SYBERIA: A DEVELOPMENT FRAMEWORK FOR R

Robert Krzyzanowski
Director of Data Engineering at Avant
syberia.io

github.com/robertzk/rfinance17

@rsyberia

rob@syberia.io
• R workflows are typically loosely organized collections of scripts
• R workflows are typically loosely organized collections of scripts

• Packages work well for recording abstract solutions to problems, but not for large projects maintained by multiple users tied to solving problems in a specific domain
• R workflows are typically loosely organized collections of scripts

• Packages work well for recording abstract solutions to problems, but not for large projects maintained by multiple users tied to solving problems in a specific domain

• Test-driven development is difficult for modeling work that is intended for real-time systems
An Industry-wide Problem

NIPS 2014 Workshop proceedings
D. Sculley, Gary Holt, Daniel Golovin, et al

“Risk factors include boundary erosion, entanglement, hidden feedback loops, undeclared consumers, data dependencies, changes in the external world, and a variety of system-level anti-patterns.”
Solution: Think like a developer
Developers are good at simplifying work that needs to be done down to its core abstractions
**Syberia** is a framework for building complex projects in R.
**Syberia** is a framework for building complex projects in R.

The modular design unit is an **engine**.
**Syberia** is a framework for building complex projects in R.

The modular design unit is an **engine**.

Today we are releasing the **modeling engine** for building and deploying production-ready machine learning products in R.
How Does It Work?

1. **Import**
2. **Munge**
3. **Model**
4. **Validate**
5. **Export**
How Does It Work?

The modeling engine is the boss.

**Mungebits** package powers feature engineering.

**Tundra package** powers model containers.
How Does It Work?

The modeling engine is the boss.

**Director** package holds the project together.

**Stagerunner** package is the workflow and execution system.
source("my_R_script.R")

• Script-driven workflow makes it harder to re-use components

• Testing is not built in unless you move to a package
resource("lib/adapters/s3")

• **Basic idea**: Everything is a resource

• All resources must be tested
  (test/lib/adapters/s3)

• Each resource type can have its own “grammar”
**Task**: Let’s import some data using Syberia.
# lib/adapters/s3.R

```r
read <- function(name) {
  s3mpi::s3read(name)
}
```

```r
write <- function(object, name) {
  s3mpi::s3store(object, name)
}
```

# package: github.com/robertzk/s3mpi
# config/routes.R

```r
list(
    "config/global" = "globals",
    "lib/adapters" = "adapters",
    "lib/classifiers" = "classifiers",
    "lib/mungebits" = "mungebits",
    "models" = "models",
    "test/models" = "test/models",
    "data" = "data")
```
The Basic Structure

```r
a <- resource("lib/adapters/s3")

a$write(iris, "tmp/iris")

# From a new R session
a <- resource("lib/adapters/s3")

identical(
  a$read("tmp/iris"),
  iris
)
```
An Example

# test/lib/adapters/s3.R

test_that("it can write a data set to S3", {  
  env <- new.env()  
  package_stub("s3mpi", "s3store", function(...) { env[[..2]] <- ..1 }, {  
    adapter <- resource()  
    adapter$write(iris, "test_key", prefix = "")  
    expect_identical(env$test_key, iris,  
      info = "iris should have been stored in the test_key in env")  
  })  
})

test_that("it can read a data set from S3", {  
  env <- list2env(list(test_key = iris))  
  package_stub("s3mpi", "s3read", function(...) { env[[..1]] }, {  
    adapter <- resource()  
    expect_identical(adapter$read("test_key", prefix = ""), env$test_key,  
      info = "iris should have been read from the test_key in env")  
  })  
})
An Example

```r
> stest("lib/adapters/s3")
...
DONE
> 
```
list(
  "config/global"   = "globals",
  "lib/adapters"    = "adapters",
  "lib/classifiers" = "classifiers",
  "lib/mungebits"   = "mungebits",
  "models"          = "models",
  "test/models"     = "test/models",
  "data"            = "data")
# lib/controllers/adapters.R

```r
function(input) {
  adapter_class <- function(r, w) {
    list(read = r, write = w)
  }
}

# Construct the adapter object.
adapter_class(
  input$read,
  input$write
)
}
```
# lib/adapters/s3.R

**read** <- function(name) {
    s3mpi::s3read(name)
}

**write** <- function(object, name) {
    s3mpi::s3store(object, name)
}

# package: github.com/robertzk/s3mpi
Think of how you might write adapters for other storage backends:

- Reading from and writing to a CSV file
- Reading from and writing to a database
- Reading from and writing to a JSON service
- Et cetera
An Example

