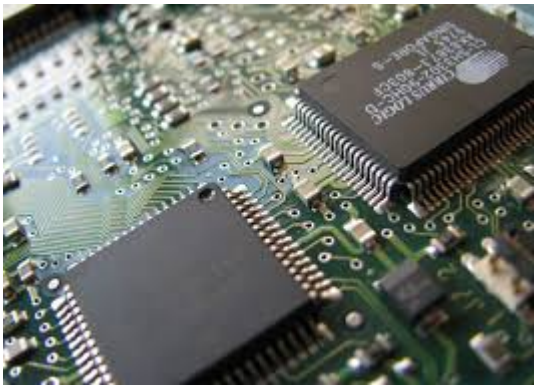
Parallel Computing Exercises: Snow and Rmpi (Part-3)



The foreach statement, which was introduced in the previous set of exercises of this series, can work with various parallel backends. This set allows to train in working with backends provided by the snow and Rmpi packages (on a single machine with multiple CPUs). The name of the former package stands for “Simple Network of Workstations”. It can employ various parallelization techniques; socket clustering is used here. The latter one is an R’s wrapper for the MPI (Message-Passing Interface), which is another paralellization technique.

The set also demonstrates that inter-process communication overhead has to be taken into account when preparing to use parallelization. If short tasks are run in parallel the overhead can offset the gains in performance from using multiple CPUs, and in some cases execution can get even slower. For parallelization to be useful, tasks that are run in parallel have to be long enough.

The exercises are based on an example of using bootstrapping to estimate the sampling distribution of linear regression coefficients. The regression is run multiple times on different sets of data derived from an original sample. The size of each derived data set is equal to the size of the original sample, which is possible because the sets are produced by random sampling with replacement. The original sample is taken from the InstEval data set, which comes with

the lme4 package, and represents lecture/instructor evaluations by students at the ETH. The estimated distribution is not analyzed in the exercises.
The exercises require the packages foreach, snow, doSNOW, Rmpi, and doMPI to be installed.
IMPORTANT NOTE: the Rmpi package depends on an MPI software, which has to be installed on the machine separately. The software can be the following:

- Windows: either the Microsoft MPI, or Open MPI library (the former one can be installed as an ordinary application).
- OS X/macOS: the Open MPI library (available through Homebrew).
- Linux: the Open MPI library (look for packages named libopenmpi (or openmpi, lib64openmpi, or similar), as well as libopenmpi-dev (or libopenmpi-devel, or similar) in your distribution's repository).

The zipped data set can be downloaded here. For other parts of the series follow the tag parallel computing.
Answers to the exercises are available here.

Exercise 1
Load the data set, and assign it to the data_large variable.

Exercise 2
Create a smaller data set that will be used to compare how parallel computing performance depends on the size of the task. Use the sample function to obtain a random subset from the loaded data. Its size has to be 10% of the size of the original dataset (in terms of rows). Assign the subset to the data_small variable.
For reproducibility, set the seed to 1234.
Print the number of rows in the data_large and data_small data sets.

Exercise 3

Write a function that will be used as a task in parallel computing. The function has to take a data set as an input, and do the following:

1. Resample the data, i.e. obtain a new sample of data based on the input data set. The number of rows in the new sample has to be equal to the one in the input data set (use the sample function as in the previous exercise, but change parameters to allow for resampling with replacement).
2. Run a linear regression on the resampled data. Use y as the dependent variable, and the others as independent variables (this can be done by using the formula y ~ . as an argument to the lm function).
3. Return a vector of coefficients of the linear regression.

Learn more about optimizing your workflow in the online course Getting Started with R for Data Science. In this course you will learn how to:

- efficiently organize your workflow to get the best performance of your entire project
- get a full introduction to using R for a data science project
- And much more

Exercise 4

Let's test how much time it takes to run the task multiple times sequentially (not in parallel). Use the foreach statement with the %do% operator (as discussed in the previous set of exercises of this series) to run it:

- 10 times with the data_large data set, and
- 100 times with the data_small data set.

Use the rbind function as an argument to foreach to combine

the results.
In both cases, measure how much time is spent on execution of the task (with the system.time function). Theoretically, the length of time spent should be roughly the same because the total number of rows processed is equal (it is 100,000 rows: 10,000 rows 10 times in the first case, and 1,000 rows 100 times in the second case), and the row length is the same. But is this the case in practice?

Exercise 5
Now we'll prepare to run the task in parallel using 2 CPU cores. First, we'll use a parallel computing backend for the foreach statement from the snow package. This requires to steps:

1. Make a cluster for parallel execution using the makeCluster function from the snow package. Pass two arguments to this function: the size of the cluster (i.e. the number of CPU cores that will be used in computations), and the type of the cluster ("SOCK" in this case).
2. Register the cluster with the registerDoSNOW function from the doSNOW package (which provides a foreach parallel adapter for the 'snow' package).

