

3 Selecting Best Variables

3.1 General Aspects in Selecting Best Variables

3.1.1 What is this about?

This chapter covers the following topics:

- The best variables ranking from conventional machine learning algorithms, either predictive or clustering.
- The nature of selecting variables with and without predictive models.
- The effect of variables working in groups (intuition and information theory).
- Exploring the best variable subset in practice using R.

Selecting the best variables is also known as feature selection, selecting the most important predictors, selecting the best predictors, among others.

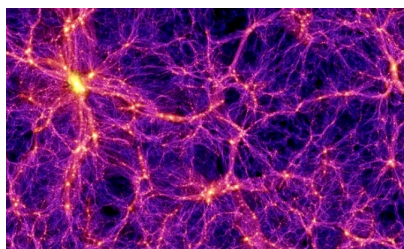


Figure 3.1: As above, so below

Image: Is it a neural network? Nope. Dark matter, from the “The Millennium Simulation Project”.

3.2 Intuition

Selecting the best variables is like doing a summary of a story, we want to focus on those few details that best describe what we’re talking about. The balance threads between talking *too much* about unnecessary details (overfitting) and talking *too little* about the essence of the story (underfitting).

Another example can be the decision of buying a new laptop: *what are the features that we care about the most? Price, color and shipping method? Color and battery life? Or just price?*

From the **Information Theory** point of view -a key point in machine learning-, the data that we are working on has **entropy** (chaos). When we select variables, we are reducing the entropy of our system by adding information.

3.3 The “best” selection?

The chapter says “best”, but we’d better mention a conceptual point, in general terms *there is no unique best variable selection*.

To start from this perspective is important, since in the exploration of many algorithms that *rank* the variables according to their predictive power, we can find different -and similar- results. That is:

- Algorithm 1 has chosen as the best variable `var_1` , followed by `var_5` and `var_14` .
- Algorithm 2 did this ranking: `var_1` , `var_5` and `var_3` .

Let’s imagine, based on algorithm 1, the accuracy is 80%, while the accuracy based on algorithm 2 is 78%. Considering that every model has its inner variance, the result can be seen as the same.

This perspective can help us to reduce time in pursuing the perfect variable selection.

However going to the extremes, there will be a set of variables that will rank high across many algorithms, and the same goes for those with little predictive power. After several runs most reliable variables will emerge quickly, so:

Conclusion: If results are not good the focus should be on improving and checking the **data preparation** step. *The next section will exemplify it.*

3.3.1 Going deeper into variable ranking

It’s quite common to find in literature and algorithms, that covers this topic an univariate analysis, which is a ranking of variables given a particular metric.

We're going to create two models: random forest and gradient boosting machine (GBM) using `caret` R package to cross-validate the data. Next, we'll compare the best variable ranking that every model returns.

```
library(caret)
library(funModeling)
library(dplyr)

# Excluding all NA rows from the data, in this case, NAs are not the main issue to solve
heart_disease=na.omit(heart_disease)

# Setting a 4-fold cross-validation
fitControl = trainControl(method = "cv",
                           number = 4,
                           classProbs = TRUE,
                           summaryFunction = twoClassSummary)

# Creating the random forest model, finding the best tuning parameter set
set.seed(999)
fit_rf = train(x=select(heart_disease, -has_heart_disease, -heart_disease_severity),
               y = heart_disease$has_heart_disease,
               method = "rf",
               trControl = fitControl,
               verbose = FALSE,
               metric = "ROC")

# Creating the gradient boosting machine model, finding the best tuning parameter set
fit_gbm = train(x=select(heart_disease, -has_heart_disease, -heart_disease_severity),
                y = heart_disease$has_heart_disease,
                method = "gbm",
                trControl = fitControl,
                verbose = FALSE,
                metric = "ROC")
```

Now we can proceed with the comparison.

The columns `importance_rf` and `importance_gbm` represent the importance measured by each algorithm. Based on each metric, there are `rank_rf` and `rank_gbm` which represent the importance order, finally `rank_diff` (`rank_rf - rank_gbm`) represents how different each algorithm rank the variables.

```
# Here we manipulate to show a nice the table described before
var_imp_rf=data.frame(varImp(fit_rf, scale=T)["importance"]) %>%
  dplyr::mutate(variable=rownames(.)) %>% dplyr::rename(importance_rf=Overall) %>%
  dplyr::arrange(-importance_rf) %>%
  dplyr::mutate(rank_rf=seq(1:nrow(.)))

var_imp_gbm=as.data.frame(varImp(fit_gbm, scale=T)["importance"]) %>%
  dplyr::mutate(variable=rownames(.)) %>% dplyr::rename(importance_gbm=Overall) %>%
  dplyr::arrange(-importance_gbm) %>%
  dplyr::mutate(rank_gbm=seq(1:nrow(.)))

final_res=merge(var_imp_rf, var_imp_gbm, by="variable")

final_res$rank_diff=final_res$rank_rf-final_res$rank_gbm

# Printing the results!
final_res
```

variable	importance_rf	rank_rf	importance_gbm	rank_gbm	rank_diff
exter_angina	29.15468	11	11.544326	6	5
slope	37.44874	10	20.799135	5	5
gender	21.72545	12	5.557935	9	3
chest_pain	97.85192	3	100.000000	1	2
fasting_blood_sugar	0.00000	14	0.000000	13	1
oldpeak	89.31555	5	51.556053	4	1
resting_blood_pressure	57.47780	8	5.690607	8	0
thal	98.15852	2	91.395284	2	0
exer_angina	37.68362	9	5.106130	10	-1
resting_electro	9.09967	13	0.000000	14	-1
num_vessels_flour	100.00000	1	88.478330	3	-2
max_heart_rate	96.89898	4	6.697837	7	-3
serum_cholesterol	62.03931	7	3.344057	11	-4
age	76.10440	6	3.140659	12	-6

Figure 3.2: Comparing different variable ranking

We can see that there are variables which are not important at all to both models (`fasting_blood_sugar`). There are others that maintain a position at the top of importance like `chest_pain` and `thal` .

Different predictive model implementations have their criteria to report what are the best features, according to that particular model. This ends up in different ranking across different algorithms. *More info about the inner importance metrics can be found at the [caret documentation](#).*

Even more, in tree based models like GBM and Random Forest there is a random component to picking up variables, and the importance is based on prior -and automatic- variable selection when building the trees. The importance of each variable depends on the others, not only on its isolated contribution: **Variables work in groups**. We'll back on this later on this chapter.

Although the ranking will vary from algorithm to algorithm, in general terms there is a correlation between all of these results as we mentioned before.

Conclusion: Every ranking list is not the “*final truth*”, it gives us orientation about where the information is.

3.4 The nature of the selection

There are two main approaches when doing variable selection:

Predictive model dependent:

Like the ones we saw before, this is the most common. The model will rank variables according to one intrinsic measure of accuracy. In tree-based models, metrics such as information gain, Gini index, node impurity. More info at ([stackoverflow.com 2017](#)) and ([stats.stackexchange.com 2017a](#)).

Not predictive model dependent:

This is interesting since they are not as popular as the other ones, but they are proved to perform really well in areas related to genomic data. They need to find those *relevant* genes (input variable) that are correlated with some disease, like cancer (target variable).

Data from this area is characterized by having a huge number of variables (in the order of thousands), which is much bigger than problems in other areas.

One algorithm to perform this is [mRMR](#), acronym for Minimum Redundancy Maximum Relevance Feature Selection. It has its own implementation in R in [mRMRe](#) package.

Another not model-dependant algorithm is `var_rank_info`, a function provided by [funModeling](#) package. It ranks the variables according to several **information theory** metrics. An example will be presented later on.

3.5 Improving variables

Variables can increase their predictive power by treating them.

This book covers by now:

- [Improvement of categorical variables](#).
- Reducing the noise in numerical variables through binning in the chapter: [Discretizing numerical variables](#).
- [How to deal with outliers in R](#).
- [Missing Data: Analysis, Handling, and Imputation of](#)

3.6 Cleaning by domain knowledge

It's not related to algorithmic procedures, but to the area from which the data comes.

Imagine data coming from a survey. This survey has one year of history, and during the first three months there was no good process control. When inserting data users could type whatever they wanted. Variables during this period will probably be spurious.

