R Markdown for CHOP R User Group

Richard Hanna (Pro tip: use your own name)

01 July, 2019

## R Markdown for CHOP R User Group

Welcome to the example .rmd file for the CHOP R User Group presentation! I hope you find it useful and informative. This will serve as a template from which you can extrapolate whatever you may need for your own reports in your day-to-day workflow!

To start, some quick tips on markdown syntax:

* “#”s will declare different heading types
* “\*\*”s on either side of a string of text will make something **bold**
* “\_ \_”s on either side of a string of text will make something *italic*
* Use “<>” when you want to include an e-mail! [hannar1@email.chop.edu](mailto:hannar1@email.chop.edu)
* Include hyperlinks in text strings using a combination of such as to the [pediRES-Q Website](https://www.pedires-q.org/)
* You can also call in-line code commands outside of chunks, like asking what is 2+2? 4
  + You will have to look at the .rmd file to make sense of that one!
* For more options and help, visit the [RMarkdown Cheatsheet](https://www.rstudio.com/wp-content/uploads/2015/02/rmarkdown-cheatsheet.pdf)

## The Heart Dataset

The dataset we will be using is a sample one I took from [Kaggle](https://www.kaggle.com/ronitf/heart-disease-uci) but it can also be found on the [UCI Machine Learning Repository](https://archive.ics.uci.edu/ml/datasets/Heart+Disease). This dataset describes 14 attributes of 303 patients with the target indicating whether or not a patient had heart disease.

Let’s load the dataset from [Kaggle](https://www.kaggle.com/ronitf/heart-disease-uci) and start perusing what’s in it.

heart <- read.csv("Data\_Files/heart.csv")  
  
# For now we will keep this commented out until we review how to use parameters.  
#heart <- read.csv(params$data)  
  
# There's something here about thinking with your heart over your head but   
# I can't quite put it together...  
head(heart)

## age sex cp trestbps chol fbs restecg thalach exang oldpeak slope ca thal  
## 1 63 1 3 145 233 1 0 150 0 2.3 0 0 1  
## 2 37 1 2 130 250 0 1 187 0 3.5 0 0 2  
## 3 41 0 1 130 204 0 0 172 0 1.4 2 0 2  
## 4 56 1 1 120 236 0 1 178 0 0.8 2 0 2  
## 5 57 0 0 120 354 0 1 163 1 0.6 2 0 2  
## 6 57 1 0 140 192 0 1 148 0 0.4 1 0 1  
## target  
## 1 1  
## 2 1  
## 3 1  
## 4 1  
## 5 1  
## 6 1

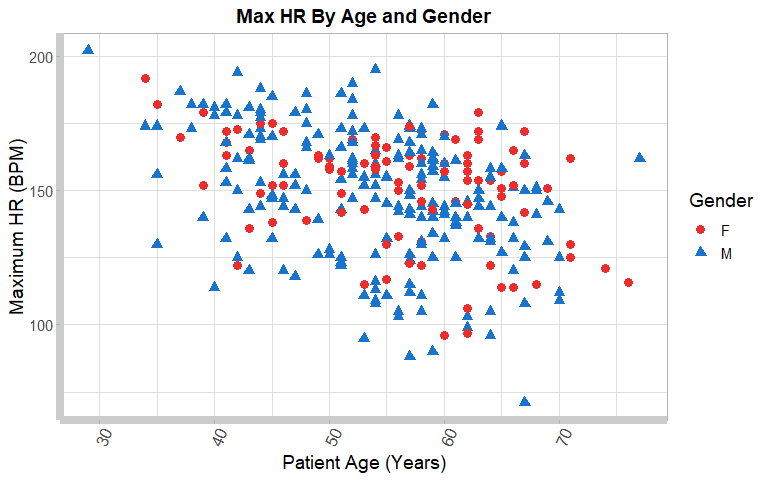
##### 

## Initial Analyses

Let’s start with a simple plot, looking into max heart rate as a function of age by gender identification. As you make larger and more complicated markdown reports, it will become helpful to keep analyses separate from your .rmd so you can easily reference locations of interest. It also helps with debugging!

source() is a great way to pull in other files. When you use source() the file placed in the argument will be run in its entirety, adding any variables to your environment. So be sure you are aware of what you place in your external file!

### Max HR by Age and Gender

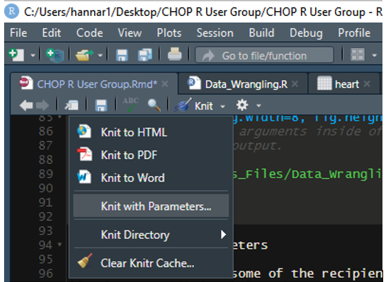


##### 

### Introducing Parameters

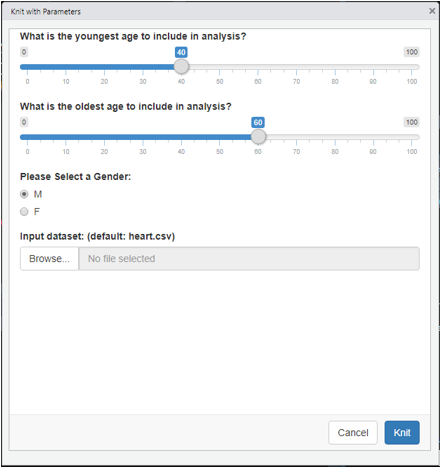
Oh no! It looks like some of the recipients of this report may want to look only at specific age groups of interest. Some wish to view only patients under the age of 60 while another group is only interested in elderly populations.