Exercise 6
Run the task 10 times with the large data set in parallel using the foreach statement with the %dopar% operator (as discussed in the previous set of exercises of this series). Measure the time spent on execution with the system.time function.
When done, use the stopCluster function from the snow package to stop the cluster.
Is the length of execution time smaller comparing to the one measured in Exercise 4?

Exercise 7
Repeat the steps listed in Exercise 5 and Exercise 6 to run

the task 100 times using the small data set.
What is the change in the execution time?

Exercise 8
Next, we'll use another parallel backend for the foreach function: the one that is provided by the Rmpi package (R's wrapper to Message-Passing Interface), and accessible through an adapter from the doMPI package. From the user perspective, it differs from the snow-based backend in the following ways:

- as mentioned above, additional software has to be installed for this backend to work (either (a) the openmpi library, available for Windows, macOS, and Linux, or (b) the Microsoft MPI library, which is available for Windows,
- when an mpi cluster is created, it immediately starts using CPUs as much as it can,
- when the work is complete, the mpi execution environment has to be terminated; if terminated, it can't be relaunched without restarting the R session (if you try to create an mpi cluster after the environment was terminated, the session will be aborted, which may result in a loss of data; see Exercise 10 for more details).

In this exercise, we'll create an mpi execution environment to run the task using 2 CPU cores. This requires actions similar to the ones performed in Exercise 5:

1. Make a cluster for parallel execution using the startMPIcluster function from the doMPI package. This function can take just one argument, which is the number of CPU cores to be used in computations.
2. Register the cluster with the registerDoMPI function from the doMPI package.

After creating a cluster, you may check whether the CPU usage on your machine increased using Resource Monitor (Windows),

Activity Monitor (macOS), top or htop commands (Linux), or other tools.

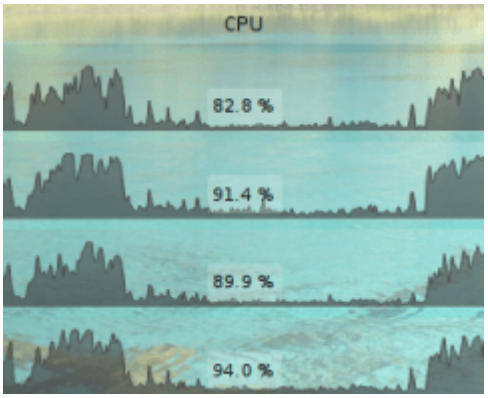
Exercise 9
Stop the cluster created in the previous exercise with the closeCluster command from the doMPI package. The CPU usage should fall immediately.

Exercise 10
Create an mpi cluster again, and use it as a backend for the foreach statement to run the task defined above:

- 10 times with the data_large data set, and
- 100 times with the data_small data set.

In both cases, start a cluster before running the task, and stop it afterwards. Measure how much time is spent on execution of the task. How the time compares to the execution time with the snow cluster (found in Exercises 6 and 7)?
When done working with the clusters, terminate the mpi execution environment with the mpi.finalize function. Note that this function always returns 1.
Important! As mentioned above, if you intend to create an mpi cluster again after the environment was terminated you have to restart the R session, otherwise the current session will be aborted, which may result in a loss of data. In RStudio, an R session can be relaunched from the Session menu (relaunching the session this way does not affect the data, you'll only need to reload libraries). In other cases, you may have to quit and restart R.

Parallel Computing Exercises: Foreach and DoParallel (Part-2)



In general, foreach is a statement for iterating over items in a collection without using any explicit counter. In R, it is also a way to run code in parallel, which may be more convenient and readable that the sfLapply function (considered in the previous set of exercises of this series) or other apply-alike functions.
Apart from being able to run code in parallel, the R's foreach has some other differences from the standard for loop. Specifically, the foreach statement:

- allows to iterate over several variables simultaneously,
- returns a value (a list, a vector, a matrix, or another object),
- is able to skip some iterations based on a condition (the last two properties make it similar to the list comprehension, which is present in Python and some other languages),
- has a special syntax that includes operators %do% (see an example in Exercise 1), %dopar%, and %:%.

The first six exercises in this set allow to train in performing basic operations with the foreach statement, and the last four ones show how to run it in parallel using

multiple CPU cores on one machine. The task will be to parallelize identical operations on a set of files (the zipped data files can be downloaded here). It is assumed that your computer has two or more CPU cores.
The exercises require the packages foreach, doParallel, and parallel. The first two packages have to be installed, and the last one comes with the standard R distribution. The packages doParallel and parallel are necessary to run foreach in parallel.
For other parts of the series follow the tag parallel computing.
Answers to the exercises are available here.

