R prep. lecture notes 2023-10-25 (R Objects)

Adam Tallman

2023-10-25

R Markdown

###Vector concepts

There are six main ``vector concepts" to learn; atomic vector, double, integer, character, logical, and coercion.

An *atomic vector* is just a set of elements. We call it ``atomic" because its just one. You can make a vector with the function c().

```
things <- c(7,6,5,4,3,2,1)
```

You can test whether something is a vector or not , with the function is.vector() and you can ask about the length of the vector using length(). We call functions like is.vector ``logical operators" because they give a true or false answer.

```
is.vector(things)
```

```
## [1] TRUE
```

```
length(things)
```

```
## [1] 7
```

We can also identify positions in the vector

```
things[3]
```

```
## [1] 5
```

A *double* is a number that *can* has decimals after. We can also call this a *numeric*, which is probably the more common term. A double vector would be a vector of doubles. The vector we created above would be an example.

An *integer* is a number that cannot have decimals. You can specifically create an integer value using uppercase L.

```
3L
```

```
## [1] 3
```

```
things_2 <- c(-1L, 2L, 4L)
typeof(things_2)
```

```
## [1] "integer"
```

R only saves a nuber as an integer if you include the L otherwise it'll save it as a double.

The only reason to ever use integers as opposed to numbers is because of *floating point errors*, whch is part of all mathematics done by computers.

```
sqrt(2)^2-2
```

```
## [1] 4.440892e-16
```

A *character* is some text. You write a character with quotations around it.

```
text <- c("Please", "write", "down", "the", "answer", "now", "everyone")
```

```
typeof(text)
```

```
## [1] "character"
```

R can also do Boolean logic

```
3>4
```

```
## [1] FALSE
```

```
4>3
```

```
## [1] TRUE
```

```
"class" %in% text
```

```
## [1] FALSE
```

```
"write" %in% text
```

```
## [1] TRUE
```

You can also have a vector that is just true or false values

```
things_3 <- c(TRUE, FALSE, FALSE, TRUE)
```

Finally, note that a single vector will be saved as one of the types. If you have a single vector with a bunch of numbers, the vector will be saved as a character type.

```
things_4 <- c(2,3,5,7,"thing")
```

```
typeof(things_4)
```

```
## [1] "character"
```

A vector you might think is a numeric vector in fact isn't. Vectors can be coerced but only in certain directions.

```
as.numeric(things_3)
```

```
## [1] 1 0 0 1
```

```
as.character(things)
```

```
## [1] "7" "6" "5" "4" "3" "2" "1"
```

So intuitively it might make sense t get the sum of numbers of things_4, but it doesn't work

```
#sum(things_4)
```

We could only make things_4 a numeric vector if we removed the final value. let's find the value, with the function match(). match() takes two arguments

```
things_4 <- c(2,3,5,7,"thing")

match("thing", things_4)
```

```
## [1] 5
```

So how can we change the database so we could get the sum over the numbers? We have to remove the character first. In order to do this we have to know how to reference elements in the vector. Every vector has indices for all its values. You can get those indices by using square brackets.

```
things[1] #retrieve value from the first position
```

```
## [1] 7
```

```
things[1:2] #retrieve value from positions 1 through 2
```

```
## [1] 7 6
```

You can output the vector *without* a specific element, but subtracting it.

```
things[c(-1,-3)] #retrieve values without positions 1 and 3
```

```
## [1] 6 4 3 2 1
```

So we could remove the character value like so.

```
things_4[-5] #remove value from position 5
```

```
## [1] "2" "3" "5" "7"
```

We can *reassign* a value to something else.

```
things_4 <- things_4[-5]
things_4
```

```
## [1] "2" "3" "5" "7"
```

We still can't sum over the value yet.

```
#sum(things_4)
```

But now we can change thsoe numbers to actual numbers because there's no more character strings in there.

```
as.numeric(things_4)
```

```
## [1] 2 3 5 7
```

That works, so we can reassign things_4 as numeric value and then sum over it

```
things_4 <- as.numeric(things_4)
sum(things_4)
```

```
## [1] 17
```

##Non-atomic vectors

Matrices are stored values in two-dimensions. The first argument gives you the number of rows and the second the number of columns

```
matrix(3,3,2)
```

```
##      [,1] [,2]
## [1,]    3    3
## [2,]    3    3
## [3,]    3    3
```

```
matrix(c(1,2,3,4,5,6), 2,3)
```

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

You can change some atomic vector into a matrix with as.matrix().

```
as.matrix(things)
```

```
##      [,1]
## [1,]    7
## [2,]    6
## [3,]    5
## [4,]    4
## [5,]    3
## [6,]    2
## [7,]    1
```

