SAS® GLOBAL FORUM 2020

MARCH 29 – APRIL 1
WASHINGTON, DC

USERS PROGRAM

{An}alytium

Analyse + Implement + Maximise

Using SAS9API and R to Create Violin Plots, Interactive 3D Plots, and a Shiny App for SAS® Data Sets

Olga Pavlova
Analytium

About SAS9API

Violin Plots

3D Plots

Shiny App

Conclusion

SAS9API
Focusing on simplicity
sas9api.io

❖ Introduction

Open-source tools are extremely popular within the data science community and R language is one of them.

While SAS Viya allows easy collaboration between SAS and open-source languages like R and Python using HTTP protocol, SAS9 lacks this feature. To address this issue we at Analytium designed our SAS9API solution. It is based on REST API and allows you to connect to your SAS server. With SAS9API help you can get and manage SAS data and metadata.

We also created a wrapper R package *rsas9api*. Among other options, it allows loading SAS data into R.

Here we present different open-source data visualization techniques: violin plots, interactive 3D plots and even more interactive Shiny apps.

You can request a free trial license for SAS9API at our website sas9api.io to try all this functionality yourself.

Olga Pavlova
Junior SAS Consultant

❖ Get required libraries

We will use popular R libraries for creating plots: *ggplot2* and *plotly*.

RColorBrewer library will allow us to create plots with nice colours.

We will need *shiny* library to create Shiny app.

Also we will use *rsas9api* package to send requests to SAS9API and to install it from GitHub we will need *devtools* package.

```
install.packages("ggplot2")
install.packages("RColorBrewer")
install.packages("plotly")
install.packages("shiny")

install.packages("devtools")
devtools::install_github("anlaytium-group/rsas9api")

library(rsas9api)
library(ggplot2)
library(RColorBrewer)
library(plotly)
library(shiny)
```

❖ Define connection properties for SAS9API

To send requests to SAS9API endpoints you need to define:

- URL for SAS9API proxy,
- SAS workspace server name.

```
sas9api_url  <- "your_url"
sas_workspace_server_name <- "your_server"
```

{An}alytium

Analyse + Implement + Maximise

Using SAS9API and R to Create Violin Plots, Interactive 3D Plots, and a Shiny App for SAS® Data Sets