Exercise 1
The foreach function (from the package of the same name) is typically used as a part of a special statement. In its simple form, the statement looks like this:

```
result <- foreach(i = 1:3) %do% sqrt(i)
```

The statement above consists of three parts:

- foreach(i = 1:3) – a call to the foreach function, with an argument that includes an iteration variable (i) and a sequence to be iterated over (1:3),
- %do% – a special operator,
- sqrt(i): an R expression, which represents an operation to be performed over the iteration variable (this part of the statement is equivalent to the body of the loop).

The code iterates over the sequence, applies an operation defined in the expression to each element of the sequence, and stores the output in the result variable.
Note that if the expression extends over several lines it has to be enclosed in curly braces. The use of the iteration variable is not mandatory: if you just want to repeat the expression n times not passing anything to that expression you can use only a sequence of the length n as input to foreach.

In this exercise:

1. Run the code above, print the result object, and find to which class it belongs.
2. Use the foreach function to reverse the result. I.e. write a line of code that receives the result object as an input, and outputs the original sequence. Print the sequence.

Exercise 2

The foreach function allows for the use of several iteration variables simultaneously. They are passed to the function as arguments, and are separated by commas.

Run the foreach function with two iteration variables to get a sequence of their sums. The variables have to iterate over a vector of integers from 1 to 3, and a vector of 5 integers of value 10. Print the result.

(Tip: if you want to use an arithmetic operator to calculate the sum then the expression must be placed in parentheses or curly braces).

What is the length of the resulting object? How does the function deal with the vectors of different length?

Exercise 3

The package iterators provides several functions that can be used to create sequences for the foreach function. For example, the irnorm function creates an object that iterates over vectors of normally distributed random numbers. It is useful when you need to use random variables drawn from one distribution in an expression that is run in parallel.

In this exercise, use the foreach and irnorm functions to iterate over 3 vectors, each containing 5 random variables. Find the largest value in each vector, and print those largest values.

Before running the foreach function set the seed to 1234.

Learn more about optimizing your workflow in the online course

Getting Started with R for Data Science. In this course you will learn how to:

- efficiently organize your workflow to get the best performance of your entire project
- get a full introduction to using R for a data science project
- And much more

Exercise 4

By default the foreach function returns a list. But it can also return sequences of other types. This requires changing the value of the .combine parameter of the function. This exercise will train how to use this parameter.

As in the previous exercise, use the foreach and irnorm functions to iterate over 3 vectors, each containing 5 random variables. But now use an expression that returns all variables generated by irnorm. Pass the .combine parameter to the foreach function with value 'c'. Print the result, and find its class and length.

Then run the code again with the 'cbind' value assigned to the .combine parameter. Print the result, find its class and size.

Note that 'c' and 'cbind' are R functions from the base package. Other functions (including user-written ones) can be used as well to combine the outputs of the expression.

Exercise 5

The results of the expression placed after the %do% operator can be combined in different ways. Look at the documentation for the foreach function to find what value has to be assigned to the .combine parameter to sum the values produced by the expression in each iteration.

Run the code used in previous exercise with that value assigned to the .combine parameter, and print the result.

Before running the code set the seed to 1234.

Exercise 6

The sequence passed to the foreach function can be filtered so

that the expression after %do% is applied only to a part of the sequence. This is done using a syntax like this:

```
result ‹- foreach(i = some_sequence) %:% when(i › 0) %do%
                                sqrt(i)
```

You can notice that the %:% operator and the when function, which contains a Boolean expression involving the iteration variable, are added to a standard foreach statement.
Modify the example above to get a vector of logs of all even integers in the range from 1 to 10. Print the result.

Exercise 7
Now let’s parallelize the execution of the foreach function. We’ll use it to read similarly named files, and perform identical calculations on data from each file.
As a first step, write a function to be run in parallel. The function takes an integer as input, and performs the following actions:

1. Create a string (character vector) with a file name by concatenating constant parts of the name (test_data_, .csv) with the integer (example of possible result when 1 is used as integer: test_data_1.csv).
2. Read the file with the obtained name from the current working directory into a data frame.
3. Calculate mean values for each column in the data frame.
4. Return a vector of those values.

Exercise 8
The second step is to create a backend for parallel execution:

1. Make a cluster for parallel execution using the makeCluster function from the parallel package; pass the size of the cluster (i.e. the number of CPU cores that you want to be used in computations) as an argument to this function .
2. Register the cluster with the registerDoParallel function from the doParallel package.

Note that by default the makeCluster function creates a PSOCK cluster, which is an enhanced version of the SOCK cluster implemented in the snow package. Accordingly, the PSOCK cluster is a pool of worker processes that exchange data with the master process via sockets. The makeCluster function can also create other types of clusters.