##Work on this a little bit before lecture, its not too important anyways

Before we move to data frames and lists, we should review how to apply operations over vectors.

```
things_5 <- c(1,4,5,6,8,9,20,36)
```

Recall length() tells you how many elements are in your vector

```
length(things_5)
```

```
## [1] 8
```

Important concepts for descriptive statistical are *mode*, *mean* and *sum*.

```
median(things_5)
```

```
## [1] 7
```

```
mean(things_5)
```

```
## [1] 11.125
```

```
sum(things_5)
```

```
## [1] 89
```

You can also do arithmetic on each of the values of thee vector.

```
things_5 * 2
```

```
## [1]  2  8 10 12 16 18 40 72
```

```
things / 2
```

```
## [1] 3.5 3.0 2.5 2.0 1.5 1.0 0.5
```

You cna do more complicated things

```
v <- c(2,3,1,2,5,6,7,8)
v * things_5
```

```
## [1]   2  12   5  12  40  54 140 288
```

```
v <- c(2,3)
v* things_5
```

```
## [1]   2  12  10  18  16  27  40 108
```

There's lots of things you can do with characters. Later in the course we'll get into that. Here's an example though, splitting a single character into a set of values.

```
sentence <- c("I hope you'll come to class next week")
strsplit(sentence, " ")
```

```
## [[1]]
## [1] "I"      "hope"   "you'll" "come"   "to"     "class"  "next"   "week"
```

##Lists

Lists are like atomic vectors in that they are one-dimensional. They are not grouping together individual values, but R objects (e.g. vectors).

```
my_office <- c("desk", "computer", "books") #This is an atomic vector
my_kitchen <- c("fridge", "sink", "oven") #This is an atomic vector
monthly_stipend <- c(1100, 1300, 1200, 1110, 1300) #This is an atomic vector
house <- c(my_office, my_kitchen, monthly_stipend) #This is a list
```

Data frames are a two dimensional version of a list. They are the most useful storage mechanism for a data analysis.

```
nettle <- read.csv("/Users/Adan Tallman/Desktop/StatisticsinLinguistics_FSU_2023_2024/07_data
/nettle_1999_climate.csv")
```

You can look at the first six rows of a dataset with the function head()

```
head(nettle)
```

```
##      Country Population Area  MGS Langs
## 1    Algeria       4.41 6.38 6.60    18
## 2     Angola       4.01 6.10 6.22    42
## 3  Australia       4.24 6.89 6.00   234
## 4 Bangladesh       5.07 5.16 7.40    37
## 5      Benin       3.69 5.05 7.14    52
## 6    Bolivia       3.88 6.04 6.92    38
```

You can get the last 6 rows by using the function tail()

```
tail(nettle)
```

```
##      Country Population Area  MGS Langs
## 69 Venezuela       4.31 5.96 7.98    40
## 70   Vietnam       4.83 5.52 8.80    88
## 71     Yemen       4.09 5.72 0.00     6
## 72     Zaire       4.56 6.37 9.44   219
## 73    Zambia       3.94 5.88 5.43    38
## 74  Zimbabwe       4.00 5.59 5.29    18
```