• Defining “adapters” abstracted away the storage backend from the underlying implementation

• Each adapter has the same interface
• Defining “adapters” abstracted away the storage backend from the underlying implementation

• Each adapter has the same interface

```r
adapter <- resource("lib/adapters/s3")
adapter$read("some_key")
adapter$write(object, "some_key")
```
• Defining “adapters” abstracted away the storage backend from the underlying implementation

• Each adapter has the same interface

```r
adapter <- resource("lib/adapters/s3")
adapter$read("some_key")
adapter$write(object, "some_key")

adapter <- resource("lib/adapters/file")
adapter$read("some_file.rds")
adapter$write(iris, "some_file.csv")
```
• Everything is a resource
• Everything is a resource

• Each resource produces a single R object
• **Everything is a resource**

• Each resource produces a single R object

• Encourages finding common interfaces and abstractions
• **Everything is a resource**

• Each resource produces a single R object

• Encourages finding common interfaces and abstractions

• Work becomes easily re usable instead of locked away in scripts
list(
# Titanic dataset is fairly popular. Here we're downloading it from a public github repo
import  = list(url = "https://raw.githubusercontent.com/haven-jeon/.../master/bicdata/data/titanic.csv"),

data  = list(
# The left-hand side defines the informal name of a mungebit that you will see when you run this model.
# The right-hand side is the mungebit invocation.
"Factor to character"  = list(column_transformation(as.character), is.factor)
,"has paren in name"  = list(multi_column_transformation(function(name) grepl("(", fixed = TRUE, name)), "name", "has_paren")
,"Name length variable"  = list(new_variable, function(name) nchar(name), "name_length")
# ~40 removed
,"Restore levels"  = list(restore_categorical_variables)
,"Rename dep_var"  = list(renamer, c("survived" = "dep_var"))
),

model  = list('gbm'
, .id_var   = 'X'
, distribution   = 'bernoulli'
, number_of_trees   = 3000
, shrinkage_factor   = 0.005
),

export  = list(R  = 'titanic')
)
list()
# Titanic dataset is fairly popular. Here we're downloading it from a public github repo
\[
\text{import} = \text{list}(\text{url} = "https://raw.githubusercontent.com/haven-jeon/.../master/bicdata/data/titanic.csv"),
\]
\[
\text{data} = \text{list}(\text{url} = "https://raw.githubusercontent.com/haven-jeon/.../master/bicdata/data/titanic.csv"),
\]
\[
\text{model} = \text{list}(\text{gbm},
\text{id_var} = \text{X},
\text{distribution} = \text{bernoulli},
\text{number_of_trees} = 3000,
\text{shrinkage_factor} = 0.005
\)
\]
\[
\text{export} = \text{list}(\text{R} = \text{titanic})
\]
# config/routes.R

```r
list(
    "config/global"   = "globals",
    "lib/adapters"    = "adapters",
    "lib/classifiers" = "classifiers",
    "lib/mungebits"   = "mungebits",
    "models"          = "models",
    "test/models"     = "test/models",
    "data"            = "data")
```
Sort data in ascending order by a given column.

```r
# lib/mungebits/orderer.R
train <- predict <- function(dataframe, col) {
  dataframe[order(dataframe[[col]]), ]
}

# From R console
m <- resource("lib/mungebits/orderer")
stopifnot(all.equal(
  m$run(iris, 1), iris[order(iris[[1]]), ]
))
```
Mean imputation for one column.

```r
# lib/mungebits/simple_impute.R
train <- function(dataframe, col) {
  input$col <- col
  input$mean <- mean(dataframe[[col]], na.rm=T)
  dataframe[is.na(dataframe[[col]]), col] <- input$mean
  dataframe
}
predict <- function(dataframe, ...) {
  col <- input$col
  dataframe[is.na(dataframe[[col]]), col] <- input$mean
  dataframe
}
```
# In R console
m <- resource("lib/mungebits/simple_impute")
iris2 <- iris; iris2[1, 1] <- NA
m$run(iris2, 1)
stopifnot(all.equal(
  m$run(iris2)[[1]],
  c(mean(iris[-1, 1]), iris2[-1, 1])
))
function(input) {
  if (isTRUE(input$column_transformation)) {
    mungebits2::mungebit$new(
      mungebits2::column_transformation(input$train),
      mungebits2::column_transformation(input$predict))
  } else {
    mungebits2::mungebit$new(
      input$train, input$predict)
  }
}
# lib/controllers/classifiers.R

```r
function(input) {
    force(input)
    function(munge_procedure = list(), default_args = list(), internal = list()) {
        input <- lapply(as.list(input), full_deflate)
        container <- tundra::tundra_container$new(resource, input$train, input$predict,
                                                  munge_procedure, full_deflate(default_args), full_deflate(internal))
        container$hooks <- lapply(container$hooks, function(fn) {
            environment(fn) <- globalenv(); fn
        })
        if (!is.null(input$read) || !is.null(input$write)) {
            attr(container, "s3mpi.serialize") <- list(read = input$read, write = input$write)
        }
        container
    }
}
```
list()
# Titanic dataset is fairly popular. Here we're downloading it from a public github repo

```r
import = list(url = "https://raw.githubusercontent.com/haven-jeon/.../master/bicdata/data/titanic.csv"),
```

