R Statistics

Descriptive Statistics

Statistical analysis in R is performed by using many built-in functions. Most of these functions are part of the R base package. These functions take R vector as an input along with the arguments and give the result. Additionally, sometimes we need to generate a user made functions to calculate some statistical measurements (e.g. mode), for that we'll explain how to build function in R.

R Functions

A function is a block of code which only runs when it is called. You can pass data, known as parameters, into a function. A function can return data as a result. To create a function, use the **function()** keyword. To call a function, use the function name followed by parenthesis, like **my_function()**:

Example:

```
> # Define a function to calculate the area of a rectangle
> calculate_area <- function(length, width) {
+   area <- length * width  # Compute the area
+   return(area)            # Return the result
+ }
>
> # Call the function
> area1 <- calculate_area(5, 3)  # Length = 5, Width = 3
> area2 <- calculate_area(10, 7) # Length = 10, Width = 7
>
> # Print the results
> print(paste("The area of the first rectangle is:", area1)
)
[1] "The area of the first rectangle is: 15"

> print(paste("The area of the second rectangle is:", area2
))
[1] "The area of the second rectangle is: 70"
```

Explanation:

- calculate_area is the name of the function.
- The length and width are **arguments** that the user provides as inputs.
- The area is calculated inside the function using the formula length * width.
- The return() statement ensures the function outputs the result when called.

How many arguments can function hold?

In R, there is technically no strict limit on the number of arguments a function can hold. You can define functions with as many arguments as needed, depending on the complexity of your task. However, managing a large number of arguments can become challenging and may affect code readability and usability.

Example:

```
> # Function with multiple arguments
> calculate_stats <- function(a, b, c, d, e) {
+   mean_value <- mean(c(a, b, c, d, e))  # Calculate mean
+   max_value <- max(c(a, b, c, d, e))    # Find maximum va
lue
+   return(list(mean = mean_value, max = max_value))
+ }
>
> # Call function
> result <- calculate_stats(1, 2, 3, 4, 5)
> print(result)
$mean
[1] 3

$max
[1] 5
```

Should the function have argument?

No, a function in R does not necessarily need to have arguments! While arguments are often used to make functions more flexible and reusable, it's completely fine to define a function without any arguments if it performs a fixed task that doesn't depend on external inputs.

Example: A Function Without Arguments

```
> # Define a function with no arguments
> say_hello <- function() {
+   print("Hello, world!")  # Prints a message
+ }
>
> # Call the function
> say_hello()
[1] "Hello, world!"
```

Let us return to Descriptive Statistics

We can calculate the central tendency measurements (Mean, Median,), dispersion measurements (Range, Variance, Standard Deviation, Max, Min) by using basic R functions, while Mode, and distribution measurements (Skewness, Kurtosis) will calculated manually (as there is no available baseic function for them in R), or by using R specific libraries.