It's easy to recognize it when during a given period of time, the variable comes empty, null or with extreme values.

We should then ask a question:

Is this data reliable? Keep in mind the predictive model will learn *as a kid*, it will not judge the data, just learn from it. If data is spurious in a given period of time, then we may remove these input cases.

To go further on this point, we should do a deeper exploratory data analysis. Both numerically and graphically.

3.7 Variables work in groups



Figure 3.3: Variables work in groups

When selecting the *best* variables, the main aim is to get those variables which carry the most information regarding a target, outcome or dependent variable.

A predictive model will find its weights or parameters based on its 1 to 'N' input variables.

Variables usually don't work isolated when explaining an event. Quoting Aristotle:

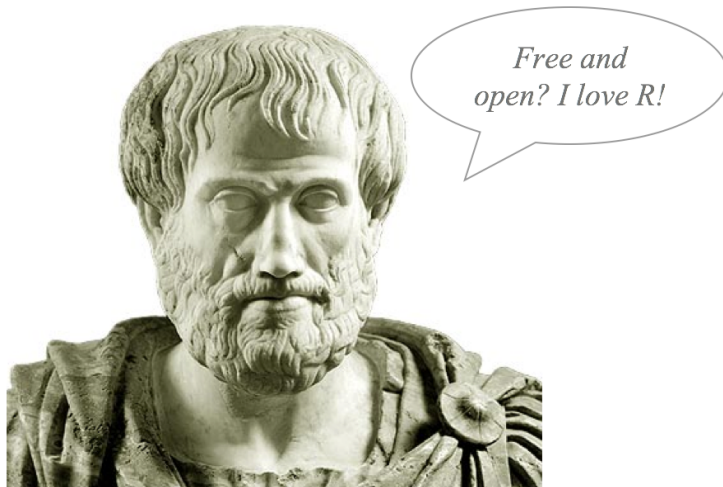
“The whole is greater than the sum of its parts.”

This is also true when selecting the *best* features:

Building a predictive model with two variables may reach a higher accuracy than the models built with only one variable.

For example: Building a model based on variable `var_1` could lead to an overall accuracy of 60%. On the other hand, building a model based on `var_2` could reach an accuracy of 72%. But when we combine these two `var_1` and `var_2` variables, we could achieve an accuracy above 80%.

3.7.1 Example in R: Variables working in groups



Aristotle (384 - 322 BC).
Philosopher and Data Scientist.

Figure 3.4: Aristotle (384 BC322 BC)

The following code illustrates what Aristotle said *some* years ago.

It creates 3 models based on different variable subsets:

- model 1 is based on `max_heart_rate` input variable
- model 2 is based on `chest_pain` input variable
- model 3 is based on `max_heart_rate` **and** `chest_pain` input variables

Each model returns the metric ROC, and the result contains the improvement of considering the two variables at the same time vs. taking each variable isolated.


```

library(caret)
library(funModeling)
library(dplyr)

# setting cross-validation 4-fold
fitControl =
  trainControl(method = "cv",
               number = 4,
               classProbs = TRUE,
               summaryFunction = twoClassSummary
               )

create_model<-function(input_variables)
{
  # create gradient boosting machine model
  # based on input variables
  fit_model = train(x=select(heart_disease,
                             one_of(input_variables)
                             ),
                    y = heart_disease$has_heart_disease,
                    method = "gbm",
                    trControl = fitControl,
                    verbose = FALSE,
                    metric = "ROC")

  # returning the ROC as the performance metric
  max_roc_value=max(fit_model$results$ROC)
  return(max_roc_value)
}

roc_1=create_model("max_heart_rate")
roc_2=create_model("chest_pain")
roc_3=create_model(c("max_heart_rate", "chest_pain"))

avg_improvement=round(100*(((roc_3-roc_1)/roc_1)+
                          ((roc_3-roc_2)/roc_2))/2,
                      2)

avg_improvement_text=sprintf("Average improvement: %s%%",

```

avg_improvement)

```

results =

    sprintf("ROC model based on 'max_heart_rate': %s.;
    based on 'chest_pain': %s; and based on both: %s",
    round(roc_1,2),
    round(roc_2,2),
    round(roc_3, 2)
    )

# printing the results!
cat(c(results, avg_improvement_text), sep="\n\n")

## ROC model based on 'max_heart_rate': 0.71.;
##   based on 'chest_pain': 0.75; and based on both: 0.81
##
## Average improvement: 11.67%

```

3.7.2 Tiny example (based on Information Theory)

Consider the following *big data* table (4 rows, 2 input variables (var_1 , var_2) and one outcome (target)):

var_1	var_2	target
a	x	red
a	z	blue
b	x	blue
b	z	red

Figure 3.5: Unity gives strength: Combining variables

If we build a predictive model based on var_1 only, what it will see?, the value a is correlated with output blue and red in the same proportion (50%):

- If var_1='a' then likelihood of target='red' is 50% (row 1)
- If var_1='b' then likelihood of target='blue' is 50% (row 2)

Same analysis goes for var_2

When the same input is related to different outcomes it's defined as **noise**. The intuition is the same as one person telling us: *"Hey it's going to rain tomorrow!"*, and another one saying: *"For sure tomorrow it's not going to rain"*. We'd think... *"OMG! do I need the umbrella or not 🤔?"*

Going back to the example, taking the two variables at the same time, the correspondence between the input and the output is unique: "If `var_1='a'` and `var_2='x'` then the likelihood of being `target='red'` is 100%". You can try other combinations.

Summing-up:

That was an example of **variables working in groups**, considering `var_1` and `var_2` at the same time increases the predictive power.

Nonetheless, it's a deeper topic to cover, considering the last analysis; how about taking an `id` column (every value is unique) to predict something? The correspondence between input-output will also be unique... but is it a useful model? There'll be more to come about information theory in this book.

3.7.3 Conclusions

- The proposed R example based on `heart_disease` data shows an average **improvement of 9%** when considering two variables at a time, not too bad. This percentage of improvement is the result of the **variables working in groups**.
- This effect appears if the variables contain information, such is the case of `max_heart_rate` and `chest_pain` (or `var_1` and `var_2`).
- Putting **noisy variables** next to good variables **will usually affect** overall performance.
- Also the **work in groups** effect is higher if the input variables **are not correlated between** them. This is difficult to optimize in practice. More on this on the next section...

3.7.4 Rank best features using information theory

As introduced at the beginning of the chapter, we can get variable importance without using a predictive model using information theory.

From version 1.6.6 the package `funModeling` introduces the function `var_rank_info`, which takes two arguments, the data and the target variable, because it follows:

```

variable_importance =
    var_rank_info(heart_disease, "has_heart_disease")

# Printing results
variable_importance

##           var      en      mi           ig      gr
## 1 heart_disease_severity 1.846 0.995 0.9950837595 0.5390655068
## 2                thal 2.032 0.209 0.2094550580 0.1680456709
## 3          exer_angina 1.767 0.139 0.1391389302 0.1526393841
## 4          exter_angina 1.767 0.139 0.1391389302 0.1526393841
## 5          chest_pain 2.527 0.205 0.2050188327 0.1180286190
## 6    num_vessels_flour 2.381 0.182 0.1815217813 0.1157736478
## 7                slope 2.177 0.112 0.1124219069 0.0868799615
## 8    serum_cholesterol 7.481 0.561 0.5605556771 0.0795557228
## 9                gender 1.842 0.057 0.0572537665 0.0632970555
## 10             oldpeak 4.874 0.249 0.2491668741 0.0603576874
## 11      max_heart_rate 6.832 0.334 0.3336174096 0.0540697329
## 12 resting_blood_pressure 5.567 0.143 0.1425548155 0.0302394591
## 13                age 5.928 0.137 0.1371752885 0.0270548944
## 14      resting_electro 2.059 0.024 0.0241482908 0.0221938072
## 15    fasting_blood_sugar 1.601 0.000 0.0004593775 0.0007579095

# Plotting
ggplot(variable_importance,
        aes(x = reorder(var, gr),
            y = gr, fill = var)
        ) +
    geom_bar(stat = "identity") +
    coord_flip() +
    theme_bw() +
    xlab("") +
    ylab("Variable Importance
        (based on Information Gain)"
        ) +
    guides(fill = FALSE)

