Block III

Sara Dovalo and Javier Muñoz Flores

21/3/2022

Contents

Introduction

The content of this section is, mainly, a further description of the dataset used as well as the exposition of the problem to solve.

The dataset selected contains several patient records of different medical measurements in order to predict wether a person is more likely to suffer a heart disease, i.e. a failure . The data has been retrieved from the public *kaggle* repository and it is the product of the combination of five different datasets from different regions of EEUU. The final dataset contains in total 918 instances and 12 attributes, which 7 of them are categorical and the five remaining are numerical:

- `Age`(*quantitative*): age of the patient in years
- `Sex`(*qualitative*): sex of the patient [M: Male, F: Female]
- `ChestPainType`(*qualitative*): Angina type, i.e. chest pain, frequently caused when the heart muscle is not able to get enough oxygen-rich blood [TA: Typical Angina, ATA: Atypical Angina, NAP: Non-Anginal Pain, ASY: Asymptomatic]
- `RestingBP`(*quantitative*): resting blood pressure [mm Hg]. A normal level is less than 180 mm Hg.
- `Cholesterol`(*quantitative*): serum cholesterol [mm/dl]
- `FastingBS`(*qualitative*): fasting blood sugar [1: if FastingBS $>$ 120 mg/dl, 0: otherwise]
- `RestingECG`(*qualitative*): resting electrocardiogram results [Normal: Normal, ST: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of $>$ 0.05 mV), LVH: showing probable or definite left ventricular hypertrophy by Estes' criteria]
- `MaxHR`(*quantitative*): maximum heart rate achieved [Numeric value between 60 and 202]
- `ExerciseAngina`: exercise-induced angina [Y: Yes, N: No]
- `Oldpeak`(*quantitative*): oldpeak = ST [Numeric value measured in depression]
- `ST_Slope`(*qualitative*): the slope of the peak exercise ST segment [Up: upsloping, Flat: flat, Down: downsloping]
- `HeartDisease`(*qualitative*): class variable [1: heart disease, 0: Normal]

Clearly, it is a binary classification problem since the target variable has two levels.

Preprocessing

First of all, it is suitable to visualize the data and to identify if there are missing values which could add noise and disrupt the performance of the future models created.

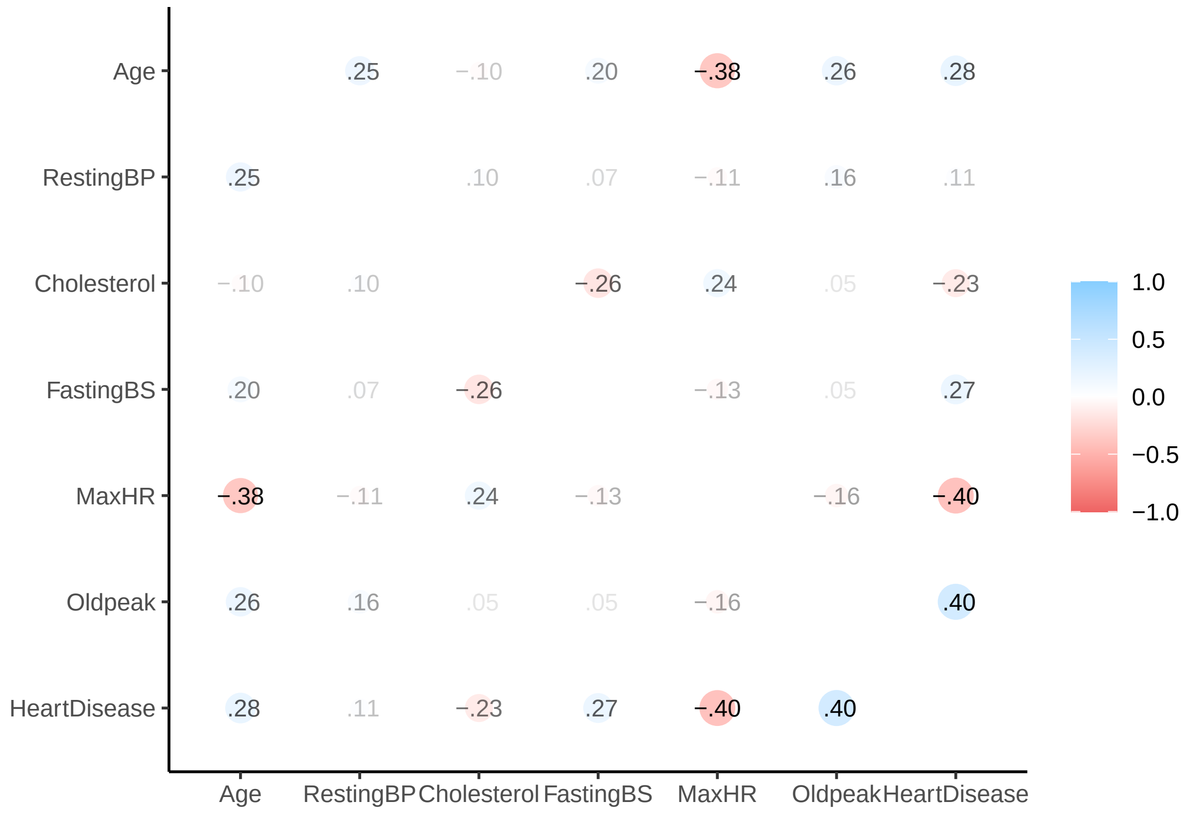
We show the correlation between the numeric variables through a nice plot to carry out a first exploratory analysis.

```r
# Plot correlation
data_num <- heart.data %>%
select(where(is.numeric))
corr <- correlate(data_num)
```

```
##
## Correlation method: 'pearson'
## Missing treated using: 'pairwise.complete.obs'
```

```r
rplot(corr, print_cor = TRUE)
```

```
## Don't know how to automatically pick scale for object of type noquote. Defaulting to continuous.
## Don’t know how to automatically pick scale for object of type noquote. Defaulting to continuous.
```



```r
# Count number of zeros (missing values)
heart.data %>% filter(Cholesterol == 0) %>% summarize(count = n())
```

```
##   count
## 1   172
```

We notice that the variable `Cholesterol` contains 172 values equal to zero. They must be considered as missing values as a person cannot that such a low level of cholesterol in blood. We decide to eliminate those observations that have a zero in that variable.

```r
# Eliminate rows which contain 0 in variable Cholesterol
heart.data = heart.data %>%
filter(Cholesterol != 0)
# Number of missing values
heart.data %>% filter(Cholesterol == 0) %>% summarize(count = n())
```

```
##   count
## 1     0
```