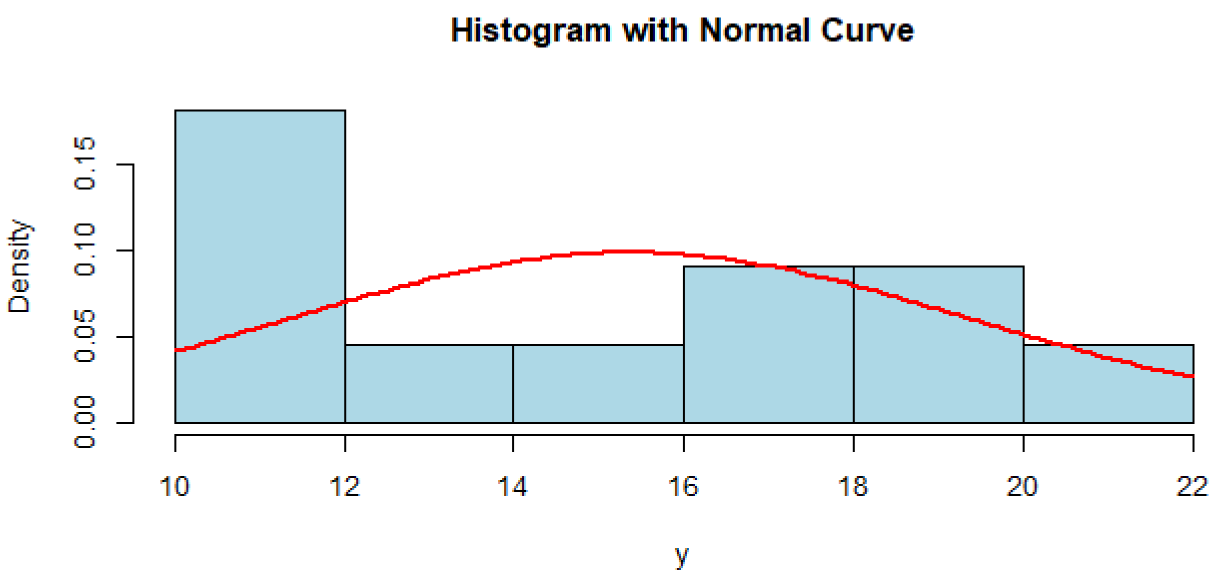
Calculating Descriptive statistics manually

Example: suppose we have the following data set: (12, 10, 11, 15, 20, 13, 12, 19, 22, 17, 18), and we want to perform descriptive statistic manually using basic R functions **(note: mode, skewness, and kurtosis have no built-in basic function in R, so we need to build them manually).**

```
> Descriptive <-function(x) {
+
+   # creating function to calculate mode
+   getmode <- function(x) {
+     uniqv <- unique(x)
+     return(uniqv[which.max(tabulate(match(x, uniqv)))])
+   }
+   #=========================================================
+   # Function to calculate skewness
+   calc_skew <- function(x) {
+     n <- length(x)        # Sample size
+     mean_x <- mean(x)     # Mean of the data
+     sd_x <- sd(x)         # Standard deviation of the data
+     skewness <- sum((x - mean_x)^3) / (n * sd_x^3) # Skew
ness formula
+     return(skewness)
+   }
+   #========================================================
+   # Function to calculate kurtosis
+   calc_kurt <- function(x) {
+     n <- length(x)                # Sample size
+     mean_x <- mean(x)             # Mean of the data
+     sd_x <- sd(x)                 # Standard deviation of
the data
+     kurtosis <- sum((x - mean_x)^4) / (n * sd_x^4) # Kurt
osis formula
+     excess_kurtosis <- kurtosis - 3  # Subtract 3 to calc
ulate excess kurtosis
+   return(excess_kurtosis)
+   }
+   #=========================================================
========
+
+   # creating data frame to include descriptive stat calcu
lations
+   df <-data.frame(
+     Mean = mean(x),
+     Median= median(x),
+     Mode = getmode(x),
+     Max = max(x),
+     Min = min(x),
+     Range = max(x) - min(x),
+     Variance = var(x),
+     SD = sd(x),
+     Skewness = calc_skew(x),
```

```
+     Excess_Kurtosis = calc_kurt(x),
+     Kurtosis = calc_kurt(x) +3
+
+    )
+
+    #==============================
+    #
+    # calculating quarantiles separately
+    print("quantiles: ")
+
+    print(quantile(x))
+
+
+    print(paste("IQR=", IQR(x,na.rm = TRUE)))
+    # ===================================
+    return(df)
+ }
>
> y <- c(12, 10, 11, 15, 20, 13, 12, 19, 22, 17, 18)
>
> My_stat <- Descriptive(y)
[1] "quantiles: "
  0%  25%  50%  75% 100%
10.0 12.0 15.0 18.5 22.0
[1] "IQR= 6.5"
>
>
> t(My_stat) # t() is a function to transpose the data fram
e
               [,1]
Mean       15.3636364
Median     15.0000000
Mode       12.0000000
Max        22.0000000
Min        10.0000000
Range      12.0000000
Variance   16.4545455
SD          4.0564203
Skewness    0.1851876
Excess_Kurtosis -1.6201770
Kurtosis         1.3798230
>
> # Normality test - Shapiro Wilk
> shapiro.test(y)
```

```
        Shapiro-Wilk normality test

data:  y
W = 0.93817, p-value = 0.4993

>
> # Normality test - histogram (this generates the plot)
> hist(y, probability = TRUE, col = "lightblue", main = "Hi
stogram with Normal Curve")
>
> # Now add the normal distribution curve
> curve(dnorm(x, mean = mean(y), sd = sd(y)),
+       col = "red", lwd = 2, add = TRUE)
```