```

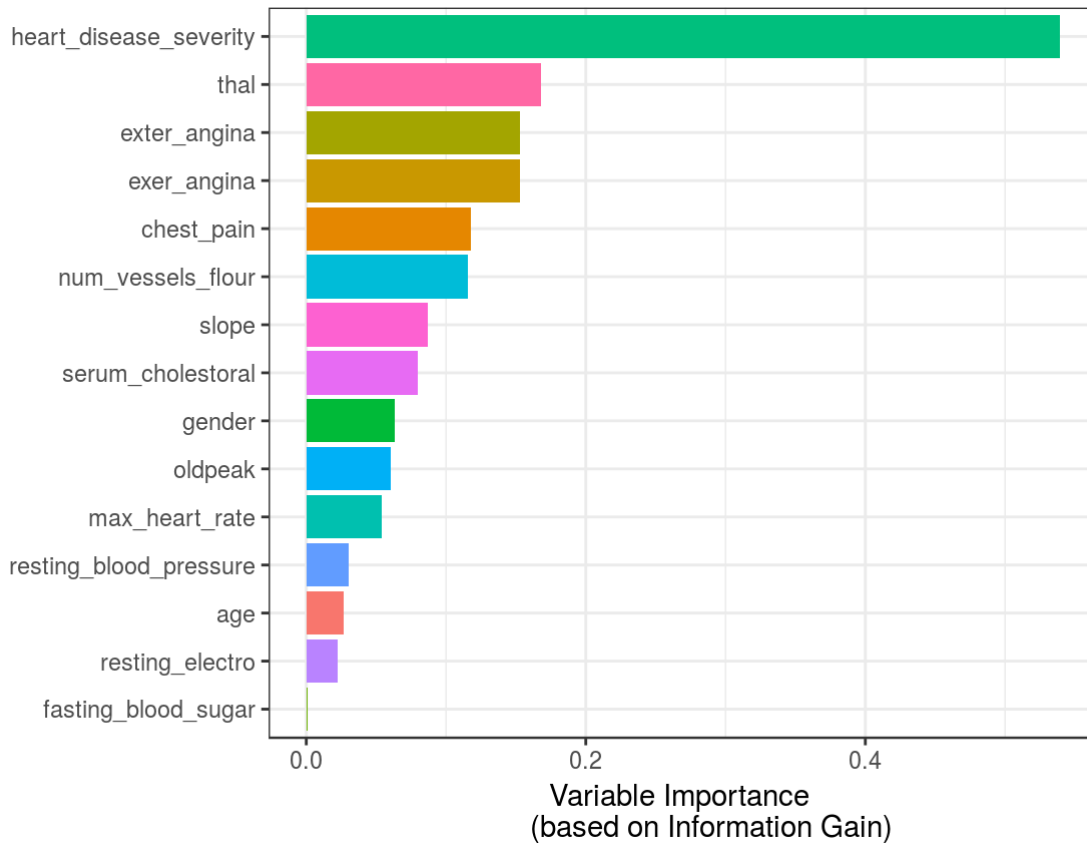


Figure 3.6: Variable Importance (based on Gain Ratio)

Is `heart_disease_severity` the feature that explains the target the most?

No, this variable was used to generate the target, thus we must exclude it. It is a typical mistake when developing a predictive model to have either an input variable that was built in the same way as the target (as in this case) or adding variables from the future as explained in [Considerations involving time](#).

Going back to the result of `var_rank_info`, the resulting metrics come from information theory:

- `en` : entropy measured in bits
- `mi` : mutual information
- `ig` : information gain
- `gr` : gain ratio

We are not going to cover what is behind these metrics at this point as this will be covered exclusively in a future chapter, However, the `gain ratio` is the most important metric here, ranged from 0 to 1, with higher being better.

Fuzzy boundaries

We've just seen how to calculate importance based on information theory metrics. This topic is not exclusive to this chapter; this concept is also present in the [Exploratory Data Analysis - Correlation and Relationship](#) section.

To *select best features* is related to *exploratory data analysis* and vice-versa.

3.8 Correlation between input variables

The ideal scenario is to build a predictive model with only variables not correlated between them. In practice, it's complicated to keep such a scenario for all variables.

Usually there will be a set of variables that are not correlated between them, but also there will be others that have at least some correlation.

In practice a suitable solution would be to exclude those variables with a **remarkably high-level** of correlation.

Regarding how to measure correlation. Results can be highly different based on linear or non-linear procedures. More info at the [Correlation](#)

What is the problem with adding correlated variables?

The problem is that we're adding complexity to the model: it's usually more time-consuming, harder to understand, less explainable, less accurate, etc. This is an effect we reviewed in [Don't predictive models handle high cardinality?](#). The general rule would be: Try to add the top N variables that are correlated with the output but not correlated between them. This leads us to the next section.

3.9 Keep it simple



Figure 3.7: Fractals in Nature

Nature operates in the shortest way possible. -Aristotle.

The principle of **Occam's razor**: Among competing hypotheses, the one with the fewest assumptions should be selected.

Re-interpreting this sentence for machine learning, those “hypotheses” can be seen as variables, so we've got:

Among different predictive models, the one with fewest variables should be selected.

(Wikipedia 2017c)

Of course, there is also the trade-off of adding-subtracting variables and the accuracy of the model.

A predictive model with a *high* number of variables will tend to do **overfitting**. While on the other hand, a model with a *low* number of variables will lead to doing **underfitting**.

The concept of *high* and *low* is **highly subjective** to the data that is being analyzed. In practice, we may have some accuracy metric, for example, the ROC value. i.e. we would see something like:

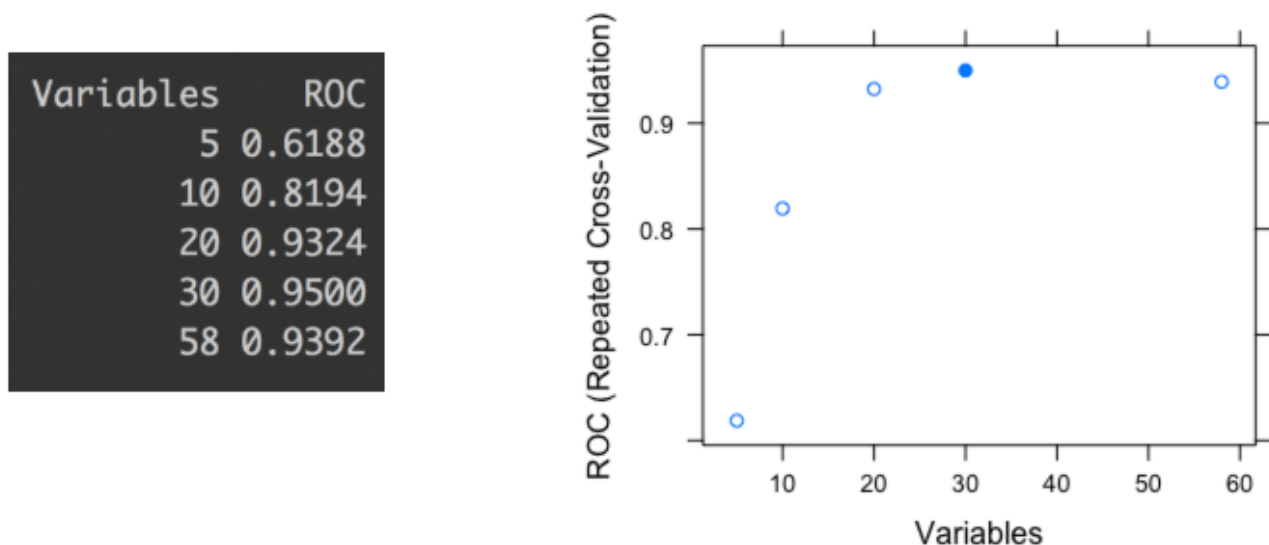


Figure 3.8: ROC values for different variables subset

The last picture shows the ROC accuracy metric given different subsets of variables (5, 10, 20, 30 and 58). Each dot represents the ROC value given a certain number of variables used to build the model.

We can check that the highest ROC appears when the model is built with 30 variables. If we based the selection only on an automated process, we might be choosing a subset which tends to overfit the data. This report was produced by library `caret` in R ((Kuhn 2017) but is analogous to any software.