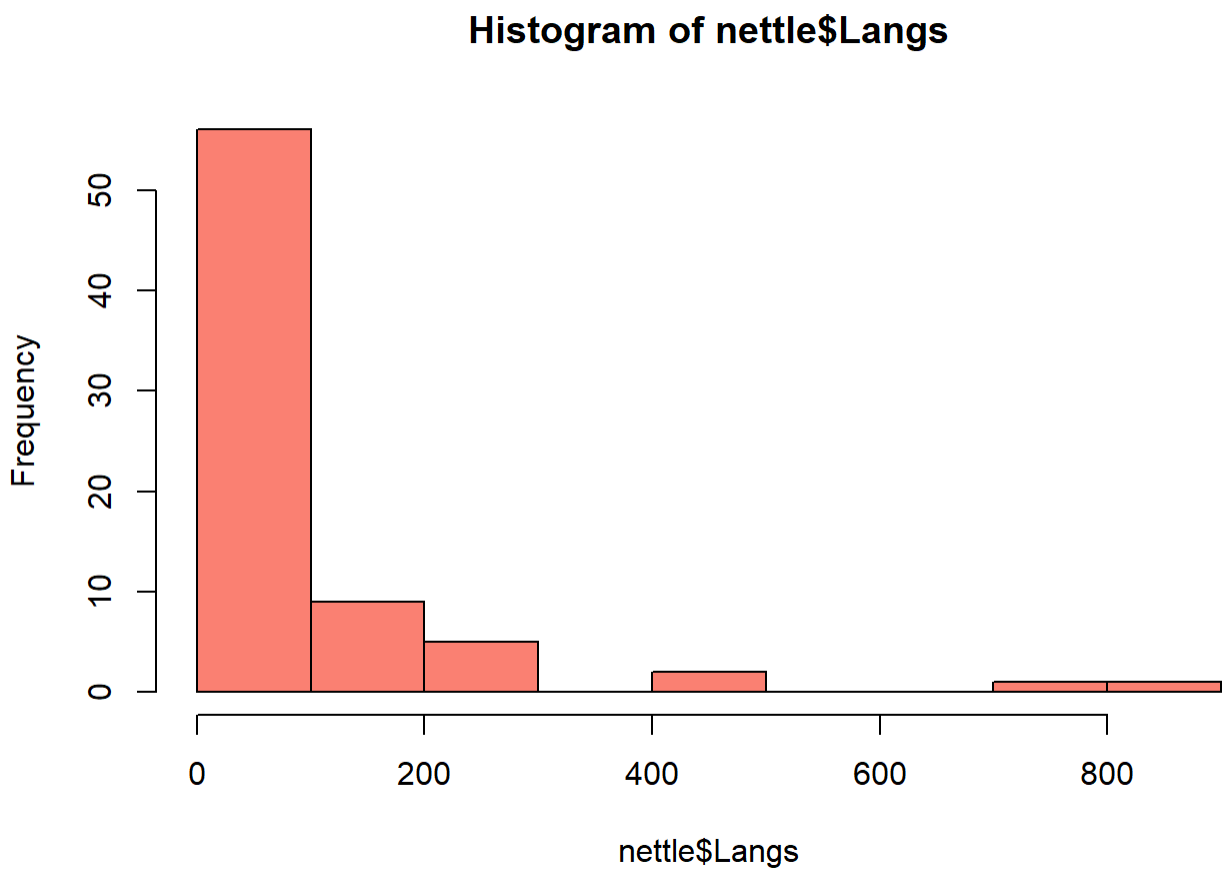
You can get a summary of the variables with the function summary().

```
summary(nettle)
```

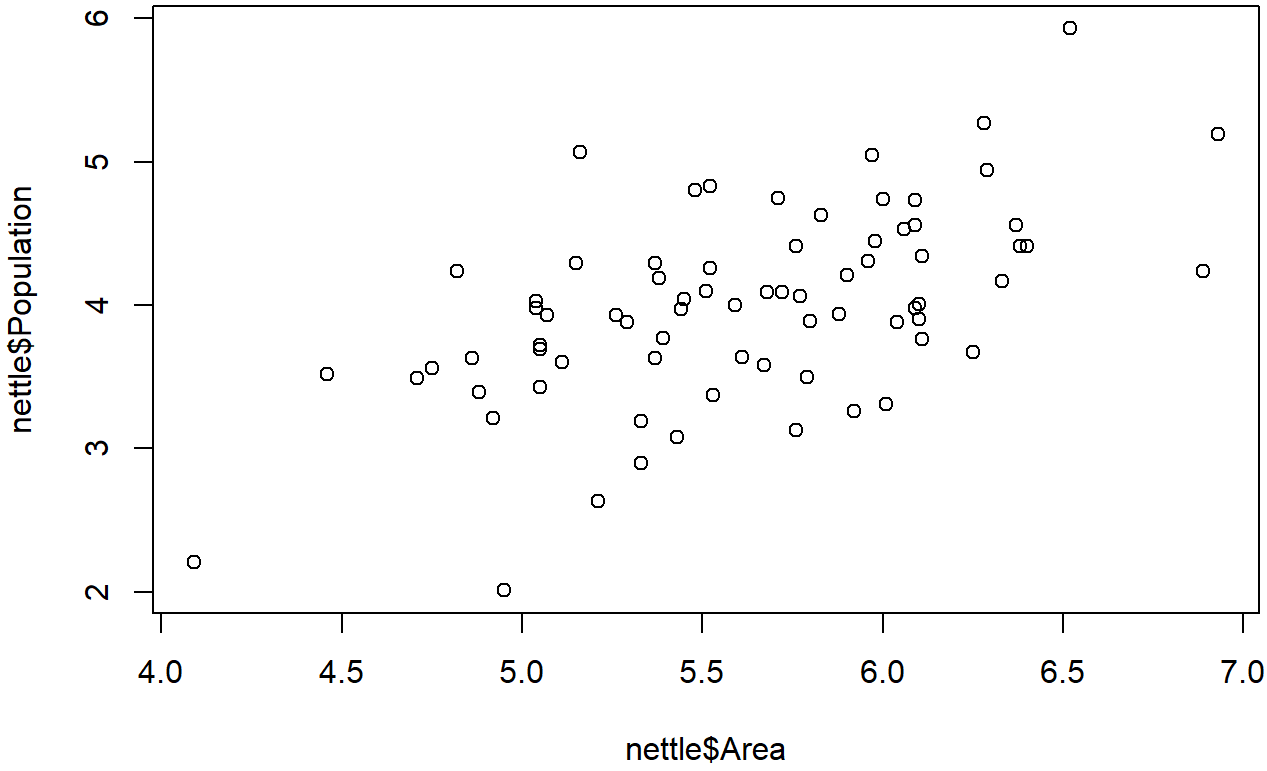
```
##    Country            Population         Area            MGS
##  Length:74          Min.   :2.010   Min.   :4.090   Min.   : 0.000
##  Class :character   1st Qu.:3.607   1st Qu.:5.223   1st Qu.: 5.348
##  Mode  :character   Median :3.990   Median :5.640   Median : 7.355
##                     Mean   :3.992   Mean   :5.618   Mean   : 7.029
##                     3rd Qu.:4.393   3rd Qu.:6.032   3rd Qu.: 9.193
##                     Max.   :5.930   Max.   :6.930   Max.   :12.000
##      Langs
##  Min.   :  1.00
##  1st Qu.: 17.25
##  Median : 40.00
##  Mean   : 89.73
##  3rd Qu.: 93.75
##  Max.   :862.00
```

