Prettify your Shiny Tables with DT: Exercises

Have you ever wanted to make your Shiny tables interactive, more functional and look better? The DT package, which stands for “DataTables”, provides an R interface to the JavaScript library “DataTables”. It allows creating high standard tables by implementing the functionalities and design features that are available through the “DataTables” library.

Even though the DT package can be used independently of Shiny, this exercise set will focus on the integration between the two packages. The exercises were built with a practical orientation in mind, and upon completing them you would be ready to tackle the vast majority of use cases for using DT with Shiny.

We will work with the light built-in dataset datasets::Orange, that holds data about growth of orange trees. Each exercise is adding some more features/functionalities to the code of the previous exercise, so be sure to not discard the code until after you’re done with all of the exercises. Answers to the exercises are available here.

Each exercises should result in a Shiny app, where the changing part is the renderDataTable() function.
To save time, you can use the following template, and just replace the placeholder for each exercise.

```
ui <- fluidPage(
br(), br(), br(),
fluidRow(column(width = 6, DT::dataTableOutput(outputId = "my_datatable")))
)

server <- function(input, output, session) {
```

```
output$my_datatable <-
}

shinyApp(ui = ui, server = server)
```

Exercise 1
Generate a minimal Shiny app that displays the dataset Orange in a datatable (default values).

Exercise 2
Remove the row names.

Exercise 3
Show only 7 rows as the default display, and allow changing the number of displayed rows to either 14, 21, 28 or 35.

Exercise 4
Align the columns text (both values and headers) to the center.

Exercise 5
Remove the search box (top right) and the table information text (bottom left).

Learn more about Shiny apps in the online courses Create Interactive Web Applications with the R Shiny Package and R Shiny Interactive Web Apps.

Exercise 6
Add a “copy” and “csv” buttons to allow saving the table to the clipboard and downloading it as a CSV respectively.

Exercise 7
Add a filter box for each column, at the bottom of the table.

Exercise 8
Allow selection of a single row only, rather than multiple rows (which is the default).

Exercise 9

Remove the option to sort the table.

Exercise 10
Finally, if a row was selected, display in the UI (textOutput) the selected tree and age values.
Hint: for this exercise you would have to make additional changes to the UI and server, rather than just changing the renderDataTable() function.

Pull the Right Strings with stringr: Exercises

By providing a set of wrappers to existing functions, the stringr package allows for simple, consistent and efficient manipulations of strings in R. Even though there are some more basic packages that offer strings-related functions, you might find yourself in need for a more complete and straightforward solution for handling strings in R.

With a simple and consistent syntax, stringr provides some very convenient functions around pattern matching, characters manipulation, whitespace handling and more. The full reference of the package can be found here.

Please find below a set of exercises that will help you practice a variety of stringr functions. The focus is on practical operations that data analysts are required to perform on a daily basis. Answers to the exercises are available here. And, don't forget to check out our other exercise sets on the stringr package by following the stringr tag.

For the following exercises we will use this data:

```
addresses <- c("14 Pine Street, Los Angeles", "152 Redwood Street, Seattle", "8 Washington Boulevard, New York")

products <- c(“TV “, ” laptop”, “portable charger”, “Wireless Keybord”, ” HeadPhones “)

long_sentences <- stringr::sentences[1:10]

field_names <- c(“order_number”, “order_date”, “customer_email”, “product_title”, “amount”)

employee_skills <- c(“John Bale (Beginner)”, “Rita Murphy (Pro)”, “Chris White (Pro)”, “Sarah Reid (Medium)”)
```

Exercise 1
Normalize the addresses vector by replacing capitalized letters with lower-case ones.

Exercise 2
Pull only the numeric part of the addresses vector.

Exercise 3
Split the addresses vector into two parts: address and city. The result should be a matrix.

Exercise 4
Now try to split the addresses vector into three parts: house number, street and city. The result should be a matrix.
Hint: use a regex lookbehind assertion

Exercise 5
In the long_sentences vector, for sentences that start with the letter “T” or end with the letter “s”, show the first or last word respectively. If the sentence both starts with a “T” and ends with an “s”, show both the first and the last words. Remember that the actual last character of a sentence is usually a period.

Learn more about string manipulation with stringr in the online course Learn R by Intensive Practice.

Exercise 6

Show only the first 20 characters of all sentences in the long_sentences vector. To indicate that you removed some characters, use two consecutive periods at the end of each sentence.

Exercise 7

Normalize the products vector by removing all unnecessary whitespaces (both from the start, the end and the middle), and by capitalizing all letters.

