Galaaz Manual

How to tightly couple Ruby and R in GraalVM

Rodrigo Botafogo

2019

Contents

1 Introduction **2**
1.1 What does Galaaz mean . . . 4

2 System Compatibility **4**

3 Dependencies **4**

4 Installation **4**

5 Usage **4**

6 Accessing R from Ruby **5**

7 gKnitting a Document **7**
7.1 gKnit and **R markdown** . . . 9
7.2 The Yaml header . . . 9
7.3 **R Markdown** formatting . . . 10
7.4 Headers . . . 10
7.5 Lists . . . 10
7.6 R chunks . . . 10
7.7 R Graphics with ggplot . . . 11
7.8 Ruby chunks . . . 13
7.9 Inline Ruby code . . . 14
7.9.1 The 'outputs' function . . . 14
7.9.2 HTML Output from Ruby Chunks . . . 15
7.10 Including Ruby files in a chunk . . . 16
7.11 Documenting Gems . . . 18
7.12 Converting to PDF . . . 20
7.13 Template based documents generation . . . 20

8 Accessing R variables **21**

9 Basic Data Types **22**
9.1 Vector . . . 22
9.1.1 Combining Vectors . . . 24
9.1.2 Vector Arithmetic . . . 25
9.1.3 Vector Indexing . . . 25
9.1.4 Extracting Native Ruby Types from a Vector . . . 26
9.2 Matrix . . . 26
9.2.1 Indexing a Matrix . . . 27
9.3 List . . . 28
9.3.1 List Indexing . . . 28
9.4 Data Frame . . . 29

9.4.1 Data Frame Indexing 29

10 Writing Expressions in Galaaz **31**
10.1 Expressions from operators 31
10.2 Expressions with R methods 32
10.3 Evaluating an Expression 32

11 Manipulating Data **33**
11.1 Filtering rows with Filter 33
11.2 Logical Operators 34
11.3 Filtering with NA (Not Available) 35
11.4 Arrange Rows with arrange 35
11.5 Selecting columns 36
11.6 Add variables to a dataframe with 'mutate' 37
11.7 Summarising data 38

12 Using Data Table **39**

13 Graphics in Galaaz **41**

14 Coding with Tidyverse **43**
14.1 Writing a function that applies to different data sets 44
14.2 Different expressions 45
14.3 Different input variables 47
14.4 Different input and output variable 48
14.5 Capturing multiple variables 49
14.6 Why does R require NSE and Galaaz does not? 50
14.7 Advanced dplyr features 50

15 Contributing **52**

References **52**

1 Introduction

Galaaz is a system for tightly coupling Ruby and R. Ruby is a powerful language, with a large community, a very large set of libraries and great for web development. However, it lacks libraries for data science, statistics, scientific plotting and machine learning. On the other hand, R is considered one of the most powerful languages for solving all of the above problems. Maybe the strongest competitor to R is Python with libraries such as NumPy, Panda, SciPy, SciKit-Learn and a couple more.

With Galaaz we do not intend to re-implement any of the scientific libraries in R, we allow for very tight coupling between the two languages to the point that the Ruby developer does not need to know that there is an R engine running.

According to Wikipedia "Ruby is a dynamic, interpreted, reflective, object-oriented, general-purpose programming language. It was designed and developed in the mid-1990s by Yukihiro"Matz" Matsumoto in Japan." It reached high popularity with the development of Ruby on Rails (RoR) by David Heinemeier Hansson. RoR is a web application framework first released around 2005. It makes extensive use of Ruby's metaprogramming features. With RoR, Ruby became very popular. According to Ruby's Tiobe index it peeked in popularity around 2008,

then declined until 2015 when it started picking up again. At the time of this writing (November 2018), the Tiobe index puts Ruby in 16th position as most popular language.

Python, a language similar to Ruby, ranks 4th in the index. Java, C and C++ take the first three positions. Ruby is often criticized for its focus on web applications. But Ruby can do much more than just web applications. Yet, for scientific computing, Ruby lags way behind Python and R. Python has Django framework for web, NumPy for numerical arrays, Pandas for data analysis. R is a free software environment for statistical computing and graphics with thousands of libraries for data analysis.

Until recently, there was no real perspective for Ruby to bridge this gap. Implementing a complete scientific computing infrastructure would take too long. Enters Oracle's GraalVM:

> GraalVM is a universal virtual machine for running applications written in JavaScript, Python 3, Ruby, R, JVM-based languages like Java, Scala, Kotlin, and LLVM-based languages such as C and C++.
>
> GraalVM removes the isolation between programming languages and enables interoperability in a shared runtime. It can run either standalone or in the context of OpenJDK, Node.js, Oracle Database, or MySQL.
>
> GraalVM allows you to write polyglot applications with a seamless way to pass values from one language to another. With GraalVM there is no copying or marshaling necessary as it is with other polyglot systems. This lets you achieve high performance when language boundaries are crossed. Most of the time there is no additional cost for crossing a language boundary at all.
>
> Often developers have to make uncomfortable compromises that require them to rewrite their software in other languages. For example:
>
> - That library is not available in my language. I need to rewrite it.
> - That language would be the perfect fit for my problem, but we cannot run it in our environment.
> - That problem is already solved in my language, but the language is too slow.
>
> With GraalVM we aim to allow developers to freely choose the right language for the task at hand without making compromises.

As stated above, GraalVM is a *universal* virtual machine that allows Ruby and R (and other languages) to run on the same environment. GraalVM allows polyglot applications to *seamlessly* interact with one another and pass values from one language to the other. Although a great idea, GraalVM still requires application writers to know several languages. To eliminate that requirement, we built Galaaz, a gem for Ruby, to tightly couple Ruby and R and allow those languages to interact in a way that the user will be unaware of such interaction. In other words, a Ruby programmer will be able to use all the capabilities of R without knowing the R syntax.

Library wrapping is a usual way of bringing features from one language into another. To improve performance, Python often wraps more efficient C libraries. For the Python developer, the existence of such C libraries is hidden. The problem with library wrapping is that for any new library, there is the need to handcraft a new wrapper.

Galaaz, instead of wrapping a single C or R library, wraps the whole R language in Ruby. Doing so, all thousands of R libraries are available immediately to Ruby developers without any new wrapping effort.

1.1 What does Galaaz mean

Galaaz is the Portuguese name for “Galahad”. From Wikipedia:

```
Sir Galahad (sometimes referred to as Galeas or Galath),
in Arthurian legend, is a knight of King Arthur's Round Table and one
of the three achievers of the Holy Grail. He is the illegitimate son
of Sir Lancelot and Elaine of Corbenic, and is renowned for his
gallantry and purity as the most perfect of all knights. Emerging quite
late in the medieval Arthurian tradition, Sir Galahad first appears in the
Lancelot-Grail cycle, and his story is taken up in later works such as
the Post-Vulgate Cycle and Sir Thomas Malory's Le Morte d'Arthur.
His name should not be mistaken with Galehaut, a different knight from
Arthurian legend.
```

2 System Compatibility

- Oracle Linux 7
- Ubuntu 18.04 LTS
- Ubuntu 16.04 LTS
- Fedora 28
- macOS 10.14 (Mojave)
- macOS 10.13 (High Sierra)

3 Dependencies

- TruffleRuby
- FastR

4 Installation

- Install GrallVM (http://www.graalvm.org/)
- Install Ruby (gu install Ruby)
- Install FastR (gu install R)
- Install rake if you want to run the specs and examples (gem install rake)

5 Usage

- Interactive shell: use ‘gstudio’ on the command line

  gstudio

```
vec = R.c(1, 2, 3, 4)
puts vec
```

```
## [1] 1 2 3 4
```

- Run all specs

galaaz specs:all

- Run graphics slideshow (80+ graphics)

  galaaz sthda:all

- Run labs from Introduction to Statistical Learning with R

  galaaz islr:all

- See all available examples

  galaaz -T

Shows a list with all available executalbe tasks. To execute a task, substitute the 'rake' word in the list with 'galaaz'. For instance, the following line shows up after 'galaaz -T'

rake master_list:scatter_plot # scatter_plot from:. . . .

execute

galaaz master_list:scatter_plot

6 Accessing R from Ruby

One of the nice aspects of Galaaz on GraalVM, is that variables and functions defined in R, can be easily accessed from Ruby. For instance, to access the 'mtcars' data frame from R in Ruby, we use the ':mtcar' symbol preceded by the '~' operator, thus '~:r_vec' retrieves the value of the 'mtcars' variable.

```
puts ~:mtcars
```

```
##                      mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4           21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag       21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710          22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Valiant             18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 240D           24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230            22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic         30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Toyota Corona       21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
```

```
## Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## Fiat X1-9           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2       26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

To access an R function from Ruby, the R function needs to be preceeded by 'R.' scoping. Bellow we see and example of creating a R::Vector by calling the 'c' R function

```
puts vec = R.c(1.0, 2.0, 3.0, 4.0)
```

```
## [1] 1 2 3 4
```

Note that 'vec' is an object of type R::Vector:

```
puts vec.class
```

```
## R::Vector
```

Every object created by a call to an R function will be of a type that inherits from R::Object. In R, there is also a function 'class'. In order to access that function we can call method 'rclass' in the R::Object:

```
puts vec.rclass
```

```
## [1] "numeric"
```

When working with R::Object(s), it is possible to use the '.' operator to pipe operations. When using '.', the object to which the '.' is applied becomes the first argument of the corresponding R function. For instance, function 'c' in R, can be used to concatenate two vectors or more vectors (in R, there are no scalar values, scalars are converted to vectors of size 1. Within Galaaz, scalar parameter is converted to a size one vector):

```
puts R.c(vec, 10, 20, 30)
```

```
## [1]  1  2  3  4 10 20 30
```

The call above to the 'c' function can also be done using '.' notation:

```
puts vec.c(10, 20, 30)
```

```
## [1]  1  2  3  4 10 20 30
```

We will talk about vector indexing in a latter section. But notice here that indexing an R::Vector will return another R::Vector:

```
puts vec[1]
```

```
## [1] 1
```

Sometimes we want to index an R::Object and get back a Ruby object that is not wrapped in an R::Object, but the native Ruby object. For this, we can index the R object with the '>>' operator:

```
puts vec >> 0
puts vec >> 2
```

```
## 1.0
## 3.0
```

It is also possible to call an R function with named arguments, by creating the function in Galaaz with named parameters. For instance, here is an example of creating a 'list' with named elements:

```
puts R.list(first_name: "Rodrigo", last_name: "Botafogo")
```

```
## $first_name
## [1] "Rodrigo"
##
## $last_name
## [1] "Botafogo"
```

Many R functions receive another function as argument. For instance, method 'map' applies a function to every element of a vector. With Galaaz, it is possible to pass a Proc, Method or Lambda in place of the expected R function. In this next example, we will add 2 to every element of our previously created vector:

```
puts vec.map { |x| x + 2 }
```

```
## [1] 3
## [1] 4
## [1] 5
## [1] 6
```

7 gKnitting a Document

This manual has been formatted usign gKnit. gKnit uses Knitr and R markdown to knit a document in Ruby or R and output it in any of the available formats for R markdown. gKnit runs atop of GraalVM, and Galaaz. In gKnit, Ruby variables are persisted between chunks, making it an ideal solution for literate programming. Also, since it is based on Galaaz, Ruby chunks can have access to R variables and Polyglot Programming with Ruby and R is quite natural.

The idea of "literate programming" was first introduced by Donald Knuth in the 1980's (Knuth 1984). The main intention of this approach was to develop software interspersing macro snippets, traditional source code, and a natural language such as English in a document that could be compiled into executable code and at the same time easily read by a human developer. According to Knuth "The practitioner of literate programming can be regarded as an essayist, whose main concern is with exposition and excellence of style."

The idea of literate programming evolved into the idea of reproducible research, in which all the data, software code, documentation, graphics etc. needed to reproduce the research and its reports could be included in a single document or set of documents that when distributed to peers could be rerun generating the same output and reports.

The R community has put a great deal of effort in reproducible research. In 2002, Sweave was introduced and it allowed mixing R code with Latex generating high quality PDF documents. A Sweave document could include code, the results of executing the code, graphics and text such that it contained the whole narrative to reproduce the research. In 2012, Knitr, developed by Yihui Xie from RStudio was released to replace Sweave and to consolidate in one single package the many extensions and add-on packages that were necessary for Sweave.

With Knitr, **R markdown** was also developed, an extension to the Markdown format. With **R markdown** and Knitr it is possible to generate reports in a multitude of formats such as HTML, markdown, Latex, PDF, dvi, etc. **R markdown** also allows the use of multiple programming languages such as R, Ruby, Python, etc. in the same document.

In **R markdown**, text is interspersed with code chunks that can be executed and both the code and its results can become part of the final report. Although **R markdown** allows multiple programming languages in the same document, only R and Python (with the reticulate package) can persist variables between chunks. For other languages, such as Ruby, every chunk will start a new process and thus all data is lost between chunks, unless it is somehow stored in a data file that is read by the next chunk.

Being able to persist data between chunks is critical for literate programming otherwise the flow of the narrative is lost by all the effort of having to save data and then reload it. Although this might, at first, seem like a small nuisance, not being able to persist data between chunks is a major issue. For example, let's take a look at the following simple example in which we want to show how to create a list and the use it. Let's first assume that data cannot be persisted between chunks. In the next chunk we create a list, then we would need to save it to file, but to save it, we need somehow to marshal the data into a binary format:

```
lst = R.list(a: 1, b: 2, c: 3)
lst.saveRDS("lst.rds")
```

then, on the next chunk, where variable 'lst' is used, we need to read back it's value

```
lst = R.readRDS("lst.rds")
puts lst
```

```
## $a
## [1] 1
##
## $b
## [1] 2
##
## $c
## [1] 3
```

Now, any single code has dozens of variables that we might want to use and reuse between chunks. Clearly, such an approach becomes quickly unmanageable. Probably, because of this problem, it is very rare to see any **R markdown** document in the Ruby community.

When variables can be used accross chunks, then no overhead is needed:

```
lst = R.list(a: 1, b: 2, c: 3)
# any other code can be added here
```

```
puts lst
```

```
## $a
## [1] 1
##
## $b
## [1] 2
##
## $c
## [1] 3
```

In the Python community, the same effort to have code and text in an integrated environment started around the first decade of 2000. In 2006 iPython 0.7.2 was released. In 2014, Fernando Pérez, spun off project Jupyter from iPython creating a web-based interactive computation environment. Jupyter can now be used with many languages, including Ruby with the iruby gem (https://github.com/SciRuby/iruby). In order to have multiple languages in a Jupyter notebook the SoS kernel was developed (https://vatlab.github.io/sos-docs/).

7.1 gKnit and R markdown

gKnit is based on knitr and **R markdown** and can knit a document written both in Ruby and/or R and output it in any of the available formats of **R markdown**. gKnit allows ruby developers to do literate programming and reproducible research by allowing them to have in a single document, text and code.

In gKnit, Ruby variables are persisted between chunks, making it an ideal solution for literate programming in this language. Also, since it is based on Galaaz, Ruby chunks can have access to R variables and Polyglot Programming with Ruby and R is quite natural.

This is not a blog post on **R markdown**, and the interested user is directed to the following links for detailed information on its capabilities and use.

- https://rmarkdown.rstudio.com/ or
- https://bookdown.org/yihui/rmarkdown/

In this post, we will describe just the main aspects of **R markdown**, so the user can start gKnitting Ruby and R documents quickly.

7.2 The Yaml header

An **R markdown** document should start with a Yaml header and be stored in a file with '.Rmd' extension. This document has the following header for gKitting an HTML document.

```
---
title: "How to do reproducible research in Ruby with gKnit"
author:
    - "Rodrigo Botafogo"
    - "Daniel Mossé - University of Pittsburgh"
tags: [Tech, Data Science, Ruby, R, GraalVM]
date: "20/02/2019"
output:
  html_document:
    self_contained: true
    keep_md: true
  pdf_document:
    includes:
      in_header: ["../../sty/galaaz.sty"]
    number_sections: yes
---
```

For more information on the options in the Yaml header, check here.

7.3 R Markdown formatting

Document formatting can be done with simple markups such as:

7.4 Headers

```
# Header 1

## Header 2

### Header 3
```

7.5 Lists

```
Unordered lists:

* Item 1
* Item 2
    + Item 2a
    + Item 2b

Ordered Lists

1. Item 1
2. Item 2
3. Item 3
    + Item 3a
    + Item 3b
```

For more R markdown formatting go to https://rmarkdown.rstudio.com/authoring_basics.html.

7.6 R chunks

Running and executing Ruby and R code is actually what really interests us is this blog. Inserting a code chunk is done by adding code in a block delimited by three back ticks followed by an open curly brace (‘{’) followed with the engine name (r, ruby, rb, include, . . .), an any optional chunk_label and options, as shown bellow:

```
```{engine_name [chunk_label], [chunk_options]}
```
```

for instance, let’s add an R chunk to the document labeled ‘first_r_chunk’. This is a very simple code just to create a variable and print it out, as follows:

```
```{r first_r_chunk}
vec <- c(1, 2, 3)
print(vec)
```
```

If this block is added to an **R markdown** document and gKnitted the result will be:

```r
vec <- c(1, 2, 3)
print(vec)
```

```
## [1] 1 2 3
```

Now let's say that we want to do some analysis in the code, but just print the result and not the code itself. For this, we need to add the option 'echo = FALSE'.

````
```{r second_r_chunk, echo = FALSE}
vec2 <- c(10, 20, 30)
vec3 <- vec * vec2
print(vec3)
```
````

Here is how this block will show up in the document. Observe that the code is not shown and we only see the execution result in a white box

```
## [1] 10 40 90
```

A description of the available chunk options can be found in https://yihui.name/knitr/.

Let's add another R chunk with a function definition. In this example, a vector 'r_vec' is created and a new function 'reduce_sum' is defined. The chunk specification is

````
```{r data_creation}
r_vec <- c(1, 2, 3, 4, 5)

reduce_sum <- function(...) {
  Reduce(sum, as.list(...))
}
```
````

and this is how it will look like once executed. From now on, to be concise in the presentation we will not show chunk definitions any longer.

```r
r_vec <- c(1, 2, 3, 4, 5)

reduce_sum <- function(...) {
  Reduce(sum, as.list(...))
}
```

We can, possibly in another chunk, access the vector and call the function as follows:

```r
print(r_vec)
```

```
## [1] 1 2 3 4 5
```

```r
print(reduce_sum(r_vec))
```

```
## [1] 15
```

7.7 R Graphics with ggplot

In the following chunk, we create a bubble chart in R using ggplot and include it in this document. Note that there is no directive in the code to include the image, this occurs automatically. The 'mpg' dataframe is natively available to R and to Galaaz as well.

For the reader not knowledgeable of ggplot, ggplot is a graphics library based on "the grammar of graphics" (Wilkinson 2005). The idea of the grammar of graphics is to build a graphics by

adding layers to the plot. More information can be found in https://towardsdatascience.com/a-comprehensive-guide-to-the-grammar-of-graphics-for-effective-visualization-of-multi-dimensional-1f92b4ed414

In the plot bellow the 'mpg' dataset from base R is used. "The data concerns city-cycle fuel consumption in miles per gallon, to be predicted in terms of 3 multivalued discrete and 5 continuous attributes." (Quinlan, 1993)

First, the 'mpg' dataset if filtered to extract only cars from the following manumactures: Audi, Ford, Honda, and Hyundai and stored in the 'mpg_select' variable. Then, the selected dataframe is passed to the ggplot function specifying in the aesthetic method (aes) that 'displacement' (disp) should be plotted in the 'x' axis and 'city mileage' should be on the 'y' axis. In the 'labs' layer we pass the 'title' and 'subtitle' for the plot. To the basic plot 'g', geom_jitter is added, that plots cars from the same manufactures with the same color (col=manufactures) and the size of the car point equal its high way consumption (size = hwy). Finally, a last layer is plotter containing a linear regression line (method = "lm") for every manufacturer.

```
# load package and data
library(ggplot2)
```

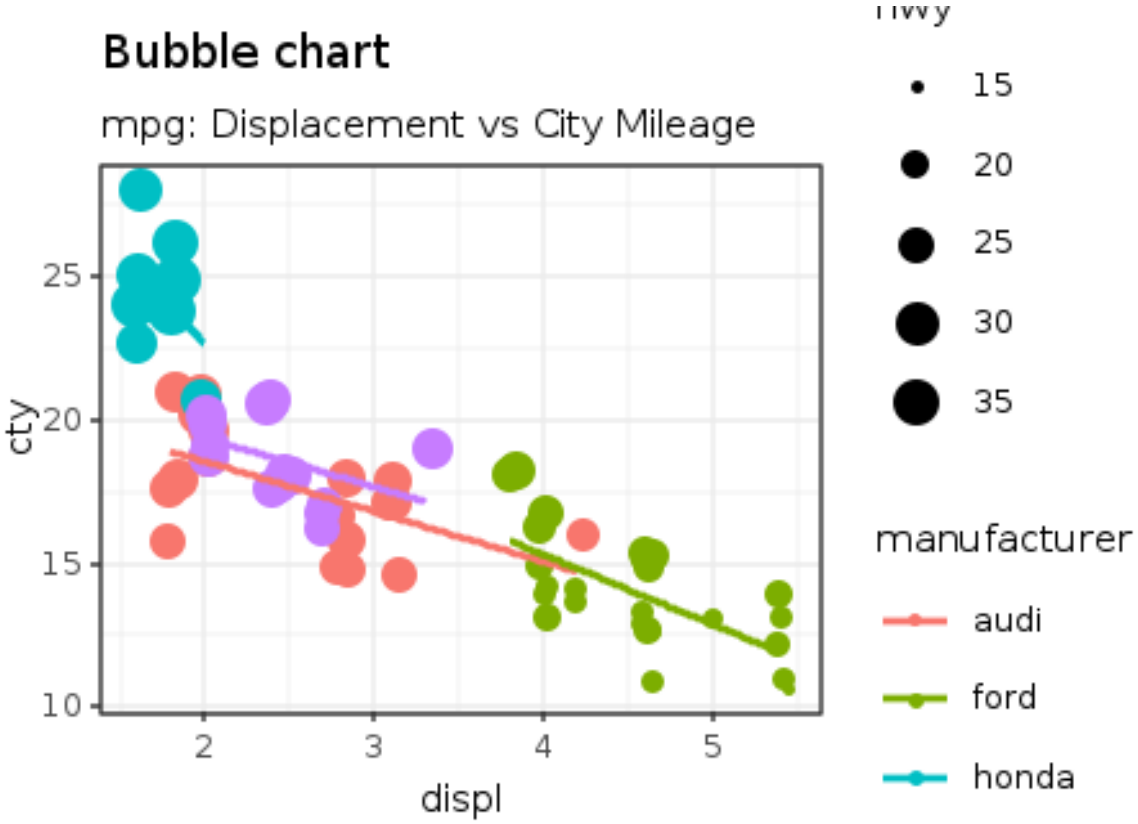
```
## Message:
##  Registered S3 methods overwritten by 'ggplot2':
##   method         from
##   [.quosures     rlang
##   c.quosures     rlang
##   print.quosures rlang
```

```
data(mpg, package="ggplot2")

mpg_select <- mpg[mpg$manufacturer %in% c("audi", "ford", "honda", "hyundai"), ]

# Scatterplot
theme_set(theme_bw())  # pre-set the bw theme.
g <- ggplot(mpg_select, aes(displ, cty)) +
  labs(subtitle="mpg: Displacement vs City Mileage",
       title="Bubble chart")

g + geom_jitter(aes(col=manufacturer, size=hwy)) +
  geom_smooth(aes(col=manufacturer), method="lm", se=F)
```



7.8 Ruby chunks

Including a Ruby chunk is just as easy as including an R chunk in the document: just change the name of the engine to 'ruby'. It is also possible to pass chunk options to the Ruby engine; however, this version does not accept all the options that are available to R chunks. Future versions will add those options.