Exercise 9

The last step is to run the foreach function to read and analyze 10 test files (contained in this archive) using the function created in Exercise 7. Combine the outputs of that function using rbind.

Perform this task twice:

1. with %do% operator, which evaluates the expression sequentially, and
2. with %dopar% operator, which evaluates the expression in parallel.

In both cases, measure the execution time using the the system.time function. Print the result of the last run.

IMPORTANT: after completing parallel computations stop the cluster (created in Exercise 8) using the stopCluster function from the parallel package.

Exercise 10

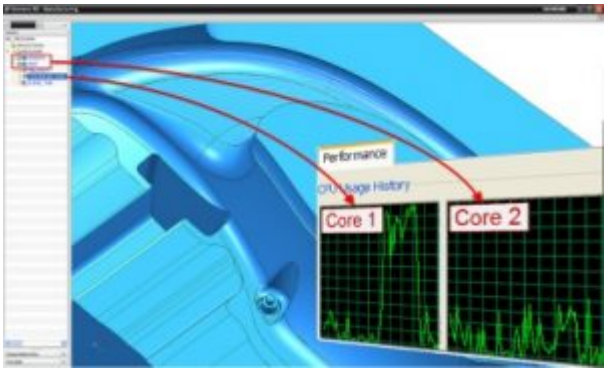
Modify the code written in the Exercise 7 and Exercise 9 to calculate the mean and the variance of values contained in the first column in each file. The resulting object must be a two-column matrix with the first column representing means, and the second column describing variances (the number of rows must be equal to the number of files).

Repeat the actions listed in Exercise 8 to prepare a cluster for parallel execution, then run the modified code in parallel.

Print the result.

Stop the cluster.

Parallel Computing Exercises: Snowfall (Part-1)



R has a lot of tools to speed up computations making use of multiple CPU cores either on one computer, or on multiple machines. This series of exercises aims to introduce the basic techniques for implementing parallel computations using multiple CPU cores on one machine.

The initial step in preparation for parallelizing computations is to decide whether the task can and should be run in parallel. Some tasks involve sequential computation, where operations in one round depend on the results of the previous round. Such computations cannot be parallelized. The next question is whether it is worth to use parallel computations. On the one hand, running tasks in parallel may reduce computer time spent on calculations. On the other hand, it requires additional time to write the code that can be run in parallel, and check whether it yields correct results.

The code that implements parallel computations basically makes three things:

- splits the task into pieces,
- runs them in parallel, and
- combines the results.

This set of exercises allows to train in using the snowfall

package to perform parallel computations. The set is based on the example of parallelizing the k-means algorithm, which splits data into clusters (i.e. splits data points into groups based on their similarity). The standard k-means algorithm is sensitive to the choice of initial points. So it is advisable to run the algorithm multiple times, with different initial points to get the best result. It is assumed that your computer has two or more CPU cores.
The data for the exercises can be downloaded here.
For other parts of the series follow the tag parallel computing.
Answers to the exercises are available here.

Exercise 1
Use the detectCores function from the parallel package to find the number of physical CPU cores on your computer. Then change the arguments of the function to find the number of logical CPU cores.

Exercise 2
Load the data set, and assign it to the df variable.

Exercise 3
Use the system.time function to measure the time spent on execution of the command fit_30 <- kmeans(df, centers = 3, nstart = 30), which finds three clusters in the data.
Note that this command runs the kmeans function 30 times sequentially with different (randomly chosen) initial points, and then selects the 'best' way of clustering (the one that minimizes the squared sum of distances between each data point and its cluster center).

performance of your entire project
- get a full introduction to using R for a data science project
- And much more

Exercise 4
Now we'll try to paralellize the runs of kmeans. The first step is to write the code that performs a single run of the kmeans function. The code has to do the following:

1. Randomly choose three rows in the data set (this can be done using the sample function).
2. Subset the data set keeping only the chosen rows (they will be used as initial points in the k-means algorithm).
3. Transform the obtained subset into a matrix.
4. Run the kmeans function using the original data set, the obtained matrix (as the centers argument), and without the nstart argument.

Exercise 5
The second step is to wrap the code written in the previous exercise into a function. It should take one argument, which is not used (see explanation on the solutions page), and should return the output of the kmeans function.
Such functions are often labelled as wrapper, but they may have any possible name.

Exercise 6
Let's prepare for parallel execution of the function:

1. Initialize a cluster for parallel computations using the sfInit function from the snowfall package. Set the parallel argument equal to TRUE. If your machine has two logical CPU's assign two to the cpus argument; if the number of CPU's exceeds two set this argument equal to the number of logical CPU's on your machine minus one.
2. Make the data set available for parallel processes with

the sfExport function.
3. Prepare the random number generation for parallel execution using the sfClusterSetupRNG. Set the seed argument equal to 1234.