Furthermore, since the dataset includes several categorical attributes, it is necessary to transform them into *factors*.

```
# Categorical variables as factors
heart.data = heart.data %>%
       mutate_each_(funs(factor(.)),c(2,3,6,7,9,11,12))
str(heart.data)
```

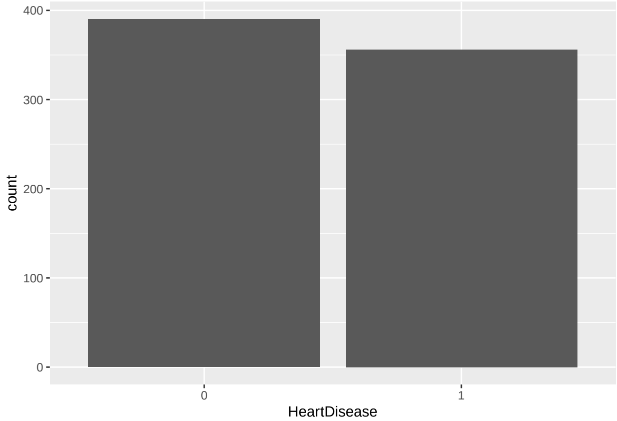
```
## 'data.frame':    746 obs. of  12 variables:
##  $ Age           : int  40 49 37 48 54 39 45 54 37 48 ...
##  $ Sex           : Factor w/ 2 levels "F","M": 2 1 2 1 2 2 1 2 2 1 ...
##  $ ChestPainType : Factor w/ 4 levels "ASY","ATA","NAP",..: 2 3 2 1 3 3 2 2 1 2 ...
##  $ RestingBP     : int  140 160 130 138 150 120 130 110 140 120 ...
##  $ Cholesterol   : int  289 180 283 214 195 339 237 208 207 284 ...
##  $ FastingBS     : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ RestingECG    : Factor w/ 3 levels "LVH","Normal",..: 2 2 3 2 2 2 2 2 2 2 ...
##  $ MaxHR         : int  172 156 98 108 122 170 170 142 130 120 ...
##  $ ExerciseAngina: Factor w/ 2 levels "N","Y": 1 1 1 2 1 1 1 1 2 1 ...
##  $ Oldpeak       : num  0 1 0 1.5 0 0 0 0 1.5 0 ...
##  $ ST_Slope      : Factor w/ 3 levels "Down","Flat",..: 3 2 3 2 3 3 3 3 2 3 ...
##  $ HeartDisease  : Factor w/ 2 levels "0","1": 1 2 1 2 1 1 1 1 2 1 ...
```

To finish this section, we check if the classes of the response `HeartDisease` are balanced or not. We visualize it through a barplot to see it more clearly.

```
# Count the observation of each class
heart.data %>%
count(HeartDisease)
```

```
##   HeartDisease   n
## 1            0 390
## 2            1 356
```

```
# Barplot
ggplot(heart.data, aes(HeartDisease)) + geom_bar()
```



The classes are balanced, it will be not needed to over/undersampling the sample.

Modelling with `H2O` package

We follow the same procedure that we did in classroom for using `h2o.automl()`, i.e. fitting, benchmarking, predicting and explaining, in that order.

Fitting

Before starting with the selection of the models, the *local H2O cluster* is initialized in order to leverage the parallelization in the virtual machine which the package provides and to use H2O functions.

The first step is to convert the data into a `h2o`object and then, to identify the names of the predictors as well as the name of the response.

Then, it is important to mention that there will not be splitting into train and test, since we will use cross-validation metrics on the leaderboard model. Thus, we have to specify the number of folds (in our case will be `nfolds = 8`) needed to the cross-validation process. However, we do not have to indicate the predictors names, instead only the`dataFrame`and the response variable. In addition, we do not exclude any algorithm in `h2o.automl()` method, but we limit the time for fitting the models (`max_runtime_secs = 30` and `max_runtime_secs_per_model = 5`)

```
# Initialize h2o
h2o.init()
```

```
##  Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         2 hours 27 minutes
##     H2O cluster timezone:       Europe/Paris
##     H2O data parsing timezone:  UTC
##     H2O cluster version:        3.36.0.3
##     H2O cluster version age:    1 month and 7 days
##     H2O cluster name:           H2O_started_from_R_34639_pgd457
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   0.92 GB
##     H2O cluster total cores:    8
##     H2O cluster allowed cores:  8
##     H2O cluster healthy:        TRUE
##     H2O Connection ip:          localhost
##     H2O Connection port:        54321
##     H2O Connection proxy:       NA
##     H2O Internal Security:      FALSE
##     R Version:                  R version 4.1.1 (2021-08-10)
```

```
h2o.no_progress()
# Send data to local H2O cluster
data.h <- as.h2o(heart.data)
# Data summary
h2o.describe(data.h)
```

```
##             Label Type Missing Zeros PosInf NegInf  Min   Max        Mean
## 1             Age  int       0     0      0      0 28.0  77.0  52.8820375
## 2             Sex enum       0   182      0      0  0.0   1.0   0.7560322
## 3   ChestPainType enum       0   370      0      0  0.0   3.0          NA
## 4       RestingBP  int       0     0      0      0 92.0 200.0 133.0227882
## 5     Cholesterol  int       0     0      0      0 85.0 603.0 244.6353887
## 6       FastingBS enum       0   621      0      0  0.0   1.0   0.1675603
## 7      RestingECG enum       0   176      0      0  0.0   2.0          NA
## 8           MaxHR  int       0     0      0      0 69.0 202.0 140.2265416
## 9  ExerciseAngina enum       0   459      0      0  0.0   1.0   0.3847185
## 10        Oldpeak real       0   317      0      0 -0.1   6.2   0.9016086
## 11       ST_Slope enum       0    43      0      0  0.0   2.0          NA
## 12   HeartDisease enum       0   390      0      0  0.0   1.0   0.4772118
##          Sigma Cardinality
## 1   9.5058879          NA
## 2   0.4297617           2
## 3          NA           4
## 4  17.2827498          NA
## 5  59.1535237          NA
## 6   0.3737260           2
## 7          NA           3
## 8  24.5241072          NA
## 9   0.4868551           2
## 10  1.0728611          NA
## 11         NA           3
## 12  0.4998155           2
```

```
# Identify the names of the response and predictors
resp_h <- "HeartDisease"
pred_h<- setdiff(names(data.h), resp_h)
# Call h2o.automl()
model_h <- h2o.automl(y = resp_h, training_frame = data.h, max_runtime_secs = 30,max_runtime_secs_per_m
seed = 1, verbosity = NULL)
```

