Descriptive Statistics in R

Victor A. Ordu

2021-07-16

Introduction

- Most of the functions related to pure statistics can be found in the *base* and *stats* package
- They are always pre-loaded by default in every R session

```
search()
```

```
## [1] ".GlobalEnv"        "package:stats"     "package:gra
## [4] "package:grDevices" "package:utils"     "package:dat
## [7] "package:methods"   "Autoloads"         "package:bas
```

Summarization of Data

Summation

- Numerical data

The sum:

$$Sum(X) = \sum_{i=1}^{N} x_i$$

where x_i is a single element in a numeric vector and N is the number of elements in that vector.

We add up the values with `sum()`:

```
x <- 1:100
sum(x)
```

```
## [1] 5050
```

- Categorical data

Here we do **counting** with `length()`

How many letters are there in the English alphabet?

```
length(LETTERS)
```

```
## [1] 26
```

Scaling

- Addition/subtraction
- Multiplication and/or division.
- When you apply a contant number, it computes on every element of the vector

```
mix <- c(5, 0, 28, -5, 29, 77, 10, 57, 28, 88, 298)


mix + 5
```

```
## [1]  10   5  33   0  34  82  15  62  33  93 303
```

```
mix * 2
```

```
## [1]  10   0  56 -10  58 154  20 114  56 176 596
```

```
mix / 3
```

```
## [1]  1.666667  0.000000  9.333333 -1.666667  9.666667 2
## [8] 19.000000  9.333333 29.333333 99.333333
```

```
mix %% 2
```

Maxima and minima

- Get the highest or lowest value in a collection.
- Functions: `max()` and `min()`

```
max(mix)
```

```
## [1] 298
```

```
min(mix)
```

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

Summarization

- Tukey's five-number summary: minimum, maximum, mean, 1st quartile, 3rd quartile
- At a glance, gives you an idea of central statistic, dispersion and extremes of the data.

```
fivenum(mix)
```

```
## [1]  -5.0   7.5  28.0  67.0 298.0
```

Sorting

Create an ordered array of values

```
sort(mix)
```

```
## [1] -5   0   5  10  28  28  29  57  77  88 298
```

In descending order

```
sort(mix, decreasing = TRUE)
```

```
## [1] 298  88  77  57  29  28  28  10   5   0  -5
```

Ranking

Get the rank of values within a collection

```
order(mix)
```

```
##  [1]  4  2  1  7  3  9  5  8  6 10 11
```

Compare with original vector for clarity:

```
comp <- data.frame(original = mix, ordered = order(mix))
comp
```

```
##    original ordered
## 1         5       4
## 2         0       2
## 3        28       1
## 4        -5       7
## 5        29       3
## 6        77       9
## 7        10       5
## 8        57       8
```

Measures of Central Tendency

Arithmetic mean

Arithmetic mean, of a population μ is defined by the equation

$$\mu = \frac{\sum_{i=1}^{N} x_i}{N}$$

- Computed with the function `mean()`

```
mean(mix)
```

```
## [1] 55.90909
```

Median

The median, the *middle* value in an ordered array, X, with N numbers can be described by the formula

$$median(X) = x_{(N+1)/2}$$

and when considering even numbered sets, we have the formula

$$median(X) = \frac{x_{N/2} + x_{(N/2)-1}}{2}$$

—

- Computed with the function `median()`

```
median(mix)
```

```
## [1] 28
```

The Difference beteen mean and median

```
# Add an outlier to the vector 'mix'
mix[length(mix) + 1] <- 1000
mix
```

```
##  [1]    5    0   28   -5   29   77   10   57   28   88
```

```
mean(mix)
```

```
## [1] 134.5833
```

```
median(mix)
```

```
## [1] 28.5
```

Mode

There is no function per se in R for computing the mode, but developing a custom function for this purpose is trivial and would be done on a case-by-case basis.