(Note that kmeans is a function from the base R packages. If your want to run in parallel a function from a downloaded package, you have also to make it available for parallel execution with the sfLibrary function).

Exercise 7
Use the sfLapply function from the snowfall package to run the wrapper function (written in Exercise 5) 30 times in parallel, and store the output of sfLapply in the result variable. Apply also the system.time function to measure the time spent on execution of sfLapply.
Note that sfLapply is a parallel version of lapply function. It takes two main arguments: (1) a vector or a list (in this case it should be a numeric vector of length 30), and (2) the function to be applied to each element of the vector or list.

Exercise 8
Stop the cluster for parallel execution with the sfStop function from the snowfall package.

Exercise 9
Explore the output of sfLapply (the result object):

1. Find out to what class it belongs.
2. Print its length.
3. Print the structure of its first element.
4. Find the value of the tot.withinss sub-element in the first element (it represents the total squared sum of distances between data points and their cluster centers in a given solution to the clustering problem). Print that value.

Exercise 10
Find an element of the result object with the lowest

tot.withinss value (there may be multiple such elements), and assign it to the best_result variable.
Compare the tot.withinss value of that variable with the corresponding value of the fit_30 variable, which was obtained in Exercise 3.

Data Manipulation with data.table (part -2)

In the last set of exercise of data.table ,we saw some interesting features of data.table .In this set we will cover some of the advanced features like set operation ,join in data.table.You should ideally complete the first part before attempting this one .
Answers to the exercises are available here.

If you obtained a different (correct) answer than those listed on the solutions page, please feel free to post your answer as a comment on that page.

Exercise 1
Create a data.table from diamonds dataset ,create key using setkey over cut and color .Now select first entry of the groups Ideal and Premium

Exercise 2

With the same dataset,select the first and last entry of the groups Ideal and Premium

Exercise 3

Earlier we have seen how we can create/update columns by reference using := .However there is a lower over head ,faster alternative in data.table . This is achieved by SET and Loop in data.table ,however this is meant for simple operations and will not work in grouped operation .Now take the diamonds data.table and make columns x,y,z value squared .For example if the value is currently 10 ,the resulting value would be 100 .You are awesome if you find out all alternative answer and check the time using system.time .

Exercise 4

In the same dataset ,capitalize first letter of column names .

Learn more about the data.table package in the online course R Data Pre-Processing & Data Management – Shape your Data!. In this course you will learn how to:

- Create, shape and manage your data using the data.table package
- Learn what other data-processing tools exist in R
- And much more

Exercise 5

Reordering column sometimes is necessary , however if your data frame is of several GBs it might be a overhead to create new data frame with new order .Data.Table provides features to overcome this . Now reorder your diamonds data.table's column by sorting alphabetically

Exercise 6

If you are not convinced with the powerful intuitive features of data.table till now , I am pretty sure you will by the end of THIS . Suppose I want to have a metric on diamonds where I want to find for each group of cut maximum of x * mean of

depth and name it my_int_feature and also I want another metric which is my_int_feature* maximum of y again for each group of cut . This is achievable by chaining but also with a single operation without chaining which is the expected answer .

Exercise 7

Suppose we want to merge iris and airquality,akin to the functionlaity of rbind . We want to do it fast and want to keep track of the rows with their original dataset , and keep all the columns of both the data set in the merged data set as well ,How do we achieve that

Exercise 8

The Next 3 exercises are on rolling Join like features of data.table ,which is useful in time series like data . Create a data.table with the following

```
set.seed(1024)
x <- data.frame(rep(letters[1:2],6),c(1,2,3,4,6,7),sample(100,6))
names(x) <- c("id","day","value")
test_dt <- setDT(x)
```

Now this mimics a sales data of 7 days for a and b . Notice that day 5 is not present for both a and b .This is not desirable in many situations , A common practise is to use the previous days data .How do we get previous days data for the id a ,You should ideally set keys and do it using join features

Exercise 9

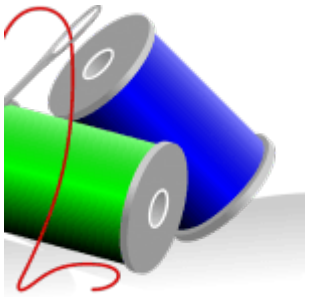
May be you dont want the previous day’s data ,you may want to copy the nearest value for day 5 ,How do we achieve that

Exercise 10

Now there may be a case when you don’t want to copy any value if the date is beyond last observation .Use your answer for question 8 to find the value for day 5 and 9 for b ,Now since 9 falls beyond last observation of 7 you might want to avoid

copying it .How do you explicitly tell your data.table to stop when it sees last observation and don’t copy previous value .This may not seem useful since you know that here 9 falls beyond 7 ,but imagine you have a series of data points and you don’t really want to copy data to observations after your last observation.This might come handy in such cases .