We can make a histogram

```
hist(nettle$Langs, col="salmon")
```

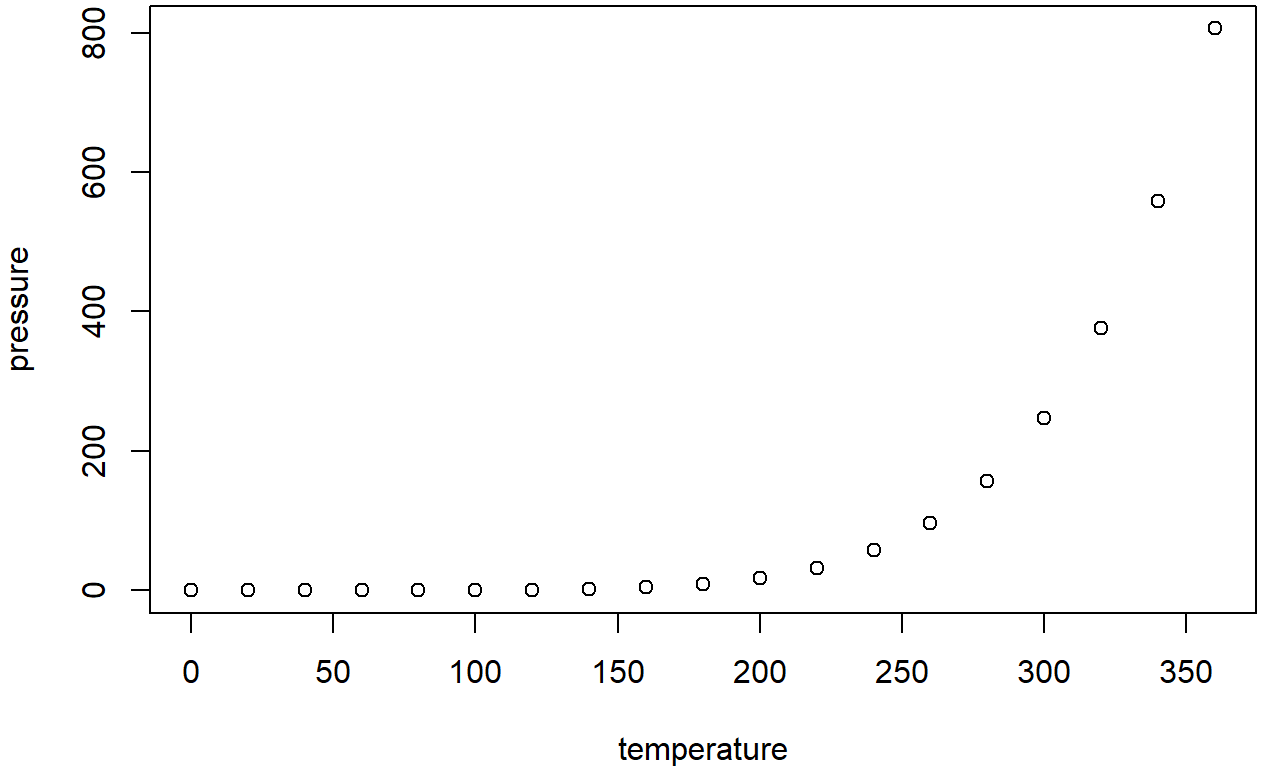



```
scatter_01 <- plot(nettle$Population~nettle$Area)
```



Including Plots

You can also embed plots, for example:



Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated the plot.