Benchmarking

In this section, we have to explore the leaderboard of models in order to carry out a first comparison.

```
# Leaderboard
lead_h <- model_h@leaderboard
names(lead_h)[5] <- "mpce" # Rename mean_per_class_error to shorten output
print(lead_h[, -6], n = nrow(lead_h)) # Exclude final column to fit the table in one page
```

```
##                                                    model_id       auc   logloss
## 1     StackedEnsemble_AllModels_2_AutoML_3_20220323_225604 0.9325771 0.3301527
## 2              GBM_grid_1_AutoML_3_20220323_225604_model_3 0.9325411 0.3271347
## 3     StackedEnsemble_AllModels_1_AutoML_3_20220323_225604 0.9324114 0.3307698
## 4  StackedEnsemble_BestOfFamily_2_AutoML_3_20220323_225604 0.9322313 0.3338607
## 5  StackedEnsemble_BestOfFamily_3_AutoML_3_20220323_225604 0.9320117 0.3359566
## 6                           GBM_1_AutoML_3_20220323_225604 0.9311906 0.3370990
## 7              GBM_grid_1_AutoML_3_20220323_225604_model_4 0.9290046 0.3415283
## 8  StackedEnsemble_BestOfFamily_1_AutoML_3_20220323_225604 0.9289830 0.3381333
## 9                           GBM_5_AutoML_3_20220323_225604 0.9284680 0.3408614
## 10                          GBM_4_AutoML_3_20220323_225604 0.9282988 0.3364268
## 11             GBM_grid_1_AutoML_3_20220323_225604_model_2 0.9281727 0.3356772
## 12                          GBM_2_AutoML_3_20220323_225604 0.9279566 0.3362758
## 13                          GBM_3_AutoML_3_20220323_225604 0.9278594 0.3369303
## 14                          DRF_1_AutoML_3_20220323_225604 0.9262136 0.3886529
## 15                          GLM_1_AutoML_3_20220323_225604 0.9259399 0.3459442
## 16             GBM_grid_1_AutoML_3_20220323_225604_model_1 0.9248992 0.3478608
## 17                          XRT_1_AutoML_3_20220323_225604 0.9221262 0.3723422
## 18                 DeepLearning_1_AutoML_3_20220323_225604 0.9131410 0.3799328
## 19    DeepLearning_grid_1_AutoML_3_20220323_225604_model_1 0.9105517 0.4469685
## 20             GBM_grid_1_AutoML_3_20220323_225604_model_5 0.9092300 0.5957935
##        aucpr      mpce        mse
## 1  0.9065720 0.1202823 0.09941243
## 2  0.9073521 0.1204624 0.09774654
## 3  0.9067751 0.1219893 0.09898539
## 4  0.9077112 0.1221766 0.10070448
## 5  0.9080762 0.1268150 0.10115304
## 6  0.9095533 0.1257131 0.10204181
## 7  0.9036310 0.1251657 0.10235549
## 8  0.9036747 0.1268727 0.10261143
## 9  0.8973031 0.1254105 0.10281113
## 10 0.9043961 0.1305964 0.10078449
## 11 0.8958956 0.1285220 0.10017451
## 12 0.8987179 0.1287669 0.10008656
## 13 0.8966380 0.1320009 0.10044678
## 14 0.9067189 0.1497695 0.10757117
## 15 0.9070012 0.1382311 0.10631715
## 16 0.8927015 0.1310861 0.10356977
## 17 0.8993209 0.1441587 0.11315271
## 18 0.8819768 0.1495895 0.11342885
## 19 0.8793046 0.1403054 0.12222662
## 20 0.8765854 0.1667387 0.20187068
##
## [20 rows x 6 columns]
```

The leader has been a *StackedEnsemble* model with an error

Prediction

We select the leader of the models and predict a few rows.

```
h2o.predict(object = model_h@leader, newdata = data.h[1:8, ])
```

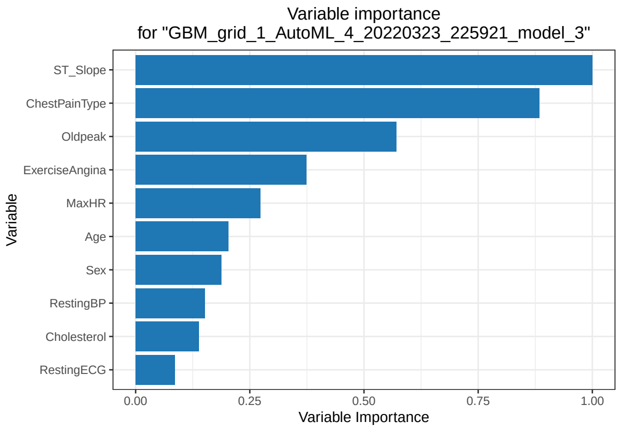
```
##   predict        p0         p1
## 1       0 0.9886658 0.01133416
## 2       0 0.5772694 0.42273057
## 3       0 0.9893544 0.01064562
## 4       1 0.1557225 0.84427751
## 5       0 0.9857569 0.01424312
## 6       0 0.9673155 0.03268450
##
## [8 rows x 3 columns]
```

Explanation

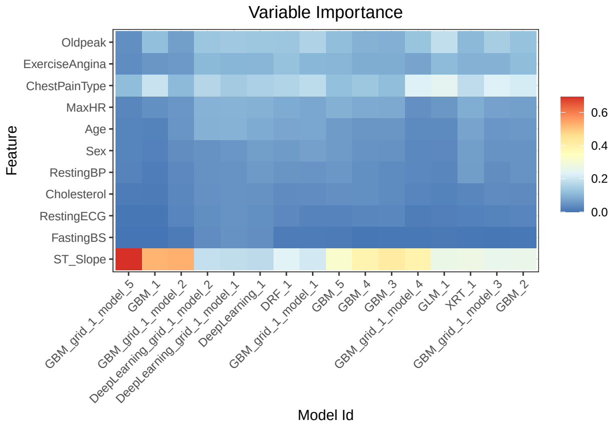
Because of the short report we must to hand in, it has not be possible to do a large interpretation of the plot results. However, it is suitable to identify what are the variables most important for the model and the correlation of them with the response. Thus, we use *SHAP* plot and *Variable Importance* plot to carry out a short explanation.

```r
exp <- h2o.explain(object = model_h, newdata = data.h)
exp
```

