Good Practice Toolbox for Analyzing Data using Unsupervised Learning Methods

Daniel Barco

Lars Gisler

Ramon Schildknecht

Marcel Ulrich

Carmela Wey

10.11.2019

1 Overview

One of the following methods could be applied to the data of interest with the code provided by this document.

- Kmeans clustering
- Hierarchical Clustering (HC) & Heat Maps
- t-Distributed Stochastic Neighbor Embedding (tSNE)
- Prinicpal Component Analysis (PCA)
- Self-Organizing Maps (SOM)

2 Initial Situation

We are a consulting company, which offers consulting around the topic of Machine Learning. Our current consultation has been commissioned by a South African wine dealer to examine his collected data. The wine merchant sells most of his wine to hotels, restaurants as well as private customers. The wine dealer has a showroom in which he presents a selection of his wines to his customers. Based on his many years of experience, he evaluates the quality of each wine in order to advise his customers in the best possible way. Therefore, he has a database in which the quality of the wines, but also other characteristics are stored. These include, for example, the pH value, acidity, alcohol content, density, etc. A total of 11 predictors are included. We are charged with generating insight for the red wine collection.

Our task now is to form certain clusters in the available data, so that the wine dealer can differentiate his red wines not only according to grape varieties, countries of origin and growing regions, but also according to other criteria.
One idea that he has expressed is that he can make an initial classification of the quality of the wine on the basis of the alcohol content or the pH value.

3 Setup

3.1 Preparation

Install package checkpoint prior to running to following code in your shell:

- install.packages(checkpoint)
- library(checkpoint)
- checkpoint("2019-09-20")

This guarantees the reproducibility of following code.

3.2 Load packages

```
library(Rtsne) # library for TSNE
library(corrplot) # it is a graphical display of a correlation matrix, confidence i
nterval
library(gplots)  # for making own color code (e.g. for heatmap representation)
library(kohonen) # functions to train self-organising maps (SOMs)
library(plot3D) # functions for viewing 2-D & 3-D data, including perspective plot
s, slice plots, scatter plots, etc.
library(factoextra)  # more complexe library than standard R library; all methods f
or distance & linkage available
library(dendextend) # can be used to represent the data in ???fan??? diagrams
library(ape) # can be used to represent the data in ???fan??? diagrams
library(RColorBrewer)  # for making own color code and show different shades of col
ors (e.g. for heatmap representation)
library(rmarkdown) # convert R Markdown documents into a variety of formats
library(DataExplorer) # automated data exploration process for analytic tasks and p
redictive modeling
library(tidyverse)  # library for SOM
library(kohonen)  # library for SOM
library(checkpoint) # package to solve the problem of package reproducibility in R
```

4 Method selection

We created an overview and comparison about the covered methods within the course “Applied Machine Learning and Predictive Modelling II”:

Method	What it is?	When to use?	What it does?	Remark
K-means	Unsupervised, parametric method (need to only pre-specify K number of clusters)	- First exploration of multidimensional data - Only a few assumptions needed, i.e., K	- Groups data based on their similarity, the resulting clusters have similar properties - Push the data into K pre-determined categories	- The possibility to define the number of clusters can be an advantage in certain situations. - Algorithm has randomness in it. Output can be different with every time the user runs the code.
Hierarchical Clustering (HC)	Unsupervised, non-parametric method (no labelled data needed)	- First exploration of multidimensional data - No assumptions needed	- Groups data based on their similarity - The resulting clusters have similar properties	- “Better” than K-means clustering, no need to specify K number of clusters a priori - Height of branches merge indicative of similarity between groups - Height of cut has comparable role as the K in K-means: it controls the number of clusters obtained - Choice of where to cut the dendrogram is not always clear
Heat Maps	Graphical combination of dendrograms with quantitative data representation	- Most commonly used to display a more generalized view of numeric values	- Heatmaps group the data in a meaningful and data-driven way - No data labelling needed	- Colors can be adapted with several libraries
PCA	Unsupervised, linear, non-parametric method	- First exploration of multidimensional data - No assumptions needed - Meaningful (especially) with datasets of more than 3 dimension	- PCA reduces dimensionality of a dataset - Data organization method for grouping variables with similar behaviour	- User does not need to make assumptions - More understandable data representation due to dimension reduction
tSNE	Unsupervised, non-linear, parametric method for dimensionality reduction	- Exploration & visualization of data - Well-suited for high-dimensional data	- “Embeds high-dimensional data in a low-dimensional space to visualize” - tSNE minimizes the difference between the similarity of points in high & in low-dimensional space	- The message in only in the way groups are made - Distances have (almost) no meaning - Easy to apply but not always intuitive to interpret the plots - Output can be different with every time the user runs the code.
SOM	- Unsupervised, nonlinear, parametric method - Type of artificial neural network	- For data visualization of high-dimensional data	- Reduces dimensionality of a dataset - Trains neural network such that parts of it become specifically responsive to certain input patterns - Produces low-dimensional, discrete representation (= map) of the input space of the training samples - Mapping from a higher-dimensional input space to a lower-dimensional map space	- Somewhat similar to K-means (SOMs with a small number of nodes behave similar to K-means) - Somewhat similar to PCA (can be considered a nonlinear generalization of PCA)

A zoomable version of the overview can be found here (https://www.evernote.com/l/Ai9WPKBaEfFEzqgLvlABkPvsZA7qA0CgPjI/).

Furthermore the following decision support tool can be used to find a suitable methods for the problem at hand:
https://mod.rapidminer.com/#app (https://mod.rapidminer.com/#app)

5 Data

5.1 Preparation

```
# clean the memory
rm(list = ls(all = TRUE))

# load data
df <- read.csv("data/winequality-red.csv")

# save dat to Rds-file
saveRDS(df, "data/winequality-red.Rds")

# count the missing values (NA) in the data frame
sum(is.na(df))
```

```
## [1] 0
```

```
# check if the columns are numeric
sapply(df, is.numeric)
```

```
##        fixed.acidity     volatile.acidity          citric.acid
##                 TRUE                 TRUE                 TRUE
##       residual.sugar            chlorides  free.sulfur.dioxide
##                 TRUE                 TRUE                 TRUE
## total.sulfur.dioxide              density                   pH
##                 TRUE                 TRUE                 TRUE
##            sulphates              alcohol              quality
##                 TRUE                 TRUE                 TRUE
```

```
# scale the data
df_scaled <- scale(df)
class(df_scaled)  # is now a matrix
```

```
## [1] "matrix"
```

```
as_tibble(df_scaled)
```

fixed.acidity <dbl>	volatile.acidity <dbl>	citric.acid <dbl>	residual.sugar <dbl>	chlorides <dbl>
-0.52819437	0.96157585	-1.391037103	-0.453076665	-0.243630469
-0.29845406	1.96682715	-1.391037103	0.043402567	0.223805180
-0.29845406	1.29665962	-1.185699492	-0.169374247	0.096322731
1.65433853	-1.38401051	1.483689439	-0.453076665	-0.264877544
-0.52819437	0.96157585	-1.391037103	-0.453076665	-0.243630469
-0.52819437	0.73818667	-1.391037103	-0.524002270	-0.264877544

fixed.acidity <dbl>	volatile.acidity <dbl>	citric.acid <dbl>	residual.sugar <dbl>	chlorides <dbl>
-0.24101899	0.40310291	-1.083030687	-0.665853479	-0.392359993
-0.58562945	0.68233938	-1.391037103	-0.949555897	-0.477348293
-0.29845406	0.29140832	-1.288368298	-0.382151061	-0.307371694
-0.47075929	-0.15537004	0.457001388	2.525798726	-0.349865844

1-10 of 1,599 rows | 1-5 of 12 columns Previous 1 2 3 4 5 6 ... 160 Next

5.2 Exploration

Description of the Variables

1. - 11.: Features &
12.: Label

1. fixed acidity: most acids involved with wine or fixed or nonvolatile (do not evaporate readily)
2. volatile acidity: the amount of acetic acid in wine, which at too high of levels can lead to an unpleasant, vinegar taste
3. citric acid: found in small quantities, citric acid can add ???freshness??? and flavor to wines
4. residual sugar: the amount of sugar remaining after fermentation stops, it???s rare to find wines with less than 1 gram/liter and wines with greater than 45 grams/liter are considered sweet 5.chlorides: the amount of salt in the wine
5. free sulfur dioxide: the free form of SO2 exists in equilibrium between molecular SO2 (as a dissolved gas) and bisulfite ion; it prevents microbial growth and the oxidation of wine
6. total sulfur dioxide: amount of free and bound forms of S02; in low concentrations, SO2 is mostly undetectable in wine, but at free SO2 concentrations over 50 ppm, SO2 becomes evident in the nose and taste of wine
7. density: the density of water is close to that of water depending on the percent alcohol and sugar content
8. pH: describes how acidic or basic a wine is on a scale from 0 (very acidic) to 14 (very basic); most wines are between 3-4 on the pH scale
9. sulphates: a wine additive which can contribute to sulfur dioxide gas (S02) levels, wich acts as an antimicrobial and antioxidant
10. alcohol: the percent alcohol content of the wine
11. quality (score between 0 and 10)

Details about the dataset can be found here (https://s3.amazonaws.com/udacity-hosted-downloads/ud651/wineQualityInfo.txt).

```
glimpse(df)
```

```
## Observations: 1,599
## Variables: 12
## $ fixed.acidity        <dbl> 7.4, 7.8, 7.8, 11.2, 7.4, 7.4, 7.9, 7.3, ...
## $ volatile.acidity     <dbl> 0.700, 0.880, 0.760, 0.280, 0.700, 0.660,...
## $ citric.acid          <dbl> 0.00, 0.00, 0.04, 0.56, 0.00, 0.00, 0.06,...
## $ residual.sugar       <dbl> 1.9, 2.6, 2.3, 1.9, 1.9, 1.8, 1.6, 1.2, 2...
## $ chlorides            <dbl> 0.076, 0.098, 0.092, 0.075, 0.076, 0.075,...
## $ free.sulfur.dioxide  <dbl> 11, 25, 15, 17, 11, 13, 15, 15, 9, 17, 15...
## $ total.sulfur.dioxide <dbl> 34, 67, 54, 60, 34, 40, 59, 21, 18, 102, ...
## $ density              <dbl> 0.9978, 0.9968, 0.9970, 0.9980, 0.9978, 0...
## $ pH                   <dbl> 3.51, 3.20, 3.26, 3.16, 3.51, 3.51, 3.30,...
## $ sulphates            <dbl> 0.56, 0.68, 0.65, 0.58, 0.56, 0.56, 0.46,...
## $ alcohol              <dbl> 9.4, 9.8, 9.8, 9.8, 9.4, 9.4, 9.4, 10.0, ...
## $ quality              <int> 5, 5, 5, 6, 5, 5, 5, 7, 7, 5, 5, 5, 5, 5,...
```

```
# View(df)
summary(df)
```

```
##  fixed.acidity   volatile.acidity  citric.acid    residual.sugar
##  Min.   : 4.60   Min.   :0.1200   Min.   :0.000   Min.   : 0.900
##  1st Qu.: 7.10   1st Qu.:0.3900   1st Qu.:0.090   1st Qu.: 1.900
##  Median : 7.90   Median :0.5200   Median :0.260   Median : 2.200
##  Mean   : 8.32   Mean   :0.5278   Mean   :0.271   Mean   : 2.539
##  3rd Qu.: 9.20   3rd Qu.:0.6400   3rd Qu.:0.420   3rd Qu.: 2.600
##  Max.   :15.90   Max.   :1.5800   Max.   :1.000   Max.   :15.500
##    chlorides       free.sulfur.dioxide total.sulfur.dioxide
##  Min.   :0.01200   Min.   : 1.00       Min.   :  6.00
##  1st Qu.:0.07000   1st Qu.: 7.00       1st Qu.: 22.00
##  Median :0.07900   Median :14.00       Median : 38.00
##  Mean   :0.08747   Mean   :15.87       Mean   : 46.47
##  3rd Qu.:0.09000   3rd Qu.:21.00       3rd Qu.: 62.00
##  Max.   :0.61100   Max.   :72.00       Max.   :289.00
##     density             pH          sulphates         alcohol
##  Min.   :0.9901   Min.   :2.740   Min.   :0.3300   Min.   : 8.40
##  1st Qu.:0.9956   1st Qu.:3.210   1st Qu.:0.5500   1st Qu.: 9.50
##  Median :0.9968   Median :3.310   Median :0.6200   Median :10.20
##  Mean   :0.9967   Mean   :3.311   Mean   :0.6581   Mean   :10.42
##  3rd Qu.:0.9978   3rd Qu.:3.400   3rd Qu.:0.7300   3rd Qu.:11.10
##  Max.   :1.0037   Max.   :4.010   Max.   :2.0000   Max.   :14.90
##     quality
##  Min.   :3.000
##  1st Qu.:5.000
##  Median :6.000
##  Mean   :5.636
##  3rd Qu.:6.000
##  Max.   :8.000
```