```
```{ruby first_ruby_chunk}
```
```

In this example, the ruby chunk is called 'first_ruby_chunk'. One important aspect of chunk labels is that they cannot be duplicated. If a chunk label is duplicated, gKnit will stop with an error.

In the following chunk, variable 'a', 'b' and 'c' are standard Ruby variables and 'vec' and 'vec2' are two vectors created by calling the 'c' method on the R module.

In Galaaz, the R module allows us to access R functions transparently. The 'c' function in R, is a function that concatenates its arguments making a vector.

It should be clear that there is no requirement in gknit to call or use any R functions. gKnit will knit standard Ruby code, or even general text without any code.

```ruby
a = [1, 2, 3]
b = "US$ 250.000"
c = "The 'outputs' function"

vec = R.c(1, 2, 3)
vec2 = R.c(10, 20, 30)
```

In the next block, variables ‘a’, ‘vec’ and ‘vec2’ are used and printed.

```
puts a
puts vec * vec2
```

```
## 1
## 2
## 3
## [1] 10 40 90
```

Note that ‘a’ is a standard Ruby Array and ‘vec’ and ‘vec2’ are vectors that behave accordingly, where multiplication works as expected.

7.9 Inline Ruby code

When using a Ruby chunk, the code and the output are formatted in blocks as seen above. This formatting is not always desired. Sometimes, we want to have the results of the Ruby evaluation included in the middle of a phrase. gKnit allows adding inline Ruby code with the ‘rb’ engine. The following chunk specification will create and inline Ruby text:

```
This is some text with inline Ruby accessing variable 'b' which has value:
```{rb puts b}
```
and is followed by some other text!
```

This is some text with inline Ruby accessing variable ‘b’ which has value: US$ 250.000 and is followed by some other text!

Note that it is important not to add any new line before of after the code block if we want everything to be in only one line, resulting in the following sentence with inline Ruby code.

7.9.1 The ‘outputs’ function

He have previously used the standard ‘puts’ method in Ruby chunks in order produce output. The result of a ‘puts’, as seen in all previous chunks that use it, is formatted inside a white box that follows the code block. Many times however, we would like to do some processing in the Ruby chunk and have the result of this processing generate and output that is “included” in the document as if we had typed it in **R markdown** document.

For example, suppose we want to create a new heading in our document, but the heading phrase is the result of some code processing: maybe it’s the first line of a file we are going to read. Method ‘outputs’ adds its output as if typed in the **R markdown** document.

Take now a look at variable ‘c’ (it was defined in a previous block above) as ‘c = “The ’outputs’ function”. “The ’outputs’ function” is actually the name of this section and it was created using the ’outputs’ function inside a Ruby chunk.

The ruby chunk to generate this heading is:

```
```{ruby heading}
outputs "### #{c}"
```
```

The three ‘###’ is the way we add a Heading 3 in **R markdown**.

7.9.2 HTML Output from Ruby Chunks

We've just seen the use of method 'outputs' to add text to the the **R markdown** document. This technique can also be used to add HTML code to the document. In **R markdown**, any html code typed directly in the document will be properly rendered.
Here, for instance, is a table definition in HTML and its output in the document:

```
<table style="width:100%">
  <tr>
    <th>Firstname</th>
    <th>Lastname</th>
    <th>Age</th>
  </tr>
  <tr>
    <td>Jill</td>
    <td>Smith</td>
    <td>50</td>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson</td>
    <td>94</td>
  </tr>
</table>
```

```
<th>Firstname</th>
<th>Lastname</th>
<th>Age</th>
```

```
<td>Jill</td>
<td>Smith</td>
<td>50</td>
```

```
<td>Eve</td>
<td>Jackson</td>
<td>94</td>
```

But manually creating HTML output is not always easy or desirable, specially if we intend the document to be rendered in other formats, for example, as Latex. Also, The above table looks ugly. The 'kableExtra' library is a great library for creating beautiful tables. Take a look at https://cran.r-project.org/web/packages/kableExtra/vignettes/awesome_table_in_html.html

In the next chunk, we output the 'mtcars' dataframe from R in a nicely formatted table. Note that we retrieve the mtcars dataframe by using '~:mtcars'.

```
R.install_and_loads('kableExtra')
outputs (~:mtcars).kable.kable_styling
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4
Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3
Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3
Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3
Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4
Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4
Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4
Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1
Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1
Dodge Challenger	15.5	8	318.0	150	2.76	3.520	16.87	0	0	3	2
AMC Javelin	15.2	8	304.0	150	3.15	3.435	17.30	0	0	3	2
Camaro Z28	13.3	8	350.0	245	3.73	3.840	15.41	0	0	3	4
Pontiac Firebird	19.2	8	400.0	175	3.08	3.845	17.05	0	0	3	2
Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1
Porsche 914-2	26.0	4	120.3	91	4.43	2.140	16.70	0	1	5	2
Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.90	1	1	5	2
Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.50	0	1	5	4
Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.50	0	1	5	6
Maserati Bora	15.0	8	301.0	335	3.54	3.570	14.60	0	1	5	8
Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.60	1	1	4	2

7.10 Including Ruby files in a chunk

R is a language that was created to be easy and fast for statisticians to use. As far as I know, it was not a language to be used for developing large systems. Of course, there are large systems and libraries in R, but the focus of the language is for developing statistical models and distribute that to peers.

Ruby on the other hand, is a language for large software development. Systems written in Ruby will have dozens, hundreds or even thousands of files. To document a large system with literate programming, we cannot expect the developer to add all the files in a single '.Rmd' file. gKnit provides the 'include' chunk engine to include a Ruby file as if it had being typed in the '.Rmd' file.

To include a file, the following chunk should be created, where is the name of the file to be included and where the extension, if it is '.rb', does not need to be added. If the 'relative' option is not included, then it is treated as TRUE. When 'relative' is true, ruby's 'require_relative'

semantics is used to load the file, when false, Ruby's $LOAD_PATH is searched to find the file and it is 'require'd.

```
```{include <filename>, relative = <TRUE/FALSE>}
```
```

Bellow we include file 'model.rb', which is in the same directory of this blog.
This code uses R 'caret' package to split a dataset in a train and test sets. The 'caret' package is a very important a useful package for doing Data Analysis, it has hundreds of functions for all steps of the Data Analysis workflow. To use 'caret' just to split a dataset is like using the proverbial cannon to kill the fly. We use it here only to show that integrating Ruby and R and using even a very complex package as 'caret' is trivial with Galaaz.

A word of advice: the 'caret' package has lots of dependencies and installing it in a Linux system is a time consuming operation. Method 'R.install_and_loads' will install the package if it is not already installed and can take a while.

```
```{include model}
```
```

```
require 'galaaz'

# Loads the R 'caret' package.  If not present, installs it
R.install_and_loads 'caret'

class Model

  attr_reader :data
  attr_reader :test
  attr_reader :train

  #============================================================
  #
  #============================================================

  def initialize(data, percent_train:, seed: 123)

    R.set__seed(seed)
    @data = data
    @percent_train = percent_train
    @seed = seed

  end

  #============================================================
  #
  #============================================================

  def partition(field)

    train_index =
      R.createDataPartition(@data.send(field), p: @percet_train,
                            list: false, times: 1)
    @train = @data[train_index, :all]
```

```
    @test = @data[-train_index, :all]

  end

end
```

```
mtcars = ~:mtcars
model = Model.new(mtcars, percent_train: 0.8)
model.partition(:mpg)
puts model.train.head
puts model.test.head
```

```
##                mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4     21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag 21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Valiant       18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Merc 280      19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 280C     17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Merc 450SE    16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
##                    mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Datsun 710        22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Duster 360        14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 240D         24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230          22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
```

7.11 Documenting Gems

gKnit also allows developers to document and load files that are not in the same directory of the '.Rmd' file.

Here is an example of loading the 'find.rb' file from TruffleRuby. In this example, relative is set to FALSE, so Ruby will look for the file in its $LOAD_PATH, and the user does not need to no it's directory.

```
```{include find, relative = FALSE}
```

```
# frozen_string_literal: true
#
# find.rb: the Find module for processing all files under a given directory.
#

#
# The +Find+ module supports the top-down traversal of a set of file paths.
#
# For example, to total the size of all files under your home directory,
# ignoring anything in a "dot" directory (e.g. $HOME/.ssh):
#
#   require 'find'
#
#   total_size = 0
```
```

```
#
#   Find.find(ENV["HOME"]) do |path|
#     if FileTest.directory?(path)
#       if File.basename(path)[0] == ?.
#         Find.prune       # Don't look any further into this directory.
#       else
#         next
#       end
#     else
#       total_size += FileTest.size(path)
#     end
#   end
#
module Find

  #
  # Calls the associated block with the name of every file and directory listed
  # as arguments, then recursively on their subdirectories, and so on.
  #
  # Returns an enumerator if no block is given.
  #
  # See the +Find+ module documentation for an example.
  #
  def find(*paths, ignore_error: true) # :yield: path
    block_given? or return enum_for(__method__, *paths, ignore_error: ignore_error)

    fs_encoding = Encoding.find("filesystem")

    paths.collect!{|d| raise Errno::ENOENT, d unless File.exist?(d); d.dup}.each do |path|
      path = path.to_path if path.respond_to? :to_path
      enc = path.encoding == Encoding::US_ASCII ? fs_encoding : path.encoding
      ps = [path]
      while file = ps.shift
        catch(:prune) do
          yield file.dup.taint
          begin
            s = File.lstat(file)
          rescue Errno::ENOENT, Errno::EACCES, Errno::ENOTDIR, Errno::ELOOP, Errno::ENAMETOO
            raise unless ignore_error
            next
          end
          if s.directory? then
            begin
              fs = Dir.children(file, encoding: enc)
            rescue Errno::ENOENT, Errno::EACCES, Errno::ENOTDIR, Errno::ELOOP, Errno::ENAMET
              raise unless ignore_error
              next
            end
            fs.sort!
            fs.reverse_each {|f|
              f = File.join(file, f)
```

```
              ps.unshift f.untaint
            }
          end
        end
      end
    end
    nil
  end

  #
  # Skips the current file or directory, restarting the loop with the next
  # entry. If the current file is a directory, that directory will not be
  # recursively entered. Meaningful only within the block associated with
  # Find::find.
  #
  # See the +Find+ module documentation for an example.
  #
  def prune
    throw :prune
  end

  module_function :find, :prune
end
```

7.12 Converting to PDF

One of the beauties of knitr is that the same input can be converted to many different outputs. One very useful format, is, of course, PDF. In order to converted an **R markdown** file to PDF it is necessary to have LaTeX installed on the system. We will not explain here how to install LaTeX as there are plenty of documents on the web showing how to proceed.

gKnit comes with a simple LaTeX style file for gknitting this blog as a PDF document. Here is the Yaml header to generate this blog in PDF format instead of HTML:

```
---
title: "gKnit - Ruby and R Knitting with Galaaz in GraalVM"
author: "Rodrigo Botafogo"
tags: [Galaaz, Ruby, R, TruffleRuby, FastR, GraalVM, knitr, gknit]
date: "29 October 2018"
output:
  pdf\_document:
    includes:
      in\_header: ["../../sty/galaaz.sty"]
    number\_sections: yes
---
```