Olga Pavlova
Analytium

About SAS9API

Violin Plots

3D Plots

Shiny App

Conclusion

SAS9API
Focusing on simplicity

sas9api.io/examples/r-violin-plot/

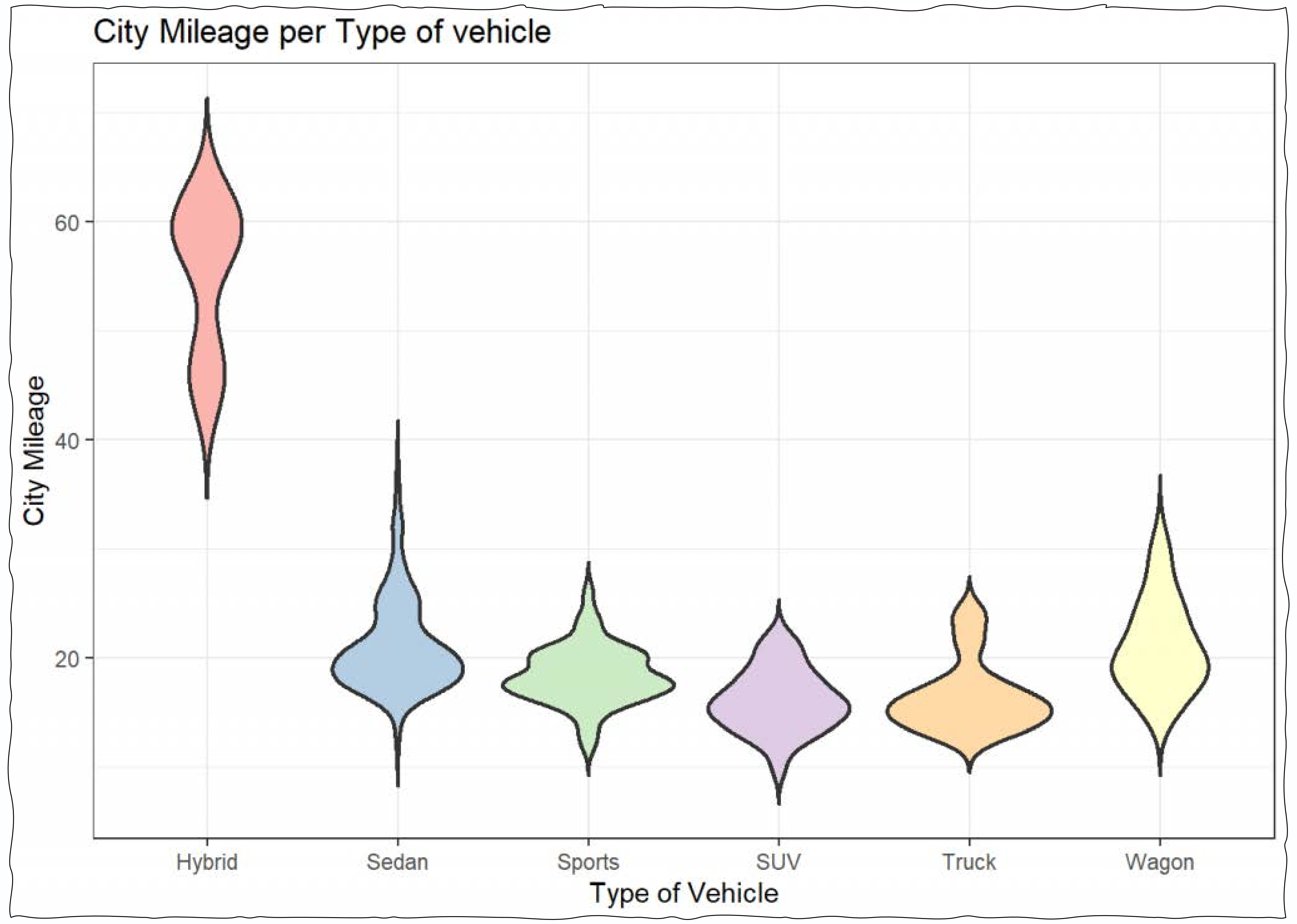
❖ Get SAS dataset data

We will be using *retrieve_data* function from *rsas9api* package. This function allows us to get data from a SAS dataset and to store it as R dataframe.

To send request using *retrieve_data* function you will need to define:

- library name of the dataset ("*SASHELP*" in this case),
- dataset name ("*CARS*" in our case),
- limit number: number of records to get from the dataset (we will use the maximum value of 10000),
- offset number: number of records to skip from the beginning of dataset (we will leave it at 0),
- *asDataFrame* flag (*TRUE* in our case, as we want our request to return a dataframe).

```
data_cars <- retrieve_data(url = sas9api_url,
                           serverName = sas_workspace_server_name,
                           libraryName = "SASHELP",
                           datasetName = "CARS",
                           limit = 10000, offset = 0,
                           asDataFrame = TRUE)
```



❖ Create violin plot

For our plot we will define x axis as *Type of vehicle* and y axis as *City mileage*.

To create a violin plot in *ggplot2* we will use *geom_violin* geometry. We will set *trim = FALSE* to have long and thin tails. To fill our violins with nice colours we will use *scale_fill_brewer* function and *palette = "Pastel1"*.

```
ggplot(data_cars, aes(x = Type, y = MPG_City, fill = Type)) +
     geom_violin(trim = FALSE, lwd = 0.75) +
     scale_fill_brewer(palette = "Pastel1") +
     labs(title = "City Mileage per Type of vehicle",
          x = "Type of Vehicle",
          y = "City Mileage") +
     theme_bw() +
     theme(legend.position = "none")
```

{An}alytium

Analyse + Implement + Maximise

Using SAS9API and R to Create Violin Plots, Interactive 3D Plots, and a Shiny App for SAS® Data Sets