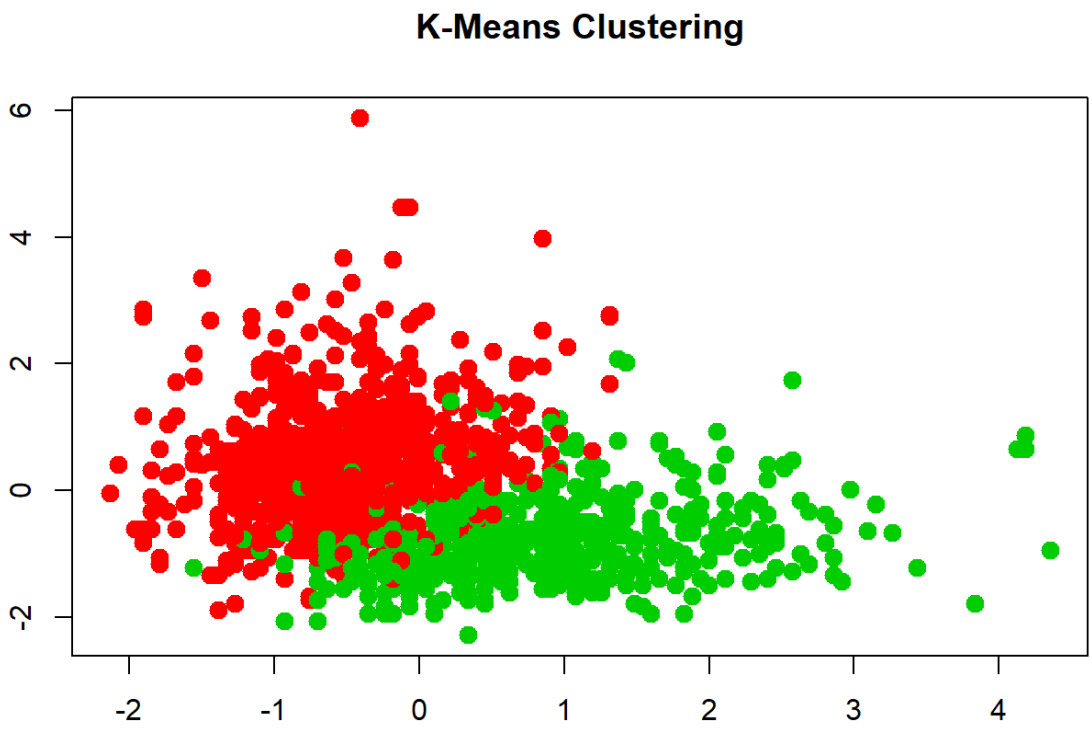
6 k-Means-Clustering

6.1 Caluclate k-means-clustering and Show Metrics

Available metrics in the returend k-Means object

- **withinss:** Vector of within-cluster sum of squares, one component per cluster.
- **tot.withinss:** Total within-cluster sum of squares, i.e. sum(withinss).
- **size:** The number of points in each cluster.
- **totss:** The total sum of squares.
- **centers:** A matrix of cluster centres.
- **cluster:** A vector of integers (from 1:k) indicating the cluster to which each point is allocated
- **betweenss:** The between-cluster sum of squares, i.e. totss-tot.withinss.
- **iter:** The number of (outer) iterations.
- **ifault:** integer: indicator of a possible algorithm problem ??? for experts.

```
# Run K-means for k=2
k <- 2 #defining to work with two clusters
km.out <- kmeans(df_scaled,k,nstart =50)
plot(df_scaled, col =(km.out$cluster+1), main ="K-Means Clustering", xlab ="", ylab
="", pch =20, cex =2)
```




```
# Check the outcome
km.out
```

```
## K-means clustering with 2 clusters of sizes 1014, 585
##
## Cluster means:
##   fixed.acidity volatile.acidity citric.acid residual.sugar  chlorides
## 1    -0.5110885        0.4208925  -0.5706485    -0.07260346 -0.1340215
## 2     0.8858867       -0.7295470   0.9891242     0.12584600  0.2323040
##   free.sulfur.dioxide total.sulfur.dioxide    density         pH
## 1           0.1027501            0.1435627 -0.2251450  0.3888984
## 2          -0.1781002           -0.2488421  0.3902513 -0.6740906
##    sulphates    alcohol    quality
## 1 -0.3284437 -0.1675215 -0.2697930
## 2  0.5693024  0.2903706  0.4676411
##
## Clustering vector:
##    [1] 1 1 1 2 1 1 1 1 1 1 1 1 1 2 1 1 2 2 1 2 2 1 2 1 1 1 1 2 1 1 1 1 1 1
##   [35] 1 1 1 2 1 1 1 1 1 2 1 1 1 1 2 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1
##   [69] 2 1 1 1 1 1 1 2 2 2 1 1 1 1 2 1 2 1 1 2 1 2 1 1 2 2 1 1 1 1 1 1 1 1
##  [103] 1 1 1 1 1 2 1 2 1 1 1 1 2 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [137] 1 1 1 1 1 1 1 1 1 1 1 2 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2
##  [171] 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 2 1 1 1 1 1 1 1 1 1 1 2 1 1 2 2 1 1
##  [205] 1 2 2 1 1 2 2 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1
##  [239] 1 1 2 2 1 2 2 1 1 1 1 1 2 1 2 1 1 1 2 1 2 2 1 1 1 1 2 2 1 2 1 2 1 2
##  [273] 2 1 1 1 1 1 2 2 2 2 2 1 2 1 1 2 1 2 2 2 2 2 1 2 2 1 1 1 1 1 2 1 1 1 2
##  [307] 1 2 2 1 2 1 1 1 1 1 1 1 2 1 2 1 1 2 2 2 2 2 2 2 2 2 1 1 1 2 2 1 2 2
##  [341] 2 2 2 2 2 2 1 1 2 2 1 2 1 1 2 1 1 2 2 2 2 1 2 2 2 2 2 2 2 2 2 1 2 2 1
##  [375] 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 2 1 2 2 1 2 2 1 2 2 1 1 1 2 2 1 2 2 2
##  [409] 2 2 1 1 1 2 1 1 2 1 2 1 2 1 1 2 1 1 1 1 1 2 2 1 2 2 2 2 1 2 2 1 2 2
##  [443] 2 2 1 1 2 2 1 2 2 2 1 2 1 2 1 1 2 2 2 1 2 1 2 2 2 2 2 1 2 2 2 2 2 1
##  [477] 2 2 1 1 2 2 2 2 2 2 2 2 2 2 1 2 2 1 1 2 1 1 2 1 1 2 2 2 2 2 2 2 2 2
##  [511] 2 2 2 2 2 2 2 2 2 2 1 2 1 2 1 2 1 1 2 2 1 2 2 2 2 2 2 1 1 2 2 1 2 1 2
##  [545] 2 2 1 2 2 2 1 2 2 1 2 2 2 2 2 2 2 1 2 2 2 2 1 1 2 1 2 1 2 2 2 2 1 1
##  [579] 1 2 2 2 2 2 2 1 2 1 1 2 2 1 2 2 1 1 2 2 1 2 1 2 1 2 1 1 2 2 2 1 2 2
##  [613] 1 2 2 1 1 2 2 2 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 2 2 1 2 1 2 1
##  [647] 1 1 2 1 2 1 2 2 2 1 2 2 1 1 1 1 1 2 2 1 2 2 2 2 1 1 1 1 2 2 2 1 1 2
##  [681] 2 1 1 1 1 1 1 1 1 2 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 2 1 1 1 1
##  [715] 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1
##  [749] 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 1
##  [783] 1 1 1 2 2 2 2 1 1 1 1 1 2 2 2 2 2 2 1 1 1 1 1 2 2 2 1 1 1 2 2 1 2 2
##  [817] 2 2 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 2 2 1 1 1 1 2 1 2 1 2 1 2 1 1 1 1 1
##  [851] 2 2 2 2 2 2 1 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 1 1 1 1 1 1 2 1
##  [885] 1 1 1 2 1 1 2 1 1 2 1 1 1 2 1 2 1 2 1 1 1 1 1 1 1 1 1 2 2 2 2 1 2 1 1
##  [919] 2 1 2 2 2 1 1 2 2 2 1 2 2 1 1 1 1 1 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2
##  [953] 2 2 1 2 2 2 2 1 1 2 1 1 2 2 2 2 1 2 1 2 2 2 1 2 1 1 1 1 2 2 1 1 2 2 1
##  [987] 2 1 1 2 1 1 1 1 1 2 1 1 1 1 1 1 2 2 1 1 1 2 2 2 2 2 2 1 1 1 2 2 2 2 1
## [1021] 2 2 1 2 2 1 1 1 1 1 1 1 1 1 1 1 2 2 1 2 2 1 1 2 2 1 1 1 1 2 2 1 2 1 2
## [1055] 1 1 2 1 2 2 2 2 2 2 2 1 1 1 2 2 1 2 2 1 1 1 1 2 2 2 2 2 2 2 1 2 1 1 2 2
## [1089] 2 2 2 2 2 1 2 1 2 2 1 1 2 1 2 1 1 1 1 1 2 2 1 2 1 1 2 2 1 1 1 1 1 1 2 1
## [1123] 1 2 1 2 2 1 1 2 2 2 1 1 2 1 2 2 2 2 2 1 1 1 2 1 1 1 2 1 2 2 2 2 2 1 1 2 1 1
## [1157] 2 1 2 2 2 2 2 2 1 1 2 2 2 2 1 1 2 1 2 2 1 1 1 1 1 1 2 2 2 2 2 1 1 1 1 1 1 1
## [1191] 2 1 2 2 1 1 1 1 1 2 2 1 1 2 2 1 2 2 2 2 2 2 2 1 1 1 2 2 2 1 2 2 2 2 2 2 1 2
## [1225] 2 1 1 1 1 1 1 2 2 1 1 2 1 1 1 1 1 1 1 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1259] 1 1 2 1 2 2 1 1 1 1 2 1 1 1 1 1 1 1 1 2 1 1 2 1 1 1 1 1 2 2 2 1 1 1 1 1
## [1293] 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 2 1 1 2 2 1 1
## [1327] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2
## [1361] 1 1 2 2 1 1 1 1 2 2 1 1 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1395] 1 1 1 1 1 1 1 1 2 2 2 1 2 2 2 1 2 1 1 1 2 2 2 2 1 2 2 1 1 1 1 1 2 2 2 1
## [1429] 1 2 2 1 1 1 1 2 2 2 2 1 1 1 2 2 1 1 1 1 1 1 1 1 2 2 2 2 1 1 2 2 1 1 1 2 2 2 1 1
```

```
## [1463] 1 1 1 1 1 1 1 1 1 1 2 1 2 1 2 1 1 2 1 2 1 1 1 1 1 1 1 2 1 1 1 1 1
## [1497] 1 1 1 1 1 1 1 1 2 1 1 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1
## [1531] 1 1 1 1 1 1 1 1 1 1 1 2 1 2 2 1 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1
## [1565] 1 1 2 1 1 1 2 1 1 1 2 1 2 1 1 1 1 1 1 1 2 2 2 1 1 1 1 1 1 1 1 1 1
## [1599] 1
##
## Within cluster sum of squares by cluster:
## [1] 8334.629 7434.904
##  (between_SS / total_SS =  17.8 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"
## [5] "tot.withinss" "betweenss"    "size"         "iter"
## [9] "ifault"
```