Character Functions (Advanced)



This set of exercises will help you to help you improve your skills with character functions in R. Most of the exercises are related with text mining, a statistical technique that analyses text using statistics. If you find them interesting I would suggest checking the library tm, this includes functions designed for this task. There are many applications of text mining, a pretty popular one is the ability to associate a text with his or her author, this was how J.K.Rowling (Harry potter author) was caught publishing a new novel series under an alias. Before proceeding, it might be helpful to look over the help pages for the nchar, tolower, toupper, grep, sub and strsplit. Take at the library stringr and the functions it includes such as str_sub.

Answers to the exercises are available here.

If you obtained a different (correct) answer than those listed on the solutions page, please feel free to post your answer as a comment on that page.

Before starting the set of exercises run the following code

lines :

```
if (!'tm' %in% installed.packages()) install.packages('tm')
library(tm)
txt = system.file("texts", "txt", package = "tm")
ovid = VCorpus(DirSource(txt, encoding = "UTF-8"),
readerControl = list(language = "lat"))
OVID = c(data.frame(text=unlist(TEXT), stringsAsFactors = F))
TEXT = lapply(ovid[1:5], as.character)
TEXT1 = TEXT[[4]]
```

Exercise 1

Delete all the punctuation marks from TEXT1

Exercise 2

How many letters does TEXT1 contains?

Exercise 3

How many words does TEXT1 contains?

Exercise 4

What is the most common word in TEXT1?

Learn more about Text analysis in the online course Text Analytics/Text Mining Using R. In this course you will learn how create, analyse and finally visualize your text based data source. Having all the steps easily outlined will be a great reference source for future work.

Exercise 5

Get an object that contains all the words with at least one capital letter (Make sure the object contains each word only once)

Exercise 6

Which are the 5 most common letter in the object OVID?

Exercise 7

Which letters from the alphabet are not in the object OVID

Exercise 8

On the OVID object, there is a character from the popular sitcom 'FRIENDS' , Who is he/she? There were six main characters (Chandler, Phoebe, Ross, Monica, Joey, Rachel)

Exercise 9

Find the line where this character is mentioned

Exercise 10

How many words finish with a vowel, how many with a consonant?

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

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.

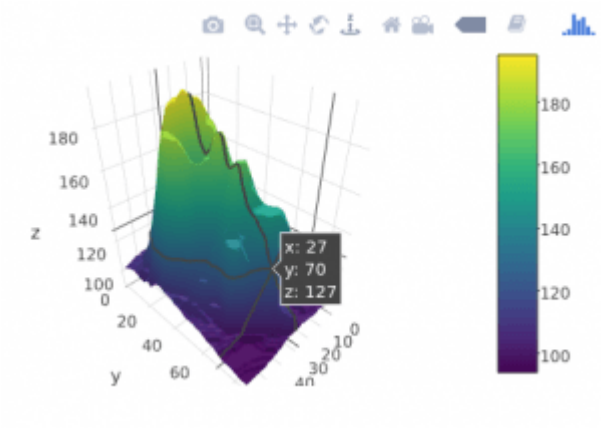
Exercise 9

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.

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.

Descriptive Analytics-Part 6: Interactive dashboard (2/2)

Descriptive Analytics is the examination of data or content, usually manually performed, to answer the question "What happened?".As this series of exercises comes to an end, the last part is going to be the development of a data product. Not everybody is able to code in R, so it is useful to be able to make GUIs in order to share your work with non-technical people. This part may be a little challenging, since it requires some basic knowledge of the shiny package. The outcome of this set of exercises will be almost like this web app (some variables are missing because I had to reduce the size of the data set).

In order to be able to solve this set of exercises you should have solved the part 0, part 1, part 2,part 3, and part 4 of this series but also you should run this script which contain some more data cleaning. In case you haven't, run this script in your machine which contains the lines of code we used to modify our data set. This is the tenth set of exercise of a series of exercises that aims to provide a descriptive analytics solution to the '2008' data set from here. This data set which contains the arrival and departure information for all domestic flights in the US from 2008 has become the "iris" data set for Big Data. The goal of Descriptive analytics is to inform the user about what is going on at the dataset. Before proceeding, it might be helpful to look over the help pages for the fluidPage, pageWithSidebar, headerPanel , sidebarPanel, selectInput, mainPanel, tabPanel, observe, verbatimTextOutput, renderPrint, shinyApp.

