Unit testing in R using testthat library Exercises

testthat is a testing framework developed by Hadley Wickham, which makes unit testing easy for developers.

Test scripts developed can be re-run after debugging or making changes to the functions without the hassle of developing the code for testing again.

testthat has a heirarchical structure made up of expectations, tests and contexts.
Visit this link to know more.

You should be familiar with creation of functions in R to know how this testing framework works.

Answers to the exercises are available here.

Exercise 1

Install and load the package **testthat** using the appropriate function.

Exercise 2

expect_that() is the function that makes the binary assertion of whether or not the value is as expected.
expect_that(x,equals(y)) reads as “it is expected that ‘a’ will be equal to ‘b'”.

Use this function to see if 5*2 equals 10

Learn more about Hadley Wickhams usefull packages in the online course R Data Pre-Processing & Data Management – Shape your Data!. In this course you will learn how to work with:

- tidyr, cleaning your data
- dplyr, shape your data
- And much more

Exercise 3

The function equals() checks for equality with a numerical tolerance. Let's see what that tolerance level is

Use appropriate function to see if 5*2 equals 10 + (1e-7).
Does the test fail?
If no, change the value to 1e-6 and see what happens.

Exercise 4

To exactly match the values is_identical_to() can be used instead of equals()

Using the appropriate function, check if 2*2 is identical to 4 + (1e-8)

Please check the documentation of this package to learn more about the available functions.

Exercise 5

Let us create a function (m) to multiply two numbers (two arguments) and check if it throws an error with character input arguments.

Check if m("2″,"3") throws an error "non-numeric argument to binary operator"

Exercise 6

Now that we know how to check for expectations, let us create

tests.
Test is a collection of expectations, where these expectations test a single item of the functionality of a process.

test_that() is the function that encapsulates the description and the code to test each expectation.

The first argument is the description and the second argument is a collection of expectations.

Create a test for function ‘m’ with description “Testing multiplication function” and add a few scenarios to it.
1. Check if m(2,3) equals 6
2. Check if m(2,c(2,3)) equals c(4,6)
3. Check if m(2,”3″) throws an error “non-numeric argument to binary operator”

Exercise 7

The User can write his own expectation using the expect() function. This expectation should compare the input value and the expectation and report the result.
The syntax to write one is as below.

```
custom_expectation <- function() {function(x) {expectation(condition, "Failure message")}}
```

Now, write an expectation is_greater_10() to check if a number is greater than 10

Exercise 8

Use the expectation defined above to check if 9 is greater than 10.

Exercise 9

tests can be put together in a file and run at once. Write tests of your choice and save them in a file.
Use the function test_file() to run all the tests in the file.

Exercise 10

Test files in a directory can be run at once using the function test_dir().
Create multiple test files and save them in a directory. Run all the tests at once using the function.