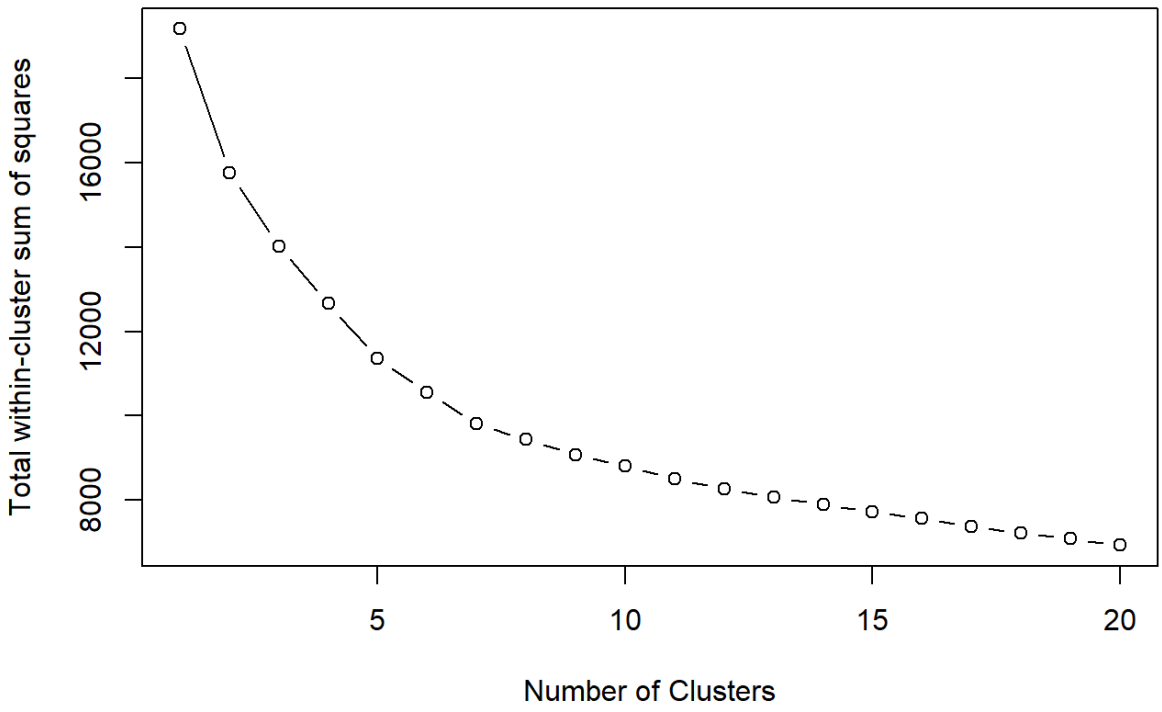
```
km.out$withinss
```

```
## [1] 8334.629 7434.904
```

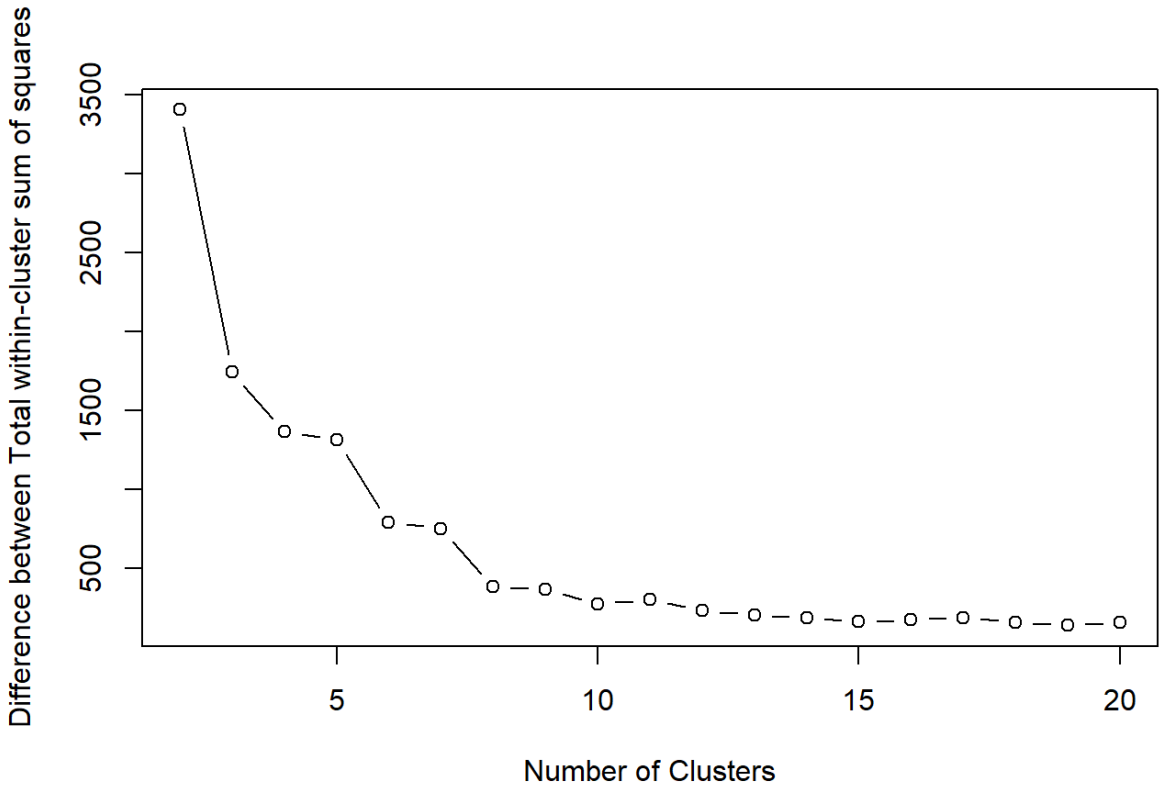
```
cat("withinss =", km.out$tot.withinss)
```

```
## withinss = 15769.53
```

```
# Find best value for k
wss <- 0 # intialise
wss_dif <- 0
number_of_clusters_tested <- 20
for (i in 1:number_of_clusters_tested){
  km.out <- kmeans(df_scaled,i,nstart =50)
  wss[i] <- km.out$tot.withinss
  if(i > 1){ # only enter condition for two clusters and higher
    wss_dif[i-1] <- wss[i-1]-wss[i] # take difference from previous "total within-c
luster sum of squares" and current one.
  }

}

plot(1:number_of_clusters_tested, wss, type="b", xlab="Number of Clusters",ylab="To
tal within-cluster sum of squares")
```




```
plot(2:number_of_clusters_tested, wss_dif, type="b", xlab="Number of Clusters",ylab
="Difference between Total within-cluster sum of squares")
```



6.2 Interpretation

- First plot: K-means Clustering with k=2 is not very useful. This plot is only useful if you have a dataset with only two dimensions.
- Second plot: With an increasing number of clusters the total within cluster sum of squares always decreases. To decide which k is the best, we look for an ellbow in the plot. In this plot there is no clear ellbow visible. Therefore we go for k = 2.
- Third plot: This plot shows the difference between the i-th and the i-1th value of the total within-cluster sum of squares. This plot should be viewed with caution as it compares the sum of squares to the previous (k-1) sum of squares. An alternative would be to subtract the mean from every sum of squares.

To sum up, for the wine case no conclusions can be drawn from this single plot. To draw conclusions, one would have to select two features and compare them on the x- and y-axis. Then you would see different clusters, here for example with k=2 i.e. two clusters, related to the certain x- and y-axis features. To show this possibility, we have created a Shiny widget, which can be tested in this additional RMarkdown: exploring_dataset_with_shiny_widget_k_means.Rmd

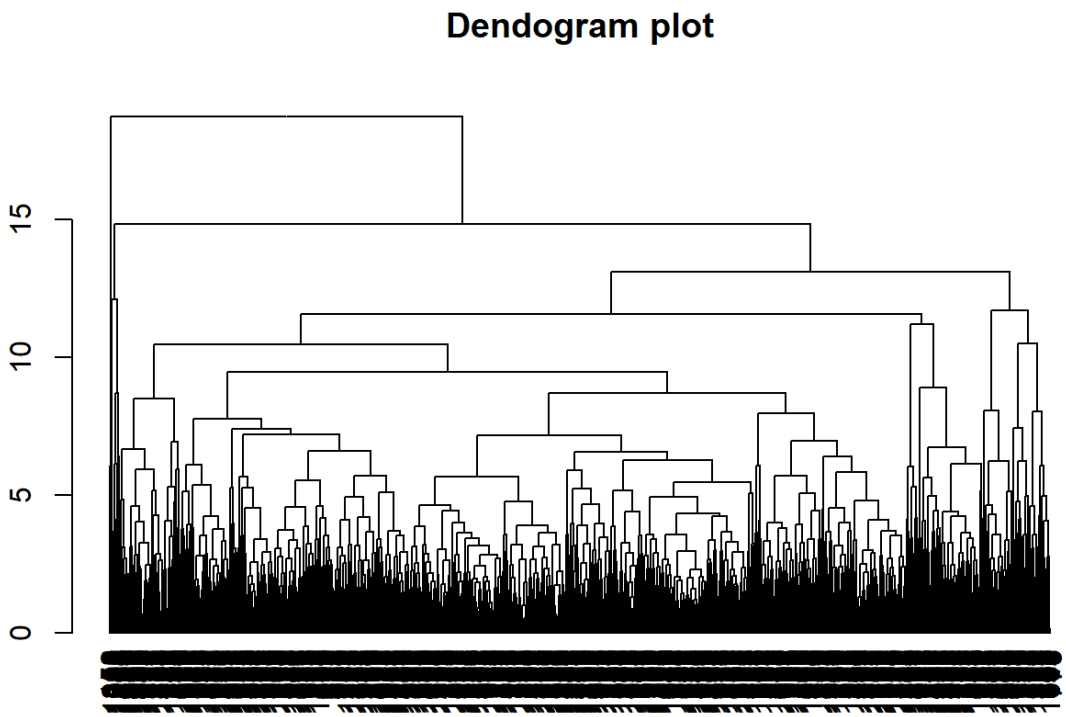
7 Hierarchical Clustering (HC)

7.1 Calculate Hierarchical Clustering and Show Metrics

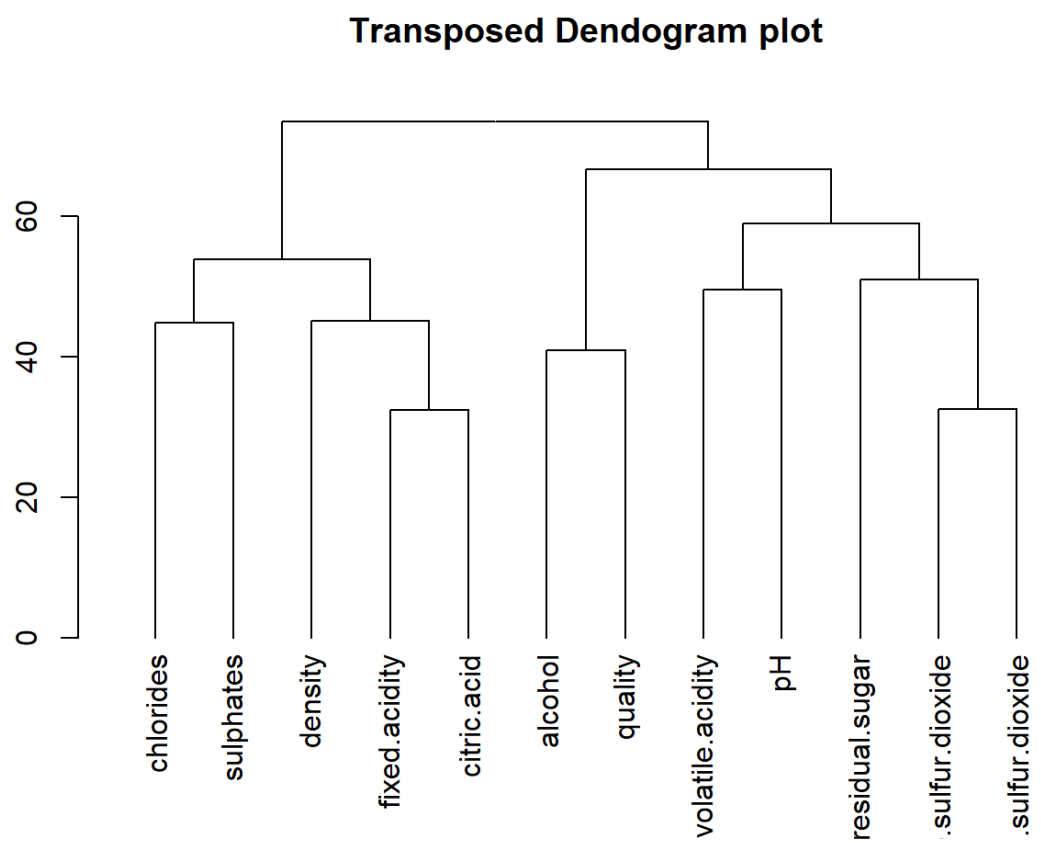
Available metrics in the returend HC object

- **labels:** labels for each of the objects being clustered.
- **method:** the cluster method that has been used.
- **dist.method:** the distance that has been used to create d (only returned if the distance object has a “method” attribute).
- **call:** the call which produced the result.
- **order:** a vector giving the permutation of the original observations suitable for plotting, in the sense that a cluster plot using this ordering and matrix merge will not have crossings of the branches.
- **height:** a set of n-1 real values (non-decreasing for ultrametric trees). The clustering height: that is, the value of the criterion associated with the clustering method for the particular agglomeration.
- **merge:** an n-1 by 2 matrix. Row i of merge describes the merging of clusters at step i of the clustering. If an element j in the row is negative, then observation -j was merged at this stage. If j is positive then the merge was with the cluster formed at the (earlier) stage j of the algorithm. Thus negative entries in merge indicate agglomerations of singletons, and positive entries indicate agglomerations of non-singletons.

