Reviewing Python and R basics

Intro to Advanced Data Science - DSC 205 - Lyon College Spring'24

Table of Contents

1. README

- In this file, we review and practice plotting with base R.
- Code along using http://tinyurl.com/r-plots-org

2. Bonus: Math plots with R

- This section is prompted by a Calculus II exercise that I came across. Especially when dealing with trigonometric functions, it can be useful to plot them and get a visual on the problem.
- Plot a complicated function, e.g. $f(x) = sin^2(x)sin(2x)$:
  1. Base R (`package:base`) has trigonometric functions pre-loaded. Check `?sin` on the command line to see documentation and function names.
  2. To plot, all you need is the `plot` function.

  3. In Emacs, you can change the code block header arguments. The following metadata will store the result in a file `graph.png` and it will also link to the file in the `#+RESULTS` output. If you don't do this, the plot will appear in a separate window.

     ```
     #+begin_src R :results output graphics file :file graph.png
     ```

- Here's the code that opens the graph in a separate window:

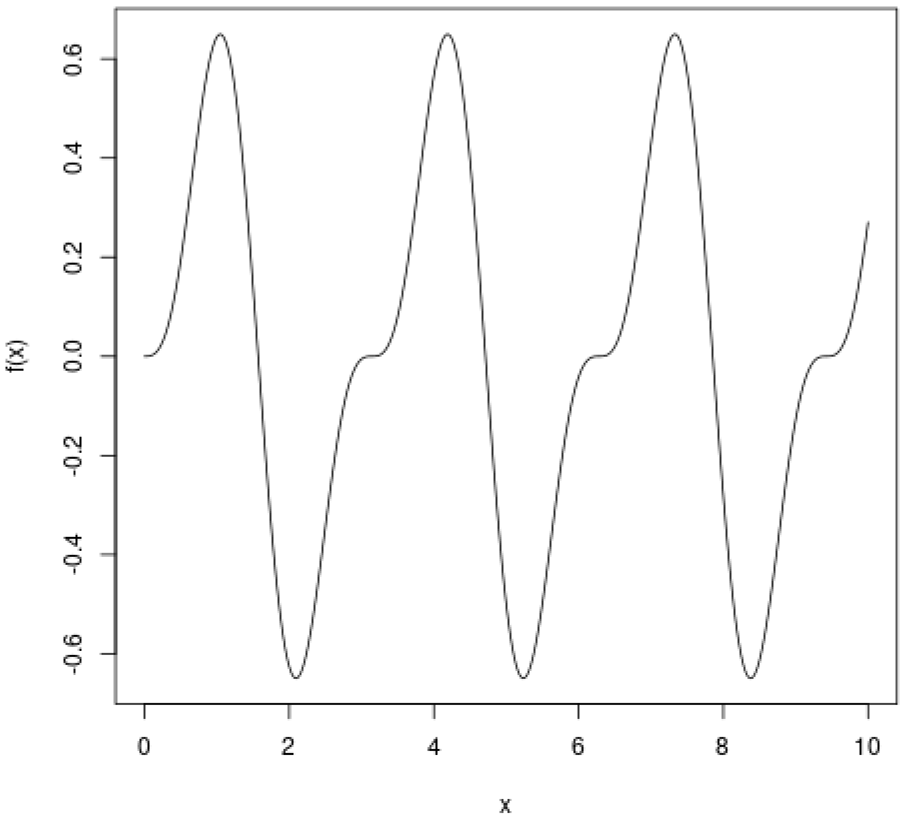
```
## function definition
f <- function(x) {
  return (sin(x)**2 * sin(2*x))
}
## arguments (independent variable)
x = seq(from = 0,to = 10,by = 0.01)

## plot as line plot
plot(x,f(x), type="l")
```

- Here's the code with the altered header line and with some customization:

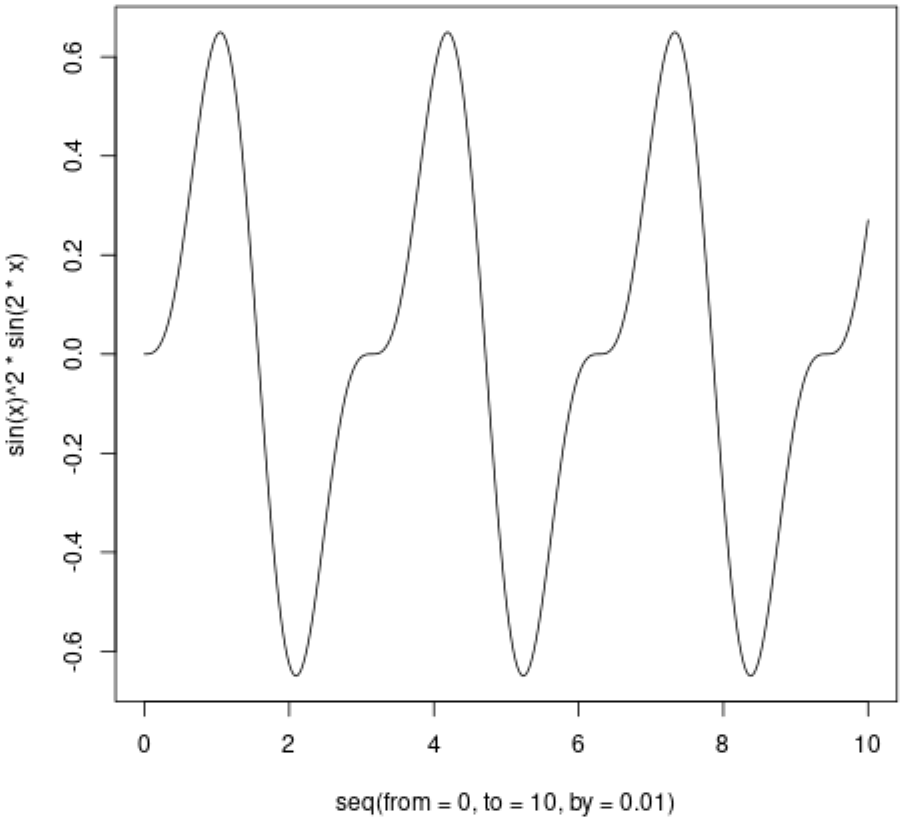
```
## function definition
f <- function(x) {
  return (sin(x)**2 * sin(2*x))
}
## arguments (independent variable)
x = seq(from = 0,to = 10,by = 0.01)

## plot as line plot
plot(x,f(x), type="l")
```



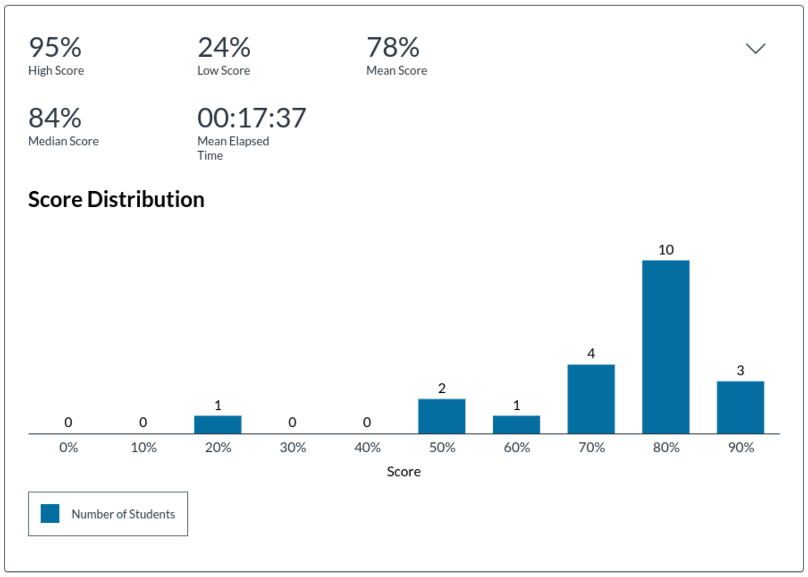
- Here is the minimal code without function, in one line:

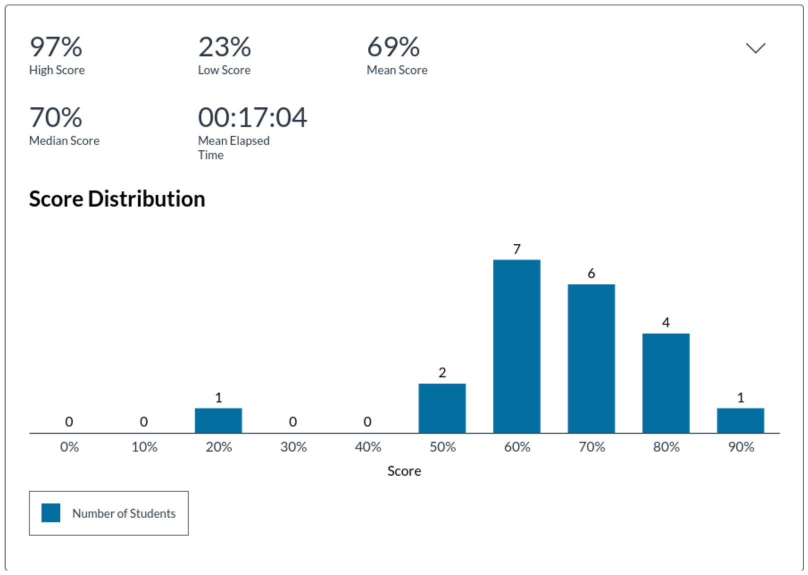
```
plot(x=seq(from = 0,to = 10,by = 0.01),
     sin(x)**2 * sin(2*x),
     type="l")
```



3. Problem

The LMS returns the following type of reports:





We want to:

1. Remake the histograms for both tests.
2. Display histograms separately as two graphs in one panel.
3. Display histograms together dodged or stacked in one panel.
4. Create boxplots and display them to compare results easily.