Measures of Dispersion

When you want to assess the variability in a set of data or a given variable

Range

The range, R, is the difference between the lowest value, x_S and the largest value, x_L, i.e.

$$R = x_L - x_S$$

In R, the function `range` returns both values as a 2-element numeric vector.

```
range(mix)
```

```
## [1]   -5 1000
```

Interquartile Range

Is the range between the first and the third quartile - A measure of the "middle fifty percent" of a dataset - It is the range between the 75th and 25th percentile

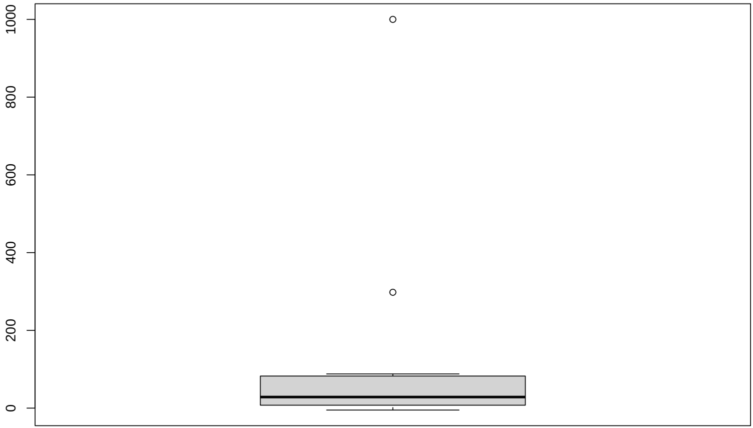
$$IQR = Q_3 - Q_1$$

```
IQR(mix)
```

```
## [1] 71
```

In the box-and-whiskers plot, the "box" displays the *IQR*.

```
boxplot(mix)
```



Variance

Dispersion around the mean, with the difference squared to create absolute values.

$$Var(X) = \frac{\sum_{i=1}^{N}(x_i - \mu)^2}{N}$$

```
var(mix)
```

```
## [1] 80993.9
```

Standard Deviation

This is essentially (elementarily) the square root of the variance

$$s.d.(X) = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \bar{x})^2}{N}}$$

```
sd(mix)
```

```
## [1] 284.5943
```

Coefficient of variation

Used to measure the variation between 2 different but related datasets It is the ratio between the standard deviation and the mean, expressed as a percentage.

$$C.V. = \frac{s.d.}{\bar{x}}(100)$$

\# Grouping of Data

Numerical vs. Categorical Data

- With numerical data we create intervals first, coverting them to discrete categories.
- Then we create a frequency tabulation

```
cut(mix, 3)  # create 3 groups using defaults
```

```
##  [1] (-6,330]    (-6,330]    (-6,330]    (-6,330]    (-6
##  [7] (-6,330]    (-6,330]    (-6,330]    (-6,330]    (-6
## Levels: (-6,330] (330,665] (665,1e+03]
```

```r
cut(mix, c(-5, 0, 100, 1000))  # Note missing value
```

```
##  [1] (0,100]     (-5,0]      (0,100]     <NA>        (0,
##  [7] (0,100]     (0,100]     (0,100]     (0,100]     (10
## Levels: (-5,0] (0,100] (100,1e+03]
```

‘

```
cut(mix, c(-4, 0, 100, 1000), include.lowest = TRUE)
```

```
##  [1] (0,100]     [-4,0]      (0,100]     <NA>        (0,
##  [7] (0,100]     (0,100]     (0,100]     (0,100]     (10
## Levels: [-4,0] (0,100] (100,1e+03]
```

```r
cut(mix, breaks = c(min(mix), median(mix), max(mix)))
```

```
##  [1] (-5,28.5]     (-5,28.5]     (-5,28.5]     <NA>
##  [6] (28.5,1e+03] (-5,28.5]     (28.5,1e+03] (-5,28.5]
## [11] (28.5,1e+03] (28.5,1e+03]
## Levels: (-5,28.5] (28.5,1e+03]
```

Frequency Distribution

- For categorical/discrete data
- In R, we use the `table()` function.

```
table(esoph$agegp)
```

```
##
## 25-34 35-44 45-54 55-64 65-74   75+
##    15    15    16    16    15    11
```

Cumulative Frequency Distribution

```
cumsum(mix)
```

```
##  [1]    5    5   33   28   57  134  144  201  229  317
```

```r
# Compare with original
cumfrq <- data.frame(original = mix, cum.freq = cumsum(mix)
cumfrq
```

```
##    original cum.freq
## 1         5        5
## 2         0        5
## 3        28       33
## 4        -5       28
## 5        29       57
## 6        77      134
## 7        10      144
## 8        57      201
## 9        28      229
## 10       88      317
## 11      298      615
## 12     1000     1615
```

Something is wrong with this arrangement. Can you spot it?

