Homework 4

Christopher Brunswick

11/1/2021

1. Use the $ operator to access the population size data and store it as the object pop. Then use the sort function to redefine pop so that it is sorted. Finally, use the [ operator to report the smallest population size.

library(dslabs)

## Warning: package 'dslabs' was built under R version 4.0.5

data(murders)  
murders$population

## [1] 4779736 710231 6392017 2915918 37253956 5029196 3574097 897934  
## [9] 601723 19687653 9920000 1360301 1567582 12830632 6483802 3046355  
## [17] 2853118 4339367 4533372 1328361 5773552 6547629 9883640 5303925  
## [25] 2967297 5988927 989415 1826341 2700551 1316470 8791894 2059179  
## [33] 19378102 9535483 672591 11536504 3751351 3831074 12702379 1052567  
## [41] 4625364 814180 6346105 25145561 2763885 625741 8001024 6724540  
## [49] 1852994 5686986 563626

pop <- murders$population  
srtpop <- sort(pop)  
srtpop[1]

## [1] 563626

1. Now instead of the smallest population size, find the index of the entry with the smallest population size. Hint: use order instead of sort

index\_small <- order(pop)  
index\_small[1]

## [1] 51

1. We can actually perform the same operation as in the previous exercise using the function which.min. Write one line of code that does this.

which.min(pop)

## [1] 51

1. Now we know how small the smallest state is and we know which row represents it. Which state is it? Define a variable states to be the state names from the murders data frame. Report the name of the state with the smallest population.

states <- murders$state  
min <- which.min(pop)  
states[min]

## [1] "Wyoming"

1. You can create a data frame using the data.frame function. Use the rank function to determine the population rank of each state from smallest population size to biggest. Save these ranks in an object called ranks, then create a data frame with the state name and its rank. Call the data frame my\_df.

ranks <- rank(pop)  
my\_df <- data.frame(state\_name=states, pop\_ranks=ranks)  
my\_df

## state\_name pop\_ranks  
## 1 Alabama 29  
## 2 Alaska 5  
## 3 Arizona 36  
## 4 Arkansas 20  
## 5 California 51  
## 6 Colorado 30  
## 7 Connecticut 23  
## 8 Delaware 7  
## 9 District of Columbia 2  
## 10 Florida 49  
## 11 Georgia 44  
## 12 Hawaii 12  
## 13 Idaho 13  
## 14 Illinois 47  
## 15 Indiana 37  
## 16 Iowa 22  
## 17 Kansas 19  
## 18 Kentucky 26  
## 19 Louisiana 27  
## 20 Maine 11  
## 21 Maryland 33  
## 22 Massachusetts 38  
## 23 Michigan 43  
## 24 Minnesota 31  
## 25 Mississippi 21  
## 26 Missouri 34  
## 27 Montana 8  
## 28 Nebraska 14  
## 29 Nevada 17  
## 30 New Hampshire 10  
## 31 New Jersey 41  
## 32 New Mexico 16  
## 33 New York 48  
## 34 North Carolina 42  
## 35 North Dakota 4  
## 36 Ohio 45  
## 37 Oklahoma 24  
## 38 Oregon 25  
## 39 Pennsylvania 46  
## 40 Rhode Island 9  
## 41 South Carolina 28  
## 42 South Dakota 6  
## 43 Tennessee 35  
## 44 Texas 50  
## 45 Utah 18  
## 46 Vermont 3  
## 47 Virginia 40  
## 48 Washington 39  
## 49 West Virginia 15  
## 50 Wisconsin 32  
## 51 Wyoming 1

1. Repeat the previous exercise, but this time order my\_df so that the states are ordered from least populous to most populous. Hint: create an object ind that stores the indexes needed to order the population values. Then use the bracket operator [ to re-order each column in the data frame.

ind <- order(pop)  
my\_df <- data.frame(state\_name=states[ind], pop\_rank=sort(ranks))  
my\_df