Exercise 8

Prepare the field_names for display, by replacing all of the underscore symbols with spaces, and by converting it to the title-case.

Exercise 9

Align all of the field_names to be with equal length, by adding whitespaces to the beginning of the relevant strings.

Exercise 10

In the employee_skills vector, look for employees that are defined as “Pro” or “Medium”. Your output should be a matrix that have the employee name in the first column, and the skill level (without parenthesis) in the second column. Employees that are not qualified should get missing values in both columns.

Step Up Your Dashboard With

Shinydashboard – Part 2: Exercises

The shinydashboard provides a well-designed dashboard theme for Shiny apps and allows for an easy assembly of a dashboard from a couple of basic building blocks. The package is widely used in commercial environments as well due to its neat features for building convenient and robust layouts.

This exercise set will help you practice all of the main features of this great package. By completing the two parts of the exercise series, you'll know that you're ready to start building well-designed Shiny apps. We will make some minimal use of the built-in data-set datasets::CO2 .(Specific descriptions of the data-set are irrelevant, but you can check them out by typing ?datasets::CO2.) Each exercise is adding some more features/functionalities to the code of the previous exercise, so be sure to not discard the code until after you're done with all of the exercises. Answers to these exercises are available here.

In the solutions page, you'll first find only the relevant component of each exercise. Then, at the end of the page, you will find the entire Shiny app code that contains all of the different components together. This exercise set is based on the output code of this exercise set. If you haven't done it, you can just use the code under "All Exercises Combined" here (at the bottom) as your basis for this exercise set.

For other parts of the series, follow the tag shinydashboard.

Exercise 1
In the "data" tab, add a box() with the title "CO2 Data."
The box should have a blue header and it should be collapsible.

Exercise 2

Add the CO2 table to the box you just created.
The table should be filtered by the plant input that was created in the previous exercise set.

Exercise 3
In the “licenses” tab, add a tabBox() that contains two panels: one titled “Data” and one titled “Icons.”
You can leave those panels empty for now.

Exercise 4
In the “contact_us” tab, add an infoBox() with some content of your choice and select its color and icon.

Exercise 5
In the “contact_us” tab, add an valueBox() with some content of your choice, and select its color and icon.

Learn more about Shiny apps in the online courses Create Interactive Web Applications with the R Shiny Package and R Shiny Interactive Web Apps.

Exercise 6
Add a “messages” drop down menu to the header.
Select its icon and badge-status and add one messageItem() to it.

Exercise 7
Add a “tasks” drop down menu to the header.
Select its icon and badge-status and add two taskItem()s to it.

Exercise 8
Add a placeholder for a “notifications” drop down menu in the header using dropdownMenuOutput().
The output ID should be “notifications.”

Exercise 9
Add the server-side code of the “notifications” drop down menu.

It should have a single notification item that shows what the current value of the plant input that was created in the previous exercise set is.

Exercise 10
Add a link to “r-exercises.com” in the header.
Hint: use tags$li(class = "dropdown", ...) together with tags$a(...).

Specalize in Geo-Spatial Visualizations With Leaflet – Part 1: Exercises

Leaflet is a JavaScript library for interactive maps. It is widely used across many platforms, and fortunately it is also implemented as a very user-friendly R package! With leaflet, you can create amazing maps within minutes that are customized exactly to your needs and embed them within your Shiny apps, markdowns, or just view them in your RStudio viewer.

In the following set of exercises, we will use geo-spatial data of the 26 cantons of Switzerland (more information about Switzerland cantons here.) The data is readily available in the leaflet package under the variable leaflet::gadmCHE. Each exercise is adding some more features/functionalities to the code of the previous exercise, so be sure not to discard the code until after you’re done with all of the exercises. Answers to these exercises are available here.

For other parts of the series, follow the tag leaflet.

Exercise 1
The first steps would be to set-up the map “scaffolding” so there’s still no need to use the data.
Create a leaflet map and select provider tiles of your choice, but not the default ones.
Hint: available provider tiles can be viewed here.

Exercise 2
Set the default view to the coordinates of the center of Switzerland with zoom level 7.
Hint: use Google to find the relevant coordinates.

Exercise 3
Restrict the zooming options to minimum level 5 and maximum level 9.

Exercise 4
Use our data-set leaflet::gadmCHE to draw the borders of the cantons.

Exercise 5
Change the border and the fill colors to the Swiss national flag colors (red and white, respectively.)
In addition, change the opacity and the weight of the border lines according to your aesthetic preferences.