We'll also have to look at the necessity to remove missing values.

4. Data preparation

To not have to wade through previous scripts, store the URL in the variable `url`, import the data, and remove the last row as before, and the first column, which is irrelevant for our stats visualization:

```
url = 'http://tinyurl.com/grades-csv'  # storing URL in variable
df <- read.csv(url,header=TRUE) # reading CSV data from the web
df <- df[-which(df$ID==2190),-1] # removing the test user row
names(df) <- c('t1','t2')  # simplifying column names
df # printing resulting data.frame
```

```
      t1    t2
1   4.83 10.00
2  13.00 11.00
3  16.33  8.50
4  19.07 14.50
5  16.83 12.00
6  10.00  9.50
7  18.00 10.33
8  15.50 10.67
9  16.83 13.00
10 17.50  9.67
11 11.50 10.67
12 15.83 10.33
13 17.00 10.50
14    NA  3.50
15 16.33 10.17
16 17.50  9.50
17 17.50 12.50
18 16.74 12.00
19 17.33  8.17
20 16.83 11.33
21    NA  9.50
```

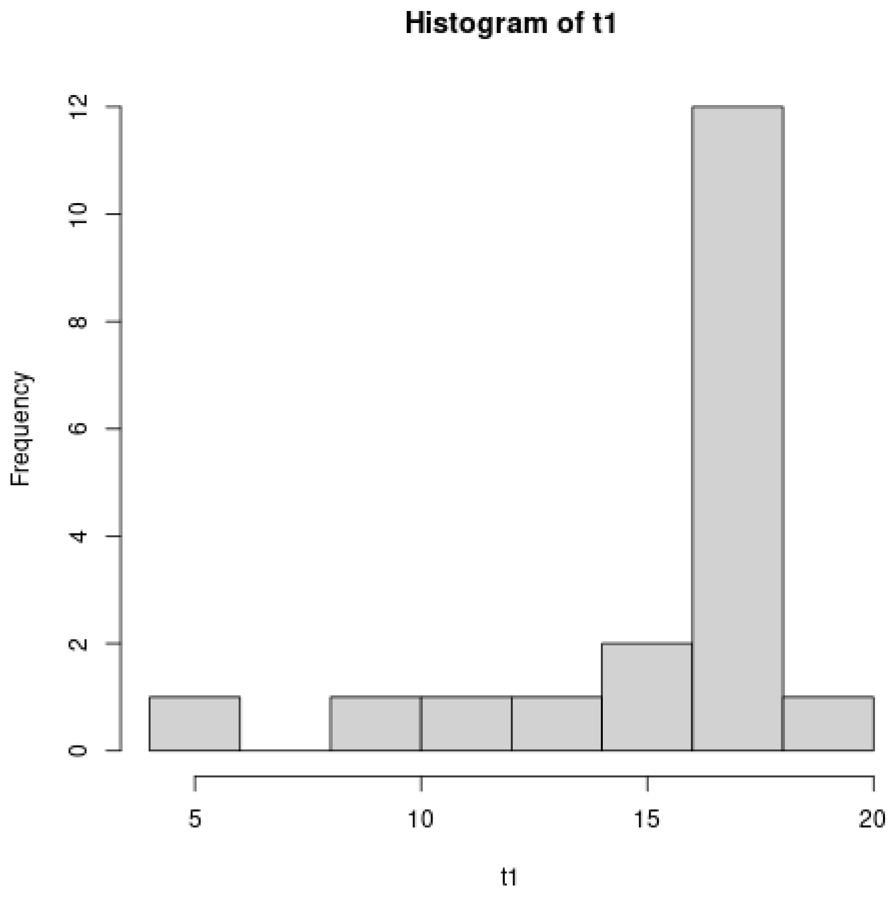
```
df <- read.csv(url,header=TRUE) # reading CSV data from the web
df <- df[-which(df$ID==2190),-1] # removing the last row and the 1st col
names(df) <- c('t1','t2')
str(df)
```

5. Histogram in R

A histogram plots frequencies over a continuous set of values.