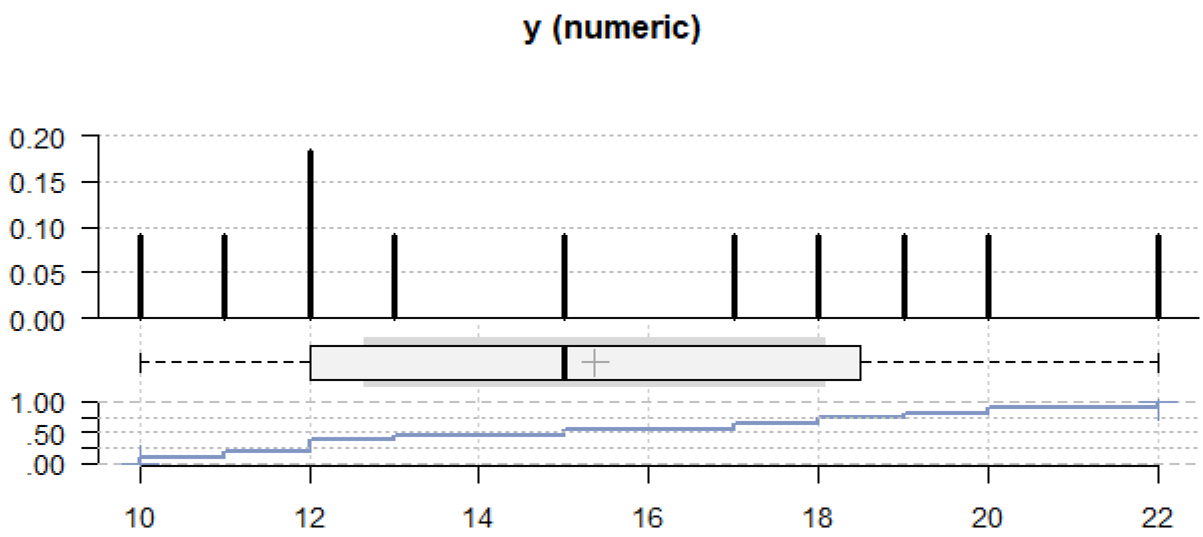


Calculating Descriptive statistics using specific R Packages

1. Using **DescTools** package: This package can provide a comprehensive descriptive statistics measurement.

```
> # Loading DescTools package
> library(DescTools)
```

```
>
> # The data set
> y <- c(12, 10, 11, 15, 20, 13, 12, 19, 22, 17, 18)
>
> Desc(y)
------------------------------------------------------------
------------
y (numeric)

  length       n     NAs  unique      0s    mean  meanCI'
      11      11       0      10       0   15.36    12.64
          100.0%    0.0%            0.0%            18.09

     .05     .10     .25  median     .75     .90      .95
   10.50   11.00   12.00   15.00   18.50   20.00    21.00

   range      sd   vcoef     mad     IQR    skew     kurt
   12.00    4.06    0.26    4.45    6.50    0.19    -1.62


    value  freq   perc  cumfreq  cumperc
1      10     1   9.1%        1     9.1%
2      11     1   9.1%        2    18.2%
3      12     2  18.2%        4    36.4%
4      13     1   9.1%        5    45.5%
5      15     1   9.1%        6    54.5%
6      17     1   9.1%        7    63.6%
7      18     1   9.1%        8    72.7%
8      19     1   9.1%        9    81.8%
9      20     1   9.1%       10    90.9%
10     22     1   9.1%       11   100.0%

' 95%-CI (classic)
```