In the code chunk below, notice we have added an additional dplyr command, filter(), and included our first two parameters, params$MinAge & params$MaxAge, which were defined in our YAML. This allows for substitutions based on knitting with parameters (see the image below):



Knitting with Parameters

When you knit you will see a GUI prompt come up. Don’t worry that we seem to be filtering by males only or haven’t uploaded a dataset, we have not yet called these parameters into action anywhere so they won’t get applied.

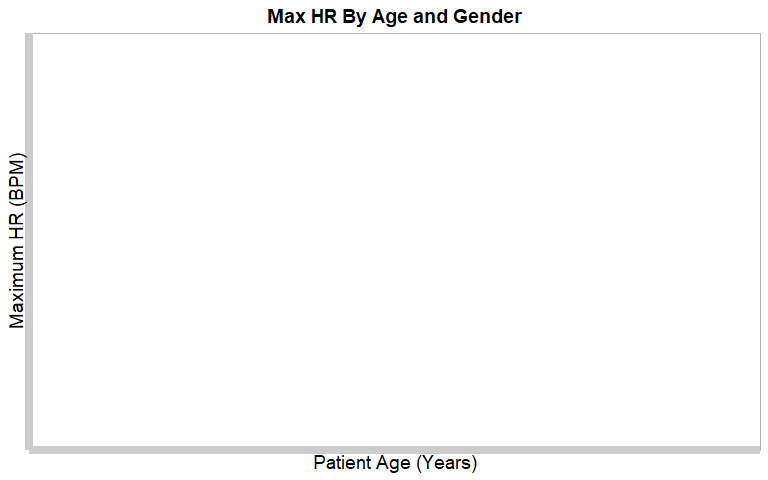


Knit Shiny GUI

##### 

Now let’s take a look at how we can incoporate parameters in our code. For this one I have selected ages between 40 and 60:

# Using a similar set-up to the code in our `source()` file:  
  
DT1\_Gender.plot <- heart %>%   
 select(age, Gender, thalach) %>%   
 filter(age > params$MinAge & age < params$MaxAge) %>%   
 ggplot(aes(x = age, y = thalach, color = Gender, shape = Gender)) +   
 geom\_point(size = 3) + my\_ggtheme +  
 xlab("Patient Age (Years)") + ylab("Maximum HR (BPM)") +  
 ggtitle("Max HR By Age and Gender") +  
 scale\_colour\_manual(values = c("firebrick2", "dodgerblue3"))  
  
DT1\_Gender.plot



## Your Turn!

After seeing how to incoporate slider parameters for age, what other methods could you use to parameterize this report? I have given two already, one with gender-based radio buttons and one with file uploads. Try to see if you can develop a plot yourself using the M/F radio button parameter!

The file upload one can be especially helpful when you have developed a framework for time period-based reporting where the only change might be the data input. Using the file upload parameter lets you intuitively select your dataset (in this case from a .csv) without having to always manually enter it in!

## Additional Tips & Tricks

Now that you’ve seen how to parameterize your scripting, it’s important to think about how to incorporate such methods in the best ways to make your project succinct, efficient, and clean. The key to parameters is reproducibility.

* One of the pitfalls to parameters is that they become increasingly difficult to test in-script (via CTRL + Enter or the like) since params$ is not a variable and requires input. That input can only be delivered via knitting the document. You may want to come up with a way to test your script before committing an overly complex algorithm to parameterization.
* Parameters play nicely with conditional logic. Using ifelse() statements coupled with parameters can allow you to knit with addition/removal of sections of your report based on the inputs you provide on knitting.
* If you have not used Shiny or other methods in R requiring interactivity and user input, parameters can be a good introduction to it.

### **Bonus Content:** Machine Learning

Since we worked with the heart dataset, you may have noticed a target column indicating if a patient had heart disease (0) or did not have heart disease (1). I am by no means a statistician or ML expert, but I figured some of you might be. I modeled this model (hehe) after a R blogpost on analyzing the infamous “Titanic” dataset which looked into passenger survival. Since this is a similar classification problem, I’ve chosen to follow this logistic regression analysis. You can find the full post [here](https://www.r-bloggers.com/how-to-perform-a-logistic-regression-in-r/)

# Sample Logistic Regression with `heart` dataset  
  
set.seed(101) # Set a seed so results can be reproducible  
  
# Remove the Gender column which is a duplicate of the sex column,  
# name this `heart.ml`  
heart.ml <- heart  
heart.ml$Gender <- NULL  
  