- Let's make one for `df$t1`. For the header arguments, we need: `:results output graphics file :file histR_t1.png` - the graph will then be linked here and it will be saved to `t1.png`:

  ```
  t1 <- df$t1  # create R object from 1st column of data frame
  hist(t1) # t1 as a positional argument
  ```



- You can check from here if there's a file (on Windows, you need to replace `ls` by `DIR`):

```
# system('ls -l histR_t1.png')
```

- You can use `table` to display the frequencies of all values ('contingency table'):

```
table(t1)
t2 <- df$t2
table(t2)
```

- You can make a barplot of the contingency table data, and you can see the rough shape of the histogram already. Label the x-axis appropriately:

```
barplot(table(t1),
        xlab="test values") # same for df$t2
```

- What is the data structure of this table? Could you plug the table values straight into a histogram? What about the table frequencies?

```
tbl1 <- table(df$t1)
str(tbl1)   # `table` is its own data structure
```

- Try to make a histogram from the frequencies `as.integer(tbl1)` and name the x-axis appropriately:

```
freq <- as.integer(tbl1) # table 1 frequencies
hist(freq,xlab='frequency')  # visualized as histogram
```

- What about the table names?

```
names(tbl1)  # tbl1 = table(df$t1)
hist(names(tbl1))
```

- But you can directly make a barplot from the table: the `names` vector is accepted as x-argument:

```
barplot(tbl1)
```

- You can also convert the table to a dataframe and rename the `factor` values (the table `names`):

```
freq_df <- as.data.frame(tbl1)
str(freq_df)
names(freq_df)[1] <- "Names"
freq_df
```

- Alternatively, combine the `names` and the frequencies in a dataframe:

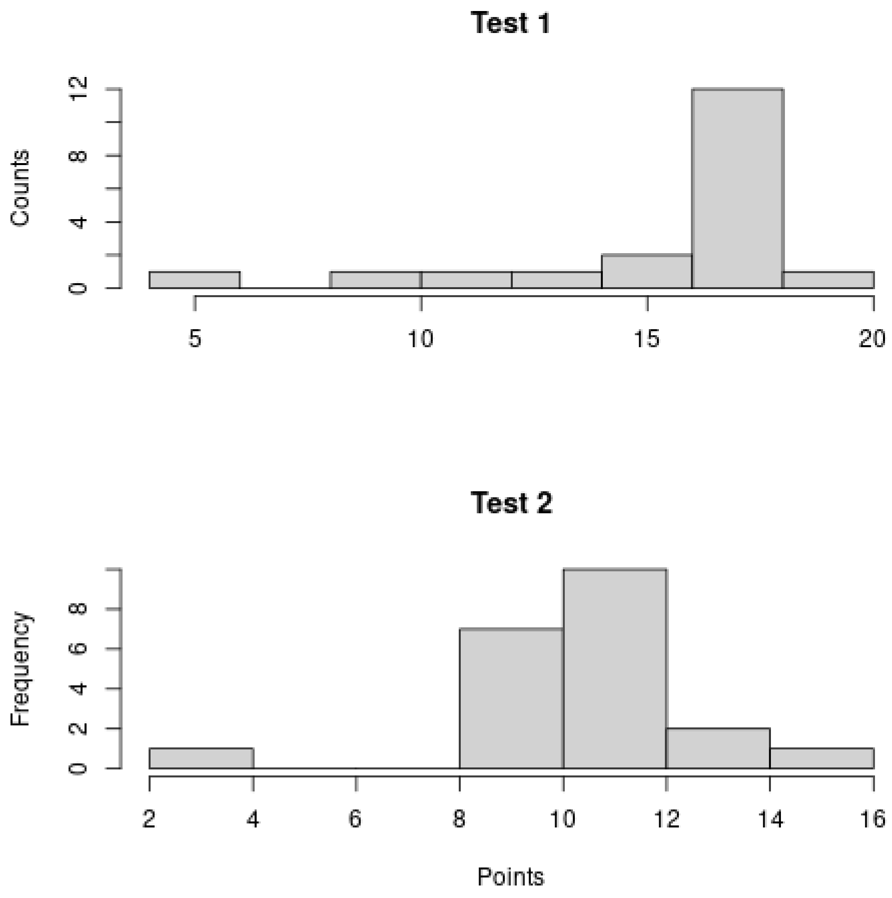
```
names <- names(tbl1)
names
freqs <- as.numeric(tbl1)
freqs
data.frame(Names=names, Freq=freqs)
```

- The binwidth is computed based on the range of the data by Sturges' formula (see help). Go back to the histogram code block and add the parameter `breaks`, then plot the histogram for different values of that parameter: 5, 10, 20.
- Check that we have `t1` and `t2`:

```
t1;t2
```