2. Using **pastecs** package: This package doesn't provide mode and Histogram:

```
> # Loading pastecs package
> library(pastecs)
>
> # The data set
> y <- c(12, 10, 11, 15, 20, 13, 12, 19, 22, 17, 18)
>
> stat.desc(y)
     nbr.val     nbr.null       nbr.na          min          max
  11.0000000    0.0000000    0.0000000   10.0000000   22.0000000
       range          sum       median         mean      SE.mean
  12.0000000  169.0000000   15.0000000   15.3636364    1.2230567
CI.mean.0.95          var      std.dev     coef.var
   2.7251402   16.4545455    4.0564203    0.2640274
```

3. Using **psych** package: This package provides (mean, median, SD, SE, skewness, kurtosis, range).

```
> # Loading psych package
> library(psych)
>
> # The data set
> y <- c(12, 10, 11, 15, 20, 13, 12, 19, 22, 17, 18)
>
```

```
> describe(y)
   vars  n  mean   sd median trimmed  mad min max range skew kurtosis
  X1  1 11 15.36 4.06     15   15.22 4.45  10  22    12 0.19    -1.62
     se
X1 1.22
```

4. Using **skimr** package: This package provides (Mean, Median, SD, Skewness, Kurtosis, IQR, Range, , Histogram) measurements.

```
> # Loading skimr package
> library(skimr)
>
> # The data set
> y <- c(12, 10, 11, 15, 20, 13, 12, 19, 22, 17, 18)
>
> skim(y)
── Data Summary ────────────────────────
                           Values
Name                       y
Number of rows             11
Number of columns          1
_______________________
Column type frequency:
  numeric                  1
________________________
Group variables            None

── Variable type: numeric ──────────────────────────────────
──────────────
  skim_variable n_missing complete_rate mean   sd p0 p25 p5
0  p75 p100
1 data                  0             1 15.4 4.06 10  12  1
5 18.5   22
  hist
1 ▇▁▃▃
```

5. Using **Hmisc** package: this package provides (Mean, Median, SD, IQR, Range) measurements.

```
> # Loading Hmisc package
> library(Hmisc)
```

```
>
> # The data set
> y <- c(12, 10, 11, 15, 20, 13, 12, 19, 22, 17, 18)
>
> describe(y)
y
       n  missing distinct     Info     Mean  pMedian      Gmd      .05
      11        0       10    0.995    15.36    15.25    4.836     10.5
     .10      .25      .50      .75      .90      .95
    11.0     12.0     15.0     18.5     20.0     21.0

Value         10    11    12    13    15    17    18    19    20    22
Frequency      1     1     2     1     1     1     1     1     1     1
Proportion 0.091 0.091 0.182 0.091 0.091 0.091 0.091 0.091 0.091 0.091

For the frequency table, variable is rounded to the nearest 0
```

Note: `Gmd` (Gini Mean Difference):

- The Gini Mean Difference is a measure of variability or inequality in the dataset.
- It calculates the average absolute difference between all pairs of values in the dataset. It's less common but offers an alternative to standard deviation for measuring spread.

Summary of package based statistics

Package	psych	skimr	pastecs	Hmisc	DescTools
Function	describe()	skim()	stat.desc()	describe()	Desc()
Mean	✓	✓	✓	✓	✓
Median	✓	✓	✓	✓	✓
Mode	✗	✗	✗	✗	✓
SD	✓	✓	✓	✓	✓
Variance	✗	✗	✓	✗	✓
SE	✓	✗	✓	✗	✓
Skewness	✓	✓	✓	✗	✓
Kurtosis	✓	✓	✓	✗	✓
IQR	✗	✓	✓	✓	✓
Range	✓	✓	✓	✓	✓
Normality Tests	✗	✗	✓	✗	✓
Histogram	✗	✓	✗	✗	✓