7.13 Template based documents generation

When a document is converted to PDF it follows a certain convertion template. We've seen above the use of 'galaaz.sty' as a basic template to generate a PDF document. Using the 'gknit-draft' app that comes with Galaaz, the same .Rmd file can be compiled to different looking PDF

documents. Galaaz automatically loads the 'rticles' R package that comes with templates for the following journals with the respective template name:

- ACM articles: acm_article
- ACS articles: acs_article
- AEA journal submissions: aea_article
- AGU journal submissions: ????
- AMS articles: ams_article
- American Statistical Association: asa_article
- Biometrics articles: biometrics_article
- Bulletin de l'AMQ journal submissions: amq_article
- CTeX documents: ctex
- Elsevier journal submissions: elsevier_article
- IEEE Transaction journal submissions: ieee_article
- JSS articles: jss_article
- MDPI journal submissions: mdpi_article
- Monthly Notices of the Royal Astronomical Society articles: mnras_article
- NNRAS journal submissions: nmras_article
- PeerJ articles: peerj_article
- Royal Society Open Science journal submissions: rsos_article
- Royal Statistical Society: rss_article
- Sage journal submissions: sage_article
- Springer journal submissions: springer_article
- Statistics in Medicine journal submissions: sim_article
- Copernicus Publications journal submissions: copernicus_article
- The R Journal articles: rjournal_article
- Frontiers articles: ???
- Taylor & Francis articles: ???
- Bulletin De L'AMQ: amq_article
- PLOS journal: plos_article
- Proceedings of the National Academy of Sciences of the USA: pnas_article

In order to create a document with one of those templates, use the following command:

```
gknit-draft --filename <my_document> --template <template> --package <package>
            --create_dir
```

So, in order to create a template for writing an R Journal, use:

```
gknit-draft --filename my_r_article --template rjournal_article --package rticles
            --create_dir
```

8 Accessing R variables

Galaaz allows Ruby to access variables created in R. For example, the 'mtcars' data set is available in R and can be accessed from Ruby by using the 'tilda' operator followed by the symbol for the variable, in this case ':mtcar'. In the code bellow method 'outputs' is used to output the 'mtcars' data set nicely formatted in HTML by use of the 'kable' and 'kable_styling' functions. Method 'outputs' is only available when used with 'gknit'.

```
outputs (~:mtcars).kable.kable_styling
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4
Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3
Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3
Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3
Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4
Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4
Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4
Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1
Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1
Dodge Challenger	15.5	8	318.0	150	2.76	3.520	16.87	0	0	3	2
AMC Javelin	15.2	8	304.0	150	3.15	3.435	17.30	0	0	3	2
Camaro Z28	13.3	8	350.0	245	3.73	3.840	15.41	0	0	3	4
Pontiac Firebird	19.2	8	400.0	175	3.08	3.845	17.05	0	0	3	2
Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1
Porsche 914-2	26.0	4	120.3	91	4.43	2.140	16.70	0	1	5	2
Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.90	1	1	5	2
Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.50	0	1	5	4
Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.50	0	1	5	6
Maserati Bora	15.0	8	301.0	335	3.54	3.570	14.60	0	1	5	8
Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.60	1	1	4	2

9 Basic Data Types

9.1 Vector

Vectors can be thought of as contiguous cells containing data. Cells are accessed through indexing operations such as x[5]. Galaaz has six basic ('atomic') vector types: logical, integer, real, complex, string (or character) and raw. The modes and storage modes for the different vector types are listed in the following table.

typeof	mode	storage.mode
logical	logical	logical
integer	numeric	integer
double	numeric	double
complex	complex	comples

typeof	mode	storage.mode
character	character	character
raw	raw	raw

Single numbers, such as 4.2, and strings, such as "four point two" are still vectors, of length 1; there are no more basic types. Vectors with length zero are possible (and useful). String vectors have mode and storage mode "character". A single element of a character vector is often referred to as a character string.

To create a vector the 'c' (concatenate) method from the 'R' module should be used:

```
vec = R.c(1, 2, 3)
puts vec
```

```
## [1] 1 2 3
```

Lets take a look at the type, mode and storage.mode of our vector vec. In order to print this out, we are creating a data frame 'df' and printing it out. A data frame, for those not familiar with it, is basically a table. Here we create the data frame and add the column name by passing named parameters for each column, such as 'typeof:', 'mode:' and 'storage___mode¿. You should also note here that the double underscore is converted to a'.'. So, when printed 'storage___mode' will actually print as 'storage.mode'.

Data frames will later be more carefully described. In R, the method used to create a data frame is 'data.frame', in Galaaz we use 'data___frame'.

```
df = R.data__frame(typeof: vec.typeof, mode: vec.mode, storage__mode: vec.storage__mode)
puts df
```

```
##    typeof    mode storage.mode
## 1 integer numeric      integer
```

If you want to create a vector with floating point numbers, then we need at least one of the vector's element to be a float, such as 1.0. R users should be careful, since in R a number like '1' is converted to float and to have an integer the R developer will use '1L'. Galaaz follows normal Ruby rules and the number 1 is an integer and 1.0 is a float.

```
vec = R.c(1.0, 2, 3)
puts vec
```

```
## [1] 1 2 3
```

```
df = R.data__frame(typeof: vec.typeof, mode: vec.mode, storage__mode: vec.storage__mode)
outputs df.kable.kable_styling
```

typeof	mode	storage.mode
double	numeric	double

In this next example we try to create a vector with a variable 'hello' that has not yet being defined. This will raise an exception that is printed out. We get two return blocks, the first with a message explaining what went wrong and the second with the full backtrace of the error.

```
vec = R.c(1, hello, 5)
```

```
## Message:
```

```
##  undefined local variable or method `hello' for #<RC:0x3d8 @out_list=nil>:RC

## Message:
##  /home/rbotafogo/desenv/galaaz/lib/util/exec_ruby.rb:103:in `get_binding'
## /home/rbotafogo/desenv/galaaz/lib/util/exec_ruby.rb:102:in `eval'
## /home/rbotafogo/desenv/galaaz/lib/util/exec_ruby.rb:102:in `exec_ruby'
## /home/rbotafogo/desenv/galaaz/lib/gknit/knitr_engine.rb:650:in `block in initialize'
## /home/rbotafogo/desenv/galaaz/lib/R_interface/ruby_callback.rb:77:in `call'
## /home/rbotafogo/desenv/galaaz/lib/R_interface/ruby_callback.rb:77:in `callback'
## (eval):3:in `function(...) {\n          rb_method(...)'
## unknown.r:1:in `in_dir'
## unknown.r:1:in `block_exec'
## /usr/local/lib/graalvm-ce-java11-20.0.0/languages/R/library/knitr/R/block.R:92:in `call_
## /usr/local/lib/graalvm-ce-java11-20.0.0/languages/R/library/knitr/R/block.R:6:in `proces
## /usr/local/lib/graalvm-ce-java11-20.0.0/languages/R/library/knitr/R/block.R:3:in `<no so
## unknown.r:1:in `withCallingHandlers'
## unknown.r:1:in `process_file'
## unknown.r:1:in `<no source>'
## unknown.r:1:in `<no source>'
## <REPL>:4:in `<repl wrapper>'
## <REPL>:1
```

Here is a vector with logical values

```
vec = R.c(true, true, false, false, true)
puts vec
```

```
## [1]  TRUE  TRUE FALSE FALSE  TRUE
```

9.1.1 Combining Vectors

The 'c' functions used to create vectors can also be used to combine two vectors:

```
vec1 = R.c(10.0, 20.0, 30.0)
vec2 = R.c(4.0, 5.0, 6.0)
vec = R.c(vec1, vec2)
puts vec
```

```
## [1] 10 20 30  4  5  6
```

In galaaz, methods can be chainned (somewhat like the pipe operator in R %>%, but more generic). In this next example, method 'c' is chainned after 'vec1'. This also looks like 'c' is a method of the vector, but in reallity, this is actually closer to the pipe operator. When Galaaz identifies that 'c' is not a method of 'vec' it actually tries to call 'R.c' with 'vec1' as the first argument concatenated with all the other available arguments. The code bellow is automatically converted to the code above.

```
vec = vec1.c(vec2)
puts vec
```

```
## [1] 10 20 30  4  5  6
```

9.1.2 Vector Arithmetic

Arithmetic operations on vectors are performed element by element:

```
puts vec1 + vec2
```

```
## [1] 14 25 36
```

```
puts vec1 * 5
```

```
## [1]  50 100 150
```

When vectors have different length, a recycling rule is applied to the shorter vector:

```
vec3 = R.c(1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0)
puts vec4 = vec1 + vec3
```

```
## [1] 11 22 33 14 25 36 17 28 39
```

9.1.3 Vector Indexing

Vectors can be indexed by using the '[]' operator:

```
puts vec4[3]
```

```
## [1] 33
```

We can also index a vector with another vector. For example, in the code bellow, we take elements 1, 3, 5, and 7 from vec3:

```
puts vec4[R.c(1, 3, 5, 7)]
```

```
## [1] 11 33 25 17
```

Repeating an index and having indices out of order is valid code:

```
puts vec4[R.c(1, 3, 3, 1)]
```

```
## [1] 11 33 33 11
```

It is also possible to index a vector with a negative number or negative vector. In these cases the indexed values are not returned:

```
puts vec4[-3]
puts vec4[-R.c(1, 3, 5, 7)]
```

```
## [1] 11 22 14 25 36 17 28 39
## [1] 22 14 36 28 39
```

If an index is out of range, a missing value (NA) will be reported.

```
puts vec4[30]
```

```
## [1] NA
```

It is also possible to index a vector by range:

```
puts vec4[(2..5)]
```

```
## [1] 22 33 14 25
```

Elements in a vector can be named using the 'names' attribute of a vector:

```
full_name = R.c("Rodrigo", "A", "Botafogo")
full_name.names = R.c("First", "Middle", "Last")
puts full_name
```

```
##      First     Middle       Last
##  "Rodrigo"        "A" "Botafogo"
```

Or it can also be named by using the 'c' function with named paramenters:

```
full_name = R.c(First: "Rodrigo", Middle: "A", Last: "Botafogo")
puts full_name
```

```
##      First     Middle       Last
##  "Rodrigo"        "A" "Botafogo"
```

9.1.4 Extracting Native Ruby Types from a Vector

Vectors created with 'R.c' are of class R::Vector. You might have noticed that when indexing a vector, a new vector is returned, even if this vector has one single element. In order to use R::Vector with other ruby classes it might be necessary to extract the actual Ruby native type from the vector. In order to do this extraction the '>>' operator is used.

```
puts vec4
puts vec4 >> 0
puts vec4 >> 4
```

```
## [1] 11 22 33 14 25 36 17 28 39
## 11.0
## 25.0
```

Note that indexing with '>>' starts at 0 and not at 1, also, we cannot do negative indexing.

9.2 Matrix

A matrix is a collection of elements organized as a two dimensional table. A matrix can be created by the 'matrix' function:

```
mat = R.matrix(R.c(1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0),
               nrow: 3,
               ncol: 3)

puts mat
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

Note that matrices data is organized by column first. It is possible to organize the matrix memory by row first passing an extra argument to the 'matrix' function:

```
mat_row = R.matrix(R.c(1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0),
                   nrow: 3,
```

```
                    ncol: 3,
                    byrow: true)

puts mat_row
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
## [3,]    7    8    9
```

9.2.1 Indexing a Matrix

A matrix can be indexed by [row, column]:

```
puts mat_row[1, 1]
puts mat_row[2, 3]
```

```
## [1] 1
## [1] 6
```

It is possible to index an entire row or column with the ':all' keyword

```
puts mat_row[1, :all]
puts mat_row[:all, 2]
```

```
## [1] 1 2 3
## [1] 2 5 8
```

Indexing with a vector is also possible for matrices. In the following example we want rows 1 and 3 and columns 2 and 3 building a 2 x 2 matrix.

```
puts mat_row[R.c(1, 3), R.c(2, 3)]
```

```
##      [,1] [,2]
## [1,]    2    3
## [2,]    8    9
```

Matrices can be combined with functions 'rbind':

```
puts mat_row.rbind(mat)
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
## [3,]    7    8    9
## [4,]    1    4    7
## [5,]    2    5    8
## [6,]    3    6    9
```

and 'cbind':

```
puts mat_row.cbind(mat)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    1    2    3    1    4    7
## [2,]    4    5    6    2    5    8
```

```
## [3,]    7    8    9    3    6    9
```

9.3 List

A list is a data structure that can contain sublists of different types, while vector and matrix can only hold one type of element.

```
nums = R.c(1.0, 2.0, 3.0)
strs = R.c("a", "b", "c", "d")
bool = R.c(true, true, false)
lst = R.list(nums: nums, strs: strs, bool: bool)
puts lst
```

```
## $nums
## [1] 1 2 3
##
## $strs
## [1] "a" "b" "c" "d"
##
## $bool
## [1]  TRUE  TRUE FALSE
```

Note that 'lst' elements are named elements.