Olga Pavlova
Analytium

About SAS9API

Violin Plots

3D Plots

Shiny App

Conclusion

SAS9API
Focusing on simplicity

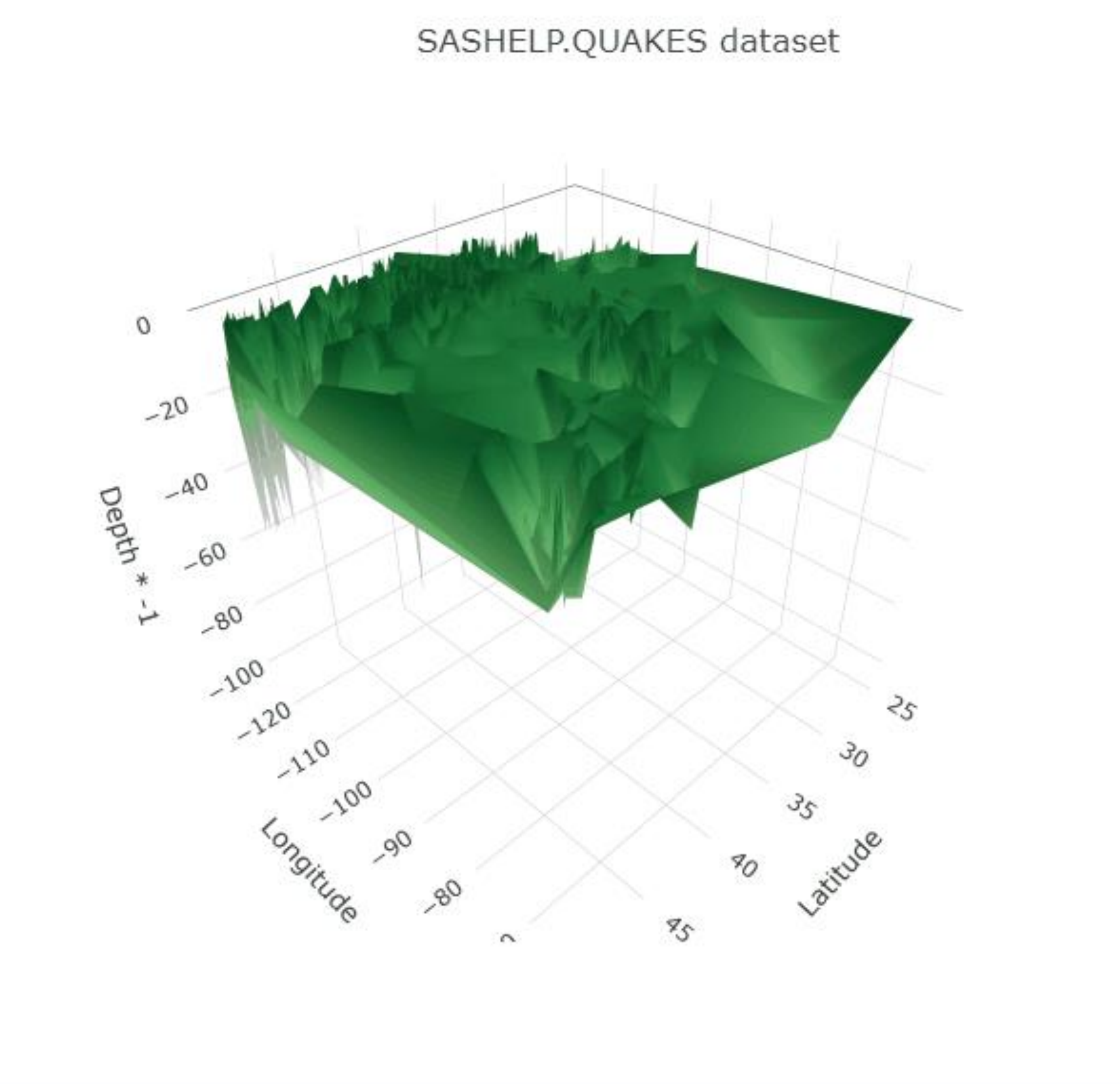
sas9api.io/examples/r-3d-plot/

❖ Get SAS dataset data

We use same *retrieve_data* function from *rsas9api* package to get the dataset (“*QUAKES*” in this case). All additional parameters are the same as in the previous example.

What is different in this case is that Quakes dataset has more than 10000 rows, so we will have to make two requests to get the full dataset. We will then bind both parts into one dataframe.

```
data_quakes1 <- retrieve_data(url = sas9api_url,
                               serverName = sas_workspace_serve_name,
                              libraryName = "SASHELP",
                              datasetName = "QUAKES",
                              limit = 10000, offset = 0,
                              asDataFrame = TRUE)
data_quakes2 <- retrieve_data(url = sas9api_url,
                               serverName = sas_workspace_serve_name,
                               libraryName = "SASHELP",
                               datasetName = "QUAKES",
                               limit = 10000, offset = 10000,
                               asDataFrame = TRUE)
data_quakes <- rbind(data_quakes1, data_quakes2)
```



❖ Create 3D plot

To create a 3D plot in *plotly* we will use function *plot_ly*. We will use *mesh3d* type to create a surface.