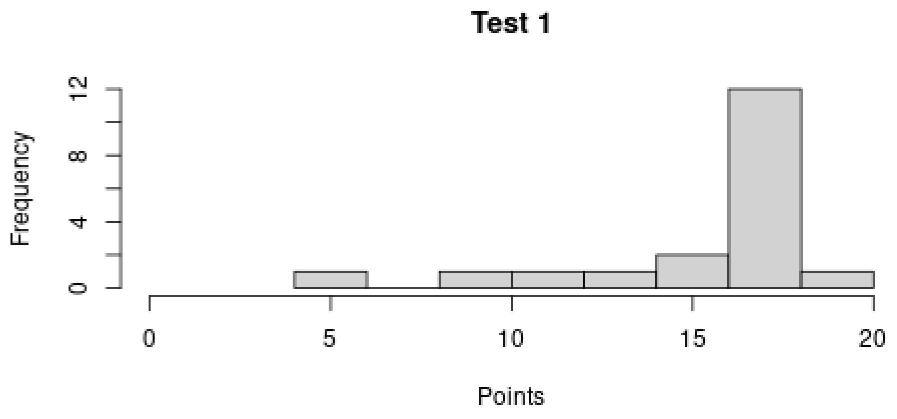
- To put the histograms for both tests in one panel, we use the `par` function and specify the number and orientation of sub-graphs with `mfrow` - notice that we now save to the file `histR_t1t2.png`:

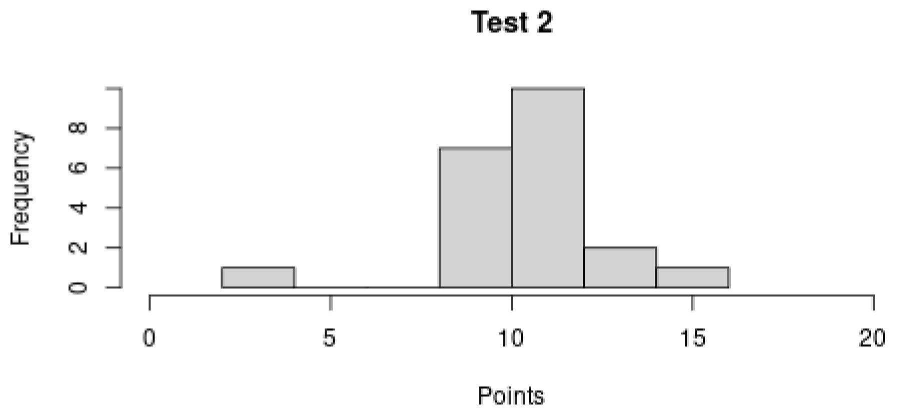
```
par(mfrow=c(2,1)) # create a 2 x 1 panel
hist(t1, main="Test 1", xlab="",ylab="Counts") # first histogram
hist(t2, main="Test 2", xlab="Points") # second histogram
par()
```



- Now, our scale program becomes obvious: the datasets have different maximum point values. This affects the x-axis. The quickest way to do this is to set the x-axis limits with `xlim`:

```
par(mfrow=c(2,1)) # create a 2 x 1 panel
hist(t1,main="Test 1",xlab="Points",xlim=c(0,20))
hist(t2,main="Test 2",xlab="Points",xlim=c(0,20))
par()
```



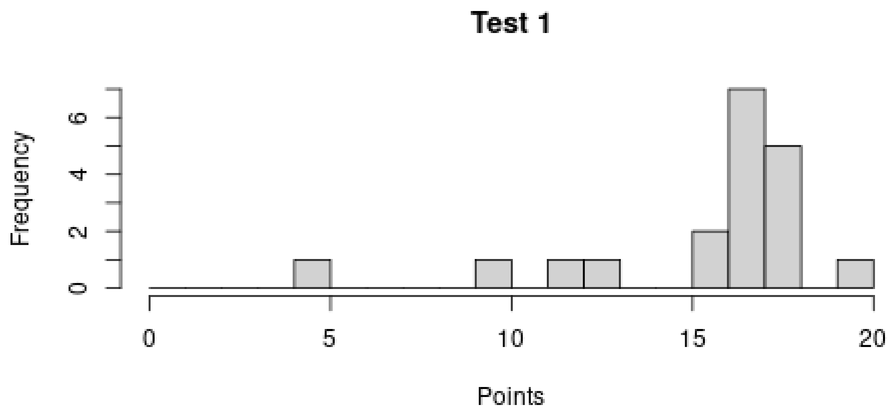


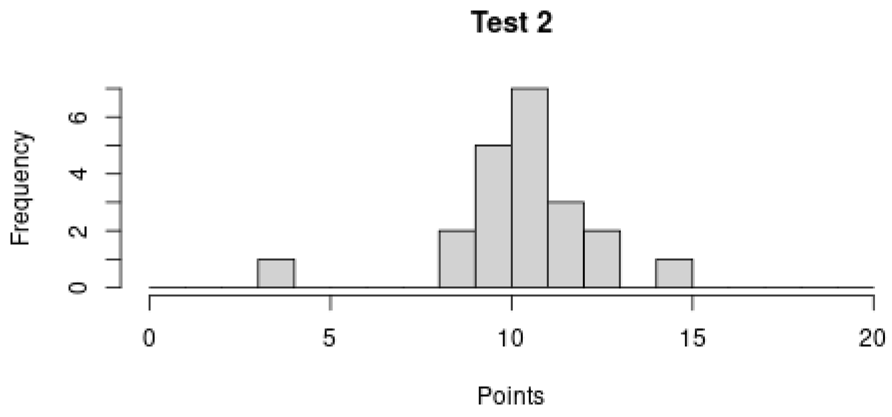
- This last result shows clearly that the peformance has decreased drastically between test 1 and test 2. A clearer picture will result from a boxplot (below).
- The boxplot is the visualization of the statistical `summary` function:

```
summary(t1)
summary(t2)
summary(data.frame(t1,t2))
```