Take a closer look at the difference between the subset of 20 and the 30; there is only an improvement of **1.8%** -from 0.9324 to 0.95- choosing **10 more variables**. In other words: *Choosing 50% more variables will impact in less than 2% of improvement.*

Even more, this 2% may be an error margin given the variance in the prediction that every predictive model has, as we seen in [Knowing the error](#) chapter.

Conclusion:

In this case, and being consequent with Occam's Razor principle, the best solution is to build the model with the subset of 20 variables.

Explaining to others -and understanding- a model with 20 variables is easier than the similar one with 30.

3.10 Variable selection in Clustering?

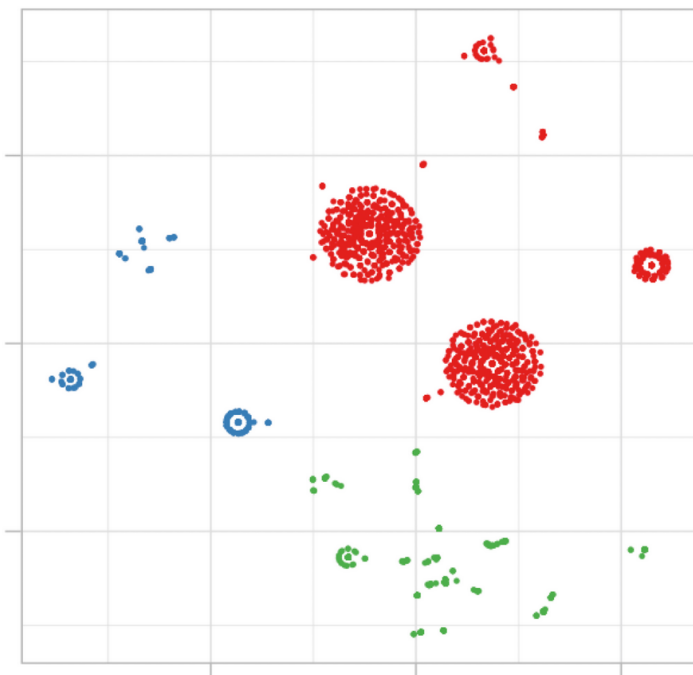


Figure 3.9: Example of cluster segmentation

This concept usually appears only in predictive modeling, i.e. having some variables to predict a target one. In clustering there is no target variable, we let the data speak, and the natural segments arise according to some distance metric.

However, **not every variable contributes in the same way to the dissimilarity in the cluster model**. Keeping it brief, if we have 3 clusters as output, and we measure the average of each variable, we expect to have these averages *quite* dissimilar between them, right?

Having built 2 cluster models, in the first one the averages of the `age` variable is 24, 33 and 26 years; while on the second one we have: 23, 31 and 46. In the second model the variable `age` is having more variability, thus it is more relevant to the model.

This was just an example considering two models, but it's the same considering just one. Those variables with **more distance** across averages will tend to **define better** the cluster than the others.

Unlike predictive modeling, in clustering *less important* variables shouldn't be removed, those variables aren't important in that particular model, but they could be if we build another one with other parameters. The cluster models' quality is highly subjective.

Finally, we could run, for example, a random forest model with the cluster as a target variable and in this way quickly collect the most important variables.

3.11 Selecting the best variables in practice

3.11.1 The short answer

Pick up the top N variables from the algorithm you're using and then re-build the model with this subset. Not every predictive model retrieves variable rankings, but if it does, use the same model (for example gradient boosting machine) to get the ranking and to build the final model.

For those models like k-nearest neighbors which don't have a built-in select best features procedure, it's valid to use the selection of another algorithm. It will lead to better results than using all the variables.

3.11.2 The long answer

- When possible, **validate** the list with someone who knows about the context, the business or the data source. Either for the top N or the bottom M variables. As regards those *bad*

variables, we may be missing something in the data munging that could be destroying their predictive power.

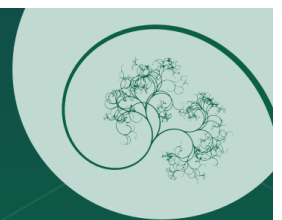
- Understand each variable, its meaning in context (business, medical, other).
- Do **exploratory data analysis** to see the distributions of the most important variables regarding a target variable, *does the selection make sense?* If the target is binary then the function [Profiling target using cross_plot](#) can be used.
- Does the average of any variable *significantly* change over time? Check for abrupt changes in distributions.
- Suspect about high cardinality top-ranked variables (like postal code, let's say above +100 categories). More information at [High Cardinality Variable in Predictive Modeling](#).
- When making the selection -as well as a predictive modeling-, try and use methods which contain some mechanism of re-sampling (like bootstrapping), and cross-validation. More information in the [knowing the error](#) chapter.
- Try other methods to find **groups of variables**, like the one mentioned before: mRMR.
- If the selection doesn't meet the needs, try creating new variables, you can check the **data preparation** chapter. Coming soon: Feature engineering chapter.

3.11.3 Generate your own knowledge

It's difficult to generalize when the nature of the data is so different, from **genetics** in which there are thousands of variables and a few rows, to web-navigation when new data is coming all the time.

The same applies to the objective of the analysis. Is it to be used in a competition where precision is highly necessary? Perhaps the solution may include more correlated variables in comparison to an ad-hoc study in which the primary goal is a simple explanation.

There is no one-size-fits-all answer to face all possible challenges; you'll find powerful insights using your experience. It's just a matter of practice.



3.12 Target profiling

3.12.1 Using `cross_plot` (dataViz)

3.12.1.1 What is this about?

This plot intent to show in real scenarios if a variable is or not important, making a visual summary of it, *(by grouping numerical variables into bins/groups)*.

3.12.1.2 Example 1: Is gender correlated with heart disease?

```
cross_plot(heart_disease, input="gender", target="has_heart_disease")
```

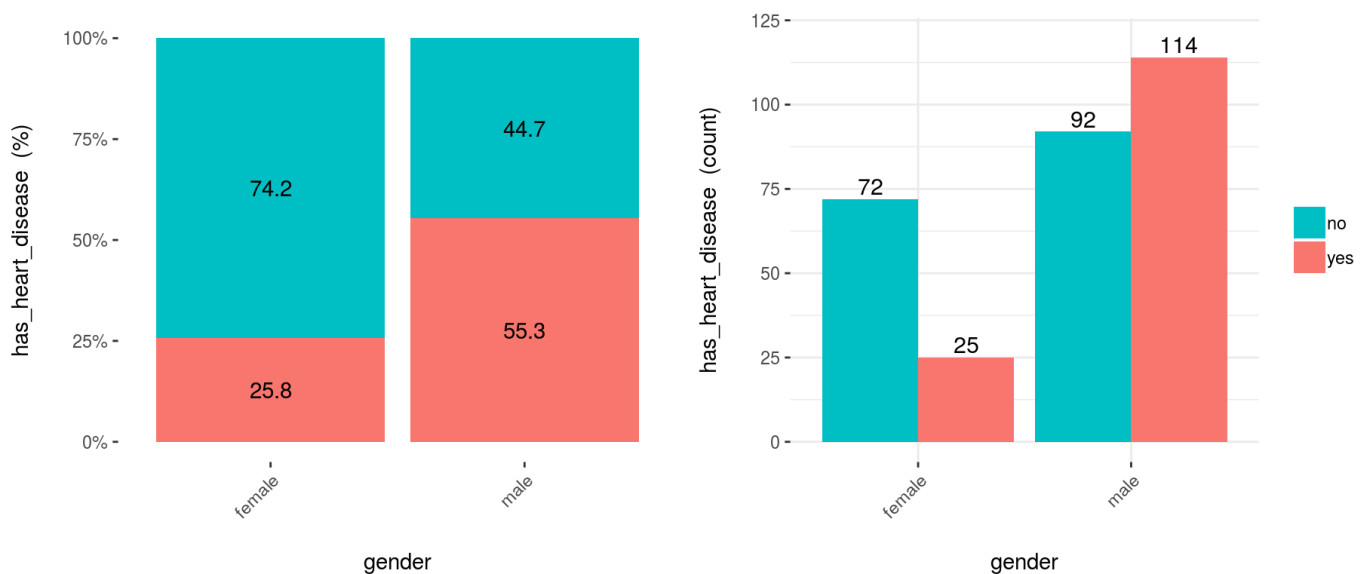


Figure 3.10: Using cross-plot to analyze and report variable importance

Last two plots have the same data source, showing the distribution of `has_heart_disease` regarding `gender`. The one on the left shows in percentage value, while the one on the right shows in absolute value.