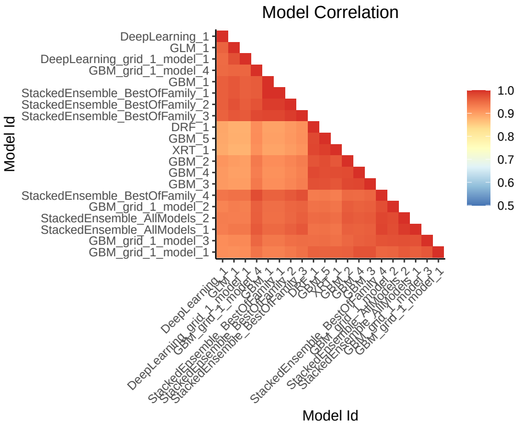
```
##
##
## Leaderboard
## ===========
##
## > Leaderboard shows models with their metrics. When provided with H2OAutoML object, the leaderboard
##
##
## |  | model_id | auc | logloss | aucpr | mean_per_class_error | rmse | mse | training_time_ms | predic
## |:---:|:---:|:---:|:---:|:---:|:---:|:---:|:---:|:---:|:---:|:---:|
## | **1** |GBM_grid_1_AutoML_4_20220323_225921_model_3 | 0.932541054451167 | 0.327134676097094 | 0.907
## | **2** |StackedEnsemble_AllModels_2_AutoML_4_20220323_225921 | 0.932541054451167 | 0.33000272272580
## | **3** |StackedEnsemble_AllModels_1_AutoML_4_20220323_225921 | 0.932411408815903 | 0.33076978939035
## | **4** |StackedEnsemble_BestOfFamily_2_AutoML_4_20220323_225921 | 0.932231345433593 | 0.33386071023
## | **5** |StackedEnsemble_BestOfFamily_3_AutoML_4_20220323_225921 | 0.932069288389513 | 0.33575991883
## | **6** |StackedEnsemble_BestOfFamily_4_AutoML_4_20220323_225921 | 0.932008066839527 | 0.33268115885
## | **7** |GBM_1_AutoML_4_20220323_225921 | 0.931190579083838 | 0.337098984517336 | 0.909553316037273
## | **8** |GBM_grid_1_AutoML_4_20220323_225921_model_4 | 0.929411552866609 | 0.3390226231438 | 0.9040
## | **9** |StackedEnsemble_BestOfFamily_1_AutoML_4_20220323_225921 | 0.92898300201671 | 0.338133288602
## | **10** |GBM_5_AutoML_4_20220323_225921 | 0.928468020743302 | 0.340861353584878 | 0.897303139940222
## | **11** |GBM_4_AutoML_4_20220323_225921 | 0.92829876116393 | 0.336426836166131 | 0.904396143187328
## | **12** |GBM_grid_1_AutoML_4_20220323_225921_model_2 | 0.928172716796312 | 0.335677225078254 | 0.89
## | **13** |GBM_2_AutoML_4_20220323_225921 | 0.92795664073754 | 0.336275767631698 | 0.898717939364543
## | **14** |GBM_3_AutoML_4_20220323_225921 | 0.927859406511092 | 0.336930268364824 | 0.896637999453036
## | **15** |DRF_1_AutoML_4_20220323_225921 | 0.926213627196773 | 0.388652886326296 | 0.906718879005219
## | **16** |GLM_1_AutoML_4_20220323_225921 | 0.925939930855661 | 0.34594420032564 | 0.907001227276824
## | **17** |GBM_grid_1_AutoML_4_20220323_225921_model_1 | 0.924899164505906 | 0.34786076655743 | 0.892
## | **18** |XRT_1_AutoML_4_20220323_225921 | 0.922126188418323 | 0.372342230184179 | 0.899320912017774
## | **19** |DeepLearning_grid_1_AutoML_4_20220323_225921_model_1 | 0.917707433016422 | 0.4180166031179
## | **20** |DeepLearning_1_AutoML_4_20220323_225921 | 0.915467444540478 | 0.373170463592199 | 0.890664
##
##
## Confusion Matrix
## ================
##
## > Confusion matrix shows a predicted class vs an actual class.
##
##
##
## GBM_grid_1_AutoML_4_20220323_225921_model_3
## -------------------------------------------
##
## |  | 0 | 1 | Error | Rate
## |:---:|:---:|:---:|:---:|:---:|
## | **0** |356 | 34 | 0.0871794871794872 |  =34/390 |
## | **1** |13 | 343 | 0.0365168539325843 |  =13/356 |
## | **Totals** |369 | 377 | 0.0630026809651475 |  =47/746 |
##
##
## Variable Importance
## ===================
##
## > The variable importance plot shows the relative importance of the most important variables in the
```



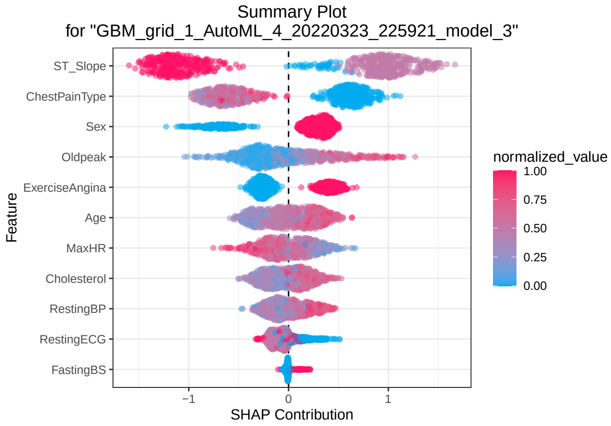
```
##
##
## Variable Importance Heatmap
## ===========================
##
## > Variable importance heatmap shows variable importance across multiple models. Some models in H2O re
```



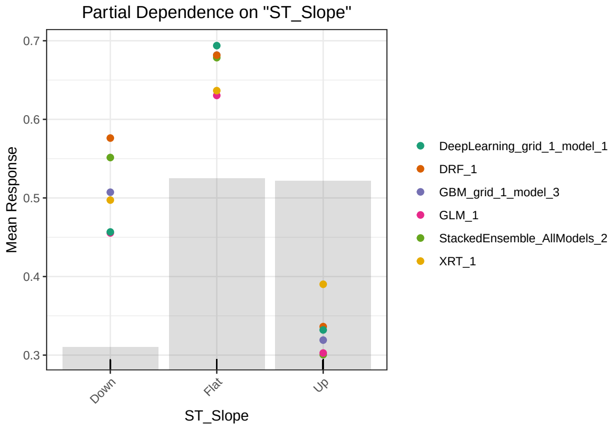
```
##
##
## Model Correlation
## =================
##
## > This plot shows the correlation between the predictions of the models. For classification, frequen
```