For our plot we will define x axis as *Latitude*, y axis as *Longitude* and z axis as *Depth*-1* (to get values below zero).

```
plot_ly(data = data_quakes,
        x = ~Latitude,
        y = ~Longitude,
        z = ~Depth*-1,
        intensity = ~Depth*-1,
        colors = "PRGn",
        type = "mesh3d")  %>%
   layout(title = "SASHELP.QUAKES dataset")
```

{An}alytium

Analyse + Implement + Maximise

Using SAS9API and R to Create Violin Plots, Interactive 3D Plots, and a Shiny App for SAS® Data Sets

Olga Pavlova
Analytium

About SAS9API

Violin Plots

3D Plots

Shiny App

Conclusion

SAS9API
Focusing on simplicity
sas9api.io/examples/shiny-app/

❖ Get SAS dataset data

We use same *retrieve_data* function from *rsas9api* package to get the dataset (“*STOCKS*” in this case). All additional parameters are the same as in previous examples.

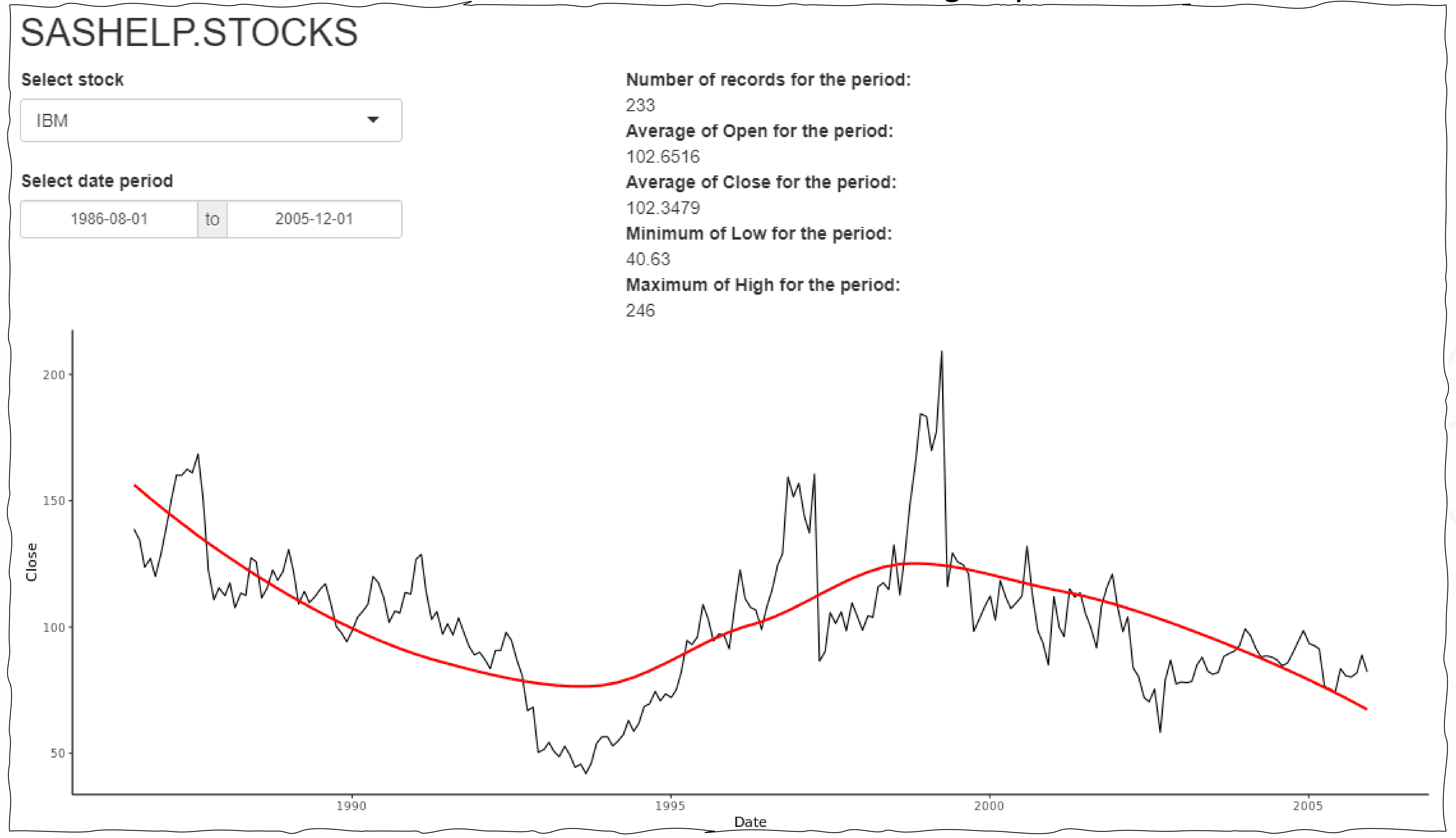
```
data_cars <- retrieve_data(url = sas9api_url,
                           serverName = sas_workspace_server_name,
                           libraryName = "SASHELP",
                           datasetName = "STOCKS",
                           limit = 10000, offset = 0,
                           asDataFrame = TRUE)
```