Exercise 6
The capital city of Switzerland, “Bern”, is located in a canton with the same name.
Add a marker that indicates the center of the “Bern” canton.
Hint: the coordinates are available in the labpt slot of the polygons data.

Exercise 7
Upon hovering over a canton, change the borders of that canton to dark red.

Exercise 8
Upon hovering over a canton, show a label with its name.

Exercise 9
Add a distance/area measurement functionality to the map.
The measurment units should be kilometers (for distances) and square-meters (for areas.)

Exercise 10
Add the mini-map feature to your map.
The mini-map should have the same provider tiles and it should start minimized.

Harvesting Data From the Web With Rvest: Exercises

The rvest package allows for simple and convenient extraction of data from the web into R, which is often called “web scraping.” Web scraping is a basic and important skill that every data analyst should master. You’ll often see it as a job requirement.

In the following exercises, you will practice your scraping skills on the “Money” section of the CNN website. All of the main functions of the rvest package will be used. Answers to these exercises are available here.

Since websites are constantly changing, some of the solutions might grow to be outdated with time. If this is the case, you are welcome to inform the author and the relevant sections will be updated.

Exercise 1
Read the HTML content of the following URL with a variable

called webpage:
https://money.cnn.com/data/us_markets/
At this point, it will also be useful to open this web page in your browser.

Exercise 2
Get the session details (status, type, size) of the above mentioned URL.

Exercise 3
Extract all of the sector names from the “Stock Sectors” table (bottom left of the web page.)

Exercise 4
Extract all of the “3 Month % Change” values from the “Stock Sectors” table.

Exercise 5
Extract the table “What’s Moving” (top middle of the web page) into a data-frame.

Exercise 6
Re-construct all of the links from the first column of the “What’s Moving” table.
Hint: the base URL is “https://money.cnn.com”

Exercise 7
Extract the titles under the “Latest News” section (bottom middle of the web page.)

Exercise 8
To understand the structure of the data in a web page, it is often useful to know what the underlying attributes are of the text you see.
Extract the attributes (and their values) of the HTML element that holds the timestamp underneath the “What’s Moving” table.

Exercise 9
Extract the values of the blue percentage-bars from the

“Trending Tickers” table (bottom right of the web page.)
Hint: in this case, the values are stored under the “class” attribute.

Exercise 10
Get the links of all of the “svg” images on the web page.

Basic Generalized Additive Models In Ecology: Exercises

Generalized Additive Models (GAM) are non-parametric models that add smoother to the data. In this exercise, we will look at GAMs using cubic spline using the mgcv package. Data-sets used can be downloaded here. The data-set is the experiment result of grassland richness over time in the Yellowstone National Park (Skkink et al. 2007).
Answers to these 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. Load the data-set and required package before running the exercise.

Exercise 1
Observe the data-set and try to classify the response and explanatory variables. We will focus on ROCK as an explanatory variable.

Exercise 2
Do some scatter-plots.

Exercise 3
Since it is not linear, try to do GAM with ROCK variables.

Exercise 4
Check the result. What can be inferred?

Exercise 5
Do some validation plots.

Exercise 6
Plot the base graph.

Exercise 7
Add “predict” across the data and add some lines.

Exercise 8
Plot the fitted values.

Why do we only use ROCK variables? It is proven to give the most fitted data without incorporation of all the explanatory variables. Try to play around with other explanatory variables to see the difference.

Melt and Cast The Shape of Your Data-Frame: Exercises

Data-sets often arrive to us in a form that is different from what we need for our modeling or visualization functions, which, in turn, don’t necessarily require the same format.

Reshaping data.frames is a step that all analysts need, but many struggle with. Practicing this meta-skill will, in the

long-run, result in more time to focus on the actual analysis.

The solutions to this set will rely on data.table, mostly melt() and dcast(), which are originally from the reshape2 package. However, you can also get practice out of it using your favorite base-R, tidy-verse or any other method, then compare the results.

Solutions are available here.

Exercise 1

Take the following data.frame *from* this form:

```
df <- data.frame(id = 1:2, q1 = c("A", "B"), q2 = c("C", "A"), stringsAsFactors = FALSE)
df
  id q1 q2
1  1  A  C
2  2  B  A
```

to this:

```
  id question value
1  1       q1     A
2  2       q1     B
3  1       q2     C
4  2       q2     A
```

Exercise 2

Do the opposite; return the data.frame back to it's original form.