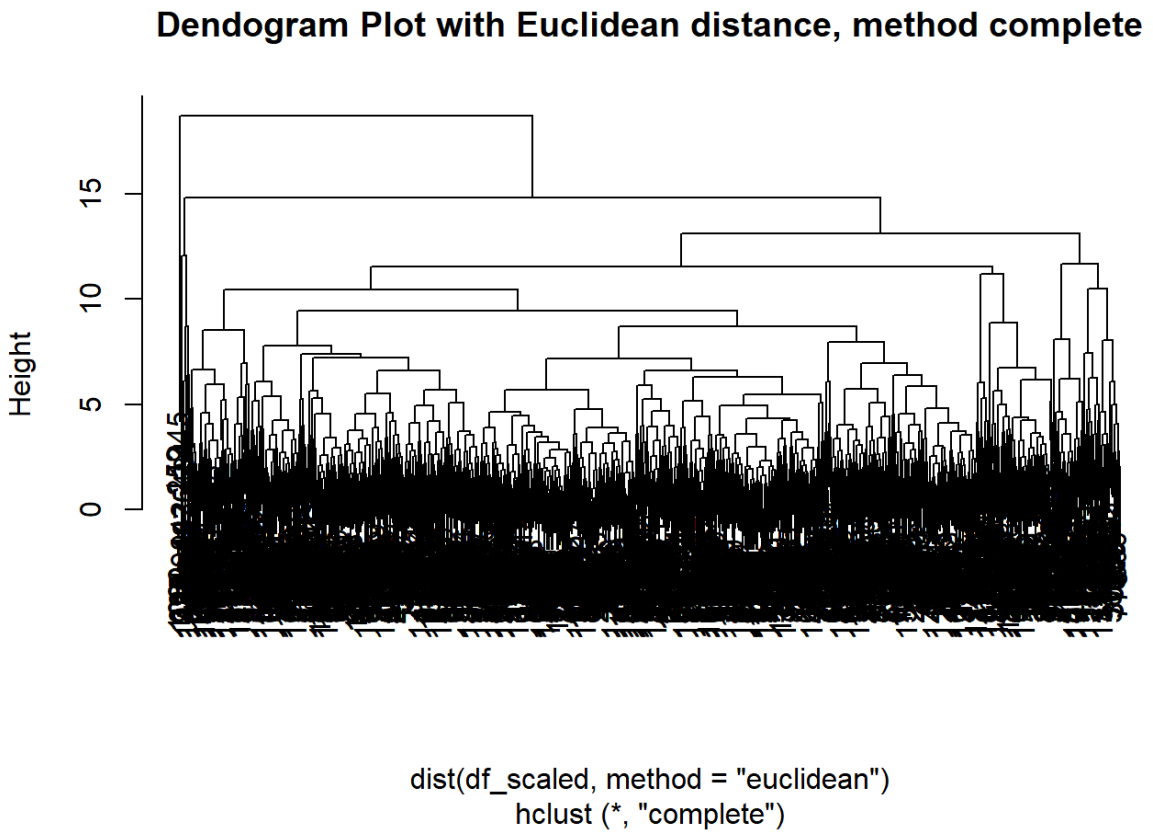
```
# Calculate distances between data (default method = euclidean):
distances <- dist(df_scaled, method = "euclidean")

# Compute hierarchical clustering based in distances calculated above:
hc <- hclust(distances)

# Computes dendrogram graphical representation:
dend <- as.dendrogram(hc)

# Graphical representation
plot(dend, main = "Dendogram plot")
```



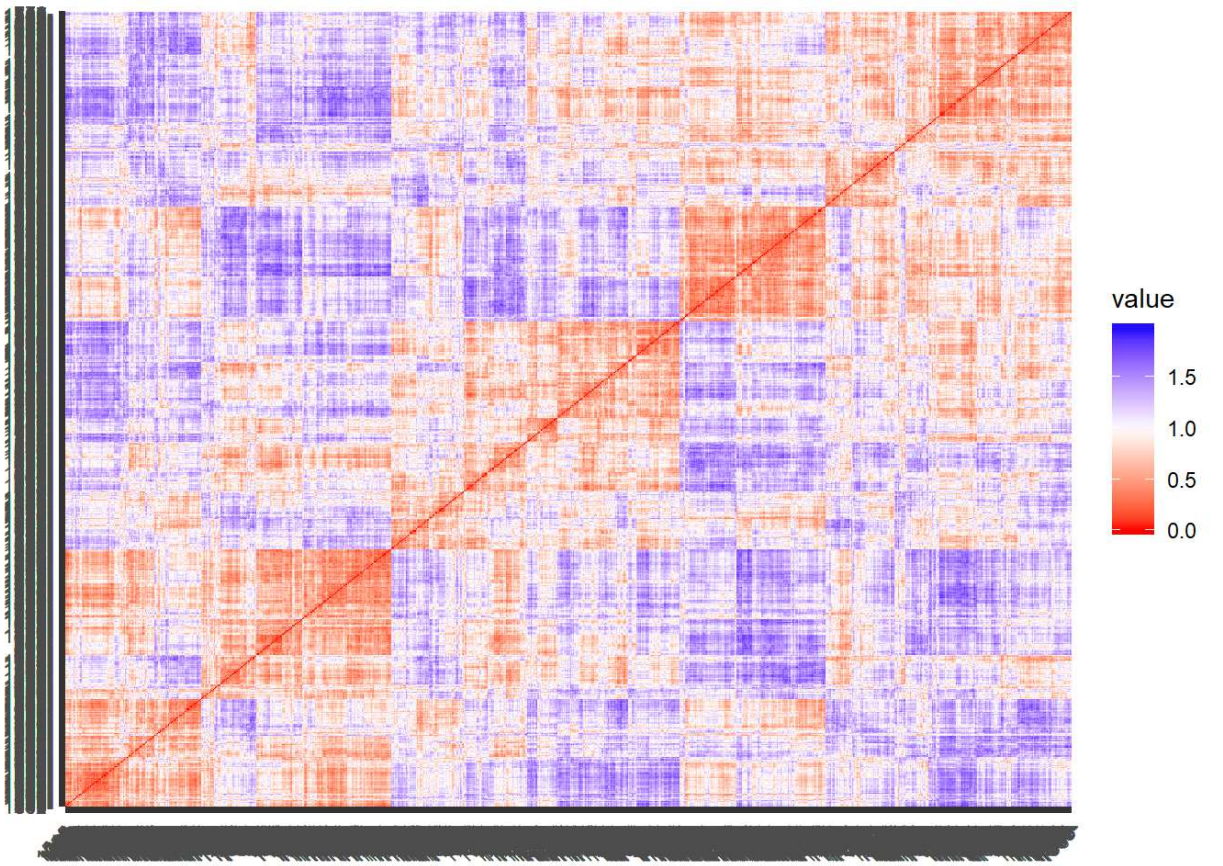
```
# Alternative, standard output representation (can be useful for ctrl+find specific
things in big trees)
# str(dend)  # computionally very expensive, takes about 2 minutes



# Transpose data:
df_transposed <- t(df_scaled)

# Calculate distances between data (default method = euclidean):
distances_t <- dist(df_transposed, method = "euclidean")

# Compute hierarchical clustering based in distances calculated above:
hc_t <- hclust(distances_t)

# Computes dendrogram graphical representation:
dend_t <- as.dendrogram(hc_t)

# Graphical representation
plot(dend_t, main = "Transposed Dendogram plot")
```



```
# Alternative, standard output representation (can be useful for ctrl+find specific
things in big trees)
# str(dend_t) # computionally very expensive, takes about 2 minutes


############################ further information ###################################
##
# Use a specific linkage or distance methode in hc clustering
# Find more information about distances and which one to choose: https://www.datano
via.com/en/lessons/clustering-distance-measures/
# Possible distance methods: "euclidean", "maximum", "manhattan","canberra","binar
y", "minkowski"
# Find more information about linkage method and which one to choose: https://www.d
atanovia.com/en/lessons/agglomerative-hierarchical-clustering/#linkage
# Possible linkage methods: "ward.D", "ward.D2", "single", "complete","average", "m
cquitty", "median","centroid"

##############################################################################################
###


# Compute hierarchical clustering with euclidean distance and complete method:
hc <- hclust(dist(df_scaled, method = "euclidean"),method="complete")
plot(hc, main = "Dendogram Plot with Euclidean distance, method complete")
```



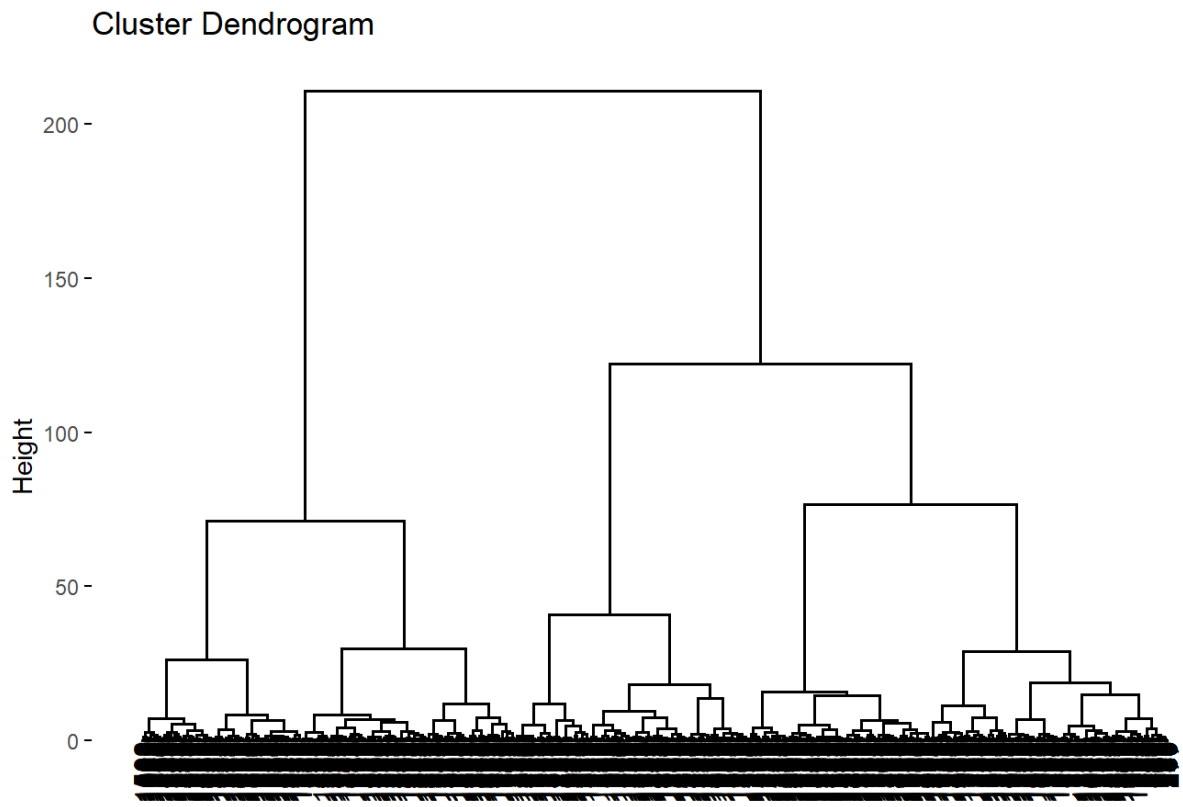
```
# Missing correlation-based methods (spearman, kendall): using library(factoextra)
 for these methods
# Compute the dissimilarity matrix with different distance types: "euclidean", "man
hattan", "pearson", "spearman", "kendall"
res.dist <- get_dist(df_scaled, method = "kendall")

# Visualize the dissimilarity matrix
fviz_dist(res.dist, lab_size = 8)  # takes some time ~1min
```