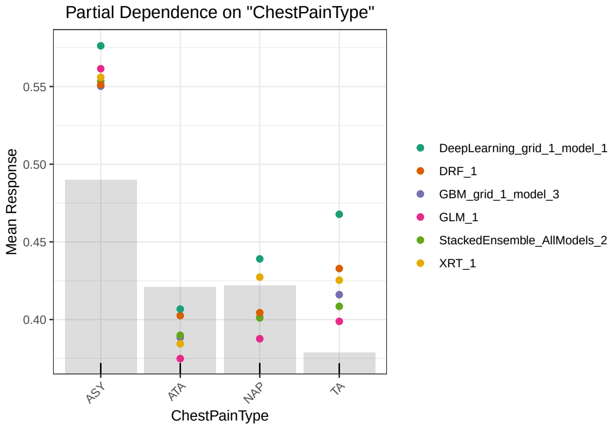


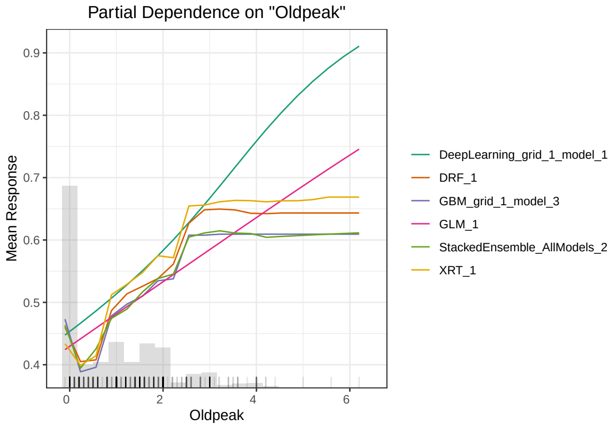
```
## Interpretable models: GLM_1_AutoML_4_20220323_225921
##
##
## SHAP Summary
## ============
##
## > SHAP summary plot shows the contribution of the features for each instance (row of data). The sum
```

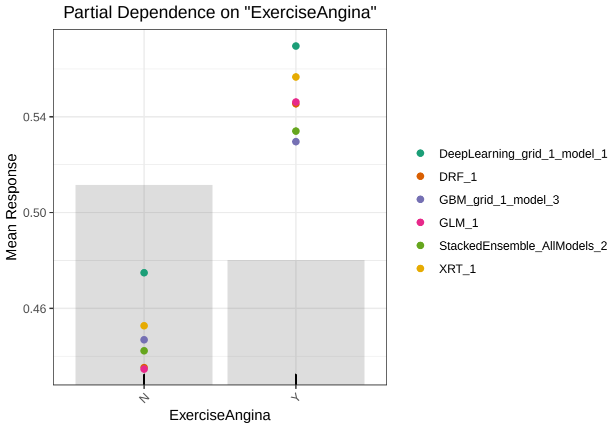


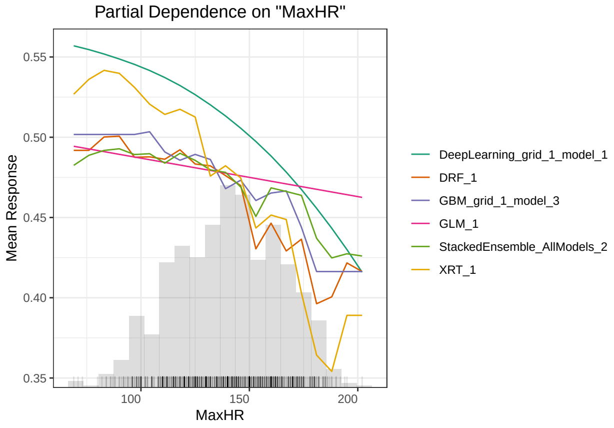
```
##
##
## Partial Dependence Plots
## ========================
##
## > Partial dependence plot (PDP) gives a graphical depiction of the marginal effect of a variable on
```











```
h2o.shutdown(prompt = FALSE)
```

Clearly, the most important variable is *ST_Slope*, an important feature which can be seen in an electrocardiogram. As expected, these unbiased features related with the functionality of the heart have an crucial role in the prediction.

SHAP plot, for instance, reveals that women are more correlated with the response than men, a point that is not so clear in a first view.

Modelling with `tidymodels` package

`Tidymodels`is an interface that unifies hundreds of functions from different packages, facilitating all stages of pre-processing, training, optimization and validation of predictive models.

The main packages that are part of the `tidymodels` ecosystem are:`brrom`, `rsample`, `parsnip`, `discrim`, `corr y` `tidypredict`, among others.

This package offers so many possibilities that they can hardly be shown with a single example.

The first step is to split the database into two subsets, the one used for training and the one used for validation. This is done with the `initial_split()` command of the `rsample` package. The training dataset, the one used for training the model and the one used to validate the model metrics can then be separated to help the selection, from a set of models applied on the same data, that one which performs best. It could be separated into two subsets (training and test), but it is a better practice to perform a cross-validation, so this one will be applied.

```
# Partition on training and test
heart_split <- initial_split(heart.data, prop = .8)
data_train <- training(heart_split)
data_validate <- testing(heart_split)
```

For simplicity we only consider 7 of the 11 explanatory variables.

```
variables <- c("Age", "Sex", "RestingBP", "Cholesterol", "FastingBS",
               "RestingECG", "MaxHR", "HeartDisease")
data_train <- data_train %>% dplyr::select(all_of(variables))
data_validate <- data_validate %>% dplyr::select(all_of(variables))
```