## state\_name pop\_rank  
## 1 Wyoming 1  
## 2 District of Columbia 2  
## 3 Vermont 3  
## 4 North Dakota 4  
## 5 Alaska 5  
## 6 South Dakota 6  
## 7 Delaware 7  
## 8 Montana 8  
## 9 Rhode Island 9  
## 10 New Hampshire 10  
## 11 Maine 11  
## 12 Hawaii 12  
## 13 Idaho 13  
## 14 Nebraska 14  
## 15 West Virginia 15  
## 16 New Mexico 16  
## 17 Nevada 17  
## 18 Utah 18  
## 19 Kansas 19  
## 20 Arkansas 20  
## 21 Mississippi 21  
## 22 Iowa 22  
## 23 Connecticut 23  
## 24 Oklahoma 24  
## 25 Oregon 25  
## 26 Kentucky 26  
## 27 Louisiana 27  
## 28 South Carolina 28  
## 29 Alabama 29  
## 30 Colorado 30  
## 31 Minnesota 31  
## 32 Wisconsin 32  
## 33 Maryland 33  
## 34 Missouri 34  
## 35 Tennessee 35  
## 36 Arizona 36  
## 37 Indiana 37  
## 38 Massachusetts 38  
## 39 Washington 39  
## 40 Virginia 40  
## 41 New Jersey 41  
## 42 North Carolina 42  
## 43 Michigan 43  
## 44 Georgia 44  
## 45 Ohio 45  
## 46 Pennsylvania 46  
## 47 Illinois 47  
## 48 New York 48  
## 49 Florida 49  
## 50 Texas 50  
## 51 California 51

1. The na\_example vector represents a series of counts. You can quickly examine the object using:

data(“na\_example”) str(na\_example) mean(na\_example)

The is.na function returns a logical vector that tells us which entries are NA. Assign this logical vector to an object called ind and determine how many NA’s does na\_example have.

is.na(na\_example)

## [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE  
## [13] FALSE FALSE FALSE FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
## [25] FALSE FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
## [37] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
## [49] FALSE TRUE TRUE TRUE FALSE FALSE FALSE FALSE FALSE FALSE TRUE FALSE  
## [61] FALSE FALSE FALSE FALSE TRUE TRUE FALSE TRUE FALSE FALSE FALSE FALSE  
## [73] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
## [85] FALSE FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE FALSE FALSE FALSE  
## [97] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
## [109] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE FALSE  
## [121] FALSE FALSE FALSE FALSE TRUE TRUE TRUE TRUE FALSE FALSE FALSE FALSE  
## [133] FALSE FALSE FALSE FALSE FALSE FALSE TRUE TRUE TRUE TRUE FALSE FALSE  
## [145] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE FALSE  
## [157] FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE  
## [169] TRUE FALSE FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE TRUE FALSE  
## [181] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE TRUE FALSE FALSE  
## [193] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE FALSE  
## [205] TRUE FALSE TRUE TRUE FALSE FALSE FALSE TRUE FALSE TRUE FALSE FALSE  
## [217] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
## [229] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE FALSE  
## [241] TRUE FALSE TRUE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
## [253] TRUE FALSE FALSE FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
## [265] TRUE FALSE TRUE FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
## [277] FALSE FALSE TRUE FALSE FALSE TRUE FALSE FALSE FALSE FALSE TRUE FALSE  
## [289] FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE  
## [301] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE  
## [313] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
## [325] TRUE TRUE FALSE FALSE FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE  
## [337] FALSE FALSE FALSE FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE TRUE  
## [349] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
## [361] TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
## [373] FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
## [385] FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE TRUE