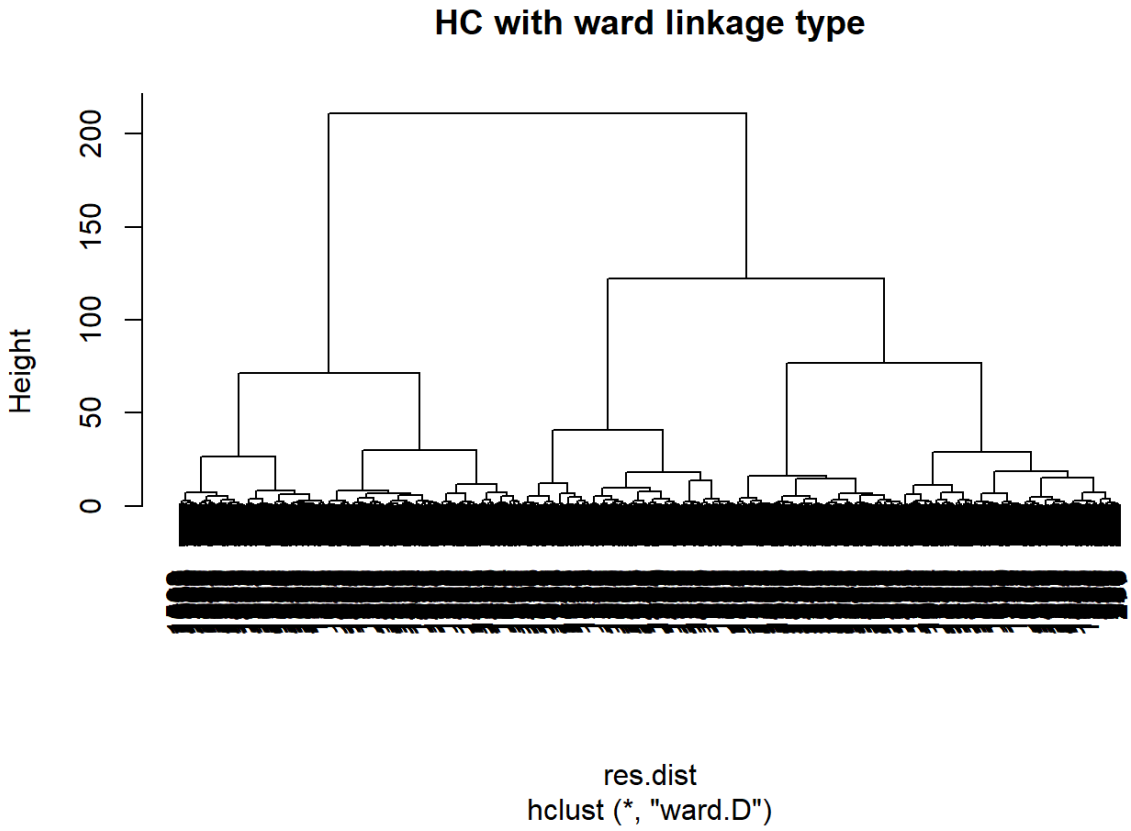

```
# Compute hierarchical clustering with different linkage types:"single", "complet
e", "average", "centroid", "ward.D", "ward.D2"
res.hc <- hclust(res.dist, method = "ward")
```

```
## The "ward" method has been renamed to "ward.D"; note new "ward.D2"
```

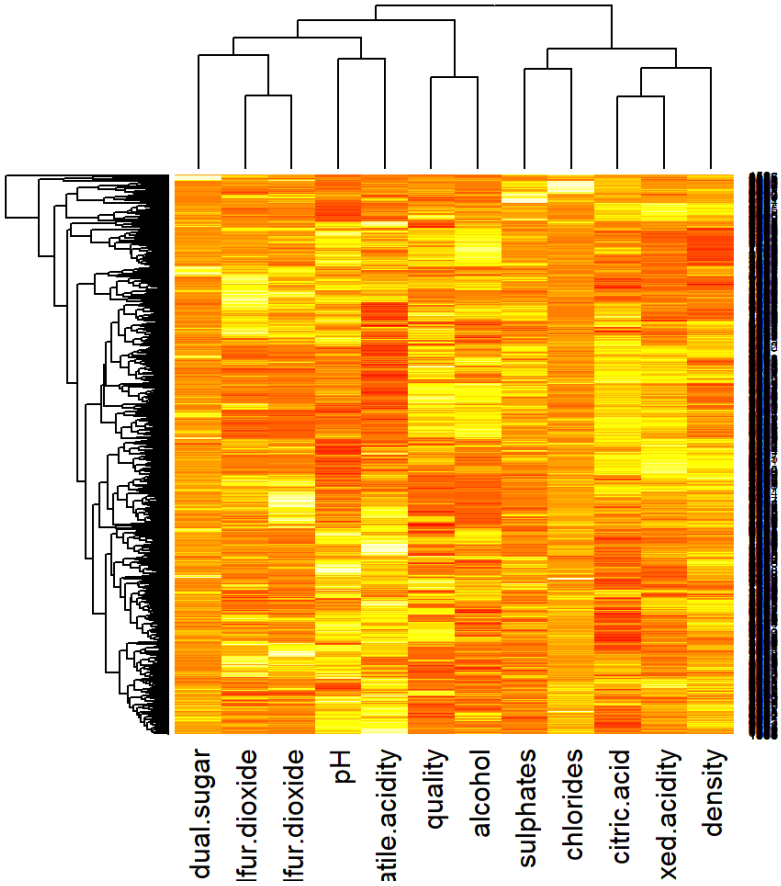
```
# Visualize the tree
fviz_dend(res.hc)
```



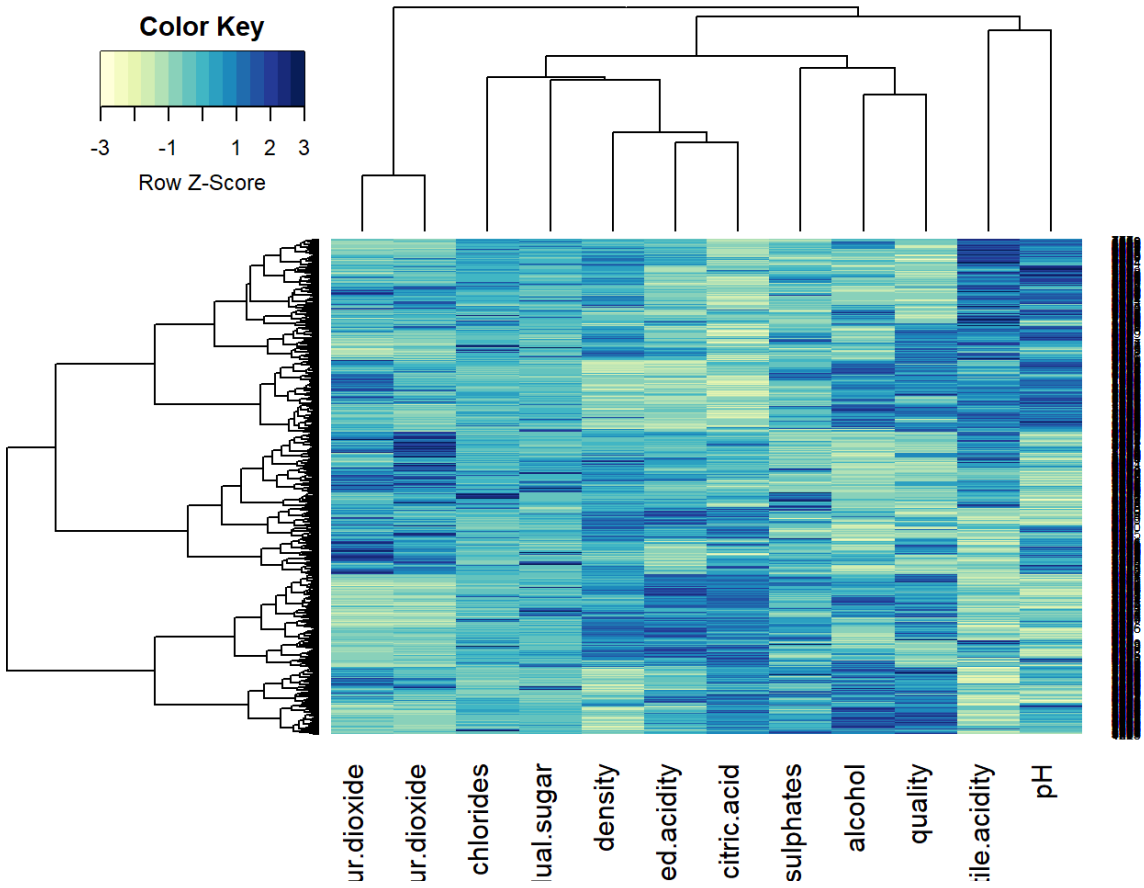

```
# Or simply
plot(res.hc, main = "HC with ward linkage type")
```




```
# Create heatmap with scaled data
heatmap(df_scaled)
```




```
# Make your own color code brewer.pal(n, name)
# n Number of different colors in the palette, minimum 3, maximum depending on pale
tte
# name Blues BuGn BuPu GnBu Greens Greys Oranges OrRd PuBu PuBuGn PuRd Purples RdPu
Reds YlGn YlGnBu YlOrBr YlOrRd
palette <- colorRampPalette(brewer.pal(9, "YlGnBu"))

# Row- and column-wise clustering, with wished linkage and distance method; hc1 wit
h regular data and hc2 with transposed data:
hc1 <- hclust(as.dist(1-cor(t(df_scaled), method="kendall")), method="ward.D2")
hc2 <- hclust(as.dist(1-cor(df_scaled, method="spearman")), method="single")

# Plot heatmap: Heatmap.2 (included in the Gplots Library) allows to define linkage
& distance methods for heatmap representaton
heatmap.2(df_scaled, Rowv=as.dendrogram(hc1), Colv=as.dendrogram(hc2), col=palette,
scale="row", density.info="none", trace="none")
```



7.2 Interpretation

- Dendogram plot: This plot is not very useful. It contains a lot of overloaded information. The height of the cut in the dendogram has comparable role as the number of K in K-means clustering: it controls the number of clusters obtained.
- Transposed Dendogram plot: This plot is way more useful and contains information to work with. The lower the altitude of a branch is, the closer the predictors are to each other. For example fixed.acidity and citric.acid have the lowest branch height which means that these two predictors are closely related to each other. Also free.sulfur.dioxide and total.sulfur.dioxide are closely related to each other which does make sense because both features are the same chemical element (sulfur). The most surprising finding from this transposed dendogram plot is that quality and alcohol concentration of wines seem to be close. It seems like wines of higher quality contain more alcohol than wines of lower quality. Although this does not imply any causality, this could be an important finding for our wine dealer.
- Dissimilarity Matrix: There is not a lot of similarity in the dataset. The diagonal members are defined as zero. This means that zero is the measure of dissimilarity between an element and itself.
- Heatmaps: High values are red, low values are yellow in the plot. The main pattern to look for is a rectangular area of about the same color. That suggests a group of rows that is correlated for the corresponding group of columns.

8 tSNE

8.1 Caluclate tSNE and Show Metrics

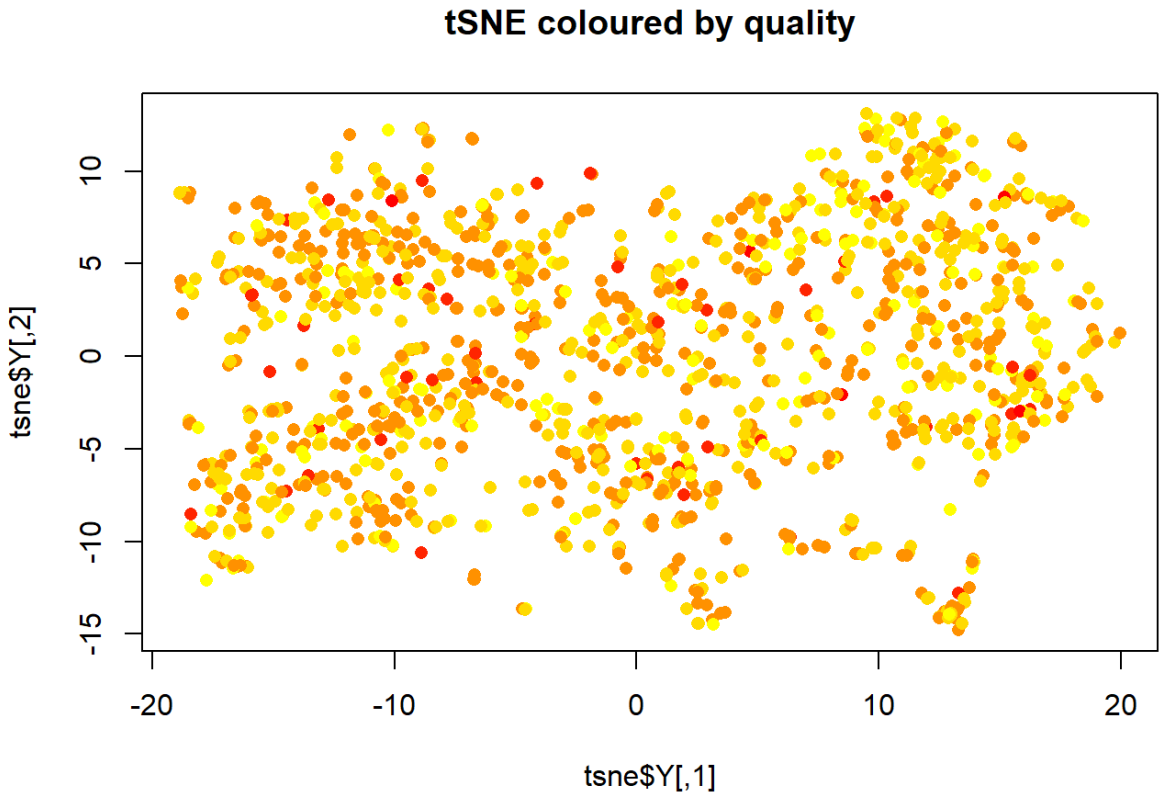
Available metrics in the returend TSNE object