Some important characteristics of vectors

- We need to discuss some important behaviours of vectors
- They have practical implications when you are carrying out analyses.

Vectorization

- A process whereby an operation carried out affects every element of a vector.
- First let's create a vector with random whole numbers

```
set.seed(134)
myvalues <- sample(1:7, size = 10, replace = TRUE)
myvalues
```

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

Arithmetic

```r
myvalues + 1
```

```
##  [1] 5 3 3 4 7 5 3 4 3 8
```

```r
myvalues * 5
```

```
##  [1] 20 10 10 15 30 20 10 15 10 35
```

```r
myvalues < 3
```

```
##  [1] FALSE  TRUE  TRUE FALSE FALSE FALSE  TRUE FALSE  TF
```

Vectorized Logical ‘AND’

```
set.seed(54)
newvalues <- sample(myvalues, size = length(myvalues), repl
identical(newvalues, myvalues)
```

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

```
myvalues < newvalues
```

```
##  [1] FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE FAL
```

```
myvalues > 2 && myvalues < 5
```

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

But with "Vectorized" & we do this check element by element.

```
myvalues > 2 & myvalues < 5
```

```
##  [1]  TRUE FALSE FALSE  TRUE FALSE  TRUE FALSE  TRUE FAL
```

- `&&` and `||` are best for logical single outcome computations e.g. `is.character(x) && is.atomic(x)`
- `&` and `|` are used to compare elements one by one along the length of two vectors. e.g. `is.na(x) | duplicated(x)`

Recycling

When 2 vectors are used together as operands in a computation e.g. $A + B$, if `length(A) != length(B)`, the shorter one will be recycled until the computation on the longer vector is completed, without prejudice to which element of the shorter is used last.

Thus,

```
x <- 5
y <- 1:8
x * y    # Recycling of lengh 1L akin to vectorization
```

```
## [1]  5 10 15 20 25 30 35 40
```

```r
x <- c(5, 10)
x * y
```

```
## [1]  5 20 15 40 25 60 35 80
```

Display side-by-side

```
data.frame(y, prod = x * y)
```

```
##   y prod
## 1 1    5
## 2 2   20
## 3 3   15
## 4 4   40
## 5 5   25
## 6 6   60
## 7 7   35
## 8 8   80
```

Demonstrate abrupt stop to recycling

```
x[3] <- 15L
x * y
```

```
## [1]  5 20 45 20 50 90 35 80
```

Viewed in a data frame…

```
##    x y x.times.y
## 1  5 1         5
## 2 10 2        20
## 3 15 3        45
## 4  5 4        20
## 5 10 5        50
## 6 15 6        90
## 7  5 7        35
## 8 10 8        80
```

Missing values

Most computations, such as the ones above will return `NA` if a missing value is present.

```
mix[7] <- NA
mix
```

```
##  [1]    5    0   28   -5   29   77   NA   57   28   88
```

```
sum(mix)
```

```
## [1] NA
```

To fix this, remove `NA` from the computation. See the `sum`'s signature:

```
args(sum)
```

```
## function (..., na.rm = FALSE)
## NULL
```

Where `...` received objects to be summed up and `na.rm` tunes our ability to discount `NA` from the computation.

Thus,

```
sum(mix, na.rm = TRUE)
```

```
## [1] 1605
```

Many other functions used for statistical computations have arguments for dealing with `NA`. Whether you leave or remove depends entirely on the circumstance at any given time.

```
mean(mix)
```

```
## [1] NA
```

```
mean(mix, na.rm = TRUE)
```

```
## [1] 145.9091
```

The real point is to always glance at the help files e.g. `?mean`. The funcion `args()` used earlier also provides a way to quickly look at a function signature, but it is not always guaranteed to be informative, especially with generic functions.

```
args(plot)

?plot
```