Data Science Certificate Course

Connor Bruce

## Course 1: R Basics

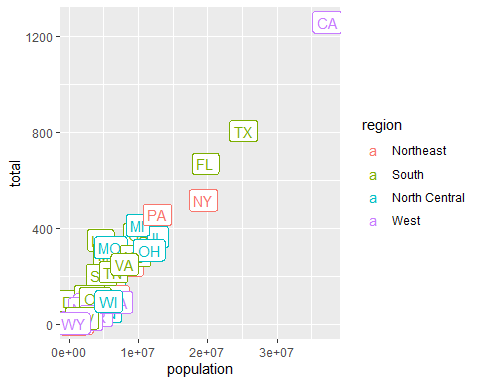
### Section 1: R Basics, Functions, and Data Types

#### 1.1 Motivation and Getting Started

* This course will focus on US crime statistics as the Case Study.
* install.packages("\_\_\_") to install packages (must be connected to internet)
* library(\_\_\_) to load package into script after it has been installed (only need to install once)

Here is an example of loading packages and using them to create a graph

library(tidyverse)  
library(dslabs)  
  
data(murders)  
  
murders %>%  
 ggplot(aes(population, total, label = abb, color =region)) +  
 geom\_label()



#### 1.2 R Basics

##### Objects

* Objects can be variables, or other, more complicated things, such as functions
* ls() outputs a list of objects. Alternatively, it can been in the Environment tab at the top right.

Objects example: Variables being used to solve quadratic formula

a = 1  
b = 1  
c = -1  
  
(-b + sqrt(b^2-4\*a\*c))/(2\*a)

## [1] 0.618034

(-b - sqrt(b^2-4\*a\*c))/(2\*a)

## [1] -1.618034

##### Functions

* To see the code behind a function, type the function without the parenthesis into the console
* Nested functions: Call a function to get an argument for another function. Example:

log(exp(1))

## [1] 1

* To get help with a function, can use the help() function using the function as the argument.

#help(log)

* Or you can check the arguments:

args(log)

## function (x, base = exp(1))   
## NULL

#### 1.3 Data Types

##### Data Frames

* class() can be used to determine the type of data.
* str() can show the structure of the data frame
* head() shows the first 6 rows
* Can use $ as a variable accessor. Will return a vector with the column of the variables

class(murders)

## [1] "data.frame"

str(murders)

## 'data.frame': 51 obs. of 5 variables:  
## $ state : chr "Alabama" "Alaska" "Arizona" "Arkansas" ...  
## $ abb : chr "AL" "AK" "AZ" "AR" ...  
## $ region : Factor w/ 4 levels "Northeast","South",..: 2 4 4 2 4 4 1 2 2 2 ...  
## $ population: num 4779736 710231 6392017 2915918 37253956 ...  
## $ total : num 135 19 232 93 1257 ...

head(murders)

## state abb region population total  
## 1 Alabama AL South 4779736 135  
## 2 Alaska AK West 710231 19  
## 3 Arizona AZ West 6392017 232  
## 4 Arkansas AR South 2915918 93  
## 5 California CA West 37253956 1257  
## 6 Colorado CO West 5029196 65

pop = murders$population  
length(pop)

## [1] 51

##### Other Data Types

* Character Strings

class(murders$state)

## [1] "character"

* Logical Variables (True/False)

z = 3 == 2  
z

## [1] FALSE

class(z)

## [1] "logical"

##### Factors

* Good for storing categorical data (more memory efficient) - numbers associated with a character string
* Recommended to avoid
* table() can be used to show how many are in each category

levels(murders$region)

## [1] "Northeast" "South" "North Central" "West"

table(murders$region)

##   
## Northeast South North Central West   
## 9 17 12 13

### Section 2: Vectors, Sorting

#### 2.1 Vectors

* Create vectors by using the concatenate function, c()
* Can assign names to the numeric values in a vector
* Two ways to assign names are shown below

codes = c(380,124,818)  
country = c("italy", "canada","egypt")  
names(codes) = country  
codes2 = c(italy = 380, canada = 124, egypt = 818)  
codes

## italy canada egypt   
## 380 124 818

codes2

## italy canada egypt   
## 380 124 818

* seq(a,b,c) provides a vector from a to b by jumps of c
* Can also do a sequence using :

seq(1,10)

## [1] 1 2 3 4 5 6 7 8 9 10

1:10

## [1] 1 2 3 4 5 6 7 8 9 10

seq(1,100,10)

## [1] 1 11 21 31 41 51 61 71 81 91

##### Subsetting

* [] are used to access elements of a vector
* Indexes at 1
* Below are examples of pulling different elements from the codes vector

codes[2]