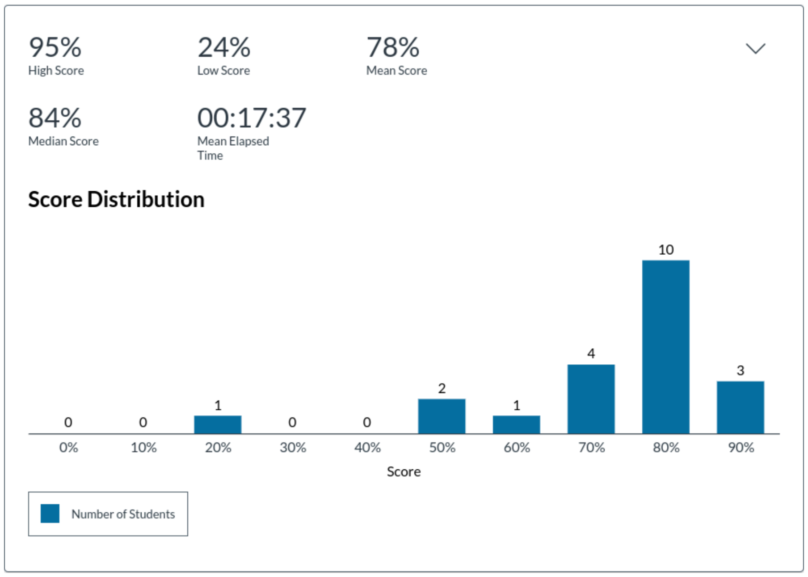
- One last improvement concerns the bin values: they are not close enough for a test where each of the questions has 1 point. To change that, you can set the `breaks` manually:

```
par(mfrow=c(2,1)) # create a 2 x 1 panel
hist(t1,   # short version: hist(t1,breaks=seq(0,20,1))
     breaks = seq(from=0,to=20,by=1),
     main="Test 1",
     xlab="Points",
     xlim=c(0,20))
hist(t2,
     breaks = seq(from=0,to=20,by=1),
     main="Test 2",
     xlab="Points",
     xlim=c(0,20))
par()
```





- Another issue (not for these data) could be if we have different number of participants. We might want to align the y-axis as well to make sure that we are looking at comparable datasets.
- However, we have not yet reproduced the LMS graphs: they show the results in percent, which automatically scales the point results.



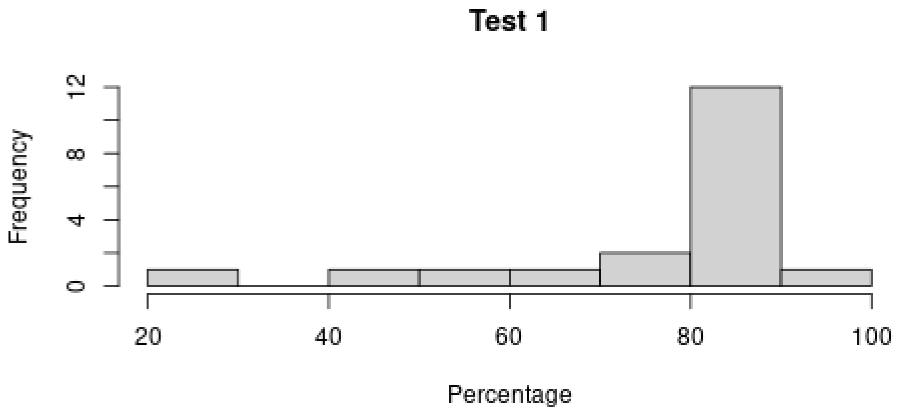
- We scale the vectors in the dataframe themselves, and we remove the 'id' column since we don't need it here at all. Vectorisation rules!