FALSE  
## [397] TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE FALSE  
## [409] FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
## [421] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
## [433] FALSE FALSE FALSE FALSE TRUE FALSE FALSE FALSE FALSE FALSE TRUE FALSE  
## [445] FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
## [457] FALSE FALSE FALSE FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE TRUE  
## [469] FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
## [481] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE  
## [493] FALSE FALSE FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
## [505] TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
## [517] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE FALSE  
## [529] FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE TRUE FALSE  
## [541] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
## [553] TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE FALSE  
## [565] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE  
## [577] FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
## [589] TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE FALSE  
## [601] FALSE FALSE TRUE FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE FALSE  
## [613] FALSE FALSE FALSE TRUE FALSE TRUE FALSE TRUE FALSE FALSE FALSE FALSE  
## [625] FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE TRUE FALSE FALSE TRUE  
## [637] FALSE FALSE FALSE FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
## [649] FALSE FALSE FALSE TRUE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
## [661] FALSE FALSE FALSE FALSE FALSE TRUE TRUE TRUE FALSE FALSE FALSE FALSE  
## [673] FALSE FALSE FALSE FALSE TRUE FALSE FALSE TRUE FALSE FALSE FALSE FALSE  
## [685] FALSE FALSE TRUE FALSE FALSE FALSE TRUE FALSE FALSE FALSE TRUE FALSE  
## [697] FALSE FALSE FALSE FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
## [709] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
## [721] FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE  
## [733] FALSE TRUE TRUE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
## [745] TRUE TRUE TRUE TRUE FALSE FALSE TRUE FALSE FALSE FALSE FALSE TRUE  
## [757] FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE FALSE FALSE TRUE FALSE  
## [769] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
## [781] FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE TRUE FALSE FALSE FALSE  
## [793] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
## [805] FALSE FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
## [817] FALSE FALSE FALSE FALSE TRUE FALSE FALSE FALSE FALSE TRUE FALSE FALSE  
## [829] FALSE FALSE FALSE TRUE FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE  
## [841] FALSE FALSE TRUE TRUE TRUE FALSE FALSE FALSE TRUE TRUE FALSE FALSE  
## [853] TRUE FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE FALSE FALSE FALSE  
## [865] FALSE FALSE FALSE FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
## [877] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE FALSE  
## [889] FALSE FALSE FALSE TRUE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
## [901] FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE TRUE FALSE TRUE FALSE  
## [913] FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE FALSE FALSE FALSE TRUE  
## [925] TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
## [937] FALSE FALSE TRUE FALSE FALSE FALSE TRUE FALSE FALSE FALSE FALSE FALSE  
## [949] FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
## [961] FALSE TRUE FALSE FALSE TRUE TRUE FALSE TRUE FALSE FALSE FALSE FALSE  
## [973] FALSE FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE FALSE FALSE FALSE  
## [985] FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE  
## [997] FALSE FALSE TRUE FALSE