- **dims:** the number of dimensions the data should be reduced to
- **perplexity:** it can be interpreted as a smooth measure of the effective number of neighbours (usuallly between 5 and 50)
- **max_iter:** Maximum iterations
- **Theta:** speed/accuracy trade-off,increase for speed but less accuracy (default 0.5)

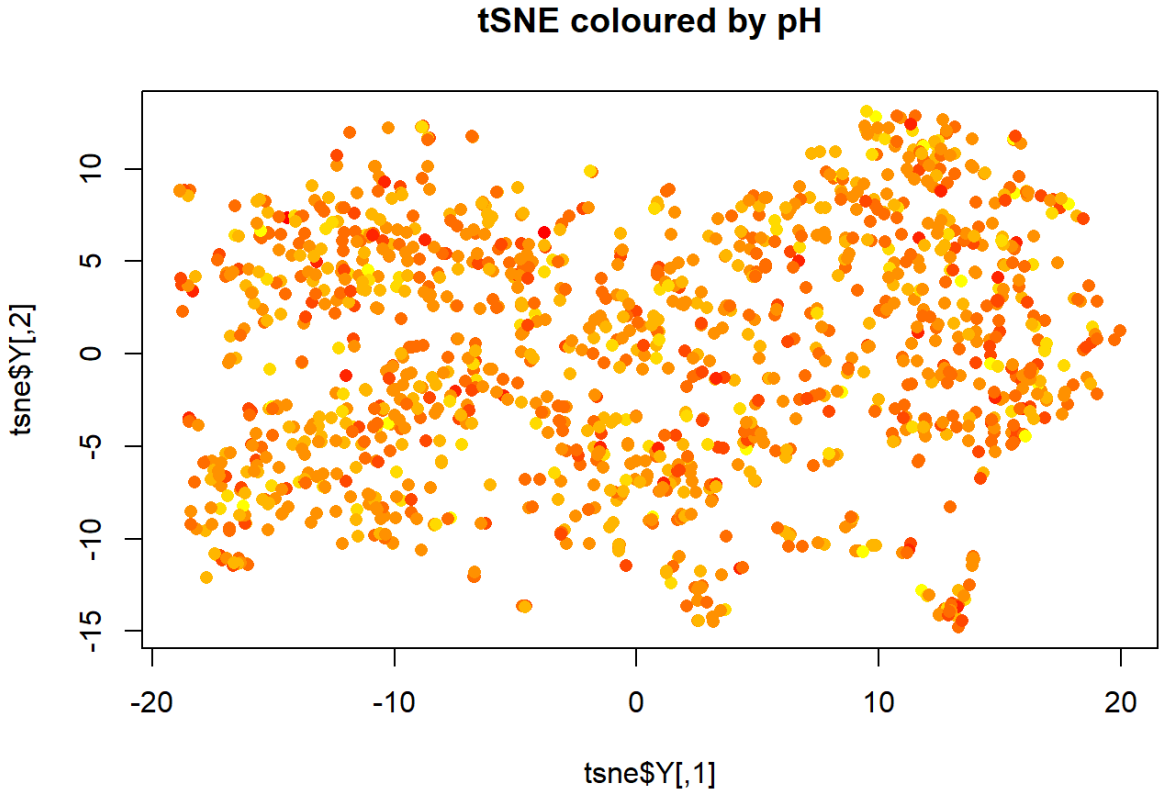
```
# the dateframe needed to be re-defined as dataframe for the code to work properly
df_scaled_df <- as.data.frame(df_scaled)

# datapoint later in the plot: data_label<-as.factor(rownames(data))
# Run tSNE:

# we need to remove duplicates

tsne <- Rtsne(df_scaled_df[!duplicated(df_scaled_df), ], dims = 2,
            perplexity=50, verbose=TRUE,
            max_iter = 500)
```

```
## Performing PCA
## Read the 1359 x 12 data matrix successfully!
## OpenMP is working. 1 threads.
## Using no_dims = 2, perplexity = 50.000000, and theta = 0.500000
## Computing input similarities...
## Building tree...
## Done in 0.33 seconds (sparsity = 0.154832)!
## Learning embedding...
## Iteration 50: error is 67.308577 (50 iterations in 0.20 seconds)
## Iteration 100: error is 67.306637 (50 iterations in 0.28 seconds)
## Iteration 150: error is 66.330376 (50 iterations in 0.22 seconds)
## Iteration 200: error is 66.298861 (50 iterations in 0.17 seconds)
## Iteration 250: error is 66.299056 (50 iterations in 0.18 seconds)
## Iteration 300: error is 1.462193 (50 iterations in 0.21 seconds)
## Iteration 350: error is 1.255665 (50 iterations in 0.16 seconds)
## Iteration 400: error is 1.205653 (50 iterations in 0.16 seconds)
## Iteration 450: error is 1.179469 (50 iterations in 0.16 seconds)
## Iteration 500: error is 1.165929 (50 iterations in 0.17 seconds)
## Fitting performed in 1.90 seconds.
```

```
## 1 ## This first section concerning colouring allows us to colour scaled numeric
 columns
#Create a function to generate a continuous color palette
rbPal <- colorRampPalette(c('red','yellow'))

#This adds a column of color values
# creates 10 colour buckets based on quality
df_scaled_df$Col_qual <- rbPal(8)[as.numeric(cut(df_scaled_df$quality,breaks = 10
))]
# creates 10 colour buckets based on ph
df_scaled_df$Col_pH <- rbPal(8)[as.numeric(cut((df_scaled_df$pH),breaks = 10))]

## 2 ## We can choose colouring to be df_scaled_df$Col or e.g. df$quality
# Plot data and labels:
plot(tsne$Y, col=df_scaled_df$Col_qual, pch=16, main = "tSNE coloured by quality")
```




```
plot(tsne$Y, col=df_scaled_df$Col_pH, pch=16, main = "tSNE coloured by pH")
```




```
str(tsne)
```

```
## List of 14
##  $ N                 : int 1359
##  $ Y                 : num [1:1359, 1:2] -11.21 -4.38 -4.77 11.34 -10.92 ...
##  $ costs             : num [1:1359] 0.000419 0.000976 0.002063 0.000773 0.00042
5 ...
##  $ itercosts         : num [1:10] 67.3 67.3 66.3 66.3 66.3 ...
##  $ origD             : int 12
##  $ perplexity        : num 50
##  $ theta             : num 0.5
##  $ max_iter          : num 500
##  $ stop_lying_iter   : int 250
##  $ mom_switch_iter   : int 250
##  $ momentum          : num 0.5
##  $ final_momentum    : num 0.8
##  $ eta               : num 200
##  $ exaggeration_factor: num 12
```

8.2 Interpretation

- Could not find any meaningful patterns or a sensible groups
- The plots were coloured by pH and quality but could not establish clear relationships
- The visual categorisation does not seem to be useful for this dataset

> The tSNE could not find any meaningful groupings for the wines.

9 Principal Component Analysis (PCA)

9.1 Calculate PCA and Show Metrics

Available metrics in the returend PCA object

- **sdev:** standard deviation of the principal components
- **rotation:** rotation matrix shows the principal components of the loadings (variables).
- **center:** mean of each variable before scaling and computing PCA
- **scale:** applied scale
- **x:** x matrix shows the principal components of the scores (of the observation)

```
# Compute PCA.
# scale=TRUE to scale the variables to have standard deviation = 1
pca_out = prcomp(df, scale=TRUE)

# Show available metrics computed by PCA:
names(pca_out)
```

```
## [1] "sdev"     "rotation" "center"   "scale"    "x"
```

```
# Access metrics computed by PCA
pca_out$sdev    # show sdev
```

```
##  [1] 1.7666827 1.4972916 1.2972739 1.1022799 0.9865412 0.8139977 0.7863319
##  [8] 0.7112472 0.6413326 0.5726425 0.4245216 0.2439629
```

```
pca_out$center  # show mean
```

```
##        fixed.acidity     volatile.acidity          citric.acid
##           8.31963727           0.52782051           0.27097561
##       residual.sugar            chlorides  free.sulfur.dioxide
##           2.53880550           0.08746654          15.87492183
## total.sulfur.dioxide              density                   pH
##          46.46779237           0.99674668           3.31111320
##            sulphates              alcohol              quality
##           0.65814884          10.42298311           5.63602251
```

```
pca_out$scale   # show scale
```

```
##        fixed.acidity     volatile.acidity          citric.acid
##          1.741096318          0.179059704          0.194801137
##       residual.sugar            chlorides  free.sulfur.dioxide
##          1.409928060          0.047065302         10.460156970
## total.sulfur.dioxide              density                   pH
##         32.895324478          0.001887334          0.154386465
##            sulphates              alcohol              quality
##          0.169506980          1.065667582          0.807569440
```

```
# Rotation matrix provides the principal component of the loadings.
dim(pca_out$rotation) # p*p matrix whereas p is the number of variables (loadings)
```

```
## [1] 12 12
```

```
round(pca_out$rotation, digit=2) # show loadings for each predictor. Round result o
n 2 digits.
```

```
##                         PC1   PC2   PC3   PC4   PC5   PC6   PC7   PC8   PC9
## fixed.acidity          0.49  0.00  0.16  0.23 -0.08  0.06 -0.31  0.20 -0.17
## volatile.acidity      -0.27  0.34  0.23 -0.04  0.30  0.30 -0.63  0.15 -0.06
## citric.acid            0.47 -0.14 -0.10  0.06 -0.12  0.14  0.24  0.30 -0.22
## residual.sugar         0.14  0.17 -0.24  0.38  0.71  0.11  0.28 -0.17  0.28
## chlorides              0.20  0.19  0.03 -0.65  0.27  0.34  0.23 -0.19 -0.42
## free.sulfur.dioxide   -0.05  0.26 -0.62  0.03 -0.16 -0.04 -0.14 -0.02 -0.32
## total.sulfur.dioxide  0.00  0.36 -0.54  0.03 -0.22  0.12 -0.11  0.09  0.12
## density                0.37  0.33  0.17  0.20  0.21 -0.43 -0.12  0.08 -0.25
## pH                    -0.43 -0.07 -0.07  0.01  0.26 -0.48  0.19  0.31 -0.46
## sulphates              0.25 -0.11 -0.21 -0.56  0.21 -0.40 -0.23  0.28  0.45
## alcohol               -0.07 -0.50 -0.22  0.09  0.26  0.39 -0.12  0.47 -0.10
## quality                0.11 -0.47 -0.22  0.04  0.14 -0.14 -0.41 -0.61 -0.24
##                         PC10  PC11  PC12
## fixed.acidity          0.18 -0.26  0.64
## volatile.acidity      -0.16  0.38  0.00
## citric.acid           -0.35  0.62 -0.07
## residual.sugar         0.05  0.09  0.18
## chlorides              0.00 -0.21  0.05
## free.sulfur.dioxide    0.59  0.24 -0.05
## total.sulfur.dioxide -0.59 -0.36  0.07
## density               -0.04 -0.23 -0.57
## pH                    -0.21 -0.01  0.34
## sulphates              0.07  0.10  0.07
## alcohol                0.11 -0.32 -0.32
## quality               -0.26  0.05  0.01
```

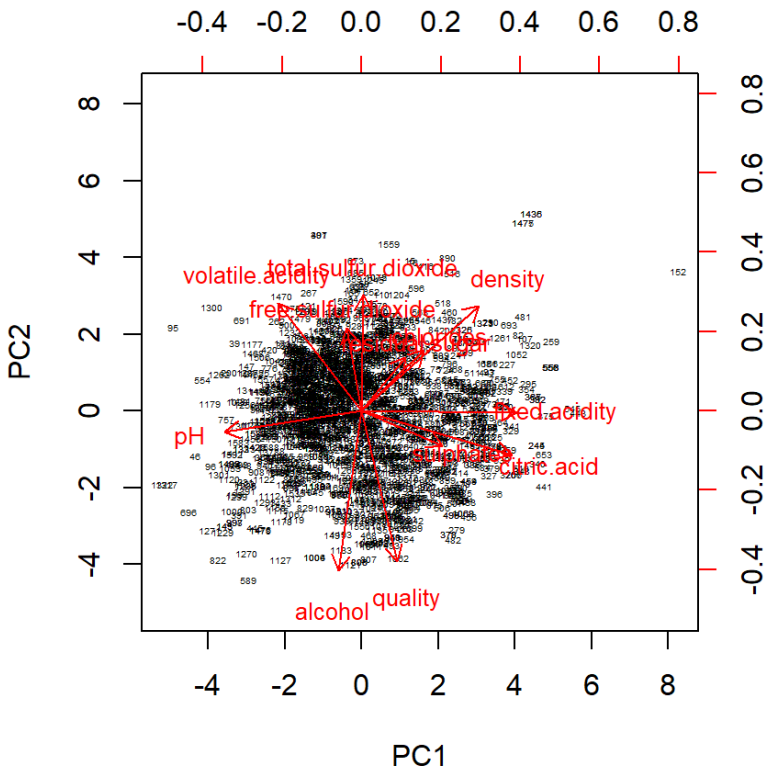
```
# x matrix provides the principal component of the scores.
dim(pca_out$x) # n * p matrix whereas n is the number of observations and p is the
 number of principal componentes
```

```
## [1] 1599   12
```

```
# pca_out$x # not executed due to size of the matrix
```