For this set of exercises you will need to install and load the package shiny.

```
install.packages('shiny')
library(shiny)
```

I have also changed the values of the DaysOfWeek variable, if you wish to do that as well the code for that is :

```
install.packages('lubridate')
library(lubridate)
flights$DayOfWeek <-
wday(as.Date(flights$Full1_Date,'%m/%d/%Y'), label=TRUE)
```

Because the app requires some time to run, I have also removed the rows with missing values from the data set just to save some time.

```
flights <-flights[which(!is.na(flights['WeatherDelay'])),]
flights <-flights[which(!is.na(flights['ArrDelay'])),]
```

Answers to the exercises are available here.

If you obtained a different (correct) answer than those listed on the solutions page, please feel free to post your answer as a comment on that page. Moreover it would be really nice of you to share the links of the apps you have developed. It would be a great contribution the community.

Exercise 1

Create the user interface and set as the header of the web app : “Descriptive Analysis”

Exercise 2

Create a side panel.

Exercise 3

Create two select list input control. The former will contain the variables: CarrierDelay, WeatherDelay, NASDelay, SecurityDelay, LateAircraftDelay. The latter will contain the variables :Dest, Origin, UniqueCarrier, TailNum, CancellationCode.

Exercise 4

Create a set of radio buttons used to select a plot from a list (Histogram, Scatter plot, Violin plot),and set as default plot the Histogram.

Exercise 5

Create a set of radio buttons used to select a plot from a list (bar plot, pie chart, rose wind),and set as default plot the bar plot.

Exercise 6

Create a main panel.

Exercise 7

Create in the main panel two tabs named “Delays” and “Categorical” that will contain the plots of the exercises 4 and 5 respectively.

Exercise 8

Now that we are done with the user interface, create the server side of the app. Create the output of the first tab, which will be the plots from exercise 4 in respect to the first set of variables from exercise 3 (notice that they are all continuous variables), bear in mind that at the scatter plot the x-axis should be the Full_Date and at the violin plot the x-axis should be the DayOfWeek as we did at the previous

set of exercises. (please check out the first tab of the app, to make things more clear).

Exercise 9

Create the output of the second tab, , which will be the plots from exercise 5 in respect to the second set of variables from exercise 3 from the exercise 5, use the knowledge you applied (or acquired at the previous exercises for the plots, make them as interesting as you can).(please check out the second tab of the app, to make things more clear).

Exercise 10

Launch the app.

Descriptive Analytics-Part 6: Interactive dashboard (1/2)

Descriptive Analytics is the examination of data or content, usually manually performed, to answer the question "What happened?".As this series of exercises comes to an end, the last part is going to be the development of a data product. Not everybody is able to code in R, so it is useful to be able to make GUIs in order to share your work with non-technical people. This part may be a little challenging, since it requires some basic knowledge of the shiny package. The outcome of this set of exercises will be almost like this web app (some variables are missing because I had to reduce the

size of the data set).

In order to be able to solve this set of exercises you should have solved the part 0, part 1, part 2,part 3, and part 4 of this series but also you should run this script which contain some more data cleaning. In case you haven't, run this script in your machine which contains the lines of code we used to modify our data set. This is the ninth set of exercise of a series of exercises that aims to provide a descriptive analytics solution to the '2008' data set from here. This data set which contains the arrival and departure information for all domestic flights in the US from 2008 has become the "iris" data set for Big Data. The goal of Descriptive analytics is to inform the user about what is going on at the dataset. Before proceeding, it might be helpful to look over the help pages for the fluidPage, pageWithSidebar, headerPanel , sidebarPanel, selectInput, mainPanel, tabPanel, verbatimTextOutput, renderPrint, shinyApp.

For this set of exercises you will need to install and load the package shiny.

```
install.packages('shiny')
library(shiny)
```

I have also changed the values of the DaysOfWeek variable, if you wish to do that as well the code for that is :

```
install.packages('lubridate')
library(lubridate)
flights$DayOfWeek <-
wday(as.Date(flights$Full1_Date,'%m/%d/%Y'), label=TRUE)
```

Because the app requires some time to run, I have also removed the rows with missing values from the data set just to save some time.

```
flights <-flights[which(!is.na(flights['WeatherDelay'])),]
flights <-flights[which(!is.na(flights['ArrDelay'])),]
```

Answers to the exercises are available here.

If you obtained a different (correct) answer than those listed on the solutions page, please feel free to post your answer as a comment on that page.

Learn more about Shiny in the online course R Shiny Interactive Web Apps – Next Level Data Visualization. In this course you will learn how to create advanced Shiny web apps; embed video, pdfs and images; add focus and zooming tools; and many other functionalities (30 lectures, 3hrs.).