3.12.1.2.1 How to extract conclusions from the plots? (Short version)

`Gender` variable seems to be a **good predictor**, since the likelihood of having heart disease is different given the female/male groups. **it gives an order to the data.**

3.12.1.3 How to extract conclusions from the plots? (Long version)

From 1st plot (%):

1. The **likelihood** of having heart disease for males is 55.3%, while for females is: 25.8%.
2. The heart disease rate for males **doubles** the rate for females (55.3 vs. 25.8, respectively).

From 2nd plot (count):

1. There is a total of **97 females**:
 - 25 of them have heart disease ($25/97=25.8\%$, which is the ratio of the 1st plot).
 - the remaining 72 have not heart disease (74.2%)
2. There is a total of **206 males**:
 - 114 of them have heart disease (55.3%)
 - the remaining 92 have not heart disease (44.7%)
3. Total cases: Summing the values of four bars: $25+72+114+92=303$.

Note: What would it have happened if instead of having the rates of 25.8% vs. 55.3% (female vs. male), they had been more similar like 30.2% vs. 30.6%). In this case variable `gender` it would have been much less relevant, since it doesn't separate the `has_heart_disease` event.

3.12.1.4 Example 2: Crossing with numerical variables

Numerical variables should be **binned** to plot them with a histogram, otherwise, the plot is not showing information, as it can be seen here:

3.12.1.4.1 Equal frequency binning

There is a function included in the package (inherited from Hmisc package): `equal_freq`, which returns the bins/buckets based on the **equal frequency criteria**. Which is *-or tries to-* have the same quantity of rows per bin.

For numerical variables, `cross_plot` has by default the `auto_binning=T`, which automatically calls the `equal_freq` function with `n_bins=10` (or the closest number).

```
cross_plot(heart_disease, input="max_heart_rate", target="has_heart_disease")
```

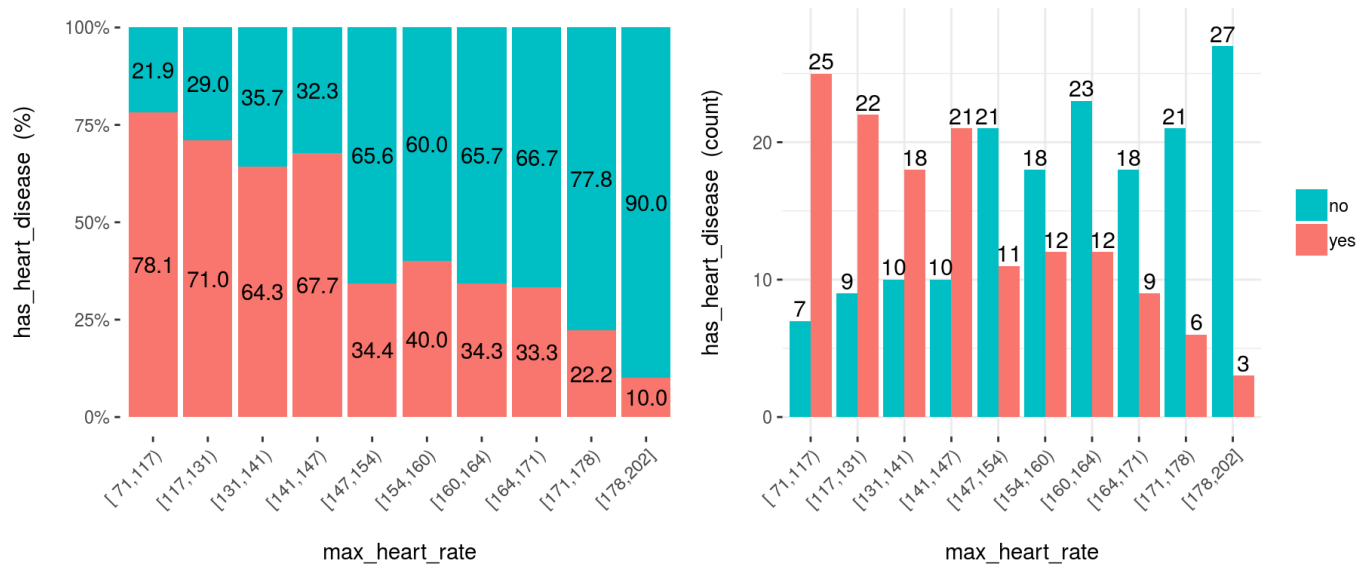


Figure 3.11: Numeric variable as input (automatic binning)

3.12.1.5 Example 3: Manual binning

If you don't want the automatic binning, then set the `auto_binning=F` in `cross_plot` function.

For example, creating `oldpeak_2` based on equal frequency, with three buckets.

```
heart_disease$oldpeak_2 =
  equal_freq(var=heart_disease$oldpeak, n_bins = 3)
summary(heart_disease$oldpeak_2)
```

```
## [0.0,0.2) [0.2,1.5) [1.5,6.2]
##          106         107         90
```

Plotting the binned variable (`auto_binning = F`):

```
cross_oldpeak_2=cross_plot(heart_disease, input="oldpeak_2", target="has_heart_disease",
```



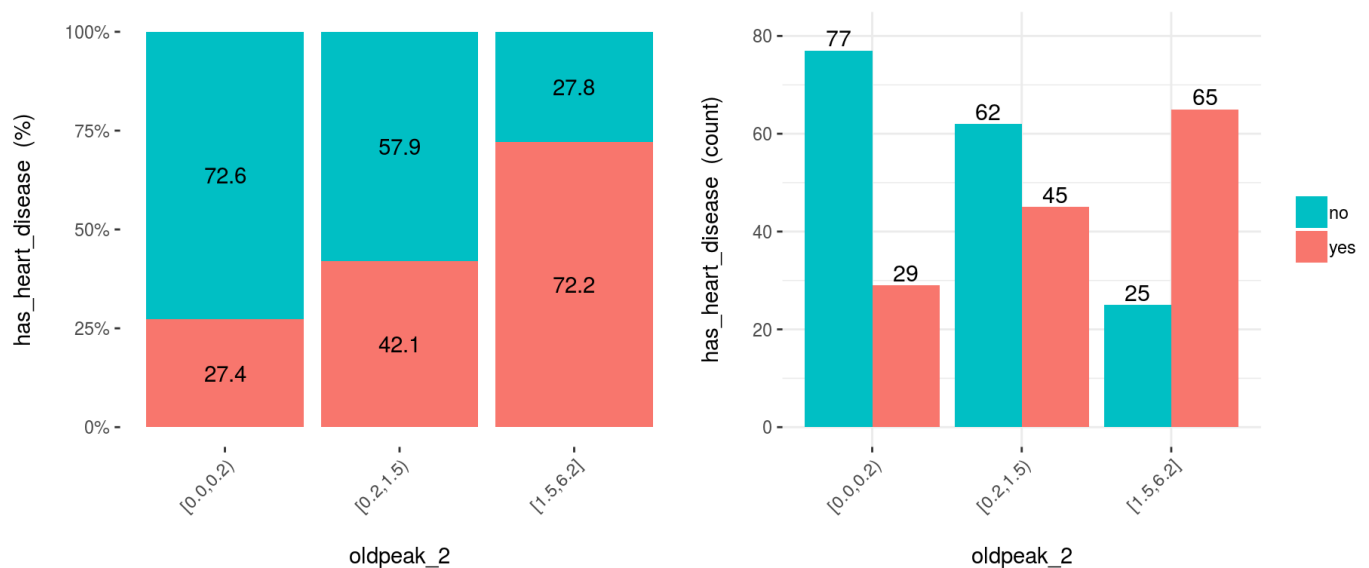


Figure 3.12: Disabling automatic binning shows the original variable

3.12.1.5.1 Conclusion

This new plot based on `oldpeak_2` shows clearly how: the likelihood of **having heart disease increases** as **oldpeak_2 increases** as well. *Again, it gives an order to the data.*

3.12.1.6 Example 4: Noise reducing

Converting variable `max_heart_rate` into a one of 10 bins:

```
heart_disease$max_heart_rate_2 =
  equal_freq(var=heart_disease$max_heart_rate, n_bins = 10)

cross_plot(heart_disease,
  input="max_heart_rate_2",
  target="has_heart_disease"
)
```