9.2 Create Biplot

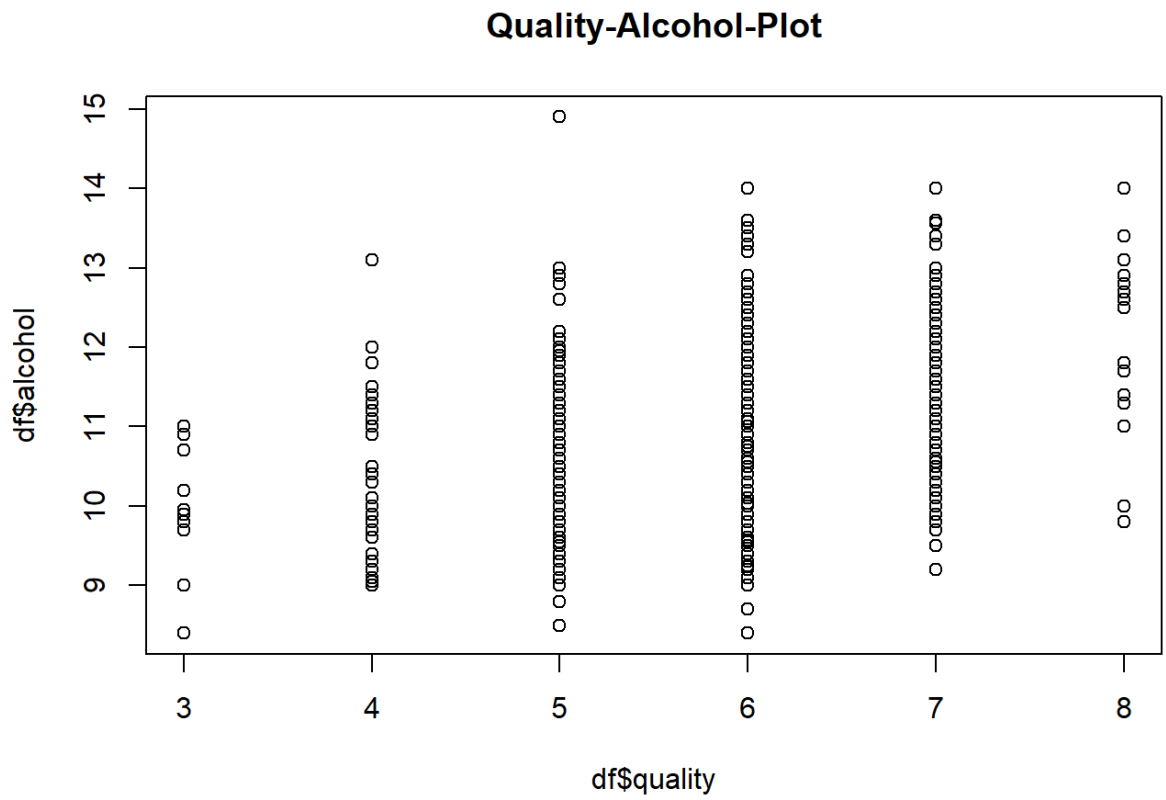
```
# Create Biplot
# scale=0 ensures that the arrows are scaled to represent the loadings;
# other values for scale give slightly different biplots with different interpretat
ions.
# cex (character expension factor): configures the labeling size of the observation
s (black) and the predictors (red)
# cex: reduze labeling size of the observations to 0.5 in order to make the plot mo
re readable.
biplot(pca_out,scale=0, cex=c(0.3,0.8))
```



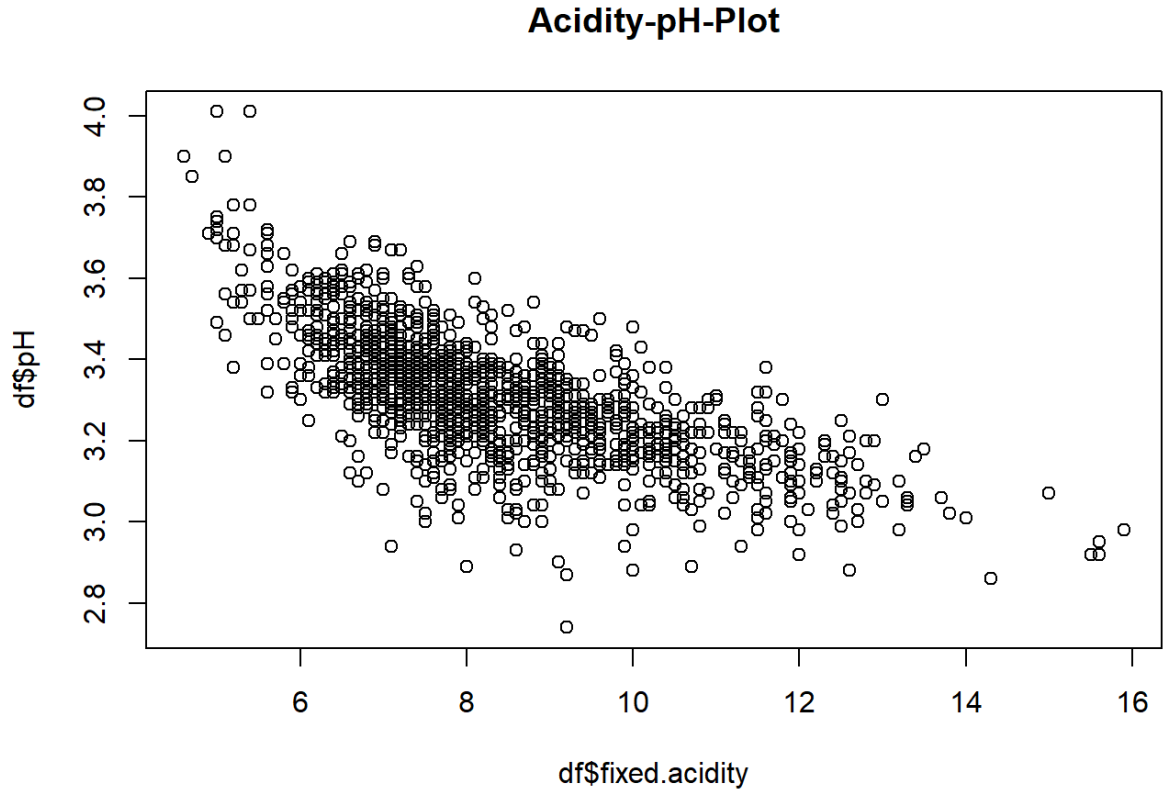
9.3 Plausibility check on Biplot

Those plots are used to check the plausibility of the biplot. Can the correlations discovered in the biplot, also be observed in a xy-plot of the predictors? It turns out in most cases it can. From the four plots below, the only plot which does not fully support the correlation discovered in the biplot, is the Quality-Sulfur-Plot. The Quality-Sulfur-Plot indicates that low and high quality wines usually have a low total sulfur dioxid, whereas mid-range quality wines have a lot more variation in the total sulfur dioxid.

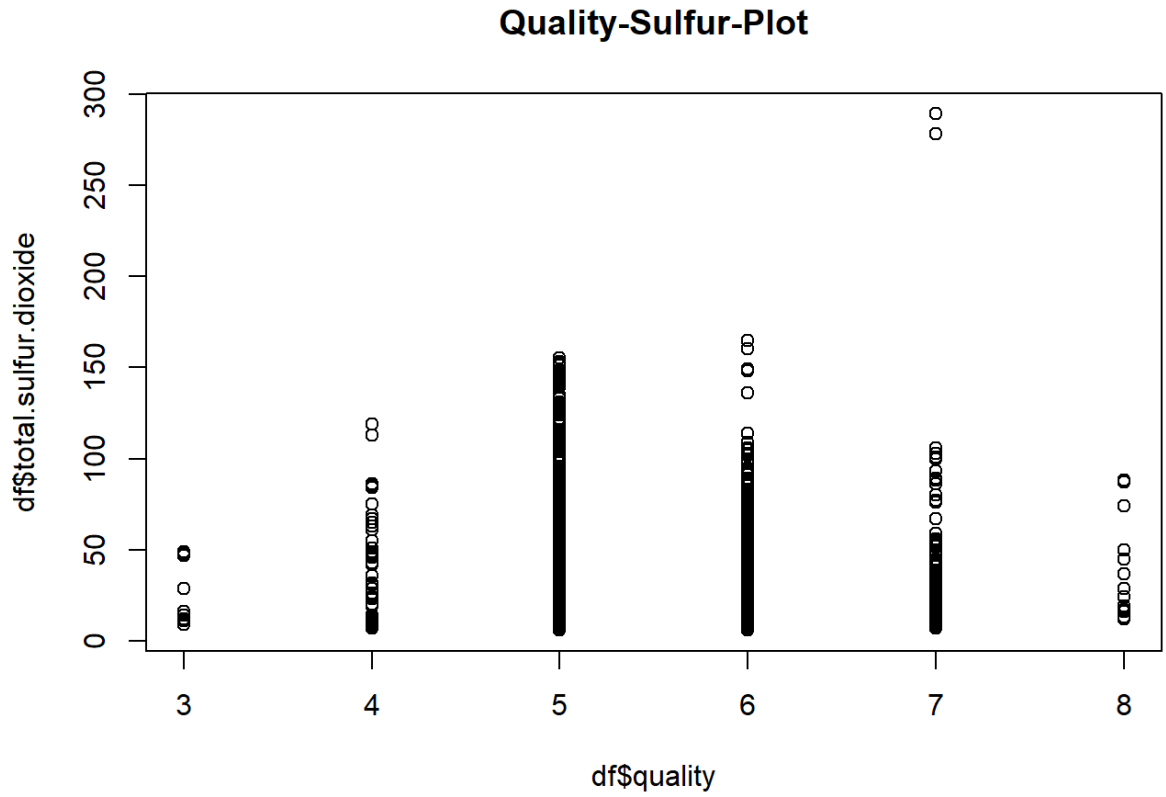
```
plot(df$quality,df$alcohol, main="Quality-Alcohol-Plot")
```



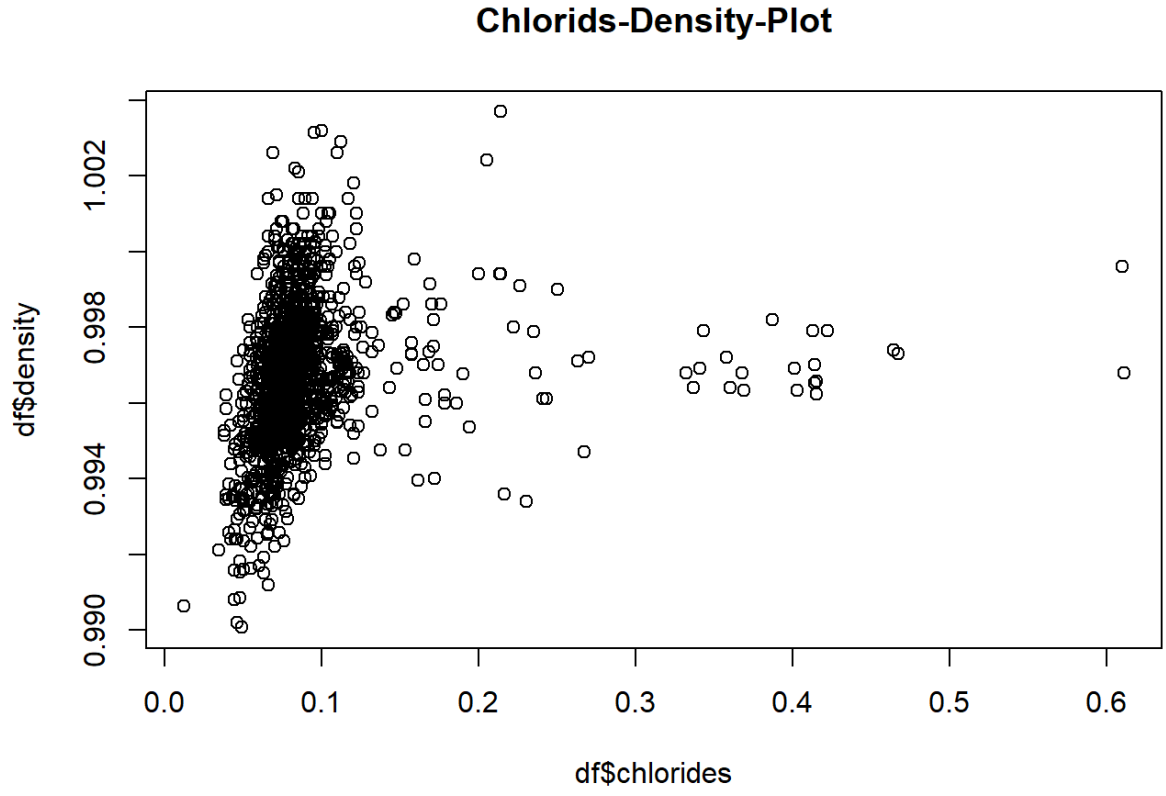

```
plot(df$fixed.acidity,df$pH, main="Acidity-pH-Plot")
```




```
plot(df$quality,df$total.sulfur.dioxide, main="Quality-Sulfur-Plot")
```




```
plot(df$chlorides,df$density, main="Chlorids-Density-Plot")
```