Exercise 3

Set up the data.frame in terms of questions, as follows:

```
  question id_1 id_2
1       q1    A    B
2       q2    C    A
```

Exercise 4

The data entry behind this data.frame went a little bit wrong. Get all the C and B entries into their corresponding columns:

```
df2 <- data.frame(
  A = c("A1", "A12", "A31", "A4"),
  B = c("B4", "C7", "C3", "B9"),
  C = c("C3", "B16", "B3", "C4")
)
```

Exercise 5

Get this data.frame:

```
df3 <- data.frame(
  Join_ID = rep(1:3, each = 2),
  Type    = rep(c("a", "b"), 3),
  v2      = c(8, 9, 7, 6, 5, 4)*10
)
```

To look like this:

```
  Join_ID a_v2 b_v2
1       1   80   90
2       2   70   60
3       3   50   40
```

Exercise 6

Revisiting a data-set used in an earlier exercise set on data exploration,
load the AER package and run the command data("Fertility"),
which loads the data-set Fertility to your work space.
Melt it into the following format, with one row per child.

```
head(ferl)
  morekids age afam hispanic other work mother_id order gender
1       no  27   no       no    no    0         1     1   male
2       no  30   no       no    no   30         2     1 female
3       no  27   no       no    no    0         3     1   male
4       no  35  yes       no    no    0         4     1   male
5       no  30   no       no    no   22         5     1 female
6       no  26   no       no    no   40         6     1   male
```

Exercise 7

Take this:

```
d1 = data.frame(
  ID=c(1,1,1,2,2,4,1,2),
  medication=c(1,2,3,1,2,7,2,8)
)
d1
  ID medication
1  1          1
2  1          2
3  1          3
4  2          1
5  2          2
6  4          7
7  1          2
8  2          8
```

to this form:

```
   ID medications
1:  1  1, 2, 3, 2
2:  2     1, 2, 8
```

```
3:  4          7
```

Note: the solution doesn’t use melt() nor dcast(), so you might look at other options.

Exercise 8

Get this:

```
dfs <- data.frame(
  Name = c(rep("name1",3),rep("name2",2)),
   MedName = c("atenolol 25mg","aspirin 81mg","sildenafil 100mg", "atenolol 50mg","enalapril 20mg")
)
dfs
   Name          MedName
1 name1    atenolol 25mg
2 name1     aspirin 81mg
3 name1 sildenafil 100mg
4 name2    atenolol 50mg
5 name2   enalapril 20mg
```

Into the following format:

```
    Name  medication_1   medication_2     medication_3
1: name1 atenolol 25mg   aspirin 81mg sildenafil 100mg
2: name2 atenolol 50mg enalapril 20mg
```

Exercise 9

Get the following data.frame organized in standard form:

```
df7 <- data.table(
  v1 = c("name1, name2", "name3", "name4, name5"),
  v2 = c("1, 2", "3", "4, 5"),
  v3 = c(1, 2, 3)
)
```

```
df7
             v1   v2 v3
1: name1, name2 1, 2  1
2:        name3    3  2
3: name4, name5 4, 5  3
```

Expected output:

```
      v1 v2 v3
1: name1  1  1
2: name2  2  1
3: name3  3  2
4: name4  4  3
5: name5  5  3
```

The solution doesn’t use melt() nor dcast() and can be suprisingly hard.

Exercise 10

Convert this:

```
df <- data.frame(
 Method = c("10.fold.CV Lasso", "10.fold.CV.1SE", "BIC",
"Modified.BIC"),
 n = c(30, 30, 50, 50, 50, 50, 100, 100),
 lambda = c(1, 3, 1, 2, 2, 0, 1, 2),
df = c(21, 17, 29, 26, 25, 32, 34, 32) )
> df
            Method   n lambda df
1 10.fold.CV Lasso  30      1 21
2   10.fold.CV.1SE  30      3 17
3              BIC  50      1 29
4     Modified.BIC  50      2 26
5 10.fold.CV Lasso  50      2 25
6   10.fold.CV.1SE  50      0 32
```

```
7              BIC 100      1 34
8     Modified.BIC 100      2 32
```

Into:

```
            Method lambda_30 lambda_50 lambda_100 df_30 df_50
df_100
1 10.fold.CV Lasso         1         2                21    25
2   10.fold.CV.1SE         3         0                17    32
3              BIC                   1           1           29
34
4     Modified.BIC                   2           2           26
32
```

(Image by Joe Alterio)

Non-Linear Models in R: Exercises

A mechanistic model for the relationship between x and y sometimes needs parameter estimation. When model linearisation does not work,we need to use non-linear modeling.
There are three main differences between non-linear and linear modeling in R:
1. Specify the exact nature of the equation.
2. Replace the lm() with nls(), which means non-linear least squares.
3. Sometimes we also need to specify the model parameters a, b and c.