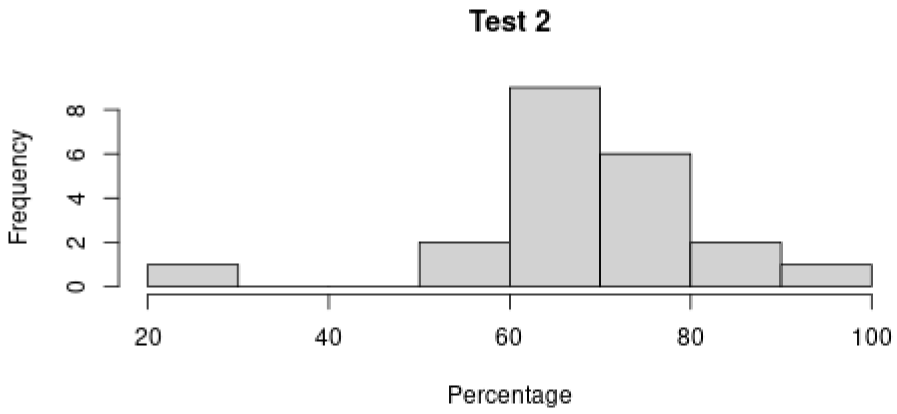
```
t1 <- (t1 / 20) * 100 # overwrite t1 with new % vector
t2 <- (t2 / 15) * 100 # overwrite t2 with new % vector
df
```

```
      t1    t2
1   4.83 10.00
2  13.00 11.00
3  16.33  8.50
4  19.07 14.50
5  16.83 12.00
6  10.00  9.50
7  18.00 10.33
8  15.50 10.67
9  16.83 13.00
10 17.50  9.67
11 11.50 10.67
12 15.83 10.33
13 17.00 10.50
14    NA  3.50
15 16.33 10.17
16 17.50  9.50
17 17.50 12.50
18 16.74 12.00
19 17.33  8.17
20 16.83 11.33
21    NA  9.50
```

- Now we redo the last plot but we do no longer need to worry about the limits or the breaks - make sure to change the name of the file:

```
par(mfrow=c(2,1)) # create a 2 x 1 panel
hist(t1,
     main="Test 1",
     xlab="Percentage")
hist(t2,
     main="Test 2",
     xlab="Percentage")
par()
```





- The original plots do not look like histograms but like barplots with exact values for the percentage, but as the data show, there are percentage ranges, which is why the histogram is more appropriate.

6. Boxplot in R

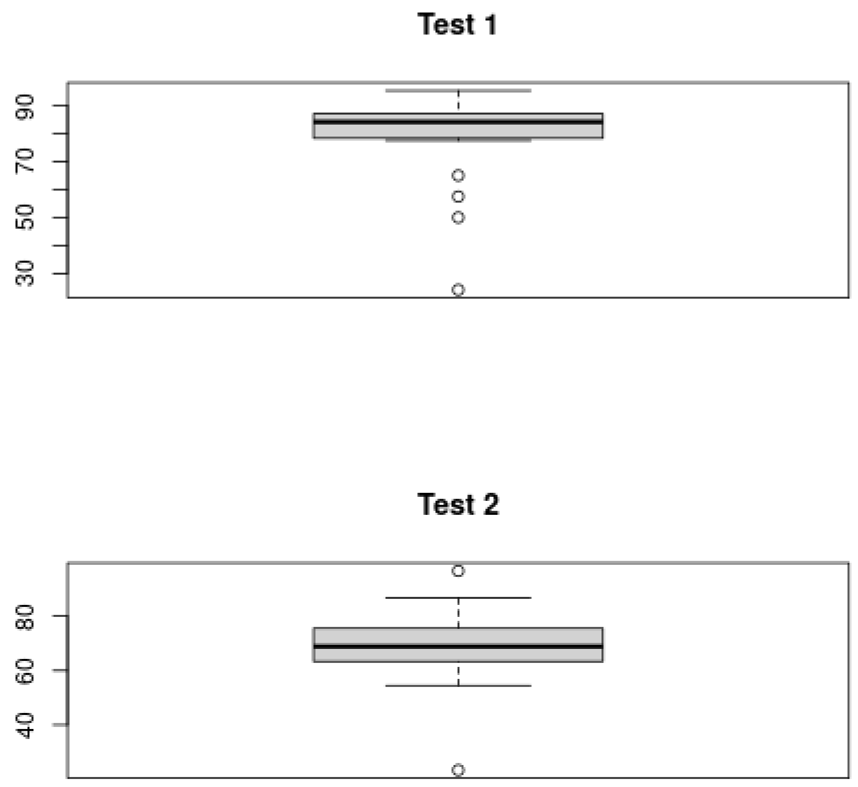
To make sure that this works, run 1:

```
url = 'http://tinyurl.com/grades-csv'  # storing URL in variable
df <- read.csv(url,header=TRUE) # reading CSV data from the web
df <- df[-which(df$ID==2190),-1] # removing the test user row
names(df) <- c('t1','t2')  # simplifying column names
df # printing resulting data.frame  # this runs the `data` code block way earlier
```

A boxplot is a graph that illustrates the statistical `summary` results.