Logistic model

Build a model

As the variable of interest `HeartDisease` is a binary variable, a logistic regression model is chosen, which will be our basis. An object shall be created that stores the formula for later use:

```
glm_fit <- glm(HeartDisease ~ Age + Sex + RestingBP + Cholesterol + FastingBS +
                 RestingECG + MaxHR , data = data_train, family = binomial)
glm_fit_formula <- as.formula(HeartDisease ~ Age + Sex + RestingBP + Cholesterol +
                 FastingBS + RestingECG + MaxHR)
```

```
summary(glm_fit)
```

```
##
## Call:
## glm(formula = HeartDisease ~ Age + Sex + RestingBP + Cholesterol +
##     FastingBS + RestingECG + MaxHR, family = binomial, data = data_train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.3175  -0.8877  -0.3495   0.8989   2.4044
##
```

```
## Coefficients:
##                    Estimate Std. Error z value Pr(>|z|)    
## (Intercept)       -1.178513   1.335328  -0.883  0.37747    
## Age                0.033622   0.011858   2.835  0.00458 ** 
## SexM               1.628546   0.250080   6.512 7.41e-11 ***
## RestingBP          0.007572   0.005960   1.270  0.20392    
## Cholesterol        0.006765   0.001759   3.845  0.00012 ***
## FastingBS1         0.365115   0.271001   1.347  0.17789    
## RestingECGNormal -0.625467   0.239922  -2.607  0.00914 ** 
## RestingECGST      -0.462013   0.315937  -1.462  0.14364    
## MaxHR             -0.030403   0.004630  -6.567 5.14e-11 ***
## ---
## Signif. codes:  0 ’***’ 0.001 ’**’ 0.01 ’*’ 0.05 ’.’ 0.1 ’ ’ 1
## 
## (Dispersion parameter for binomial family taken to be 1)
## 
##     Null deviance: 822.98  on 595  degrees of freedom
## Residual deviance: 648.09  on 587  degrees of freedom
## AIC: 666.09
## 
## Number of Fisher Scoring iterations: 4
```

```
tidy(glm_fit)
```

```
## # A tibble: 9 x 5
##   term             estimate std.error statistic  p.value
##   <chr>               <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept)      -1.18      1.34       -0.883 3.77e- 1
## 2 Age               0.0336    0.0119      2.84  4.58e- 3
## 3 SexM              1.63      0.250       6.51  7.41e-11
## 4 RestingBP         0.00757   0.00596     1.27  2.04e- 1
## 5 Cholesterol       0.00676   0.00176     3.85  1.20e- 4
## 6 FastingBS1        0.365     0.271       1.35  1.78e- 1
## 7 RestingECGNormal -0.625     0.240      -2.61  9.14e- 3
## 8 RestingECGST     -0.462     0.316      -1.46  1.44e- 1
## 9 MaxHR            -0.0304    0.00463    -6.57  5.14e-11
```

```
glance(glm_fit)
```

```
## # A tibble: 1 x 8
##   null.deviance df.null logLik   AIC   BIC deviance df.residual  nobs
##           <dbl>   <int>  <dbl> <dbl> <dbl>    <dbl>       <int> <int>
## 1          823.     595  -324.  666.  706.     648.         587   596
```

Preprocess our data with recipes

The `resample` package will be used to evaluate the model. Ten cross-validations, each of 8 *folds*, will be used, therefore 80 samples will be obtained to evaluate the accuracy of the model.

```
set.seed(1234)
folds_with_repeats <- rsample::vfold_cv(data = data_train, v = 8, repeats = 10)
print(folds_with_repeats)
```

```
## #  8-fold cross-validation repeated 10 times
## # A tibble: 80 x 3
##    splits           id       id2
##    <list>           <chr>    <chr>
##  1 <split [521/75]> Repeat01 Fold1
##  2 <split [521/75]> Repeat01 Fold2
##  3 <split [521/75]> Repeat01 Fold3
##  4 <split [521/75]> Repeat01 Fold4
##  5 <split [522/74]> Repeat01 Fold5
##  6 <split [522/74]> Repeat01 Fold6
##  7 <split [522/74]> Repeat01 Fold7
##  8 <split [522/74]> Repeat01 Fold8
##  9 <split [521/75]> Repeat02 Fold1
## 10 <split [521/75]> Repeat02 Fold2
## # ... with 70 more rows
```

The command `rsample::vfold_cv()` generates the 80 subsets of equal size, varying the seed of the subdivision. The result generates an object with three variables: the repetition (`id`), the fold (`id2`) and splits. `splits` is a variable that stores a list of size (*v*repeats*)*3, where *v* is the number of folds and *repeat* the number of repetitions. Each element of the list is an object of the tibble class with several variables:

- `data`: matrix of dimension n*p, where n is the number of records and p the number of variables.
- `in_id`: logical index of the position of the records in the data that belong to the records being analysed. This set of records is called analisys, and the one left out is called assessment.
- `id`: stores the id of the fold and of the repetition to which the data corresponds.

Prediction

Next, to assess the accuracy of the model for each of the samples generated with the cross-validation we will create a function that will mainly do the following:

1. A logistic regression and model fit for the *analysis* data.
2. Prediction on the *assessment* data using one of the most prominent packages of the `tidymodels` package, `broom`.
3. Checking whether the prediction was performed correctly.

```
res_leftout <- function(samplecv, model) {
  # Fit the model
  glm_model <- glm(model, data = analysis(samplecv), family = binomial)

  # Identify the dataset left out
  holdout <- assessment(samplecv)

  # Perfoms prediction on the data set hold out using augment()
  res <- broom::augment(glm_model, newdata = holdout)

  # lvls will be the levels of the factor with the predictions
  # the prediction (res$.fitted) is transformed into a nominal variable
  # depending on whether it is greater or less than zero
  # If greater than zero, then No (no heart attack), otherwise Yes.
  lvls <- levels(holdout$HeartDisease)
  predictions <- factor(ifelse(res$.fitted > 0, lvls[2], lvls[1]), levels = lvls)

  # We check if the prediction is correct
  res$correct <- predictions == holdout$HeartDisease

  # Return the dataset with the additional columns
  res
}
```

For example, if we implement this procedure for the first of the 80 samples generated we obtain:

```
firstsample <- res_leftout(folds_with_repeats$splits[[1]], glm_fit_formula)
dim(firstsample)
```

```
## [1] 75 11
```

```
dim(rsample::assessment(folds_with_repeats$splits[[1]]))
```

```
## [1] 75  8
```

Therefore, the name of the columns added for the first of the samples is:

```
print(firstsample[1:7, setdiff("correct", names("HeartDisease"))])
```

```
## # A tibble: 7 x 1
##   correct
##   <lgl>
## 1 TRUE
## 2 TRUE
## 3 TRUE
## 4 TRUE
## 5 TRUE
## 6 FALSE
## 7 FALSE
```

As we can see, half of the predictions in the first sample are incorrect. For this particular model, `.fitted` refers to the linear predictor of the *log-odds*.