In this exercise, we will use the same data-set as the previous exercise in polynomial regression here. Download the data-set here.
A quick overview of the data-set:
Response variable = number of invertebrates (INDIV)
Explanatory variable = the area of each clump (AREA)
Additional possible response variables = Species richness of invertebrates (SPECIES)
Answers to these 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
Load the data-set; specify the model. Try to use the power function with nls() and a=0.1 and b=1 as the initial parameter number.

Exercise 2
Do a quick check by creating a plot residual vs. a fitted model, since a normal plot will not work.

Exercise 3
Try to build a self-start function of the powered model.

Exercise 4
Generate the asymptotic model.

Exercise 5
Compared the asymptotic model to the powered one using AIC. What can we infer?

Exercise 6
Plot the model in one graph.

Exercise 7
Predict across the data and plot all three lines.

Intro To Time Series Analysis – Part 2: Exercises

In the exercises below, we will explore more in the Time Series analysis. The previous exercise can be found here. Please follow this in sequence.
Answers to these exercises are available here.

Exercise 1

Load the AirPassengers data. Check its class and see the start and end of the series.

Exercise 2
Check the cycle of the Time-Series AirPassengers.

Exercise 3

Create a lag-plot using the gglag-plot from the forecast package. Check how the relationship changes as the lag increases.

Exercise 4

Also, plot the correlation for each of the lags. You can see when the lag is above 6, the correlation drops, climbs up in 12 and again drops in 18.
Exercise 5

Plot the histogram of the AirPassengers using a gghistogram from the forecast.

Exercise 6

Use tsdisplay to plot auto-correlation, time-series and partial auto-correlation together in the same plot.

Exercise 7

Find the outliers in the time-series.

Sharpening the Knives in the data.table Toolbox: Exercises

If knowledge is power, then knowledge of data.table is something of a super power, at least in the realm of data manipulation in R.

In this exercise set, we will use some of the more obscure functions from the data.table package. The solutions will use set(), inrange(), chmatch(), uniqueN(), tstrsplit(), rowid(), shift(), copy(), address(), setnames() and last(). You are free to use more, as long as they are part of data.table. The objective is to get (more) familiar with these functions and be able to call on them in real-life, giving us fewer reasons to leave the fast and neat data.table universe.

Solutions are available here.

PS. If you are unfamiliar with data.table, we recommend you start with the exercises covering the basics of data.table.

Exercise 1

Load the gapminder data-set from the gapminder package. Save it to an object called “gp” and convert it to a data.table. How many different countries are covered by the data?

Exercise 2

Create a lag term for GDP per capita. That is the value of GDP at the last observation (which are 5 years apart) for each country.

Exercise 3

Using the data.table syntax, calculate the GDP per capita growth from 2002 to 2007 for each country. Extract the one with the highest value for each continent.

Exercise 4

Save the column names in a vector named “temp” and change the name of the year column in “gp” to “anno” (just because); print the temp. Oh my, what just happened? Check the memory address of temp and names(gp), respectively.

Exercise 5

Overwrite “gp” with the original data again. Now make a *copy passed by value* into temp (before you change the year to anno) so you can keep the original variable names. Check the addresses again. Also, change factors to characters and don’t forget to convert to data.table again.

Exercise 6

A data.table of the number of goals each team in group A made in the FIFA world championship is given below. Import this into R and add a column with the countries’ population in 2017 to the data.table, rounded to the nearest million.

```
gA_2014 <- data.table(
  country   = c("Brazil", "Mexico", "Croatia", "Cameroon"),
  goals2014 = c(7, 4, 6, 1)
)
gA_2014
   country goals2014
```

```
1:   Brazil         7
2:   Mexico         4
3:  Croatia         6
4: Cameroon         1
```

Exercise 7

Calculate the number of years since the country reached $8k in GDP per capita at each relevant observation as accurately as the data allows.

Exercise 8

Add a subtly different variable using rowid(). That is the number of the observations among observations where the GDP is below 8k up to and including the given observation. Which country, in each continent, has the most observations above 8k? If there are ties, then list all of the those tied at the top.

Exercise 9

Use inrange() to extract countries that have their life expectancy either below 40 or above 80 in 2002.

Exercise 10

Now, the soccer/football data from exercise 6 came with goals made and goals made against each team as the following:

```
gA_2014b <- data.table(
  country   = c("Brazil", "Mexico", "Croatia", "Cameroon"),
  goals2014 = c("7-2", "4-1", "6-6", "1-9")
)
```

How can you split the goals column into two relevant columns?

(Image by National Museum Wales)