Exercise 1

Create the user interface and set as the header of the web app : “Descriptive Analysis”

Exercise 2

Create a side panel.

Exercise 3

Create a select list input control that contains the functions : summary, str, head, tail, names, summary.

Exercise 4

Create a select list input control that contains the functions : mean, median, max, min, range, sd.

Exercise 5

Create a select list input control that contains the variables : ActualElapsedTime, CRSElapsedTime, AirTime, ArrDelay, DepDelay, TaxiIn, TaxiOut.

Exercise 6

Create a main panel.

Exercise 7

Create in the main panel two tabs named “Content” and “Measures” that will contain the output of the functions of exercise 3 and exercise 4 respectively.

Exercise 8

Now that we are done with the user interface, create the server side of the app and the output that is supposed to print the functions of the exercise 3. (please check out the first tab of the app, to make things more clear).

Exercise 9

Create the output of the second tab, combining the functions of exercise 4 and the variables from the exercise 5.(please check out the second tab of the app, to make things more clear).

Exercise 10

Launch the app.

Descriptive Analytics-Part 5: Data Visualisation (Spatial data)

Descriptive Analytics is the examination of data or content, usually manually performed, to answer the question "What happened?".

In order to be able to solve this set of exercises you should have solved the part 0, part 1, part 2,part 3, and part 4 of this series but also you should run this script which contain some more data cleaning. In case you haven't, run this script in your machine which contains the lines of code we used to modify our data set. This is the eighth set of exercise of a series of exercises that aims to provide a descriptive analytics solution to the '2008' data set from here. This data set which contains the arrival and departure information for all domestic flights in the US from 2008 has become the "iris" data set for Big Data.In order to solve this set of exercises, you have to download this data set which provide us the coordinates of each airport.Please find the script used to create a merged dataset here . I don't expect you to do the pre-processing yourself since it is beyond the scope of this set but I highly encourage you to give it a try, in case you did that with a better or more efficient way than I did, please post your solution at the comment section(it will be highly appreciated). Moreover we will remove the rows with missing values (various delays) because the methods that we will use are computationally expensive so having a big data set is just a waste of time. The goal of Descriptive analytics is to inform the user about what is going on at the dataset. A great way to do that fast and effectively is by performing data visualisation. Data visualisation is also a form of art, it has to be simple, comprehended and full of information. On this set of exercises we will explore different ways of visualising spatial using the famous ggmap package. Before proceeding, it might be helpful to look over the help pages

for the get_map, ggmap, facet_wrap.

For this set of exercises you will need to install and load the packages ggplot2, dplyr, and ggmap.

```
install.packages('ggplot2')
library(ggplot2)
install.packages('dplyr')
library(dplyr)
install.packages('ggmap')
library(ggmap)
```

I have also changed the values of the DaysOfWeek variable, if you wish to do that as well the code for that is :

```
install.packages('lubridate')
library(lubridate)
flights$DayOfWeek <- wday(as.Date(flights$Full1_Date,'%m/%d/%Y'), label=TRUE)
```

Answers to the exercises are available here.

If you obtained a different (correct) answer than those listed on the solutions page, please feel free to post your answer as a comment on that page.

Exercise 1

Query the map of United States using the get_map function.
It is recommended to experiment with the various types of maps and select the one that you think is the best. (I have used the toner-lite from Stamen Maps.)

Exercise 2

Print the map that you have selected.

Exercise 3

Modify the printed map in order to print out a bigger image(extent) and assign it to a m object.

Exercise 4

Plot the destination airports of the flights on the map.

Exercise 5

Plot the destination airports of the flights on the map, the size of the points should be based on the number of flights that arrived to the destination airports.

Exercise 6

Plot the destination airports of the flights on the map, the colour of the points should be based on the number of flights that arrived to the destination airport. Make it a bit prettier, use the scale_colour_gradient and set the lows and the highs of your preferences.

Exercise 7

Plot the destination airports of the flights on the map, the colour of the points should be based on the number of flights that arrived to the destination airport and the size of the points should be based on the total delay of arrival of the flights that arrived at the destination airport.
Something is not right here, right?

Exercise 8

Plot the destination airports of the flights on the map, the colour of the points should be based on the number of flights that arrived to the destination airport and the size of the points should be based on the total delay of arrival divided by the number of flights per destination.

Exercise 9

Plot the destination airports for everyday of the week (hint : facet_wrap)

Exercise 10

Plot the destination airports of the flights on the map, the colour of the points should be based on the number of flights that arrived to the destination airports, the size of the points should be based on the total delay of arrival of the flights that arrived at the destination airport for everyday of the week.
(This may be a bit more challenging , if you can’t solve it go to the solutions and try to understand the reason I did what I did, if you have any questions please post them at the comment section).