Unit testing in R using testthat library – Solutions

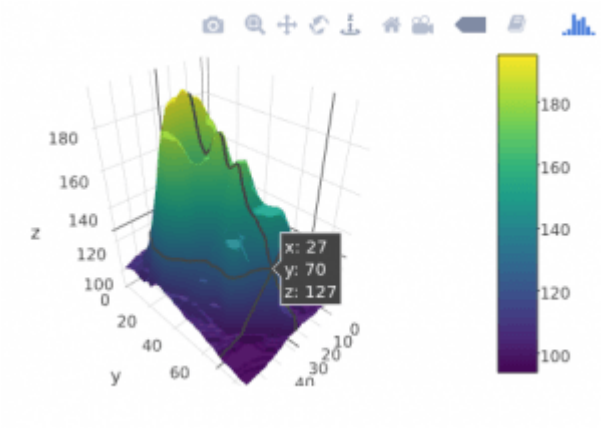
Below are the solutions to these exercises on unit testing in R.

```
####################
#                  #
#    Exercise 1    #
#                  #
####################

install.packages("testthat")
library(testthat) # loading the package


####################
#                  #
#    Exercise 2    #
#                  #
####################

#Passes
expect_that(5*2, equals(10))

####################
#                  #
```

```
#    Exercise 3    #
#                  #
####################

#Passes
expect_that(5*2, equals(10+ 1e-7))
#fails
expect_that(5*2, equals(10+ 1e-6))

####################
#                  #
#    Exercise 4    #
#                  #
####################

expect_that(2*2, is_identical_to(4+ 1e-8))

####################
#                  #
#    Exercise 5    #
#                  #
####################

m <- function(x,y){return(x*y)}
expect_that(m("2","3"), throws_error("non-numeric argument to
binary operator"))


####################
#                  #
#    Exercise 6    #
#                  #
####################

test_that("Testing multiplication function",{
  expect_that(m(2,3), equals(6))
  expect_that(m(2,c(2,3)), equals(c(4,6)))
   expect_that(m(2,"3"), throws_error("non-numeric argument to
binary operator"))
})
```

```
####################
#                  #
#    Exercise 7    #
#                  #
####################

is_greater_10 <- function() {
  function(x){
    expect(x > 10, "Not greater than 10")
  }
}

####################
#                  #
#    Exercise 8    #
#                  #
####################

expect_that(9, is_greater_10())

####################
#                  #
#    Exercise 9    #
#                  #
####################

test_file(<Path to the file with test script>)

####################
#                  #
#    Exercise 10   #
#                  #
####################

test_dir(<Path to the directory with test files>)
```

Plotly : Advanced plots and features



Plotly is a d3 based graphing library used to produce interactive and high quality graphs in R. In the following exercises, we will look at some advanced plots and features avaialable in the package. Please note that the list here is not exhaustive.
We will use datasets available in base R packages.

You are expected to have the knowledge of generating basic plots using plotly package before attempting this exercise set. It is recommended that you go through this exercise set to test your knowledge on plotly basics.

Answers to the exercises are available here.

Refer to this link for more help on the plotly functions.
For a quick info on the functions and arguments required to build basic plotly plots, refer to this cheat sheet.

Learn more about Plotly in Section 17 *Interactive Visualizations with Plotly* of the online course Data Science and Machine Learning Bootcamp with R.

Exercise 1

Install and load the latest version of plotly package.

Generate separate histograms for the first four columns of iris dataset and save the plots in objects p1, p2, p3 and p4.
HINT: Use plot_ly() function with x argument and type="histogram". Use name argument to give appropriate name

for the trace.

Exercise 2

a. Use subplot() function to generate a plot with all the plot objects from previous exercise as the arguments.

b. Use the nrows argument to arrange 2 plots per row.

Exercise 3

a. Generate a scatter plot for the iris dataset with first column on the x-axis and second column on the y-axis. Save the plot object.

b. Generate a 2d histogram using the add_histogram2d() function.Save the plot object.
HINT: Use the function plot_ly() with the same x and y arguments and pass the plot object to the 2-d histogram function.

Exercise 4

Generate a subplot with the scatter plot and the 2-d histogram created in the previous exercise.

Notice how the scatter plot can be represented in a more interesting way. Cells in the 2-d histogram are binned and represented with the color on the scale based on the cell population/density.

Exercise 5

Set the value of shareX and shareY arguments in the subplot() function to scale the plots to the same range of x and y.

Exercise 6

Now, let us build a 3-d surface plot. The syntax to build such plot is as below.
```
plot_ly(z = matrix(1:100, nrow = 10)) %>% add_surface()
```

Click, hold and drag the cursor to see the plot surface.

Build a 3-d surface plot using the volcano dataset available in the base R distribution.

Exercise 7

Let’s look at few helpful and commonly used arguments from the layout() function.

Create and save a scatter plot object with first and second columns of iris dataset as x and y arguments respectively. Colour the markers based on the species

Exercise 8

a. Add an appropriate title to the plot using the layout function and title argument.
b. Add an appropriate x-axis label using the xaxis argument. xaxis takes a list of attribute values. Refer to the R reference page for more help.
c. Add an appropriate y-axis label.

Exercise 9

a. Use the range attribute in the list of values given to the xaxis argument to set the x-axis range from 1 to 10.
b. Similarly, set the y-axis range from 1 to 5.