- The creation of panels and subpanels, the customization and the scaling carries over from the last histogram, since these are graphical parameters. We first create a quick and dirty boxplot, and customize in the next step:

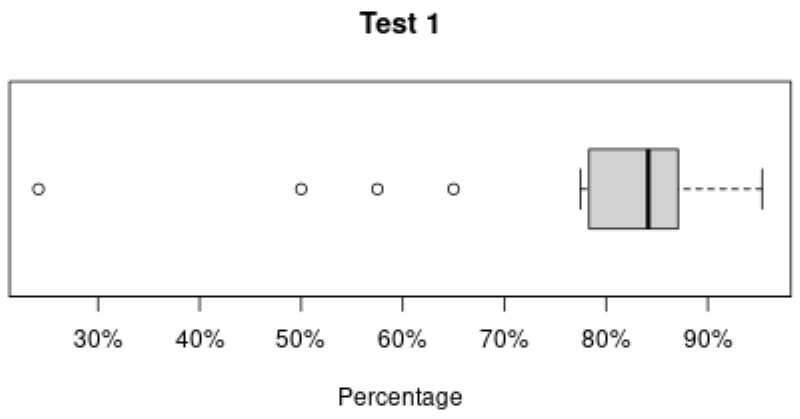
```
par(mfrow=c(2,1)) # create a 2 x 1 panel
boxplot(t1,
        main="Test 1")
boxplot(t2,
        main="Test 2")
par()
```

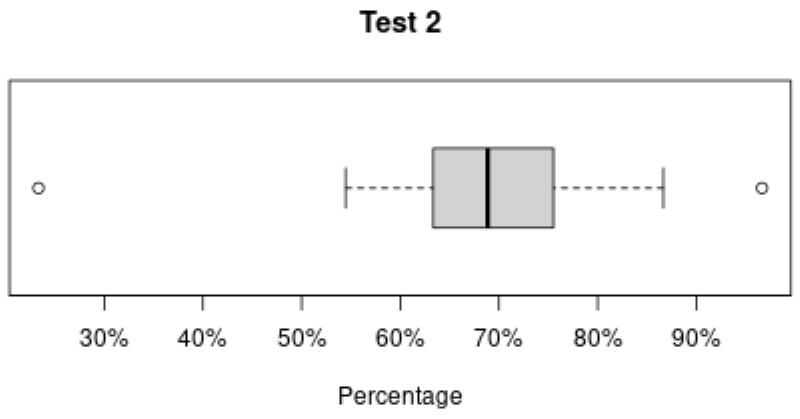


- These results are hard to compare. We're going to:

    1. turn the boxplots on their side with the parameter `horizontal=TRUE`
    2. label the x-axis as before with `xlab`
    3. remove the standard x-axis annotation with `xaxt='n'`
    4. redefine the x-axis ticks with the `axis` function.

```
par(mfrow=c(2,1)) # create a 2 x 1 panel
boxplot(t1,
        horizontal=TRUE, # rotate plot by 90 degrees
        main="Test 1",
        xlab="Percentage",
        xaxt='n') # Suppress default x-axis
## redraw axis data
axis(side=1,
     at=seq(0,100,by=10),
     labels=paste0(seq(0, 100, by=10), "%")) # Add custom x-axis
boxplot(t2,
        horizontal=TRUE,
        main="Test 2",
        xlab="Percentage",
        xaxt='n') # Suppress default x-axis
## redraw axis data
axis(side=1,
     at=seq(0,100,by=10),
     labels=paste0(seq(0, 100, by=10), "%")) # Add custom x-axis
par()
```





7. Summary

We covered:

- The practice of plotting mathematical functions, particularly trigonometric functions, in R, which serves as a valuable tool for visualizing complex equations.
- Addressing real-world data science problems, such as making histograms and creating boxplots, to compare test results visually. This section is crucial for developing skills in data analysis and interpretation.

- Data preparation techniques, which involve importing, cleaning, and simplifying data for effective statistical visualization. This foundational skill is essential for any data science endeavor.

8. Glossary

TERM	DEFINIION
`"package:base"`	Name of base package in `search()`
`NULL`	Represents a non-existing object
`which`	Extracts indices (Boolean argument)
`plot`	Used for creating a graph in R. Generic
`hist`	Generates a histogram in R (numeric distribution)
`boxplot`	Creates a boxplot in R (numeric distribution)
`seq`	Generates regular sequences of numbers in R (as vectors)
`table`	Creates a contingency table of the counts of categorical values
`barplot`	Draws a bar plot in R to visualize categoric data)
`str`	Displays the internal structure of an R object.
`names`	Gets or sets the names attribute of an object.
`as.integer`	Converts data types to integer in R.
`as.data.frame`	Converts an object to a data frame in R.
`read.csv`	Reads a file in table format and creates a data frame from it.
`axis`	Adds an axis to a plot with specific attributes like side, at, and labels.
`par`	Used to set or query graphical parameters in R.
`mfrow`	Sets panels for multi-panelled plots, e.g. `mfrow=c(1,2)`

Date: Time-stamp: <2024-02-09 Fri 07:23>

Author: Marcus Birkenkrahe (pledged)

Created: 2024-03-04 Mon 12:52

Validate