] TRUE
> "data" %in% y  #data does not belong to y... CASE MATTERS!
[1] FALSE
```

Set operations II

```
> x = c(2,4,6) # a vector with 3 elements
> y = c(-1,4,-1) # another vector with 3 elements
>
> union(x, y) #Union of 2 sets - not repeating elements
[1]  2  4  6 -1
> intersect(x, y)  #Intersection of 2 sets... not repeating elements also
[1] 4
> setdiff(x, y) ##elements of x that are not in y
[1] 2 6
```

Lists: I - assigning entries to a list

```
> x = vector("list",3) # a list with 3 entries
>
> x[[1]] = c(1,2,3) # assign a vector to its first entry
>
> x[[2]] = "I love Data Science!!!!"    # assign a string to its second entry
>
> x[[3]] = matrix(1:6,nrow=3)  # assign a matrix to its third entry
> x
[[1]]
[1] 1 2 3

[[2]]
[1] "I love Data Science!!!!"

[[3]]
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
```

Lists II - Retrieving elements of a list

```
> x = vector("list",3) # a list with 3 entries ... another way to make one
> x[[1]] = c(1,2,3)
> x[[2]] = "Wohoo!!! Data 106"
> x[[3]] = matrix(1:6,nrow=3)
>
> x[[2]] # retrieve second component of the list x
[1] "Wohoo!!! Data 106"
```

Lists III - Another way to assign entries to a list but also assigning objects

```
> a = c(1,2,3)
> b = "YAY... I really enjoy this class!"
> c = matrix(1:6,nrow=3)
> y = list("vector" = a, "string" = b, "matrix" = c)
> y
$vector
[1] 1 2 3

$string
[1] "YAY... I really enjoy this class!"

$matrix
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
```

"Coerce"

- `as.numeric` coerces an object to be numeric
- `as.factor` coerces an object to be a factor
- `as.marix` coerces an object to be a matrix
- `as.logical` coerces an object to be logical
- `as.data.frame` coerces an object to be a data frame
- `as.integer` coerces an object to be an integer
- so on ...

Length

```
> a = 0:10
> length(a) #returns the number of components of a vector
[1] 11
```

Dimensions

```
> x=dim(matrix(1:6,nrow=3,ncol=2)) #returns the dimension of matrix or data frame
> x
[1] 3 2
> x[1] #Returns the 3rd entry
[1] 3
```

Adapted from License