ind <- is.na(na\_example)  
na\_example[ind]

## [1] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA  
## [26] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA  
## [51] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA  
## [76] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA  
## [101] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA  
## [126] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA

length(na\_example[ind])

## [1] 145

1. Now compute the average again, but only for the entries that are not NA. Hint: remember the ! operator.

na.omit(na\_example)

## [1] 2 1 3 2 1 3 1 4 3 2 2 2 2 1 4 1 1 2 1 2 2 1 2 5 2 2 3 1 2 4 1 1 1 4 5 2 3  
## [38] 4 1 2 4 1 1 2 1 5 1 1 5 1 3 1 4 4 7 3 2 1 4 1 2 2 3 2 1 2 2 4 3 4 2 3 1 3  
## [75] 2 1 1 1 3 1 3 1 2 2 1 2 2 1 1 4 1 1 2 3 3 2 2 3 3 3 4 1 1 1 2 4 3 4 3 1 2  
## [112] 1 1 5 1 2 1 3 5 3 2 2 3 5 3 1 1 4 2 4 3 3 2 3 2 6 1 1 2 2 1 3 1 1 5 2 4 2  
## [149] 5 1 4 3 3 4 3 1 4 1 1 3 1 1 3 5 2 2 2 3 1 2 2 3 2 1 2 1 2 1 1 3 1 2 2 1 3  
## [186] 2 2 1 1 2 3 1 1 1 4 3 4 2 2 1 4 1 5 1 4 3 1 1 5 2 3 3 2 4 3 2 5 2 3 4 6 2  
## [223] 2 2 2 2 3 3 2 2 4 3 1 4 2 2 4 6 2 3 1 2 2 1 1 3 2 3 3 1 1 4 2 1 1 3 2 1 2  
## [260] 3 1 2 3 3 2 1 2 3 5 5 1 2 3 3 1 1 2 4 2 1 1 1 3 2 1 1 3 4 1 2 1 1 3 3 1 1  
## [297] 3 5 3 2 3 4 1 4 3 1 2 1 2 2 1 2 2 6 1 2 4 5 3 4 2 1 1 4 2 1 1 1 1 2 1 4 4  
## [334] 1 3 3 3 2 1 2 1 1 4 2 1 4 4 1 2 3 2 2 2 1 4 3 6 1 2 3 1 3 2 2 2 1 1 3 2 1  
## [371] 1 1 3 2 2 4 4 4 1 1 4 3 1 3 1 3 2 4 2 2 2 3 2 1 4 3 1 4 3 1 3 2 3 1 3 1 4  
## [408] 1 1 1 2 4 3 1 2 2 2 3 2 3 1 1 3 2 1 1 2 2 2 2 3 3 1 1 2 1 2 1 1 3 3 1 3 1  
## [445] 1 1 1 1 2 5 1 1 2 2 1 1 1 4 1 2 4 1 3 2 1 1 2 1 1 4 2 3 3 1 5 3 1 1 2 1 1  
## [482] 3 1 3 2 4 2 3 2 1 2 1 1 1 2 2 3 1 5 2 2 3 2 2 2 1 5 3 2 3 1 3 1 2 2 2 1 2  
## [519] 2 4 6 1 2 1 1 2 2 3 3 2 3 3 4 2 2 4 1 1 2 2 3 1 1 1 3 2 5 7 1 4 3 3 1 1 1  
## [556] 1 1 3 2 4 2 2 3 1 4 3 2 2 2 3 2 4 2 2 4 6 3 3 1 4 4 2 1 1 6 3 3 2 1 1 6 1  
## [593] 5 1 2 6 2 4 1 3 1 2 1 1 3 1 2 4 2 1 3 2 4 3 2 2 1 1 5 6 4 2 2 2 2 4 1 2 2  
## [630] 2 2 4 5 4 3 3 3 2 4 2 4 2 1 2 4 3 2 2 3 1 3 4 1 2 1 2 3 1 2 1 2 1 2 1 2 2  
## [667] 2 2 1 1 3 3 1 3 4 3 4 2 3 2 1 3 2 4 2 2 3 1 2 4 3 3 4 1 4 2 1 1 1 3 1 5 2  
## [704] 2 4 2 1 3 1 2 1 2 1 2 1 1 3 2 3 2 2 1 4 2 2 4 2 3 1 5 5 2 2 2 2 1 3 1 3 2  
## [741] 4 2 4 4 1 2 3 2 3 3 2 3 2 2 2 1 3 2 4 2 3 3 2 2 3 2 1 2 4 1 1 1 1 4 3 2 3  
## [778] 2 1 3 2 1 1 1 2 2 2 3 3 2 4 5 2 2 2 1 2 3 1 3 3 4 3 1 1 1 4 3 5 1 1 2 2 2  
## [815] 2 2 5 2 2 3 1 2 3 1 2 2 3 1 1 2 5 3 5 1 1 4 2 1 3 1 1 2 4 3 3 3 1 1 2 2 1  
## [852] 1 2 2 2  
## attr(,"na.action")  
## [1] 12 17 27 50 51 52 59 65 66 68 91 117 125 126 127 128 139 140  
## [19] 141 142 153 158 168 169 172 179 189 190 203 205 207 208 212 214 237 241  
## [37] 243 244 253 257 265 267 269 279 282 287 290 298 310 325 326 330 341 348  
## [55] 361 374 392 395 397 407 410 437 443 446 461 468 470 490 496 505 527 536  
## [73] 539 553 561 576 578 589 599 603 609 616 618 620 630 633 636 641 652 653  
## [91] 666 667 668 677 680 687 691 695 701 726 734 735 736 745 746 747 748 751  
## [109] 756 762 767 788 789 807 821 826 832 838 843 844 845 849 850 853 859 869  
## [127] 887 892 893 906 909 911 918 924 925 939 943 950 962 965 966 968 979 990  
## [145] 999  
## attr(,"class")  
## [1] "omit"

no\_na <- na.omit(na\_example)  
mean(no\_na)

## [1] 2.301754