data = list(

# The left-hand side defines the informal name of a mungebit that you will see when you run this model.
# The right-hand side is the mungebit invocation.
"Factor to character" = list(column_transformation(as.character), is.factor)
,"has paren in name" = list(multi_column_transformation(function(name) grepl("(", fixed = TRUE, name)), "name", "has paren")
,"Name length variable" = list(new_variable, function(name) nchar(name), "name_length")
# ~40 removed
,"Restore levels" = list(restore_categorical_variables)
,"Rename dep_var" = list(renamer, c("survived" = "dep_var"))
)

```r
model = list('gbm'
 , id_var = 'X'
 , distribution = 'bernoulli'
 , number_of_trees = 3000
 , shrinkage_factor = 0.005
),
```

```r
export = list(R = 'titanic')
```
)
list(
    # Import and data stage
    model = list('gbm'
        , .id_var = 'X'
        , distribution = 'bernoulli'
        , number_of_trees = 3000
        , shrinkage_factor = 0.005
    )
)

# Export stage
)


```r
# lib/controllers/models/models.R

construct_stage_runner <- Ramd::define("construct_stage_runner")[[1]](resource)
preprocessor <- Ramd::define("preprocessor")[[1]]

function(args, resource, output, director, modified, any_dependencies_modified) {
  parent.env(source_env)(construct_stage_runner) <- environment()
  if (is.element("raw", names(args))) return(output)
  require(objectdiff)
  message("Loading model: ", resource)
  tests <- file.path("test", resource)
  has_tests <- director$exists(tests)
  if (has_tests) { 
    testrunner <- stageRunner$new(new.env(), director$resource(tests))
    testrunner$transform(function(fn) {
      library(testthat); force(fn)
      function(after) fn(cached_env, after)
    })
  }

  model_version <- gsub("\"\\w+/\"", "", resource)
  if (isTRUE(args$refresh) || identical(resource, director$cache_get("last_model"))) {
    stagerunner <- construct_stage_runner(output, model_version)
  } else if (modified || any_dependencies_modified) {
    message(crayon::yellow("Copying cached environments..."))
    stagerunner <- construct_stage_runner(output, model_version)
    stagerunner$coalesce(director$cache_get("last_model_runner"))
  } else if (!director$cache_exists("last_model_runner")) {
    stagerunner <- construct_stage_runner(output, model_version)
  } else {
    stagerunner <- director$cache_get("last_model_runner")
  }

  if (has_tests) stagerunner$overlay(testrunner, "tests", flat = TRUE)
  director$cache_set("last_model", resource)
  director$cache_set("last_model_runner", stagerunner)
}
```

---

```r
# ../models/preprocessor.R

preprocessor <- function(resource, director, source_env) {
  source_env$extending <- function(model_version, expr) {
    eval.parent(substitute(within(resource(file.path("models/", model_version)), raw = TRUE), { expr })))
  }

  source_env$model_version <- version <- gsub("^[/]+[/]+[/]+", "", resource)
  source_env$model_name <- basename(version)
  source_env$output <- function(suffix = ", create = TRUE, dir = file.path(director$root(), "tmp") {
    fn <- file.path(dir, version, suffix)
    if (create & !file.exists(dir <- dirname(fn)))
      dir.create(dir, recursive = TRUE)
    fn
  }

  lexicals <- director.resource("lib/shared/lexicals")
  for (x in ls(lexicals)) source_env[[x]] <- lexicals[[x]]
  director.resource("lib/shared/source_mungebits")<-

  model <- resource()
  if (nzchar(Sys.getenv("CI"))) {
    model$import <- NULL
  }

  model
}
```

---

```r
# ../models/construct_stage_runner.R

model_env <- function() {
  if (identical(getOption("environment_type"), "environment")) {
```
• Other useful abstractions with their own grammar:

  – **Modules**: Nested bundles of related R functions.
  – **Indicators**: \( y \sim x_1 + x_2 \) tied to an ETL backend
  – **Stages**: Execution tasks for our stage runner.
  – **Jobs**: Build reports, monitoring checks, etc.
  – **Queries**: “Object-relational mapper” for R
Running A Model

run("example") # Runs models/dev/example.R

list(
  import = list(R = "iris"),
  data = list(
    "Create dep var" = list(renamer,
      c("Sepal.Length" = "dep_var")),
    "Example var" = list(example),
    "Create ID var" = list(
      multi_column_transformation(seq_along),
      "dep_var", "id")
  ),
  model = list("lm", .id_var = "id"),
  export = list(R = "model")
)
run("example")  # Runs models/dev/example.R

```r
list(
    import = list(R = "iris"),
    data = list(
        "Create dep var" = list(renamer,
            c("Sepal.Length" = "dep_var")),
        "Example var" = list(example),
        "Create ID var" = list(
            multi_column_transformation(seq_along),
            "dep_var", "id")
    ),
    model = list("lm", .id_var = "id"),
    export = list(R = "model")
)

model$predict(iris)  # [1] 5.005 4.757 4.890 ...
```
run(, "data/3")  # Re-runs one munge step

```r
list(
    import = list(R = "iris"),
    data  = list(
        "Create dep var" = list(renamer
            c("Sepal.Length" = "dep_var")),
        "Example var"   = list(example),
        "Create ID var" = list( multi_column_transformation(seq_along),
                          "dep_var", "id")
    ),
    model  = list("lm", .id_var = "id"),
    export = list(R = "model")
)

setdiff(ls(B), ls(A))  # [1] "id"
```
Each resource requires an accompanying test

```
test_project()
```
Dependency Management

Everyone working on the project has the same set of dependencies
Dependency Management

```
# lockfile.yml
packages:

  - name: devtools
    version: 1.12.0
    repo: hadley/devtools
    ref: v1.12.0

  - name: checkr
    version: 0.1.4
    repo: syberia/checkr

  - name: Ramd
    version: 0.3.8
    repo: robertzk/Ramd

  - name: statsUtils
    version: 0.1.4
    repo: robertzk/statsUtils

  - name: mungebits2
    version: 0.1.0.9014
    repo: syberia/mungebits2

  - name: syberiaMungebits2
    version: 0.1.2.9002
    repo: syberia/syberiaMungebits2

  - name: director
    version: 0.3.0.5.9000
    repo: syberia/director

  - name: tundra
    version: 0.3.0.9000
    repo: syberia/tundra

  - name: syberia
    version: 0.6.1.9009
    repo: syberia/syberia
    ref: 0.6.1.9009

  - name: s3mpi
    version: 0.2.40
    repo: robertzk/s3mpi

  - name: objectdiff
    version: 0.2.3.9003
    repo: robertzk/objectdiff

  - name: stagerunner
    version: 0.5.6
    repo: syberia/stagerunner
```
Scales to large teams of contributors working on thousands of R models

- 🔄 25,778 commits
- 🌱 517 branches
- 👤 34 contributors
Currently **2 main engines** are open sourced:

- Modeling engine
- Base engine *(dependency of modeling engine)*
Currently **2 main engines** are open sourced:

- Modeling engine
- Base engine *(dependency of modeling engine)*

Using Syberia for HFT automation:

- Define a backtesting engine & grammar *(built on top of base engine)*
Currently **2 main engines** are open sourced:

- Modeling engine
- Base engine (dependency of modeling engine)

Using Syberia for HFT automation:

- Define a backtesting engine & grammar (built on top of base engine)
- Use it to visualize + test strategies in R
Currently **2 main engines** are open sourced:

- Modeling engine
- Base engine *(dependency of modeling engine)*

Using Syberia for HFT automation:

- Define a backtesting engine & grammar *(built on top of base engine)*
- Use it to visualize + test strategies in R
- Define resources to transpile R -> VHDL
Currently **2 main engines** are open sourced:

- Modeling engine
- Base engine (dependency of modeling engine)

Using Syberia for HFT automation:

- Define a backtesting engine & grammar (built on top of base engine)
- Use it to visualize + test strategies in R
- Define resources to transpile R -> VHDL
- Push to FPGA arrays and trade realtime
A huge **thank you** to all Avantees that contributed to making Syberia happen:

- Abel Castillo
- David Feldman
- Jason French
- Kirill Sevastyanenko
- Peter Hurford
- Ryland Ely
- Tong Lu
syberia.io

github.com/robertzk/rfinance17
@rsyberia
rob@syberia.io