## canada   
## 124

codes[c(1,3)]

## italy egypt   
## 380 818

codes["canada"]

## canada   
## 124

codes[c("egypt","italy")]

## egypt italy   
## 818 380

##### Vector Coercion

* Coercion is an attempt by r to be flexible with data types.
* The below code should throw an error because all elements of a vector must be the same type, however R coerces 1 and 3 into character variables in order to avoid and error

x = c(1,"canada",3)  
x

## [1] "1" "canada" "3"

* Variables can be converted into different types

x = 1:5  
x

## [1] 1 2 3 4 5

y = as.character(x)  
y

## [1] "1" "2" "3" "4" "5"

as.numeric(y)

## [1] 1 2 3 4 5

* Can only coerce in certain ways and if impossible, will replace with NA

x = c("1", "b", "2")  
as.numeric(x)

## Warning: NAs introduced by coercion

## [1] 1 NA 2

#### 2.2 Sorting

* sort() sorting into ascending order
* order() provides the index of the values in ascending order
* Can use the index to extract the necessary information

sort(murders$total)

## [1] 2 4 5 5 7 8 11 12 12 16 19 21 22 27 32  
## [16] 36 38 53 63 65 67 84 93 93 97 97 99 111 116 118  
## [31] 120 135 142 207 219 232 246 250 286 293 310 321 351 364 376  
## [46] 413 457 517 669 805 1257

order(murders$total)

## [1] 46 35 30 51 12 42 20 13 27 40 2 16 45 49 28 38 8 24 17 6 32 29 4 48 7  
## [26] 50 9 37 18 22 25 1 15 41 43 3 31 47 34 21 36 26 19 14 11 23 39 33 10 44  
## [51] 5

index = order(murders$total)  
murders$abb[index]

## [1] "VT" "ND" "NH" "WY" "HI" "SD" "ME" "ID" "MT" "RI" "AK" "IA" "UT" "WV" "NE"  
## [16] "OR" "DE" "MN" "KS" "CO" "NM" "NV" "AR" "WA" "CT" "WI" "DC" "OK" "KY" "MA"  
## [31] "MS" "AL" "IN" "SC" "TN" "AZ" "NJ" "VA" "NC" "MD" "OH" "MO" "LA" "IL" "GA"  
## [46] "MI" "PA" "NY" "FL" "TX" "CA"

* max() gives the maximum value in a vector, which.max() give the index of that value

max(murders$total)

## [1] 1257

which.max(murders$total)

## [1] 5

i\_max = which.max(murders$total)  
murders$state[i\_max]

## [1] "California"

* rank() gives the “rank” of each value in the vector (1 being the smallest value) (also works with min)

x = c(31,4,15,92,65)  
rank(x)

## [1] 3 1 2 5 4

* Here is an overall comparison of sort(), order(), and rank()

x = c(31,4,15,92,65)  
sortx = sort(x)  
orderx = order(x)  
rankx = rank(x)  
sorderank = matrix(c(x,sortx,orderx,rankx), ncol = 4)  
colnames(sorderank) = c("original", "sort", "order", "rank")  
rownames(sorderank) = c(" "," "," "," "," ")  
sorderank

## original sort order rank  
## 31 4 2 3  
## 4 15 3 1  
## 15 31 1 2  
## 92 65 5 5  
## 65 92 4 4

* Can use is.na() to find which entries in a vector are NA (returns as a vector of logical variables)
* True = 1, False = 0
* Can also use [!\_\_\_] to take everything from a vector that is not something.

#### 2.3 Vector Arithmetic

* We can see that California has the most murders and can also see that California also has the largest population

murders$state[which.max(murders$population)]

## [1] "California"

* Therefore, we should be looking at the murders per capita
* It is possible to scale a vector. e.g vector to inches to centimeters

height = c(12,23,15,32,42,26,51,10)  
height \* 2.54

## [1] 30.48 58.42 38.10 81.28 106.68 66.04 129.54 25.40

* Now we can do this with the murders vector to get a rate of murders per 100,000 and then order those to find which has the highest rate and the lowest rate. It can be seen that, while California has the most murders, it ranks 14th in terms of murder rate.

murder\_rate = murders$total/murders$population\*100000  
murders$state[order(murder\_rate, decreasing = T)]