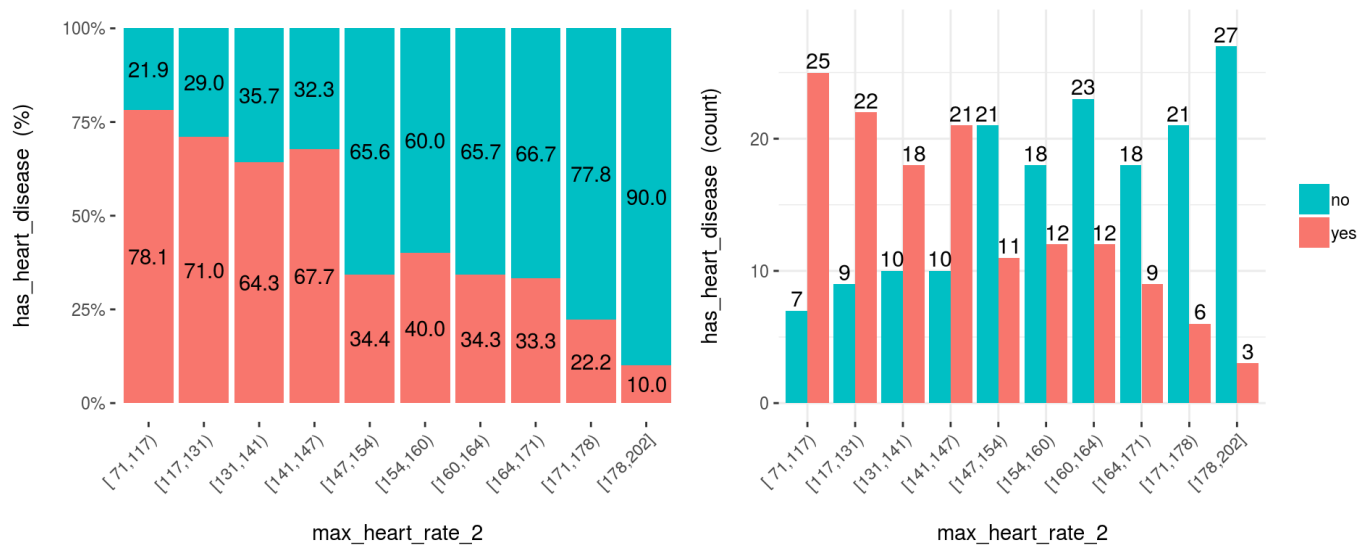


Figure 3.13: Plotting using custom binning

At first glance, `max_heart_rate_2` shows a negative and linear relationship. However, there are some buckets which add noise to the relationship. For example, the bucket `(141, 146]` has a higher heart disease rate than the previous bucket, and it was expected to have a lower. *This could be noise in data.*

Key note: One way to reduce the **noise** (at the cost of **losing** some information), is to split with less bins:

```
heart_disease$max_heart_rate_3 =
  equal_freq(var=heart_disease$max_heart_rate, n_bins = 5)

cross_plot(heart_disease,
  input="max_heart_rate_3",
  target="has_heart_disease"
)
```

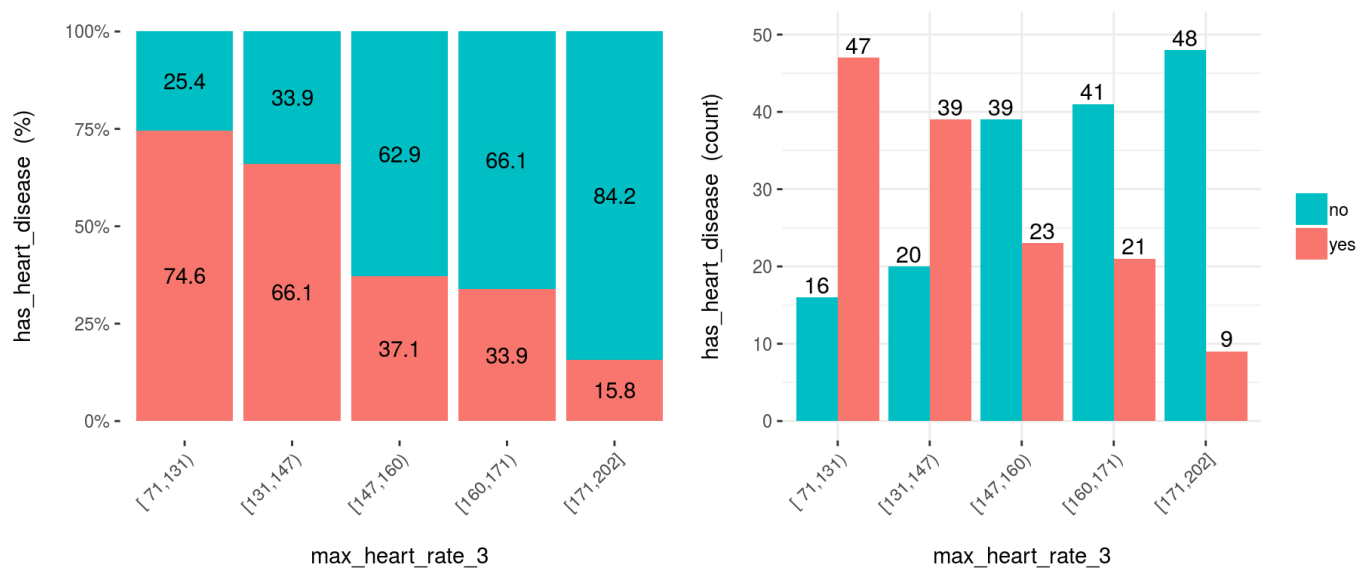


Figure 3.14: Reducing the bins may help to better expose the relationship

Conclusion: As it can be seen, now the relationship is much clean and clear. Bucket 'N' has a higher rate than 'N+1', which implies a negative correlation.

How about saving the cross_plot result into a folder?

Just set the parameter `path_out` with the folder you want -It creates a new one if it doesn't exists-.

```
cross_plot(heart_disease, input="max_heart_rate_3", target="has_heart_disease", path_out
```

It creates the folder `my_plots` into the working directory.

3.12.1.7 Example 5: cross_plot on multiple variables

Imagine you want to run `cross_plot` for several variables at the same time. To achieve this goal just define a vector containing the variable names.

If you want to analyze these 3 variables:

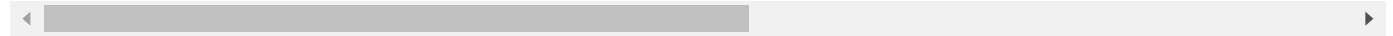
```
vars_to_analyze=c("age", "oldpeak", "max_heart_rate")
```

```
cross_plot(data=heart_disease, target="has_heart_disease", input=vars_to_analyze)
```

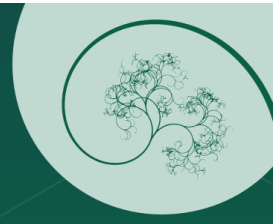

3.12.1.8 Exporting plots

`plotar` and `cross_plot` can handle from 1 to N input variables, and plots generated by them can be easily exported in high quality with parameter `path_out` .

```
plotar(data=heart_disease, input=c('max_heart_rate', 'resting_blood_pressure'), target=
```



Data Science Live Book



3.12.2 Using BoxPlots

3.12.2.1 What is this about?

The use of Boxplots in importance variable analysis gives a quick view of how different the quartiles are among the various values in a binary target variable.

```
# Loading funModeling !
```

```
library(funModeling)
```

```
data(heart_disease)
```

```
plotar(data=heart_disease, input="age", target="has_heart_disease", plot_type = "boxplot
```