# Create a sample train and test split of the data using the `caTools` package  
sample <- sample.int(n = nrow(heart.ml), size = floor(.75\*nrow(heart.ml)), replace = F)  
train <- heart.ml[sample, ]  
test <- heart.ml[-sample, ]  
  
# Fit a logistic regression model to the training dataset  
model <- glm(target ~.,family=binomial(link='logit'),data=train)  
  
# Assess the results of the model  
summary(model)

##   
## Call:  
## glm(formula = target ~ ., family = binomial(link = "logit"),   
## data = train)  
##   
## Deviance Residuals:   
## Min 1Q Median 3Q Max   
## -2.2615 -0.3680 0.1809 0.5623 2.6066   
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) 2.823904 3.046561 0.927 0.353970   
## age -0.003105 0.028312 -0.110 0.912679   
## sex -1.751672 0.540082 -3.243 0.001181 \*\*   
## cp 0.884728 0.232180 3.811 0.000139 \*\*\*  
## trestbps -0.015167 0.012510 -1.212 0.225379   
## chol -0.004518 0.004763 -0.949 0.342770   
## fbs -0.543546 0.630641 -0.862 0.388746   
## restecg 0.314394 0.404012 0.778 0.436462   
## thalach 0.022844 0.012287 1.859 0.062990 .   
## exang -0.901519 0.472051 -1.910 0.056160 .   
## oldpeak -0.475283 0.255552 -1.860 0.062910 .   
## slope 0.596006 0.403002 1.479 0.139163   
## ca -0.925272 0.232311 -3.983 6.81e-05 \*\*\*  
## thal -0.831422 0.337516 -2.463 0.013764 \*   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 311.47 on 226 degrees of freedom  
## Residual deviance: 155.96 on 213 degrees of freedom  
## AIC: 183.96  
##   
## Number of Fisher Scoring iterations: 6

It would appear our most signficiant factors influencing this model are the variables sex, cp (chest pain type), and ca (number of major vessels colored by flourosopy) as indicated by their p-values and significance codes (\*\*\*). The table of deviance can be produced by running an anova() command:

anova(model)

## Analysis of Deviance Table  
##   
## Model: binomial, link: logit  
##   
## Response: target  
##   
## Terms added sequentially (first to last)  
##   
##   
## Df Deviance Resid. Df Resid. Dev  
## NULL 226 311.47  
## age 1 15.028 225 296.44  
## sex 1 30.195 224 266.25  
## cp 1 39.811 223 226.44  
## trestbps 1 5.044 222 221.39  
## chol 1 0.930 221 220.46  
## fbs 1 0.492 220 219.97  
## restecg 1 0.880 219 219.09  
## thalach 1 18.114 218 200.98  
## exang 1 3.966 217 197.01  
## oldpeak 1 14.107 216 182.90  
## slope 1 0.559 215 182.34  
## ca 1 20.206 214 162.14  
## thal 1 6.180 213 155.96

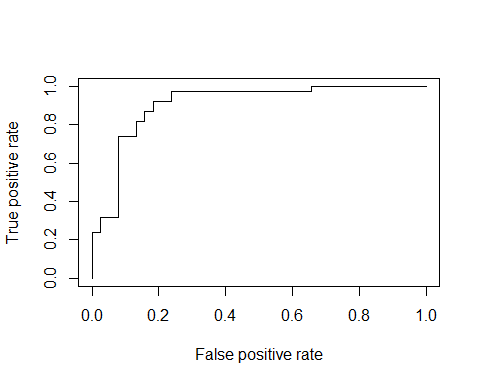
To test the model’s accuracy, fit the results using R’s predict() command and establish a threshold at 0.5 to turn the values into classifiers of 1 or 0. then assess the error between the model response and the true values from the target dataset.

# Test model accuracy  
# Use `predict()` to run the model on we created on the test dataset using the  
# initial 13 variables.  
fitted.results <- predict(model,newdata=test[1:13],type='response')   
  
# Assign all values >0.5 to "1" and all values <=0.5 to "0"  
fitted.results <- ifelse(fitted.results > 0.5,1,0)  
  
# Determine the rate at which our predicted results to not match the true results  
misClasificError <- mean(fitted.results != test$target)  
  
# Print the rate of correct predictions as accuracy  
print(paste('Accuracy: ', round(1-misClasificError, 4)\*100, "%"))

## [1] "Accuracy: 85.53 %"

Finally, plot a receiver operating characteristics (ROC) curve and assess the area-under-the-curve (AUC) value:

p <- predict(model, newdata=test[1:13], type="response")  
pr <- prediction(p, test$target)  
prf <- performance(pr, measure = "tpr", x.measure = "fpr")  
plot(prf)



auc <- performance(pr, measure = "auc")  
auc <- auc@y.values[[1]]  
  
print(paste('AUC Value: ', round(auc, 4)))

## [1] "AUC Value: 0.9065"

Not bad! Looks like our model, given the available data, is able to reasonably predict wether or not a patient had heart disease.

## Come Join CHOP R!



CHOP R QR