9.3.1 List Indexing

List indexing, also called slicing, is done using the '[]' operator and the '[[]]' operator. Let's first start with the '[]' operator. The list above has three sublist indexing with '[]' will return one of the sublists.

```
puts lst[1]
```

```
## $nums
## [1] 1 2 3
```

Note that when using '[]' a new list is returned. When using the double square bracket operator the value returned is the actual element of the list in the given position and not a slice of the original list

```
puts lst[[1]]
```

```
## [1] 1 2 3
```

When elements are named, as dones with lst, indexing can be done by name:

```
puts lst[['bool']][[1]] >> 0
```

```
## true
```

In this example, first the 'bool' element of the list was extracted, not as a list, but as a vector, then the first element of the vector was extracted (note that vectors also accept the '[[]]' operator) and then the vector was indexed by its first element, extracting the native Ruby type.

9.4 Data Frame

A data frame is a table like structure in which each column has the same number of rows. Data frames are the basic structure for storing data for data analysis. We have already seen a data frame previously when we accessed variable '~:mtcars'. In order to create a data frame, function 'data___frame' is used:

```
df = R.data__frame(
  year: R.c(2010, 2011, 2012),
  income: R.c(1000.0, 1500.0, 2000.0))

puts df
```

```
##   year income
## 1 2010   1000
## 2 2011   1500
## 3 2012   2000
```

9.4.1 Data Frame Indexing

A data frame can be indexed the same way as a matrix, by using '[row, column]', where row and column can either be a numeric or the name of the row or column

```
puts (~:mtcars).head
puts (~:mtcars)[1, 2]
puts (~:mtcars)['Datsun 710', 'mpg']
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
## [1] 6
## [1] 22.8
```

Extracting a column from a data frame as a vector can be done by using the double square bracket operator:

```
puts (~:mtcars)[['mpg']]
```

```
##  [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2
## [15] 10.4 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4
## [29] 15.8 19.7 15.0 21.4
```

A data frame column can also be accessed as if it were an instance variable of the data frame:

```
puts (~:mtcars).mpg
```

```
##  [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2
## [15] 10.4 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4
## [29] 15.8 19.7 15.0 21.4
```

Slicing a data frame can be done by indexing it with a vector (we use 'head' to reduce the output):

```
puts (~:mtcars)[R.c('mpg', 'hp')].head
```

```
##                    mpg  hp
## Mazda RX4         21.0 110
## Mazda RX4 Wag     21.0 110
## Datsun 710        22.8  93
## Hornet 4 Drive    21.4 110
## Hornet Sportabout 18.7 175
## Valiant           18.1 105
```

A row slice can be obtained by indexing by row and using the ':all' keyword for the column:

```
puts (~:mtcars)[R.c('Datsun 710', 'Camaro Z28'), :all]
```

```
##             mpg cyl disp  hp drat   wt  qsec vs am gear carb
## Datsun 710 22.8   4  108  93 3.85 2.32 18.61  1  1    4    1
## Camaro Z28 13.3   8  350 245 3.73 3.84 15.41  0  0    3    4
```

Finally, a data frame can also be indexed with a logical vector. In this next example, the 'am' column of :mtcars is compared with 0 (with method 'eq'). When 'am' is equal to 0 the car is automatic. So, by doing '(~:mtcars).am.eq 0' a logical vector is created with 'true' whenever 'am' is 0 and 'false' otherwise.

```
# obtain a vector with 'true' for cars with automatic transmission
automatic = (~:mtcars).am.eq 0
puts automatic
```

```
##  [1] FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [12]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE  TRUE  TRUE
## [23]  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

Using this logical vector, the data frame is indexed, returning a new data frame in which all cars have automatic transmission.

```
# slice the data frame by using this vector
puts (~:mtcars)[automatic, :all]
```

```
##                     mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Hornet 4 Drive     21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout  18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Valiant            18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Duster 360         14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 240D          24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230           22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 280           19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 280C          17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Merc 450SE         16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Merc 450SL         17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Merc 450SLC        15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## Cadillac Fleetwood 10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4  8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Chrysler Imperial  14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Toyota Corona      21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
```

```
## Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
```

10 Writing Expressions in Galaaz

Galaaz extends Ruby to work with complex expressions, similar to R's expressions build with 'quote' (base R) or 'quo' (tidyverse). Let's take a look at some of those expressions.

10.1 Expressions from operators

The code bellow creates an expression summing two symbols

```
exp1 = :a + :b
puts exp1
```

```
## a + b
```

We can build any complex mathematical expression

```
exp2 = (:a + :b) * 2.0 + :c ** 2 / :z
puts exp2
```

```
## (a + b) * 2 + c^2L/z
```

It is also possible to use inequality operators in building expressions

```
exp3 = (:a + :b) >= :z
puts exp3
```

```
## a + b >= z
```

Galaaz provides both symbolic representations for operators, such as (>, <, !=) as functional notation for those operators such as (.gt, .ge, etc.). So the same expression written above can also be written as

```
exp4 = (:a + :b).ge :z
puts exp4
```

```
## a + b >= z
```

Two type of expression can only be created with the functional representation of the operators, those are expressions involving '==', and '='. In order to write an expression involving '==' we need to use the method '.eq' and for '=' we need the function '.assign'

```
exp5 = (:a + :b).eq :z
puts exp5
```

```
## a + b == z
```

```
exp6 = :y.assign :a + :b
puts exp6
```

```
## y <- a + b
```

In general we think that using the functional notation is preferable to using the symbolic notation as otherwise, we end up writing invalid expressions such as

```
exp_wrong = (:a + :b) == :z
puts exp_wrong
```

and it might be difficult to understand what is going on here. The problem lies with the fact that when using '==' we are comparing expression (:a + :b) to expression :z with '=='. When the comparison is executed, the system tries to evaluate :a, :b and :z, and those symbols at this time are not bound to anything and we get a "object 'a' not found" message. If we only use functional notation, this type of error will not occur.

10.2 Expressions with R methods

It is often necessary to create an expression that uses a method or function. For instance, in mathematics, it's quite natural to write an expressin such as $y = sin(x)$. In this case, the 'sin' function is part of the expression and should not immediately executed. Now, let's say that 'x' is an angle of 45° and we acttually want our expression to be $y = 0.850....$ When we want the function to be part of the expression, we call the function preceeding it by the letter E, such as 'E.sin(x)'

```
exp7 = :y.assign E.sin(:x)
puts exp7
```

```
## y <- sin(x)
```

Expressions can also be written using '.' notation:

```
exp8 = :y.assign :x.sin
puts exp8
```

```
## y <- sin(x)
```

When a function has multiple arguments, the first one can be used before the '.':

```
exp9 = :x.c(:y)
puts exp9
```

```
## c(x, y)
```

10.3 Evaluating an Expression

Expressions can be evaluated by calling function 'eval' with a binding. A binding can be provided with a list:

```
exp = (:a + :b) * 2.0 + :c ** 2 / :z
puts exp.eval(R.list(a: 10, b: 20, c: 30, z: 40))
```

```
## [1] 82.5
```

... with a data frame:

```
df = R.data__frame(
  a: R.c(1, 2, 3),
  b: R.c(10, 20, 30),
  c: R.c(100, 200, 300),
```

```
  z: R.c(1000, 2000, 3000))

puts exp.eval(df)
```

```
## [1] 32 64 96
```

11 Manipulating Data

One of the major benefits of Galaaz is to bring strong data manipulation to Ruby. The following examples were extracted from Hardley's "R for Data Science" (https://r4ds.had.co.nz/). This is a highly recommended book for those not already familiar with the 'tidyverse' style of programming in R. In the sections to follow, we will limit ourselves to convert the R code to Galaaz.

For these examples, we will investigate the nycflights13 data set available on the package by the same name. We use function 'R.install_and_loads' that checks if the library is available locally, and if not, installs it. This data frame contains all 336,776 flights that departed from New York City in 2013. The data comes from the US Bureau of Transportation Statistics.

Dplyr uses 'tibbles' in place of data frames; unfortunately, tibbles do not print yet properly in Galaaz due to a bug in fastR. In order to print a tibble we need to convert it to a data frame using the 'as___data___frame' method.

```
R.install_and_loads('nycflights13')
R.library('dplyr')
```

```
flights = ~:flights
puts flights.head
```

```
## # A tibble: 6 x 19
##    year month   day dep_time sched_dep_time dep_delay arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>
## 1  2013     1     1      517            515         2      830
## 2  2013     1     1      533            529         4      850
## 3  2013     1     1      542            540         2      923
## 4  2013     1     1      544            545        -1     1004
## 5  2013     1     1      554            600        -6      812
## 6  2013     1     1      554            558        -4      740
## # ... with 12 more variables: sched_arr_time <int>, arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
## #   time_hour <dttm>
```

11.1 Filtering rows with Filter

In this example we filter the flights data set by giving to the filter function two expressions: the first :month.eq 1

```
puts flights.filter((:month.eq 1), (:day.eq 1)).head
```

```
## # A tibble: 6 x 19
##    year month   day dep_time sched_dep_time dep_delay arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>
```

```
## 1  2013     1     1      517            515         2      830
## 2  2013     1     1      533            529         4      850
## 3  2013     1     1      542            540         2      923
## 4  2013     1     1      544            545        -1     1004
## 5  2013     1     1      554            600        -6      812
## 6  2013     1     1      554            558        -4      740
## # ... with 12 more variables: sched_arr_time <int>, arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
## #   time_hour <dttm>
```

11.2 Logical Operators

All flights that departed in November of December

```
puts flights.filter((:month.eq 11) | (:month.eq 12)).head
```

```
## # A tibble: 6 x 19
##    year month   day dep_time sched_dep_time dep_delay arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>
## 1  2013    11     1        5           2359         6      352
## 2  2013    11     1       35           2250       105      123
## 3  2013    11     1      455            500        -5      641
## 4  2013    11     1      539            545        -6      856
## 5  2013    11     1      542            545        -3      831
## 6  2013    11     1      549            600       -11      912
## # ... with 12 more variables: sched_arr_time <int>, arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
## #   time_hour <dttm>
```

The same as above, but using the 'in' operator. In R, it is possible to define many operators by doing %%. The %in% operator checks if a value is in a vector. In order to use those operators from Galaaz the '._'method is used, where the first argument is the operator's symbol, in this case':in' and the second argument is the vector:

```
puts flights.filter(:month._ :in, R.c(11, 12)).head
```

```
## # A tibble: 6 x 19
##    year month   day dep_time sched_dep_time dep_delay arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>
## 1  2013    11     1        5           2359         6      352
## 2  2013    11     1       35           2250       105      123
## 3  2013    11     1      455            500        -5      641
## 4  2013    11     1      539            545        -6      856
## 5  2013    11     1      542            545        -3      831
## 6  2013    11     1      549            600       -11      912
## # ... with 12 more variables: sched_arr_time <int>, arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
## #   time_hour <dttm>
```

11.3 Filtering with NA (Not Available)

Let's first create a 'tibble' with a Not Available value (R::NA). Tibbles are a modern version of a data frame and operate very similarly to one. It differs in how it outputs the values and the result of some subsetting operations that are more consistent than what is obtained from data frame.