## [1] "District of Columbia" "Louisiana" "Missouri"   
## [4] "Maryland" "South Carolina" "Delaware"   
## [7] "Michigan" "Mississippi" "Georgia"   
## [10] "Arizona" "Pennsylvania" "Tennessee"   
## [13] "Florida" "California" "New Mexico"   
## [16] "Texas" "Arkansas" "Virginia"   
## [19] "Nevada" "North Carolina" "Oklahoma"   
## [22] "Illinois" "Alabama" "New Jersey"   
## [25] "Connecticut" "Ohio" "Alaska"   
## [28] "Kentucky" "New York" "Kansas"   
## [31] "Indiana" "Massachusetts" "Nebraska"   
## [34] "Wisconsin" "Rhode Island" "West Virginia"   
## [37] "Washington" "Colorado" "Montana"   
## [40] "Minnesota" "South Dakota" "Oregon"   
## [43] "Wyoming" "Maine" "Utah"   
## [46] "Idaho" "Iowa" "North Dakota"   
## [49] "Hawaii" "New Hampshire" "Vermont"

### Section 3: Indexing, Data Wrangling, Plots

#### 3.1 Indexing

* We can subset a vector based on properties of another vector.
* We can use logical operators to index vectors.
* Suppose we want to figure out which states have a murder rate of <= .71 per 100,000:

index = murder\_rate <= .71  
index

## [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE  
## [13] FALSE FALSE FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
## [25] FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE FALSE FALSE TRUE FALSE  
## [37] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE  
## [49] FALSE FALSE FALSE

murders$state[index]

## [1] "Hawaii" "Iowa" "New Hampshire" "North Dakota"   
## [5] "Vermont"

sum(index)

## [1] 5

* Now supposed we want to move to a state in the Western region that has a murder rate that is <= 1.

west = murders$region == "West"  
safe = murder\_rate <= 1  
index = safe & west  
murders$state[index]

## [1] "Hawaii" "Idaho" "Oregon" "Utah" "Wyoming"

##### Indexing Functions

* which() gives entries of a logical vector that are true
* match() looks for entries in a vector and returns the index needed to access them.
* %in% determines whether each element of a first vector is in a second vector.

x = c(FALSE, TRUE, FALSE, TRUE, TRUE, FALSE)  
which(x)

## [1] 2 4 5

index = match(c("New York","Florida","Texas"), murders$state)  
index

## [1] 33 10 44

murders$state[index]

## [1] "New York" "Florida" "Texas"

x = c("a","b","c","d","e")  
y = c("a","d","f")  
y %in% x

## [1] TRUE TRUE FALSE

c("Boston", "Dakota", "Washington") %in% murders$state

## [1] FALSE FALSE TRUE

#### 3.2: Data Wrangling

First, the dplyr package must be installed and loaded in

library(dplyr)

dplyr Functions:

* mutate() can be used to change a data table by adding a new column or changing an existing one. Works in the dataframe, not the workspace
* filter() can be used to filter the data by subsetting rows.
* select() can be used to subset data by selecting specific columns.

Using this, we can now edit our murders data frame to add the information we want to add:

murders = mutate(murders, rate = total/population\*100000)  
head(murders)

## state abb region population total rate  
## 1 Alabama AL South 4779736 135 2.824424  
## 2 Alaska AK West 710231 19 2.675186  
## 3 Arizona AZ West 6392017 232 3.629527  
## 4 Arkansas AR South 2915918 93 3.189390  
## 5 California CA West 37253956 1257 3.374138  
## 6 Colorado CO West 5029196 65 1.292453

filter(murders, rate <= .71)

## state abb region population total rate  
## 1 Hawaii HI West 1360301 7 0.5145920  
## 2 Iowa IA North Central 3046355 21 0.6893484  
## 3 New Hampshire NH Northeast 1316470 5 0.3798036  
## 4 North Dakota ND North Central 672591 4 0.5947151  
## 5 Vermont VT Northeast 625741 2 0.3196211

new\_table = select(murders, state, region, rate)  
filter(new\_table, rate <= .71)

## state region rate  
## 1 Hawaii West 0.5145920  
## 2 Iowa North Central 0.6893484  
## 3 New Hampshire Northeast 0.3798036  
## 4 North Dakota North Central 0.5947151  
## 5 Vermont Northeast 0.3196211

In order to streamline, results can be sent from one function to another function using the pipe operator %>%:

murders %>% select(state, region, rate) %>% filter(rate <= .71)

## state region rate  
## 1 Hawaii West 0.5145920  
## 2 Iowa North Central 0.6893484  
## 3 New Hampshire Northeast 0.3798036  
## 4 North Dakota North Central 0.5947151  
## 5 Vermont Northeast 0.3196211