Exercise 10

Try different layout options to further customize the font, axes etc… of the plot.

Plotly : Advanced plots and features – Solutions

Below are the solutions to these exercises on Plotly advanced features.

Learn more about Plotly in Section 17 *Interactive Visualizations with Plotly* of the online course Data Science and Machine Learning Bootcamp with R.

```
####################
#                  #
#    Exercise 1    #
#                  #
####################

library(plotly) # loading the package
p1 <- plot_ly(x = iris[,1], type = "histogram", name =
names(iris)[1])
p2 <- plot_ly(x = iris[,2], type = "histogram", name =
names(iris)[2])
p3 <- plot_ly(x = iris[,3], type = "histogram", name =
names(iris)[3])
p4 <- plot_ly(x = iris[,4], type = "histogram", name =
names(iris)[4])


####################
#                  #
#    Exercise 2    #
#                  #
####################

subplot(p1, p2, p3, p4)
subplot(p1, p2, p3, p4, nrows = 2)

####################
#                  #
```

```
#    Exercise 3    #
#                  #
####################

p <- plot_ly(x  = iris[,1], y = iris[,2], mode = "markers",
type = "scatter")
q <- add_histogram2d(plot_ly(x = iris[,1], y = iris[,2]))

####################
#                  #
#    Exercise 4    #
#                  #
####################

subplot(p,q)

####################
#                  #
#    Exercise 5    #
#                  #
####################

subplot(p,q, shareX = TRUE, shareY = TRUE)

####################
#                  #
#    Exercise 6    #
#                  #
####################

plot_ly(z = volcano) %>% add_surface()

####################
#                  #
#    Exercise 7    #
#                  #
####################

p <- plot_ly(x= iris[,1], y = iris[,2], type="scatter", mode =
"markers", color = iris[,5])
```

```
####################
#                  #
#    Exercise 8    #
#                  #
####################

p %>% layout(title = "Scatter plot")
p %>% layout(title = "Scatter plot", xaxis = list(title =
"First column"))
p %>% layout(title = "Scatter plot", xaxis = list(title =
"First column"), yaxis = list(title = "Second column"))

####################
#                  #
#    Exercise 9    #
#                  #
####################

p %>% layout(title = "Scatter plot", xaxis = list(title =
"First column", range = c(1,10)), yaxis = list(title = "Second
column"))
p %>% layout(title = "Scatter plot", xaxis = list(title =
"First column", range = c(1,10)), yaxis = list(title = "Second
column", range = c(1,5)))

####################
#                  #
#    Exercise 10   #
#                  #
####################

#Learn more about the layout options here
'https://plot.ly/r/#layout-options'. Layout options can also
be found in the R reference link.
```

Getting started with Plotly: basic Plots

Plotly is a d3 based graphing library used to produce interactive and high quality graphs in R. In the following exercises, we will look at the basic plots' syntax and some basic features in the plotly functions.
We will use datasets available in base R packages.

Refer to the documentation of the plotly packages when in need of help with the functions and arguments.
Also, refer to this link for figure references

Installation of the package is straight-forward as it is available on CRAN.
Use this command install.packages('plotly') to install the package. Do not forget to load the package before trying out the exercises.

Answers to the exercises are available here.

Learn more about Plotly in Section 17 *Interactive Visualizations with Plotly* of the online course Data Science and Machine Learning Bootcamp with R.

Exercise 1

a. Generate a histogram for the first column of iris dataset using the plot_ly() function.
For the first run, supply only one argument to the function and see the message it returns.
identify the argument that you need to include to specify a chart type.

Exercise 2

Identify the argument to specify number of bins and generate a histogram with 20 bins using the same data.

Notice the behaviour of plot on mouse hover. There are also a few options available on the right top corner of the plot to zoom, save the plot etc.

Exercise 3

a. Generate a scatter plot for the first two columns of the iris dataset. Identify and use the appropriate arguments to get the plot with first column on the x axis and second column on y axis.
b. Using the color argument, specify the color such that data points are colored based on the Species column. HINT: color argument takes a vector of same length of number of data points data points with the level info.