```
df = R.tibble(x: R.c(1, R::NA, 3))
puts df
```

```
## # A tibble: 3 x 1
##       x
##   <int>
## 1     1
## 2
## 3     3
```

Now filtering by :x > 1 shows all lines that satisfy this condition, where the row with R:NA does not.

```
puts df.filter(:x > 1)
```

```
## # A tibble: 1 x 1
##       x
##   <int>
## 1     3
```

To match an NA use method 'is___na'

```
puts df.filter((:x.is__na) | (:x > 1))
```

```
## # A tibble: 2 x 1
##       x
##   <int>
## 1
## 2     3
```

11.4 Arrange Rows with arrange

Arrange reorders the rows of a data frame by the given arguments.

```
puts flights.arrange(:year, :month, :day).head
```

```
## # A tibble: 6 x 19
##    year month   day dep_time sched_dep_time dep_delay arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>
## 1  2013     1     1      517            515         2      830
## 2  2013     1     1      533            529         4      850
## 3  2013     1     1      542            540         2      923
## 4  2013     1     1      544            545        -1     1004
## 5  2013     1     1      554            600        -6      812
## 6  2013     1     1      554            558        -4      740
## # ... with 12 more variables: sched_arr_time <int>, arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
```

```
## #   time_hour <dttm>
```

To arrange in descending order, use function 'desc'

```
puts flights.arrange(:dep_delay.desc).head
```

```
## # A tibble: 6 x 19
##    year month   day dep_time sched_dep_time dep_delay arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>
## 1  2013     1     9      641            900      1301     1242
## 2  2013     6    15     1432           1935      1137     1607
## 3  2013     1    10     1121           1635      1126     1239
## 4  2013     9    20     1139           1845      1014     1457
## 5  2013     7    22      845           1600      1005     1044
## 6  2013     4    10     1100           1900       960     1342
## # ... with 12 more variables: sched_arr_time <int>, arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
## #   time_hour <dttm>
```

11.5 Selecting columns

To select specific columns from a dataset we use function 'select':

```
puts flights.select(:year, :month, :day).head
```

```
## # A tibble: 6 x 3
##    year month   day
##   <int> <int> <int>
## 1  2013     1     1
## 2  2013     1     1
## 3  2013     1     1
## 4  2013     1     1
## 5  2013     1     1
## 6  2013     1     1
```

It is also possible to select column in a given range

```
puts flights.select(:year.up_to :day).head
```

```
## # A tibble: 6 x 3
##    year month   day
##   <int> <int> <int>
## 1  2013     1     1
## 2  2013     1     1
## 3  2013     1     1
## 4  2013     1     1
## 5  2013     1     1
## 6  2013     1     1
```

Select all columns that start with a given name sequence

```
puts flights.select(E.starts_with('arr')).head
```

```
## # A tibble: 6 x 2
```

```
##   arr_time arr_delay
##      <int>     <dbl>
## 1      830        11
## 2      850        20
## 3      923        33
## 4     1004       -18
## 5      812       -25
## 6      740        12
```

Other functions that can be used:

- ends_with("xyz"): matches names that end with "xyz".
- contains("ijk"): matches names that contain "ijk".
- matches("(.)\1"): selects variables that match a regular expression. This one matches any variables that contain repeated characters.
- num_range("x", (1..3)): matches x1, x2 and x3

A helper function that comes in handy when we just want to rearrange column order is 'Everything':

```
puts flights.select(:year, :month, :day, E.everything).head
```

```
## # A tibble: 6 x 19
##    year month   day dep_time sched_dep_time dep_delay arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>
## 1  2013     1     1      517            515         2      830
## 2  2013     1     1      533            529         4      850
## 3  2013     1     1      542            540         2      923
## 4  2013     1     1      544            545        -1     1004
## 5  2013     1     1      554            600        -6      812
## 6  2013     1     1      554            558        -4      740
## # ... with 12 more variables: sched_arr_time <int>, arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
## #   time_hour <dttm>
```

11.6 Add variables to a dataframe with 'mutate'

```
flights_sm = flights.
               select((:year.up_to :day),
                      E.ends_with('delay'),
                      :distance,
                      :air_time)

puts flights_sm.head
```

```
## # A tibble: 6 x 7
##    year month   day dep_delay arr_delay distance air_time
##   <int> <int> <int>     <dbl>     <dbl>    <dbl>    <dbl>
## 1  2013     1     1         2        11     1400      227
## 2  2013     1     1         4        20     1416      227
```

```
## 3  2013     1     1         2        33     1089      160
## 4  2013     1     1        -1       -18     1576      183
## 5  2013     1     1        -6       -25      762      116
## 6  2013     1     1        -4        12      719      150
```

```
flights_sm = flights_sm.
               mutate(gain: :dep_delay - :arr_delay,
                      speed: :distance / :air_time * 60)
puts flights_sm.head
```

```
## # A tibble: 6 x 9
##    year month   day dep_delay arr_delay distance air_time  gain speed
##   <int> <int> <int>     <dbl>     <dbl>    <dbl>    <dbl> <dbl> <dbl>
## 1  2013     1     1         2        11     1400      227    -9  370.
## 2  2013     1     1         4        20     1416      227   -16  374.
## 3  2013     1     1         2        33     1089      160   -31  408.
## 4  2013     1     1        -1       -18     1576      183    17  517.
## 5  2013     1     1        -6       -25      762      116    19  394.
## 6  2013     1     1        -4        12      719      150   -16  288.
```

11.7 Summarising data

Function 'summarise' calculates summaries for the data frame. When no 'group_by' is used a single value is obtained from the data frame:

```
puts flights.summarise(delay: E.mean(:dep_delay, na__rm: true))
```

```
## # A tibble: 1 x 1
##   delay
##   <dbl>
## 1  12.6
```

When a data frame is grouped with 'group_by' summaries apply to the given group:

```
by_day = flights.group_by(:year, :month, :day)
puts by_day.summarise(delay: :dep_delay.mean(na__rm: true)).head
```

```
## # A tibble: 6 x 4
## # Groups:   year, month [1]
##    year month   day delay
## * <int> <int> <int> <dbl>
## 1  2013     1     1 11.5
## 2  2013     1     2 13.9
## 3  2013     1     3 11.0
## 4  2013     1     4  8.95
## 5  2013     1     5  5.73
## 6  2013     1     6  7.15
```

Next we put many operations together by pipping them one after the other:

```
delays = flights.
           group_by(:dest).
           summarise(
             count: E.n,
```

```
            dist: :distance.mean(na__rm: true),
            delay: :arr_delay.mean(na__rm: true)).
          filter(:count > 20, :dest != "NHL")

puts delays.head
```

```
## # A tibble: 6 x 4
##   dest  count  dist delay
##   <chr> <int> <dbl> <dbl>
## 1 ABQ     254 1826   4.38
## 2 ACK     265  199   4.85
## 3 ALB     439  143  14.4
## 4 ATL   17215  757. 11.3
## 5 AUS    2439 1514.  6.02
## 6 AVL     275  584.  8.00
```

12 Using Data Table

```
R.library('data.table')
R.install_and_loads('curl')

input = "https://raw.githubusercontent.com/Rdatatable/data.table/master/vignettes/flights14
flights = R.fread(input)
puts flights
puts flights.dim
```

```
##         year month day dep_delay arr_delay carrier origin dest air_time
##      1: 2014     1   1        14        13      AA    JFK  LAX      359
##      2: 2014     1   1        -3        13      AA    JFK  LAX      363
##      3: 2014     1   1         2         9      AA    JFK  LAX      351
##      4: 2014     1   1        -8       -26      AA    LGA  PBI      157
##      5: 2014     1   1         2         1      AA    JFK  LAX      350
##     ---
## 253312: 2014    10  31         1       -30      UA    LGA  IAH      201
## 253313: 2014    10  31        -5       -14      UA    EWR  IAH      189
## 253314: 2014    10  31        -8        16      MQ    LGA  RDU       83
## 253315: 2014    10  31        -4        15      MQ    LGA  DTW       75
## 253316: 2014    10  31        -5         1      MQ    LGA  SDF      110
##         distance hour
##      1:     2475    9
##      2:     2475   11
##      3:     2475   19
##      4:     1035    7
##      5:     2475   13
##     ---
## 253312:     1416   14
## 253313:     1400    8
## 253314:      431   11
## 253315:      502   11
```

```
## 253316:      659    8
## [1] 253316     11
```

```
data_table = R.data__table(
  ID: R.c("b","b","b","a","a","c"),
  a: (1..6),
  b: (7..12),
  c: (13..18)
)

puts data_table
puts data_table.ID
```

```
##    ID a  b  c
## 1:  b 1  7 13
## 2:  b 2  8 14
## 3:  b 3  9 15
## 4:  a 4 10 16
## 5:  a 5 11 17
## 6:  c 6 12 18
## [1] "b" "b" "b" "a" "a" "c"
```

```
# subset rows in i
ans = flights[(:origin.eq "JFK") & (:month.eq 6)]
puts ans.head

# Get the first two rows from flights.

ans = flights[(1..2)]
puts ans

# Sort flights first by column origin in ascending order, and then by dest in descending or

# ans = flights[E.order(:origin, -(:dest))]
# puts ans.head
```

```
##    year month day dep_delay arr_delay carrier origin dest air_time
## 1: 2014     6   1        -9        -5      AA    JFK  LAX      324
## 2: 2014     6   1       -10       -13      AA    JFK  LAX      329
## 3: 2014     6   1        18        -1      AA    JFK  LAX      326
## 4: 2014     6   1        -6       -16      AA    JFK  LAX      320
## 5: 2014     6   1        -4       -45      AA    JFK  LAX      326
## 6: 2014     6   1        -6       -23      AA    JFK  LAX      329
##    distance hour
## 1:     2475    8
## 2:     2475   12
## 3:     2475    7
## 4:     2475   10
## 5:     2475   18
## 6:     2475   14
##    year month day dep_delay arr_delay carrier origin dest air_time
## 1: 2014     1   1        14        13      AA    JFK  LAX      359
```

```
## 2: 2014     1   1        -3        13      AA    JFK  LAX      363
##    distance hour
## 1:     2475    9
## 2:     2475   11
```

```ruby
# Select column(s) in j
# select arr_delay column, but return it as a vector.

ans = flights[:all, :arr_delay]
puts ans.head

# Select arr_delay column, but return as a data.table instead.

ans = flights[:all, :arr_delay.list]
puts ans.head

ans = flights[:all, E.list(:arr_delay, :dep_delay)]
```

```
## [1]  13  13   9 -26   1   0
##    arr_delay
## 1:        13
## 2:        13
## 3:         9
## 4:       -26
## 5:         1
## 6:         0
```

13 Graphics in Galaaz

Creating graphics in Galaaz is quite easy, as it can use all the power of ggplot2. There are many resources in the web that teaches ggplot, so here we give a quick example of ggplot integration with Ruby. We continue to use the :mtcars dataset and we will plot a diverging bar plot, showing cars that have 'above' or 'below' gas consuption. Let's first prepare the data frame with the necessary data:

```ruby
# copy the R variable :mtcars to the Ruby mtcars variable
mtcars = ~:mtcars

# create a new column 'car_name' to store the car names so that it can be
# used for plotting. The 'rownames' of the data frame cannot be used as
# data for plotting
mtcars.car_name = R.rownames(:mtcars)