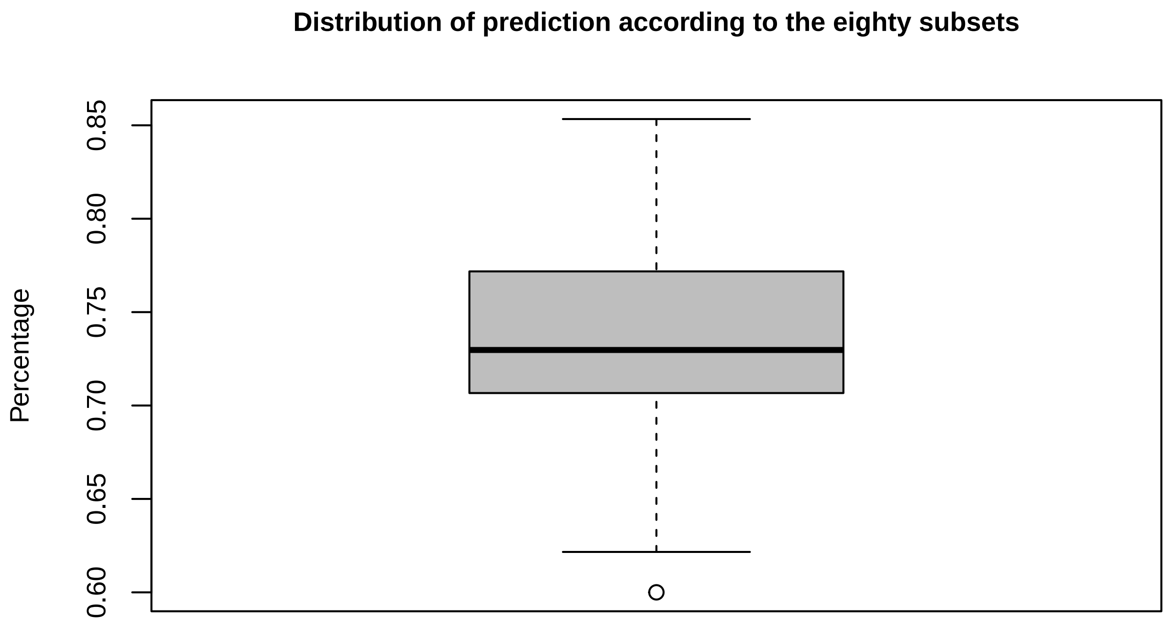
To do the above with the rest of the 79 samples obtained, we will apply the `purrr::map()` function, which applies the requested on each list:

```
folds_with_repeats$results <- purrr::map(folds_with_repeats$splits, res_leftout,
                                         glm_fit_formula)
print(folds_with_repeats)
```

```
## #  8-fold cross-validation repeated 10 times
## # A tibble: 80 x 4
##    splits           id       id2   results
##    <list>           <chr>    <chr> <list>
##  1 <split [521/75]> Repeat01 Fold1 <tibble [75 x 11]>
##  2 <split [521/75]> Repeat01 Fold2 <tibble [75 x 11]>
##  3 <split [521/75]> Repeat01 Fold3 <tibble [75 x 11]>
##  4 <split [521/75]> Repeat01 Fold4 <tibble [75 x 11]>
##  5 <split [522/74]> Repeat01 Fold5 <tibble [74 x 11]>
##  6 <split [522/74]> Repeat01 Fold6 <tibble [74 x 11]>
##  7 <split [522/74]> Repeat01 Fold7 <tibble [74 x 11]>
##  8 <split [522/74]> Repeat01 Fold8 <tibble [74 x 11]>
##  9 <split [521/75]> Repeat02 Fold1 <tibble [75 x 11]>
## 10 <split [521/75]> Repeat02 Fold2 <tibble [75 x 11]>
## # ... with 70 more rows
```

We can now calculate the metric for all *assessment* datasets. The percentage of heart diseases with a correct prediction is calculated:

```
folds_with_repeats$results <- map_dbl(folds_with_repeats$results, function(x) {mean(x$correct)})
boxplot(folds_with_repeats$results,
        main = 'Distribution of prediction according to the eighty subsets',
        col = 'grey', ylab = 'Percentage', cex.main = 0.8, cex.lab = 0.8, cex.axis = 0.8)
```



The accuracy to be exceeded from this baseline is 0.73. Let's see if we can overcome it by applying a classification method such as k-means.

Implementation of k-means algorithm

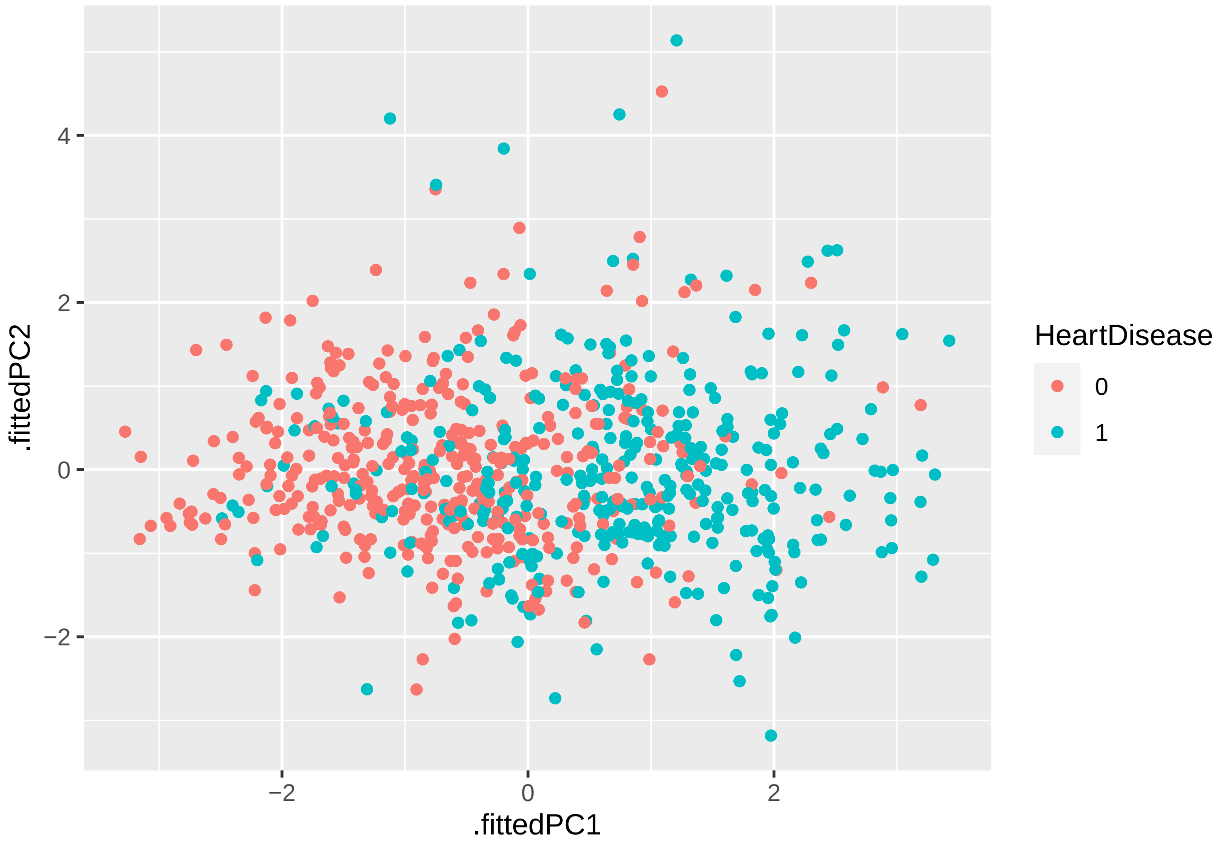
We attempt to implement a new method in order to achieve a further analysis of the data. Before that, we apply PCA and select the principal components which explain most of the variability of the dataset. To apply both techniques, it is compulsory to select the numeric attributes and highly recommendable to scale them.

```
# PCA components
pca_comp <- heart.data %>%
select(where(is.numeric)) %>%
prcomp(scale. = TRUE)
tidy(pca_comp)
```

```
## # A tibble: 3,730 x 3
##      row    PC   value
##    <int> <dbl>   <dbl>
##  1     1     1 -1.56
##  2     1     2  1.40
##  3     1     3 -0.654
##  4     1     4  0.0201
##  5     1     5 -0.347
##  6     2     1 -0.0289
##  7     2     2 -0.255
##  8     2     3 -2.00
##  9     2     4  0.162
## 10     2     5 -0.343
## # ... with 3,720 more rows
```