##### Data Frames

* Data frames can be created using the data.frame() function
* Columns are created using name = data, name = data, name = data, …
* Warning: data.frame turns character variables into factors, to avoid, add stringsAsFactors = FALSE Example:

grades = data.frame(names = c("John", "Juan", "Jean","Yao"),exam\_1 = c(95,80,90,85),exam\_2 = c(90,85,85,90), stringsAsFactors = F)  
grades

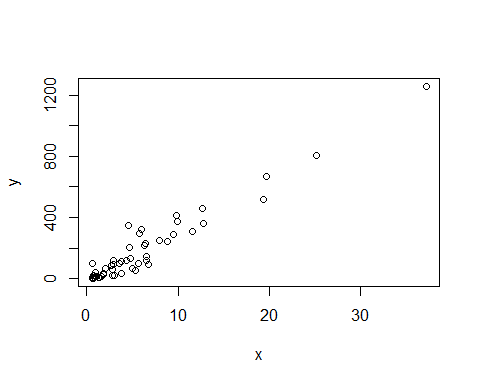
## names exam\_1 exam\_2  
## 1 John 95 90  
## 2 Juan 80 85  
## 3 Jean 90 85  
## 4 Yao 85 90

#### 3.3: Basic Plots

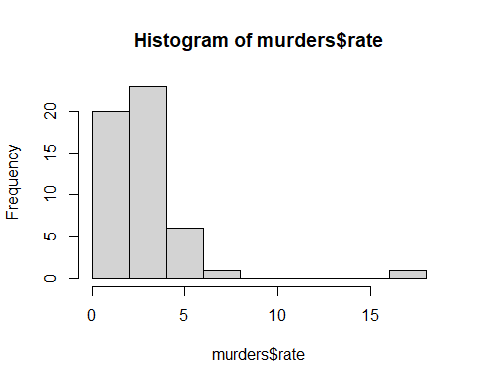
More will be learned on plots during the course based on the ggplot2 package.

Below are 3 different types of plots: scatterplot, histogram, and boxplot. All are related to the murders data set that we have been working with:

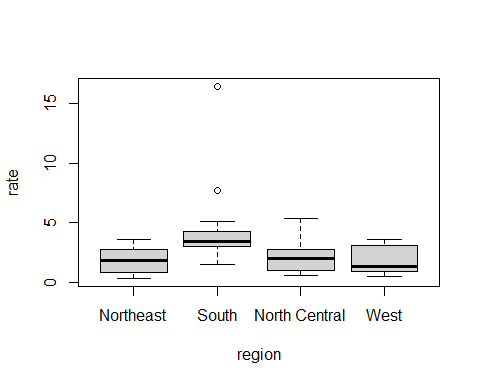
x = murders$population / 10^6  
y = murders$total  
plot(x,y)



hist(murders$rate)



boxplot(rate~region, data=murders)



### Section 4: Programming Basics

#### 4.2 Conditionals

* Most common conditional is the if/else statement

Let’s use the if/else statement to see if any of the states have a murder rate lower that .5, and print them if they do:

ind = which.min(murder\_rate)  
if(murder\_rate[ind] < .5) {  
 print(murders$state[ind])  
} else {  
 print("No state has murder rate that low")  
}

## [1] "Vermont"

if(murder\_rate[ind] < .25) {  
 print(murders$state[ind])  
} else {  
 print("No state has murder rate that low")  
}

## [1] "No state has murder rate that low"

* Can also use ifelse(boolean statement, return if true, return if false) statement
* The ifelse statement can be useful because it can be used on vectors

a=0  
ifelse(a > 0,1/a, NA)

## [1] NA

a=5  
ifelse(a > 0,1/a, NA)

## [1] 0.2

a = c(0,1,2,-4,5)  
ifelse(a > 0,1/a, NA)

## [1] NA 1.0 0.5 NA 0.2

* Another example using na\_example. Say we want to replace all the NAs in the data set with 0, we can use ifelse to do this:

data(na\_example)  
sum(is.na(na\_example))

## [1] 145

no\_nas = ifelse(is.na(na\_example),0,na\_example)

* any and all are useful function: any takes a vector and returns TRUE if there is at least one TRUE, all returns TRUE if all of the entries are TRUE:

z = c(T,T,F)  
z2 = c(F,F,F)  
z3 = c(T,T,T)  
any(z)

## [1] TRUE

all(z)

## [1] FALSE

any(z2)

## [1] FALSE

all(z2)

## [1] FALSE

any(z3)

## [1] TRUE

all(z3)