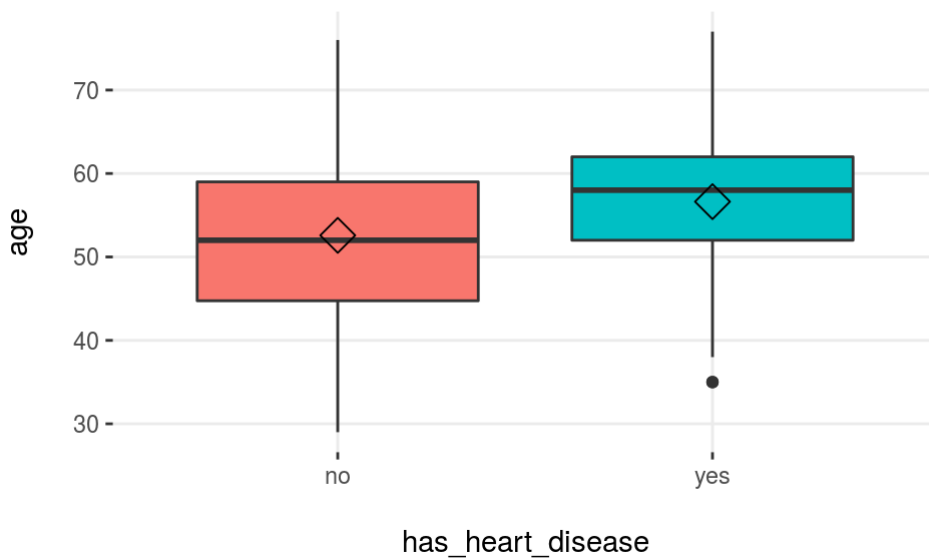


Figure 3.15: Profiling target using boxplot

*Rhomboid near the mean line represents the **median**.*

Box plot are used to see **percentiles graphically**.

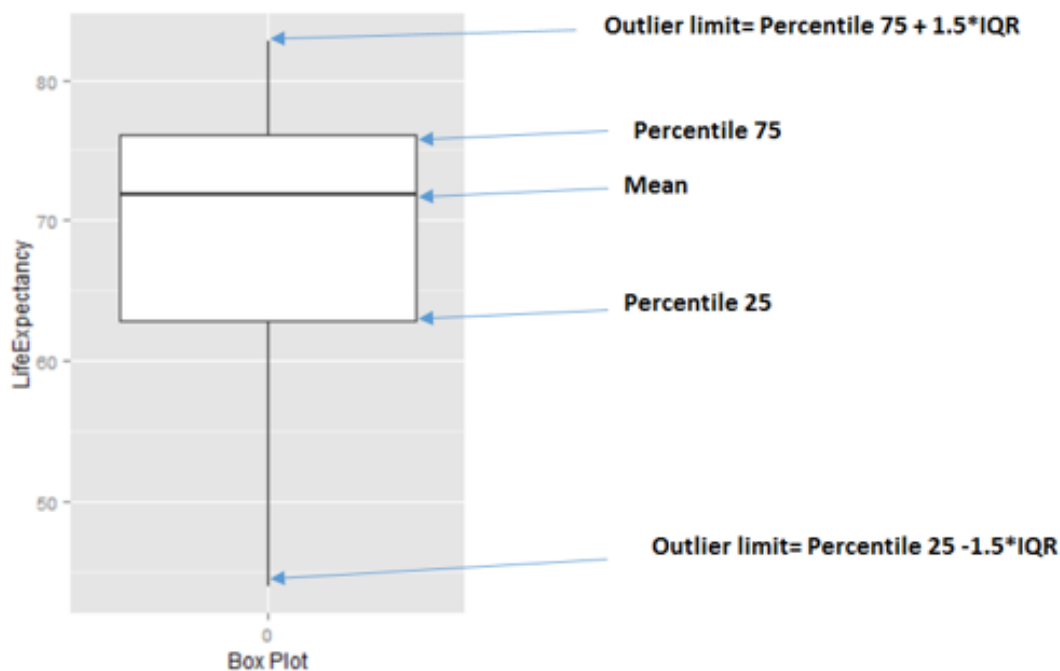


Figure 3.16: How to interpret a boxplot

When to use boxplots?

When we need to analyze different percentiles across the classes to predict. Note this is a powerful technique since the bias produced due to outliers doesn't affect as much as it does to the mean.

3.12.2.2 Boxplot: Good vs. Bad variable

Using more than one variable as inputs is useful in order to compare boxplots quickly, and thus getting the best variables...

```
plotar(data=heart_disease, input=c('max_heart_rate', 'resting_blood_pressure'), target=
```

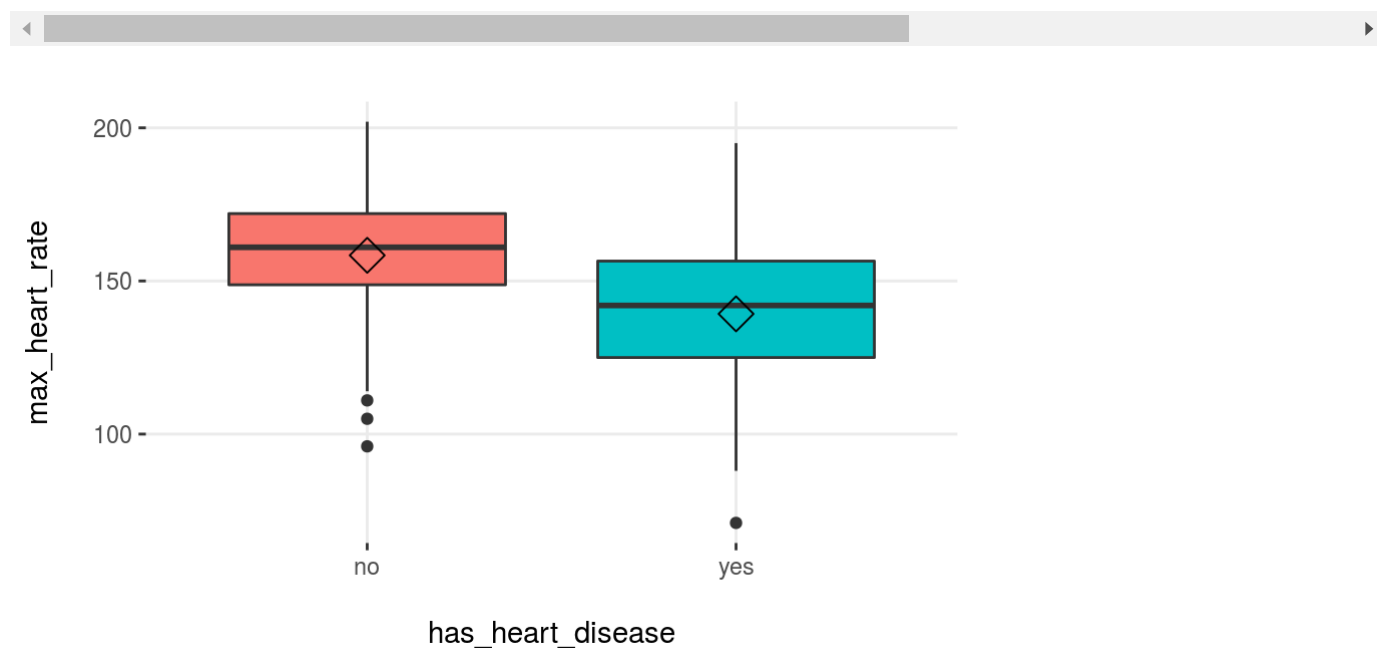


Figure 3.17: plotar function for multiple variables

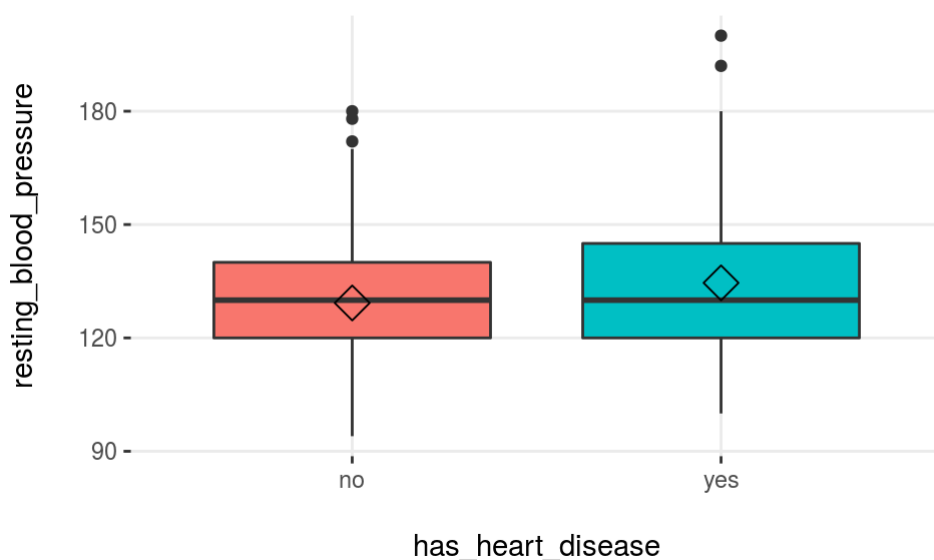


Figure 3.17: plotar function for multiple variables

We can conclude that `max_heart_rate` is a better predictor than `resting_blood_pressure`.