# compute normalized mpg and add it to a new column called mpg_z
# Note that the mean value for mpg can be obtained by calling the 'mean'
# function on the vector 'mtcars.mpg'.  The same with the standard
# deviation 'sd'.  The vector is then rounded to two digits with 'round 2'
mtcars.mpg_z = ((mtcars.mpg - mtcars.mpg.mean)/mtcars.mpg.sd).round 2

# create a new column 'mpg_type'. Function 'ifelse' is a vectorized function
# that looks at every element of the mpg_z vector and if the value is below
```

```
# 0, returns 'below', otherwise returns 'above'
mtcars.mpg_type = (mtcars.mpg_z < 0).ifelse("below", "above")

# order the mtcar data set by the mpg_z vector from smaler to larger values
mtcars = mtcars[mtcars.mpg_z.order, :all]

# convert the car_name column to a factor to retain sorted order in plot
mtcars.car_name = mtcars.car_name.factor levels: mtcars.car_name

# let's look at the final data frame
puts mtcars.head
```

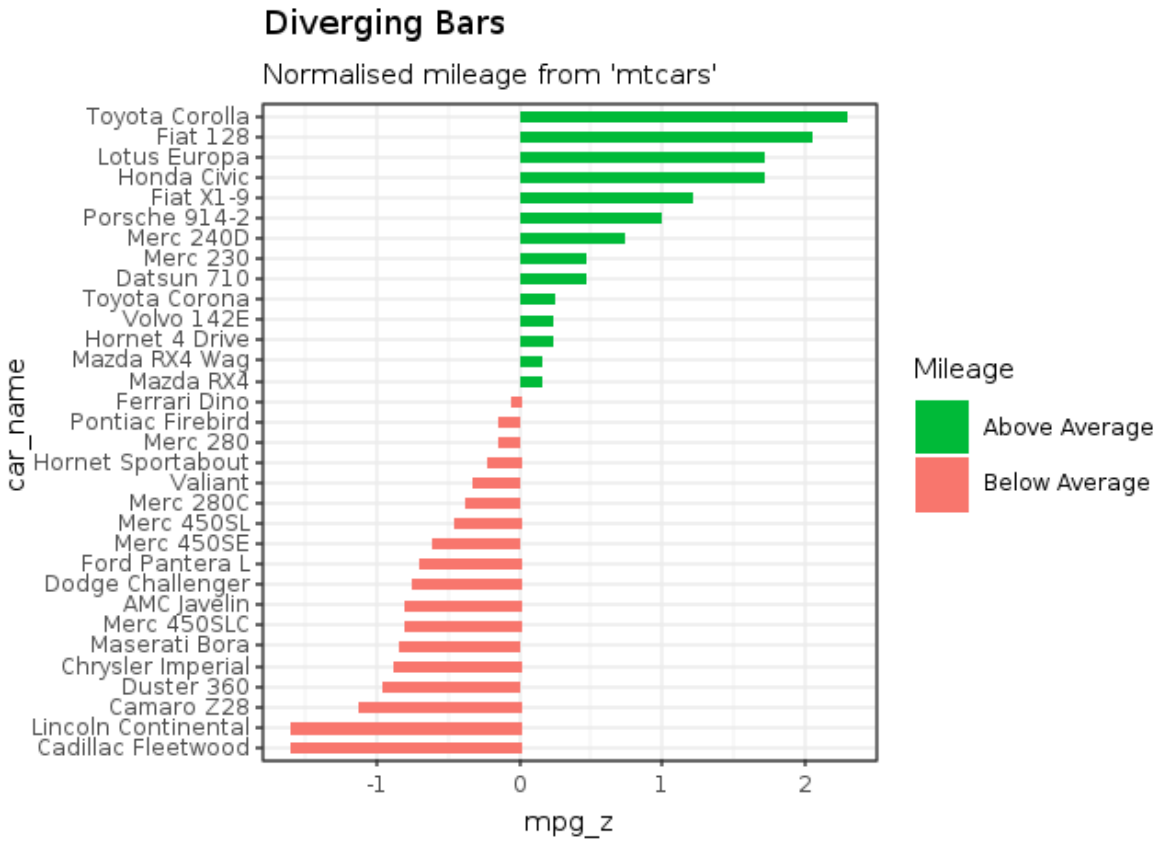
```
##                      mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Cadillac Fleetwood  10.4   8  472 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4   8  460 215 3.00 5.424 17.82  0  0    3    4
## Camaro Z28          13.3   8  350 245 3.73 3.840 15.41  0  0    3    4
## Duster 360          14.3   8  360 245 3.21 3.570 15.84  0  0    3    4
## Chrysler Imperial   14.7   8  440 230 3.23 5.345 17.42  0  0    3    4
## Maserati Bora       15.0   8  301 335 3.54 3.570 14.60  0  1    5    8
##                                car_name mpg_z mpg_type
## Cadillac Fleetwood   Cadillac Fleetwood -1.61    below
## Lincoln Continental Lincoln Continental -1.61    below
## Camaro Z28                   Camaro Z28 -1.13    below
## Duster 360                   Duster 360 -0.96    below
## Chrysler Imperial     Chrysler Imperial -0.89    below
## Maserati Bora             Maserati Bora -0.84    below
```

Now, lets plot the diverging bar plot. When using gKnit, there is no need to call 'R.awt' to create a plotting device, since gKnit does take care of it. Galaaz provides integration with ggplot. The interested reader should check online for more information on ggplot, since it is outside the scope of this manual describing how ggplot works. We give here but a brief description on how this plot is generated.

ggplot implements the 'grammar of graphics'. In this approach, plots are build by adding layers to the plot. On the first layer we describe what we want on the 'x' and 'y' axis of the plot. In this case, we have 'car_name' on the 'x' axis and 'mpg_z' on the 'y' axis. Then the type of graph is specified by adding 'geom_bar' (for a bar graph). We specify that our bars should be filled using 'mpg_type', which is either 'above' or 'bellow' giving then two colours for filling. On the next layer we specify the labels for the graph, then we add the title and subtitle. Finally, in a bar chart usually bars go on the vertical direction, but in this graph we want the bars to be horizontally layed so we add 'coord_flip'.

```
require 'ggplot'

puts mtcars.ggplot(E.aes(x: :car_name, y: :mpg_z, label: :mpg_z)) +
     R.geom_bar(E.aes(fill: :mpg_type), stat: 'identity', width: 0.5) +
     R.scale_fill_manual(name: 'Mileage',
                         labels: R.c('Above Average', 'Below Average'),
                         values: R.c('above': '#00ba38', 'below': '#f8766d')) +
     R.labs(subtitle: "Normalised mileage from 'mtcars'",
            title: "Diverging Bars") +
     R.coord_flip
```



14 Coding with Tidyverse

In R, and when coding with 'tidyverse', arguments to a function are usually not *referencially transparent*. That is, you can't replace a value with a seemingly equivalent object that you've defined elsewhere. To see the problem, let's first define a data frame:

```
df = R.data__frame(x: (1..3), y: (3..1))
puts df
```

```
##   x y
## 1 1 3
## 2 2 2
## 3 3 1
```

and now, let's look at this code:

```
my_var <- x
filter(df, my_var == 1)
```

It generates the following error: "object 'x' not found.

However, in Galaaz, arguments are referencially transparent as can be seen by the code bellow. Note initally that 'my_var = :x' will not give the error "object 'x' not found" since ':x' is treated as an expression and assigned to my_var. Then when doing (my_var.eq 1), my_var is a variable that resolves to ':x' and it becomes equivalent to (:x.eq 1) which is what we want.

```
my_var = :x
puts df.filter(my_var.eq 1)
```

```
##   x y
## 1 1 3
```

As stated by Hardley

> dplyr code is ambiguous. Depending on what variables are defined where, filter(df, x == y) could be equivalent to any of:

```
df[df$x == df$y, ]
df[df$x == y, ]
df[x == df$y, ]
df[x == y, ]
```

In galaaz this ambiguity does not exist, filter(df, x.eq y) is not a valid expression as expressions are build with symbols. In doing filter(df, :x.eq y) we are looking for elements of the 'x' column that are equal to a previously defined y variable. Finally in filter(df, :x.eq :y) we are looking for elements in which the 'x' column value is equal to the 'y' column value. This can be seen in the following two chunks of code:

```
y = 1
x = 2

# looking for values where the 'x' column is equal to the 'y' column
puts df.filter(:x.eq :y)
```

```
##   x y
## 1 2 2
```

```
# looking for values where the 'x' column is equal to the 'y' variable
# in this case, the number 1
puts df.filter(:x.eq y)
```

```
##   x y
## 1 1 3
```

14.1 Writing a function that applies to different data sets

Let's suppose that we want to write a function that receives as the first argument a data frame and as second argument an expression that adds a column to the data frame that is equal to the sum of elements in column 'a' plus 'x'.

Here is the intended behaviour using the 'mutate' function of 'dplyr':

```
mutate(df1, y = a + x)
mutate(df2, y = a + x)
mutate(df3, y = a + x)
mutate(df4, y = a + x)
```

The naive approach to writing an R function to solve this problem is:

```
mutate_y <- function(df) {
  mutate(df, y = a + x)
}
```

Unfortunately, in R, this function can fail silently if one of the variables isn't present in the data frame, but is present in the global environment. We will not go through here how to solve this problem in R.

In Galaaz the method mutate_y bellow will work fine and will never fail silently.

```
def mutate_y(df)
  df.mutate(:y.assign :a + :x)
end
```

Here we create a data frame that has only one column named 'x':

```
df1 = R.data__frame(x: (1..3))
puts df1
```

```
##   x
## 1 1
## 2 2
## 3 3
```

Note that method mutate_y will fail independetly from the fact that variable 'a' is defined and in the scope of the method. Variable 'a' has no relationship with the symbol ':a' used in the definition of 'mutate_y' above:

```
a = 10
mutate_y(df1)
```

```
## Message:
##  Error in mutate_impl(.data, dots) :
##   Evaluation error: object 'a' not found.
## In addition: Warning message:
## In mutate_impl(.data, dots) :
##   mismatched protect/unprotect (unprotect with empty protect stack) (RError)
## Translated to internal error
```

14.2 Different expressions

Let's move to the next problem as presented by Hardley where trying to write a function in R that will receive two argumens, the first a variable and the second an expression is not trivial. Bellow we create a data frame and we want to write a function that groups data by a variable and summarises it by an expression:

```
set.seed(123)

df <- data.frame(
  g1 = c(1, 1, 2, 2, 2),
  g2 = c(1, 2, 1, 2, 1),
  a = sample(5),
  b = sample(5)
)

as.data.frame(df)
```

```
##   g1 g2 a b
## 1  1  1 3 3
## 2  1  2 2 1
## 3  2  1 5 2
## 4  2  2 4 5
## 5  2  1 1 4
```

```
d2 <- df %>%
  group_by(g1) %>%
  summarise(a = mean(a))

as.data.frame(d2)
```

```
##   g1        a
## 1  1 2.500000
## 2  2 3.333333
```

```
d2 <- df %>%
  group_by(g2) %>%
  summarise(a = mean(a))

as.data.frame(d2)
```

```
##   g2 a
## 1  1 3
## 2  2 3
```

As shown by Hardley, one might expect this function to do the trick:

```
my_summarise <- function(df, group_var) {
  df %>%
    group_by(group_var) %>%
    summarise(a = mean(a))
}

# my_summarise(df, g1)
#> Error: Column `group_var` is unknown
```

In order to solve this problem, coding with dplyr requires the introduction of many new concepts and functions such as 'quo', 'quos', 'enquo', 'enquos', '!!' (bang bang), '!!!' (triple bang). Again, we'll leave to Hardley the explanation on how to use all those functions.

Now, let's try to implement the same function in galaaz. The next code block first prints the 'df' data frame defined previously in R (to access an R variable from Galaaz, we use the tilda operator '~' applied to the R variable name as symbol, i.e., ':df'.

