PRISM for model developers

Mohsen Sadatsafavi

February 19, 2019

# PRISM: PRogrammable Interface for Statistical/Simulation Models

This document explains the technical aspect of using PRISM for model developers. The emphasis is on R developers who want to access their models within R.

PRISM has a client R package that can be downloaded from GitHUb.

For doing so, you need to have installed the devtools package.

library(devtools)

install\_github(“<https://github.com/aminadibi/prism>”)

We will connect to an exemplary model called epicR

library(prism)  
prism::connect\_to\_model("prismServer", api\_key="123456", address ="localhost:5656")

## call is http://localhost:5656/ocpu/library/prismServer/R/gateway\_json2

## http://localhost:5656/ocpu/tmp/x0c0ca26aa2/R/.val

## This is an exemplary model for PRISM tutorial

## $result  
## [1] TRUE  
##   
## $session\_id  
## [1] "NBK896V5WU"  
##   
## $version  
## [1] ""  
##   
## $description  
## [1] "This is an exemplary model for PRISM tutorial"

This function returns a boolean operator for TRUE=success, FALSE=failure to connect.

Note that this function automatically stors session\_id in an internal variable so for the most part this session\_id is automatically passed to the server.

## Level 1 compatibility

Each model MUST expose one function to be compatible with PRISM. This function is called model\_run.

model\_run<-function(model\_input)  
{  
 #Process the input  
   
 #Run the model  
   
 #Return the results as a named list  
}

For example, imagine our model has 2 parameters. It return the backgroun mortality of an individual with these values

model\_run<-function(model\_input)  
{  
 age<-model\_input[["age"]]  
 sex<-model\_input[["sex"]]  
  
 if(is.null(age) || is.null(sex) || is.na(match(sex,c(1,2))) || (age>0 && age<120)) return(NULL)  
  
 return(list(result=age\*sex))  
}

Onn the client side,

res<-prism::model\_run(list(age=2,sex='Female'))

## call is http://localhost:5656/ocpu/library/prismServer/R/gateway\_json1

## http://localhost:5656/ocpu/tmp/x0a920a68ae/R/.val

print(res)

## $result  
## [1] 4

This is pretty much the basic level of compatibility with PRISM.

## Getting graphical output from the server

Now let’s upgrade the model such that it produces a plot on the server. Plots are stored on the server in the same sequence as they are generated.

model\_run<-function(model\_input)  
{  
 age<-model\_input[["age"]]  
 sex<-model\_input[["sex"]]  
  
 sex<-match(sex,c('Male','Female'))  
  
 if(is.null(age) || is.null(sex) || is.na(sex) || !(age>0 && age<120)) return(NULL)  
  
 return(list(result=age\*sex))  
}

res<-prism::model\_run(list(age=2,sex='Male',vector=c(1,2,3,4,5)))

## call is http://localhost:5656/ocpu/library/prismServer/R/gateway\_json1

## http://localhost:5656/ocpu/tmp/x0988eb3694/R/.val

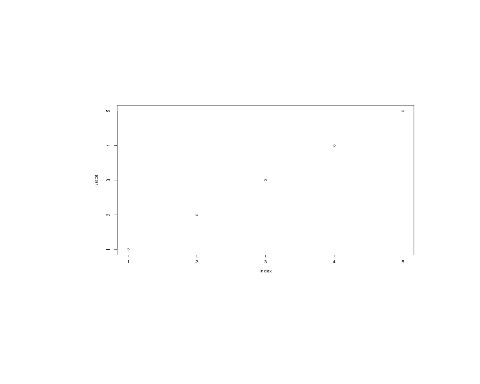
print(res)

## $result  
## [1] 2

x<-get\_plots()

## call is http://localhost:5656/ocpu/tmp/x0988eb3694/

for(i in 1:length(x))  
 plot(x[[i]])



## Making your model web-app ready

PRISM is enable to turn your model into complex, good-looking apps. We start form the ‘basic’ requirement.

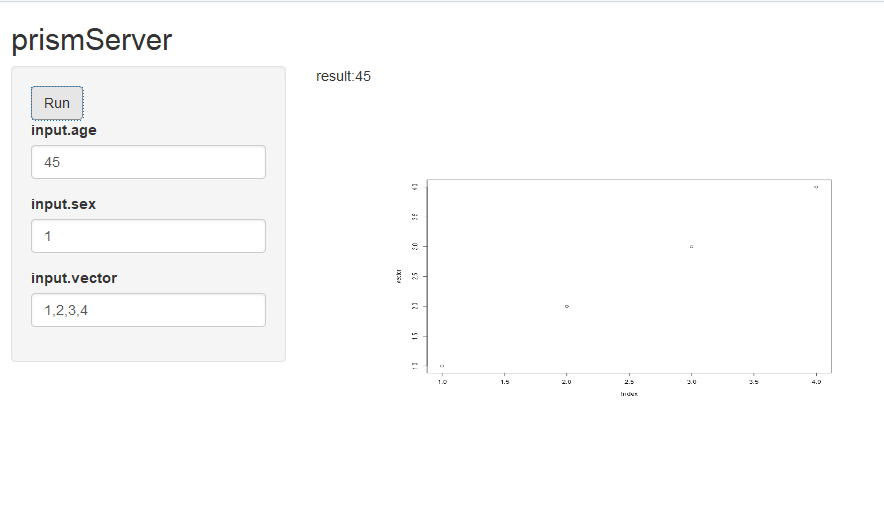
Because at the minimu, the Web app requires to know the input stracture of yourmodel, you need to supply the following function

get\_default\_input()

This function should return a non-heiararchical named list of all the inputs

get\_default\_input<-function()  
{  
 input<-list(age=45,sex=1,vector=c(1,2,3,4))  
 return(input)  
}

If your model has supplied this, then we you create a simple web app on the fly!



A caption