As a general rule, a variable will rank as **more important** if boxplots are **not aligned** horizontally.

Statistical tests: percentiles are another used feature used by them in order to determine -for example- if means across groups are or not the same.

3.12.2.3 Exporting plots

`plotar` and `cross_plot` can handle from 1 to N input variables, and plots generated by them can be easily exported in high quality with parameter `path_out` .

```
plotar(data=heart_disease, input=c('max_heart_rate', 'resting_blood_pressure'), target=
```



- **Keep in mind this when using Histograms and BoxPlots** They are nice to see when the variable:
 - Has a good spread -not concentrated on a bunch of 3, 4..6.. different values, **and**
 - It has not extreme outliers... *(this point can be treated with `prep_outLiers` function present in this package)*

3.12.3 Using Density Histograms

3.12.3.1 What is this about?

Density histograms are quite standard in any book/resource when plotting distributions. To use them in selecting variables gives a quick view on how well certain variable separates the class.

```
plotar(data=heart_disease, input="age", target="has_heart_disease", plot_type = "histder
```



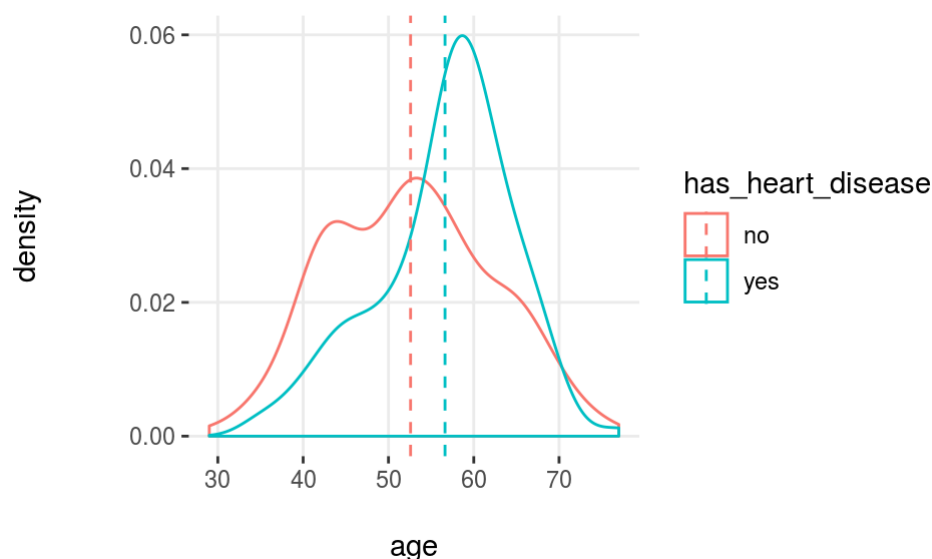


Figure 3.18: Profiling target using density histograms

Note: The dashed-line represents variable mean.

Density histograms are helpful to visualize the general shape of a numeric distribution.

This *general shape* is calculated based on a technique called **Kernel Smoother**, its general idea is to reduce high/low peaks (noise) present in near points/bars by estimating the function that describes the points. Here some pictures to illustrate the concept:

https://en.wikipedia.org/wiki/Kernel_smoother

3.12.3.2 What is the relationship with a statistical test?

Something similar is what a **statistical test** sees: they measured **how different** the curves are reflecting it in some statistics like the p-value using in the frequentist approach. It gives to the analyst reliable information to determine if the curves have -for example- the same mean.

3.12.3.3 Good vs. bad variable

```
plotar(data=heart_disease, input=c('resting_blood_pressure', 'max_heart_rate'), target=
```



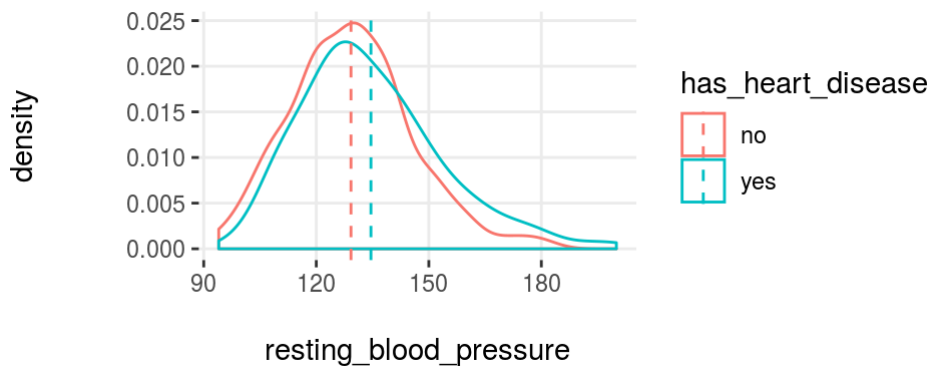


Figure 3.19: plotar function for multiple variables

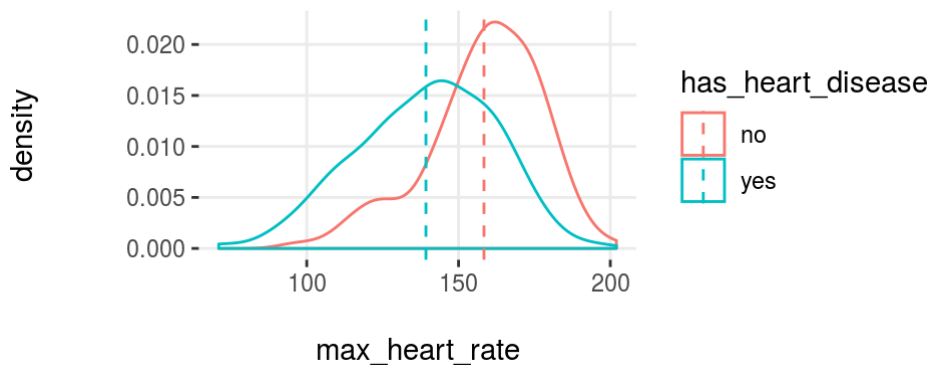
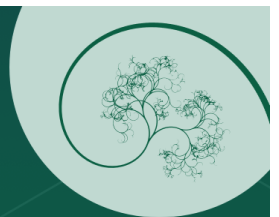


Figure 3.19: plotar function for multiple variables

And the model will see the same... if the curves are quite overlapped, like it is in `resting_blood_pressure`, then it's **not a good predictor** as if they were **more spaced** -like `max_heart_rate`.

- **Key in mind this when using Histograms & BoxPlots** They are nice to see when the variable:
 - Has a good spread -not concentrated on a bunch of 3, 4..6.. different values, **and**
 - It has not extreme outliers... *(this point can be treated with `prep_outliers` function present in this package)*



References

stackoverflow.com. 2017. "What Is Entropy and Information Gain?"

<http://stackoverflow.com/questions/1859554/what-is-entropy-and-information-gain>.

stats.stackexchange.com. 2015. "Gradient Boosting Machine Vs Random Forest."

<https://stats.stackexchange.com/questions/173390/gradient-boosting-tree-vs-random-forest>.

2017a. "How to Interpret Mean Decrease in Accuracy and Mean Decrease Gini in Random Forest Models." <http://stats.stackexchange.com/questions/197827/how-to-interpret-mean-decrease-in-accuracy-and-mean-decrease-gini-in-random-fore>.

Wikipedia. 2017c. "Occam's Razor."

https://en.wikipedia.org/wiki/Occam's_razor#Probability_theory_and_statistics.

Kuhn, Max. 2017. "Recursive Feature Elimination in R Package Caret."

<https://topepo.github.io/caret/recursive-feature-elimination.html>.