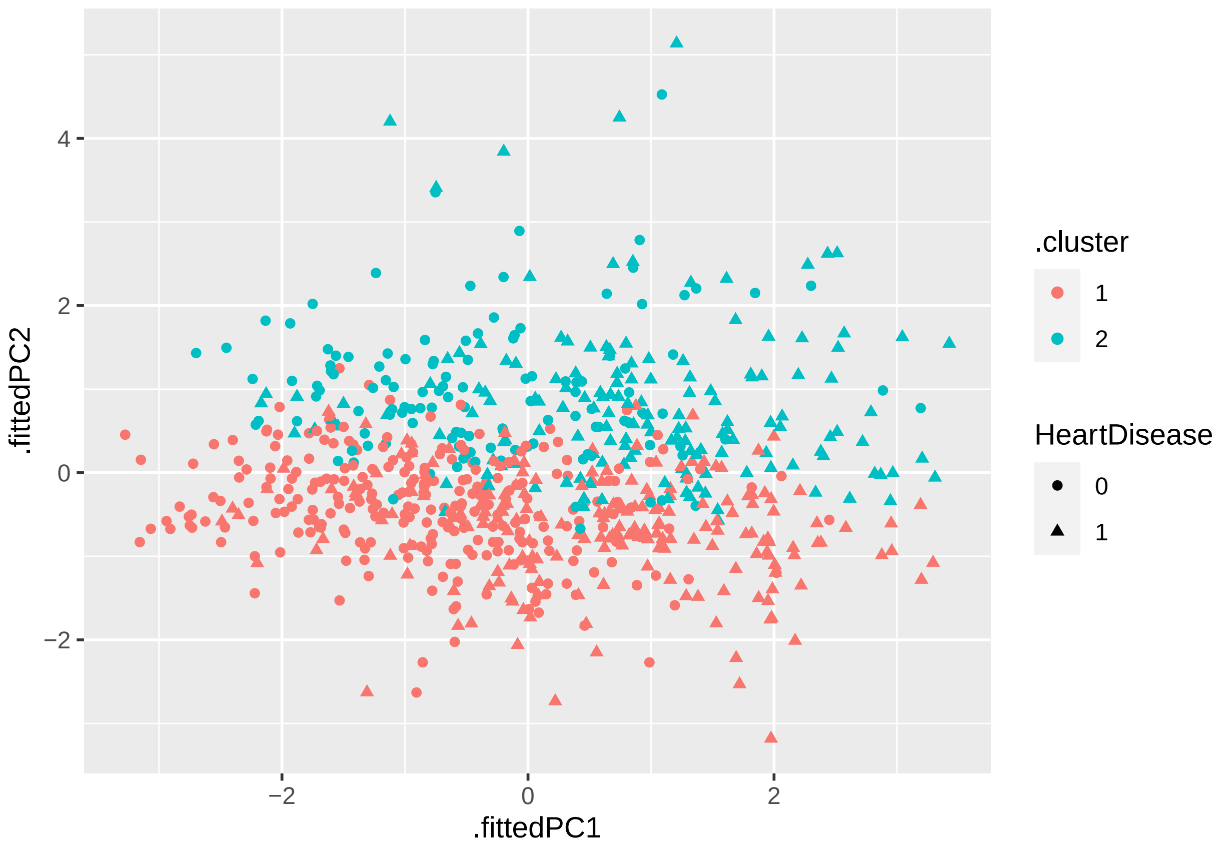
First, we plot the first two PCA components and visualize if the classes form differentiated clusters. By using the `augment()` function, we can complement the results as the PCA technique as the *k-means* algorithm. It is very useful in order to plot the categorization *k-means* in terms of the first two PCA components.

```
aug_pca <- augment(pca_comp, data = heart.data)
ggplot(data = aug_pca, aes(x = .fittedPC1, y = .fittedPC2)) +
geom_point(aes(col = HeartDisease))
```



We cannot distinguish two groups in the data. We apply *k-means* method and again visualize through the first two PCA components the division in clusters which has carried out the algorithm.

```
k_m <- heart.data %>%
select(where(is.numeric)) %>%
kmeans(centers = 2)
aug_k_m <- augment(k_m, data = heart.data)
aug_k_m <- cbind(aug_k_m, aug_pca[".fittedPC1"], aug_pca[".fittedPC2"])
ggplot(data = aug_k_m, mapping = aes(x = .fittedPC1, y = .fittedPC2)) +
geom_point(aes(col = .cluster, pch = HeartDisease))
```



It seems that now there is less overlapping, if we look at the cluster *k-means* division. It suggests that cluster *k-means* division may distinguish two different population in a better way than the classes do it.