❖ Create UI part

Shiny has several options for UI layout. We use fluid rows layout in this example. Our UI will have a title defined using *titlePanel* function and two rows defined by *fluidRow* function.

The first column of the first row contains input elements: drop-down list created by *selectInput* function and two date fields defined by *dateRangeInput* function. The second column will have the output as *Number of records for the period*. The second row contains the plot.

```
ui <- fluidPage(
        titlePanel("SASHELP.STOCKS"),
        fluidRow(column(width = 4,
          selectInput(inputId = "selectStock", label = "Select stock",
                      choices = c("IBM", "Intel", "Microsoft")),
          dateRangeInput(inputId = "Dates", label = "Select dates",
                         start = "1986-08-01", end = "2005-12-01")),
          column(width = 7, offset = 1,
                 strong("Number of records for the period:"),
                 textOutput("number"))),
        fluidRow(column(width = 12, plotOutput("plot1"))))
```



SASHELP.STOCKS

Select stock
IBM

Select date period
1986-08-01 to 2005-12-01

Number of records for the period:
233
Average of Open for the period:
102.6516
Average of Close for the period:
102.3479
Minimum of Low for the period:
40.63
Maximum of High for the period:
246

❖ Create Server part

Server part is actually the one that does all the calculations.

So first we need to subset our dataset using the input data: start and end date of the period and stock name. And then we need to calculate the number of records used and create the plot.

```
server <- function(input, output) {
        datasubset <- reactive({
          data$Date <- as.Date(data$Date)
          subset(data, Date >= input$Dates[1] &
              Date <= input$Dates[2] & Stock == input$selectStock)})
          output$number <- renderText({nrow(datasubset())})
          output$plot1 <- renderPlot({ggplot(datasubset(),
                            aes(x = Date, y = Close, group = Stock)) +
                            geom_line() +
                            geom_smooth(method = "loess",
                                        se = FALSE, colour = "red") +
                            theme_classic()})}
```

❖ Run Shiny app

To start our Shiny app run:

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

{An}alytium

Analyse + Implement + Maximise

Using SAS9API and R to Create Violin Plots, Interactive 3D Plots, and a Shiny App for SAS® Data Sets

Olga Pavlova
Analytium

About SAS9API

Violin Plots

3D Plots

Shiny App

Conclusion

SAS9API

Focusing on simplicity

sas9api.io

❖ Conclusion

We presented here how SAS9API can be used to get data from SAS datasets into R. Our rsas9api package for R language has easy to use functions for all SAS9API endpoints and those go well beyond data extraction. You can not only read but also update data in datasets, get metadata information about different objects and much more. You can see the full list of available endpoints at Docs page on our website: sas9api.io/docs/.

SAS9API is based on REST API so its functionality can be used with any language that can send HTTP requests. And apart from R package we also created a similar wrapper package for Python that can be found here: github.com/anlaytium-group/python-sas9api.

❖ References

- Code on GitHub: github.com/sascommunities/sas-global-forum-2020/tree/master/papers/4901-2020-Pavlova
- GitHub page for rsas9api package: github.com/anlaytium-group/rsas9api
- Documentation on SAS9API endpoints: sas9api.io/docs/
- More examples of using SAS9API: sas9api.io/examples/
- Free trial license for SAS9API: sas9api.io/free-trial/

❖ Acknowledgement

I would like to thank SAS for this excellent opportunity to present our SAS9API solution.

I am also very grateful for being selected to receive one of the SAS International Professional Awards.

❖ Contact Information

Olga Pavlova
Email: olga.pavlova@analytium.co.uk
LinkedIn: linkedin.com/in/ms-olga-pavlova/

SAS® GLOBAL FORUM 2020

USERS PROGRAM

MARCH 29 – APRIL 1 | WASHINGTON, DC | #SASGF