## [1] TRUE

#### 4.3 Functions

* Functions are useful for operations that need to be done multiple times.
* Values/variables in functions only live in the function, not in the global environment
* Format to create a function: my\_function = function(VARIABLES){operation that operate on VARIABLES and return value}

Example for creating the mean function:

avg = function(x){  
 s = sum(x)  
 n = length(x)  
 s/n  
}  
avg(c(1,2,3,4,5,6,7))

## [1] 4

x = 1:100  
identical(mean(x),avg(x))

## [1] TRUE

Function that computes either arithmetic or geometric average:

avg2 = function(x, arithmetic = T){  
 n = length(x)  
 ifelse(arithmetic, sum(x)/n, prod(x)^(1/n))  
}  
avg2(1:10)

## [1] 5.5

avg2(1:10, F)

## [1] 4.528729

#### 4.4 For Loops

* Say we want to prove the formula sum of 1 to n = :

compute\_s\_n = function(n){  
 x = 1:n  
 sum(x)  
}  
compute\_s\_n(3)

## [1] 6

compute\_s\_n(100)

## [1] 5050

* For loops allow us to change define the range that a variable takes:
* For loop format looks like: for(i in range of values){operations that use i in that range of values}
* Whatever variable in the range will be set to the end value

Simple Example:

for(i in 1:5){  
 print(i)  
}

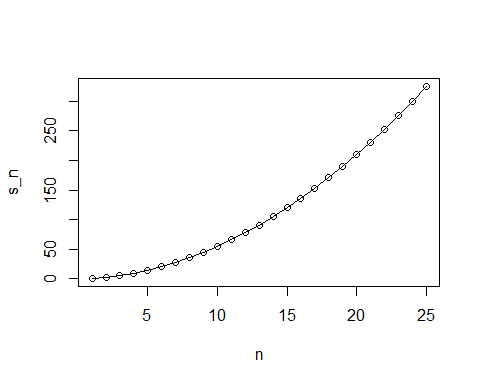
## [1] 1  
## [1] 2  
## [1] 3  
## [1] 4  
## [1] 5

i

## [1] 5

Here is a for loop for the sum problem:

m = 25  
#create an empty vector  
s\_n = vector(length=m)  
for(n in 1:m){  
 s\_n[n]=compute\_s\_n(n)  
}  
n = 1:m  
plot(n,s\_n)  
lines(n,(n\*(n+1))/2)

 Notice that the line matches the points perfectly

##### Other Functions

* For loops are rarely used because there is usually a more powerful way to do the same task
* Examples are: apply, sapply, tapply, and mapply

## Course 2: Data Visualization

### Section 1: Intro to Data Visualization and Distributions

#### 1.1 Intro to Data Visualization

Data Types:

* Discrete Numerical: numbers from a countably infinite list
* Continuous Numerical: can be any number
* Categorical: data can be broken up into categories (e.g Male and Female, or regions of the US)
* Ordinate: Ordered categorical (can also be numbers) (e.g tall, average, short)

#### 1.2 Intro to Distributions

* For categorical data, the distribution simply describes the proportions of each unique category

library(dslabs)  
data(heights)  
head(heights)

## sex height  
## 1 Male 75  
## 2 Male 70  
## 3 Male 68  
## 4 Male 74  
## 5 Male 61  
## 6 Female 65

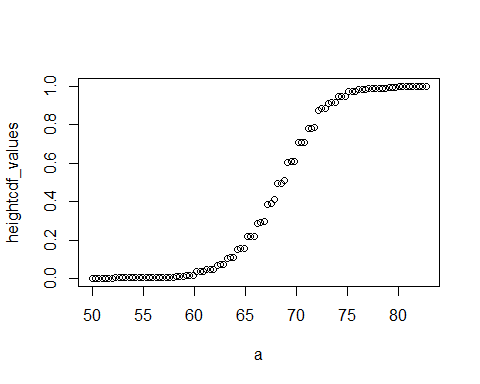
prop.table(table(heights$sex))

##   
## Female Male   
## 0.2266667 0.7733333

* When there are more categories, a bar plot is better at describing the distribution.
* Cumulative Distribution Function (CDF) is a function that reports the the proportion of data below a certain value:
* The proportion of data that are between two values (a and b) can be calculated using:
* If we want to get the proportion of data that is above a certain value:

Here is an example of creating a cdf related for the heights:

a <- seq(min(heights$height), max(heights$height), length = 100)   
heightcdf\_function <- function(x) {   
 mean(heights$height <= x)  
}  
heightcdf\_values <- sapply(a, heightcdf\_function)  
plot(a, heightcdf\_values)



heightcdf\_function(72)

## [1] 0.8704762