Exercise 4

To the same code in the previous exercise, use the size argument to specify the markers' size based on the third column of the iris dataset.

Exercise 5

a. Generate a line plot for 100 values of random normal distribution with the default mean and standard deviation.
HINT: Use index values on x axis. Use type="scatter" and mode="lines" to get a line chart

Exercise 6

b. Save the previous plot in an object p. Use layout function to add an appropriate title to the plot.

Exercise 7

To learn how to generate a bar plot, we will simulate sample data and look at how to prepare the data before plotting.

Run the below code to generate the data.
cat <- c(rep("A", 2), rep("B", 4), rep("C", 8))

Data has to be in the form of Levels and their Counts . Therefore, using table() function to summarize and as.data.frame to create a data frame.
df <- as.data.frame(table(cat))

a. Now, using type="bar" and the appropriate x and y arguments, create a bar graph.
b. Add color to each bar using color argument, based on the categorical levels.

Exercise 8

Pie chart is an alternative way of representing categorical data, but, when the levels in the data are more, bar chart is preferred.
Data has to be prepared in the same way as we do it for bar chart. The arguments vary a little. Instead of x and y, we use labels and values.

Generate a pie chart using the same data from the previous exercise and appropriate arguments.

Notice how the tick marks and the axes appear behind the chart.

Exercise 9

For the first column in the iris dataset, generate a box plot using the box trace type.
HINT:For a simple box plot, just the y argument and type argumnets are given.

Exercise 10

Add another argument to the code used in the previous exercies to generate mutliple box plots for the first column of iris dataset, where each box corresponds to data of particular

Species.
HINT : Remember the argument we used to specify the color for each species in Exercise 3.

In the coming exercise sets on plotly, we will work on some advanced plots and see how to use argumnets from the layout function to manipulate the plot layout and settings.

Getting started with Plotly: basic Plots – Solutions

Below are the solutions to these exercises on plotly basic plots

Learn more about Plotly in Section 17 *Interactive Visualizations with Plotly* of the online course Data Science and Machine Learning Bootcamp with R.

```
library(plotly)

###############
#             #
# Exercise 1  #
#             #
###############

plot_ly(x = iris[,1])
#type = "histogram" is the argument to be included to specify the chart type

###############
#             #
# Exercise 2  #
```

```
#             #
###############

#nbinsx is the argument to specify number of bins.

plot_ly(x = iris[,1], type = "histogram", nbinsx = 20)

###############
#             #
# Exercise 3  #
#             #
###############

plot_ly(x = iris[,1], y = iris[,2], type = "scatter", mode =
"markers")
plot_ly(x = iris[,1], y = iris[,2], type = "scatter", mode =
"markers", color = iris[,"Species"])

###############
#             #
# Exercise 4  #
#             #
###############

plot_ly(x = iris[,1], y = iris[,2], type = "scatter", mode =
"markers", color = iris[,"Species"], size = iris[,3])

###############
#             #
# Exercise 5  #
#             #
###############

plot_ly(x = 1:100, y = rnorm(100), type = "scatter", mode =
"lines")

###############
#             #
# Exercise 6  #
#             #
###############
```

```
p <- plot_ly(x = 1:100, y = rnorm(100), type = "scatter", mode
= "lines")
layout(p , title = "Line plot")

###############
#             #
# Exercise 7  #
#             #
###############
cat <- c(rep("A", 2), rep("B", 4), rep("C", 8))
df <- as.data.frame(table(cat))

plot_ly(x = df[,1], y = df[, 2], type = "bar")
plot_ly(x = df[,1], y = df[, 2], type = "bar", color = df[,1])

###############
#             #
# Exercise 8  #
#             #
###############

plot_ly(labels = df[,1], values = df[,2], type = "pie")

###############
#             #
# Exercise 9  #
#             #
###############

plot_ly(y =iris[,1], type = "box")

###############
#             #
# Exercise 10  #
#             #
###############

plot_ly(y =iris[,1], type = "box", color = iris[,"Species"])
```