```
puts ~:df
```

```
##   g1 g2 a b
## 1  1  1 3 3
## 2  1  2 2 1
## 3  2  1 5 2
## 4  2  2 4 5
## 5  2  1 1 4
```

We then create the 'my_summarize' method and call it passing the R data frame and the group by variable ':g1':

```
def my_summarize(df, group_var)
  df.group_by(group_var).
    summarize(a: :a.mean)
end
```

```
puts my_summarize(:df, :g1)
```

```
## # A tibble: 2 x 2
##      g1     a
##   <dbl> <dbl>
## 1     1  2.5
## 2     2  3.33
```

It works!!! Well, let's make sure this was not just some coincidence

```
puts my_summarize(:df, :g2)
```

```
## # A tibble: 2 x 2
##      g2     a
##   <dbl> <dbl>
## 1     1     3
## 2     2     3
```

Great, everything is fine! No magic, no new functions, no complexities, just normal, standard Ruby code. If you've ever done NSE in R, this certainly feels much safer and easy to implement.

14.3 Different input variables

In the previous section we've managed to get rid of all NSE formulation for a simple example, but does this remain true for more complex examples, or will the Galaaz way prove inpractical for more complex code?

In the next example Hardley proposes us to write a function that given an expression such as 'a' or 'a * b', calculates three summaries. What we want a function that does the same as these R statements:

```
summarise(df, mean = mean(a), sum = sum(a), n = n())
#> # A tibble: 1 x 3
#>    mean   sum     n
#>   <dbl> <int> <int>
#> 1     3    15     5

summarise(df, mean = mean(a * b), sum = sum(a * b), n = n())
#> # A tibble: 1 x 3
#>    mean   sum     n
#>   <dbl> <int> <int>
#> 1   9    45     5
```

Let's try it in galaaz:

```
def my_summarise2(df, expr)
  df.summarize(
    mean: E.mean(expr),
    sum: E.sum(expr),
    n: E.n
  )
end

puts my_summarise2((~:df), :a)
```

```
puts "\n"
puts my_summarise2((~:df), :a * :b)
```

```
##   mean sum n
## 1    3  15 5
##
##   mean sum n
## 1    9  45 5
```

Once again, there is no need to use any special theory or functions. The only point to be careful about is the use of 'E' to build expressions from functions 'mean', 'sum' and 'n'.

14.4 Different input and output variable

Now the next challenge presented by Hardley is to vary the name of the output variables based on the received expression. So, if the input expression is 'a', we want our data frame columns to be named 'mean_a' and 'sum_a'. Now, if the input expression is 'b', columns should be named 'mean_b' and 'sum_b'.

```
mutate(df, mean_a = mean(a), sum_a = sum(a))
#> # A tibble: 5 x 6
#>      g1    g2     a     b mean_a sum_a
#>   <dbl> <dbl> <int> <int>  <dbl> <int>
#> 1     1     1     1     3      3    15
#> 2     1     2     4     2      3    15
#> 3     2     1     2     1      3    15
#> 4     2     2     5     4      3    15
#> # ... with 1 more row

mutate(df, mean_b = mean(b), sum_b = sum(b))
#> # A tibble: 5 x 6
#>      g1    g2     a     b mean_b sum_b
#>   <dbl> <dbl> <int> <int>  <dbl> <int>
#> 1     1     1     1     3      3    15
#> 2     1     2     4     2      3    15
#> 3     2     1     2     1      3    15
#> 4     2     2     5     4      3    15
#> # ... with 1 more row
```

In order to solve this problem in R, Hardley needs to introduce some more new functions and notations: 'quo_name' and the ':=' operator from package 'rlang'

Here is our Ruby code:

```
def my_mutate(df, expr)
  mean_name = "mean_#{expr.to_s}"
  sum_name = "sum_#{expr.to_s}"

  df.mutate(mean_name => E.mean(expr),
            sum_name => E.sum(expr))
end

puts my_mutate((~:df), :a)
```

```
puts "\n"
puts my_mutate((~:df), :b)
```

```
##   g1 g2 a b mean_a sum_a
## 1  1  1 3 3      3    15
## 2  1  2 2 1      3    15
## 3  2  1 5 2      3    15
## 4  2  2 4 5      3    15
## 5  2  1 1 4      3    15
##
##   g1 g2 a b mean_b sum_b
## 1  1  1 3 3      3    15
## 2  1  2 2 1      3    15
## 3  2  1 5 2      3    15
## 4  2  2 4 5      3    15
## 5  2  1 1 4      3    15
```

It really seems that "Non Standard Evaluation" is actually quite standard in Galaaz! But, you might have noticed a small change in the way the arguments to the mutate method were called. In a previous example we used df.summarise(mean: E.mean(:a), ...) where the column name was followed by a ':' colom. In this example, we have df.mutate(mean_name => E.mean(expr), ...) and variable mean_name is not followed by ':' but by '=>'. This is standard Ruby notation.

[explain....]

14.5 Capturing multiple variables

Moving on with new complexities, Hardley proposes us to solve the problem in which the summarise function will receive any number of grouping variables.

This again is quite standard Ruby. In order to receive an undefined number of paramenters the paramenter is preceded by '*':

```
def my_summarise3(df, *group_vars)
  df.group_by(*group_vars).
    summarise(a: E.mean(:a))
end

puts my_summarise3((~:df), :g1, :g2)
```

```
## # A tibble: 4 x 3
## # Groups:   g1 [?]
##      g1    g2     a
##   <dbl> <dbl> <dbl>
## 1     1     1     3
## 2     1     2     2
## 3     2     1     3
## 4     2     2     4
```

14.6 Why does R require NSE and Galaaz does not?

NSE introduces a number of new concepts, such as 'quoting', 'quasiquotation', 'unquoting' and 'unquote-splicing', while in Galaaz none of those concepts are needed. What gives?

R is an extremely flexible language and it has lazy evaluation of parameters. When in R a function is called as 'summarise(df, a = b)', the summarise function receives the litteral 'a = b' parameter and can work with this as if it were a string. In R, it is not clear what a and b are, they can be expressions or they can be variables, it is up to the function to decide what 'a = b' means.

In Ruby, there is no lazy evaluation of parameters and 'a' is always a variable and so is 'b'. Variables assume their value as soon as they are used, so 'x = a' is immediately evaluate and variable 'x' will receive the value of variable 'a' as soon as the Ruby statement is executed. Ruby also provides the notion of a symbol; ':a' is a symbol and does not evaluate to anything. Galaaz uses Ruby symbols to build expressions that are not bound to anything: ':a.eq :b' is clearly an expression and has no relationship whatsoever with the statment 'a = b'. By using symbols, variables and expressions all the possible ambiguities that are found in R are eliminated in Galaaz.

The main problem that remains, is that in R, functions are not clearly documented as what type of input they are expecting, they might be expecting regular variables or they might be expecting expressions and the R function will know how to deal with an input of the form 'a = b', now for the Ruby developer it might not be immediately clear if it should call the function passing the value 'true' if variable 'a' is equal to variable 'b' or if it should call the function passing the expression ':a.eq :b'.

14.7 Advanced dplyr features

In the blog: Programming with dplyr by using dplyr (https://www.r-bloggers.com/programming-with-dplyr-by-using-dplyr/) Iñaki Úcar shows surprise that some R users are trying to code in dplyr avoiding the use of NSE. For instance he says:

> Take the example of seplyr. It stands for standard evaluation dplyr, and enables us to program over dplyr without having "to bring in (or study) any deep-theory or heavy-weight tools such as rlang/tidyeval".

For me, there isn't really any surprise that users are trying to avoid dplyr deep-theory. R users frequently are not programmers and learning to code is already hard business, on top of that, having to learn how to 'quote' or 'enquo' or 'quos' or 'enquos' is not necessarily a 'piece of cake'. So much so, that 'tidyeval' has some more advanced functions that instead of using quoted expressions, uses strings as arguments.

In the following examples, we show the use of functions 'group_by_at', 'summarise_at' and 'rename_at' that receive strings as argument. The data frame used in 'starwars' that describes features of characters in the Starwars movies:

```
puts (~:starwars).head
```

```
## # A tibble: 6 x 13
##   name  height  mass hair_color skin_color eye_color birth_year gender
##   <chr>  <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr>
## 1 Luke~    172    77 blond      fair       blue            19   male
## 2 C-3PO    167    75 <NA>       gold       yellow         112   <NA>
```

```
## 3 R2-D2     96    32 <NA>       white, bl~ red             33   <NA>
## 4 Dart~    202   136 none       white      yellow          41.9 male
## 5 Leia~    150    49 brown      light      brown           19   female
## 6 Owen~    178   120 brown, gr~ light      blue            52   male
## # ... with 5 more variables: homeworld <chr>, species <chr>, films <list>,
## #   vehicles <list>, starships <list>
```

The grouped_mean function bellow will receive a grouping variable and calculate summaries for the value_variables given:

```
grouped_mean <- function(data, grouping_variables, value_variables) {
  data %>%
    group_by_at(grouping_variables) %>%
    mutate(count = n()) %>%
    summarise_at(c(value_variables, "count"), mean, na.rm = TRUE) %>%
    rename_at(value_variables, funs(paste0("mean_", .)))
    }

gm = starwars %>%
   grouped_mean("eye_color", c("mass", "birth_year"))

as.data.frame(gm)
```

```
##        eye_color mean_mass mean_birth_year count
## 1          black  76.28571        33.00000    10
## 2           blue  86.51667        67.06923    19
## 3      blue-gray  77.00000        57.00000     1
## 4          brown  66.09231       108.96429    21
## 5           dark       NaN             NaN     1
## 6           gold       NaN             NaN     1
## 7  green, yellow 159.00000             NaN     1
## 8          hazel  66.00000        34.50000     3
## 9         orange 282.33333       231.00000     8
## 10          pink       NaN             NaN     1
## 11           red  81.40000        33.66667     5
## 12     red, blue       NaN             NaN     1
## 13       unknown  31.50000             NaN     3
## 14         white  48.00000             NaN     1
## 15        yellow  81.11111        76.38000    11
```

The same code with Galaaz, becomes:

```
def grouped_mean(data, grouping_variables, value_variables)
  data.
    group_by_at(grouping_variables).
    mutate(count: E.n).
    summarise_at(E.c(value_variables, "count"), ~:mean, na__rm: true).
    rename_at(value_variables, E.funs(E.paste0("mean_", value_variables)))
end

puts grouped_mean((~:starwars), "eye_color", E.c("mass", "birth_year"))
```

```
## # A tibble: 15 x 4
##    eye_color     mean_mass mean_birth_year count
```

```
##    <chr>             <dbl>          <dbl> <dbl>
##  1 black              76.3           33      10
##  2 blue               86.5           67.1    19
##  3 blue-gray          77             57       1
##  4 brown              66.1          109.     21
##  5 dark              NaN            NaN       1
##  6 gold              NaN            NaN       1
##  7 green, yellow     159            NaN       1
##  8 hazel              66             34.5     3
##  9 orange            282.           231       8
## 10 pink              NaN            NaN       1
## 11 red                81.4           33.7     5
## 12 red, blue         NaN            NaN       1
## 13 unknown            31.5          NaN       3
## 14 white              48            NaN       1
## 15 yellow             81.1           76.4    11
```

[TO BE CONTINUED...]

15 Contributing

- Fork it
- Create your feature branch (git checkout -b my-new-feature)
- Write Tests!
- Commit your changes (git commit -am 'Add some feature')
- Push to the branch (git push origin my-new-feature)
- Create new Pull Request

References

Knuth, Donald E. 1984. "Literate Programming." *Comput. J.* 27 (2). Oxford, UK: Oxford University Press: 97–111. doi:10.1093/comjnl/27.2.97.

Wilkinson, Leland. 2005. *The Grammar of Graphics (Statistics and Computing)*. Berlin, Heidelberg: Springer-Verlag.