9.4 Show variance explained by certain Principal Components

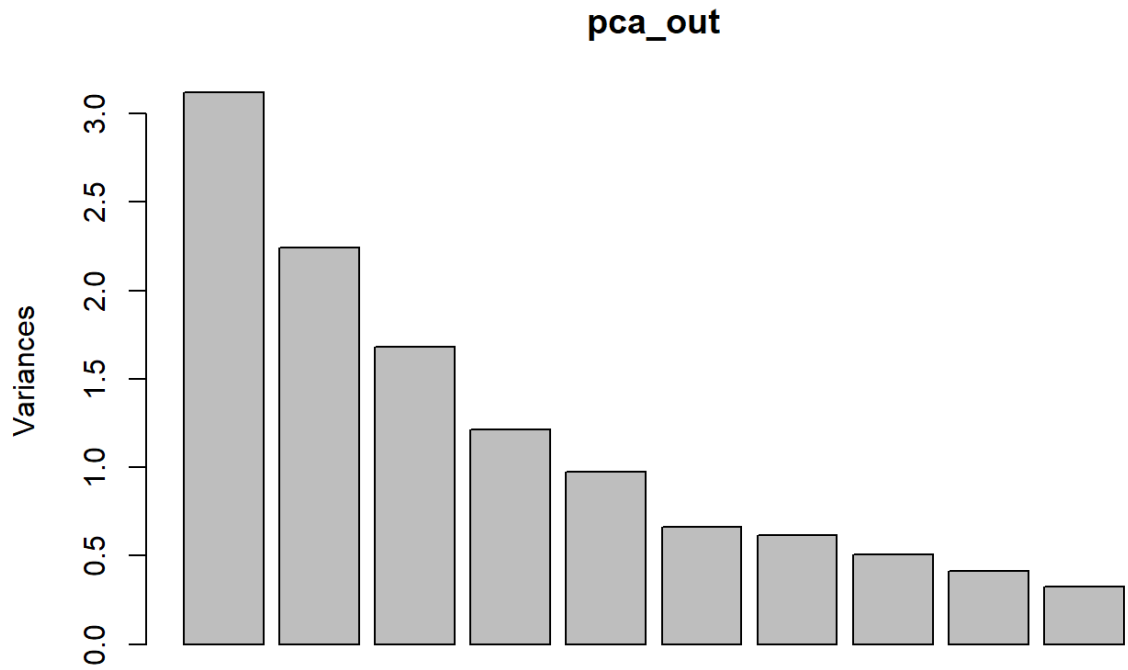
summary() on a prcomp object:

Summary shows the standard deviation, proportion of variance and cumulative proportion of variance of each principal component

Result:

From the Plot “Cumulative Proportion of Variance per PC” once can see that 6 Principal Components explain more than 80% of the variance in the data.

```
# screeplot shows the variance of each prinicipal component
screeplot(pca_out)
```



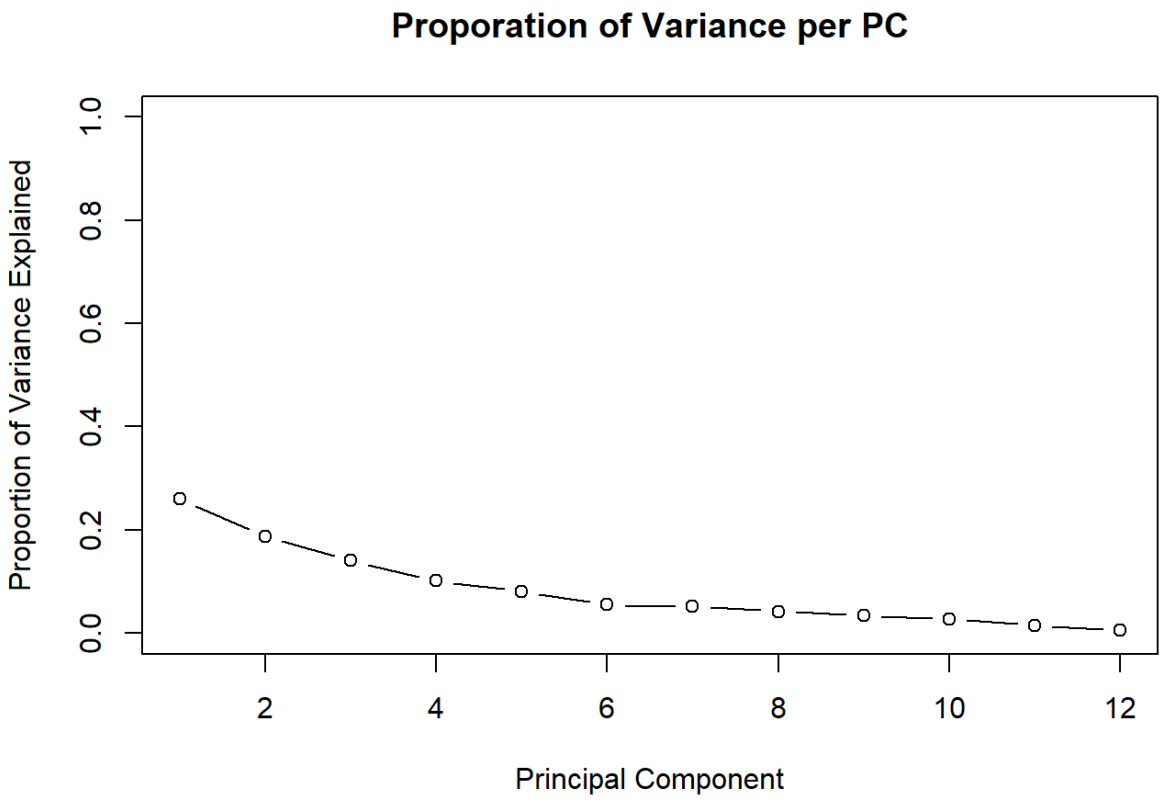

```
# Alternative:
# It's more usueful to have the proportion (%) of variance of each principal compon
ent

# Numeric values about proportion of variance can be retrieved with the summary fun
ction
summary(pca_out)
```

```
## Importance of components:
##                           PC1    PC2    PC3    PC4     PC5     PC6     PC7
## Standard deviation     1.7667 1.4973 1.2973 1.1023 0.98654 0.81400 0.78633
## Proportion of Variance 0.2601 0.1868 0.1402 0.1013 0.08111 0.05522 0.05153
## Cumulative Proportion  0.2601 0.4469 0.5872 0.6884 0.76952 0.82474 0.87626
##                           PC8     PC9    PC10    PC11    PC12
## Standard deviation     0.71125 0.64133 0.57264 0.42452 0.24396
## Proportion of Variance 0.04216 0.03428 0.02733 0.01502 0.00496
## Cumulative Proportion  0.91842 0.95270 0.98002 0.99504 1.00000
```

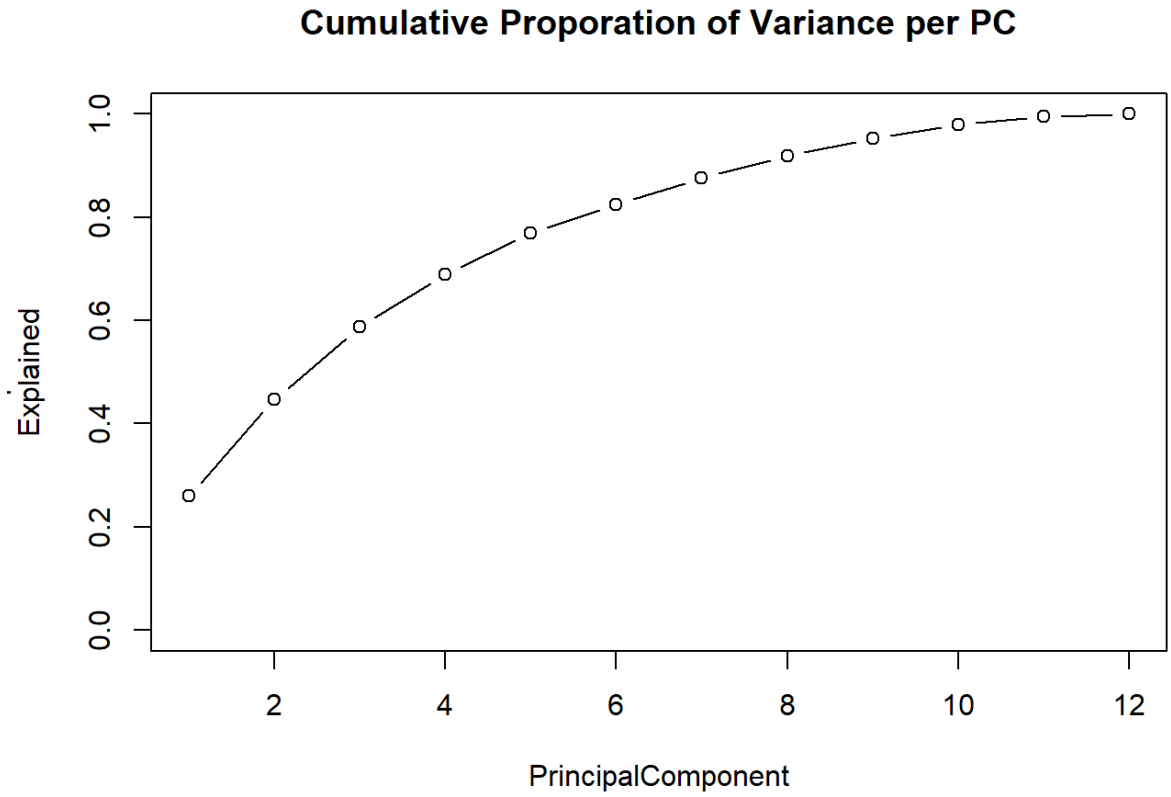
```
# Compute the proportion of variance explained by each principal component
# Calculation: variance explained by each principal component / total variance expl
ained by all principal components)

# Calculate variance of prinicipal components
pca_out_var <- pca_out$sdev^2

# calculate proportion of variance for each principal component
pve <- pca_out_var/sum(pca_out_var)
pve
```

```
##  [1] 0.260097308 0.186823504 0.140243308 0.101251739 0.081105302
##  [6] 0.055216020 0.051526483 0.042156046 0.034275628 0.027326616
## [11] 0.015018219 0.004959826
```

```
# plots
plot(pve, main="Proporation of Variance per PC",xlab="Principal Component",ylab="Pr
oportion of Variance Explained",ylim=c(0,1),type='b')
```




```
plot(cumsum(pve),main="Cumulative Proporation of Variance per PC",xlab="PrincipalCo
mponent",ylab="Cumulative Proportion of Variance
Explained",ylim=c(0,1),type='b')
```



9.5 Interpretation

- PC2 mostly represents quality, alcohol content and sulfur content of a wine. Alcohol and qulity appear to be positivly correlated. So wines with a higher alcohol content tend to have a higher quality rating. But based on this we cannot conculde any causal relationship between the alcohol content of a wine and its quality. There might be other confounding factors affecting both variables.
- Wines with a high quality rating and more alcohol are located at the bottom of the biplot.
- Wines with a high sulfur content are respresented at the top of the biplot.
- PC1 mostly represents how acidic or basic (alkaline) a wine is. All predictors related to acidity (fixed.acidity (0.49), volatile acidity (-0.27), citric.acid (0.47) adn pH (-0.43)) have relatively high loadings for PC1. Fixed.acitiy and pH value point in the opposite direction in the biplot (negatively correlated). pH value is a metric to measure the acidity. A low pH value indicates high acidity whereas a high pH value indicate a low acidity (alkaline). So the result in the biplot is conclusive.
- Although there are no clear clusters from the biplot, the dealer could use PC1 as a scoring for the acidity of a wine. The left side of the biplot represent the morebasic (alkaline) wines, whereas the right side represents the more acidic wines.
- Also the dealer could use the biplot as a visual instrument to select high quality wines that are either acidic or basic (alkaline). Also if a customer prefers a wine with high alcohol content, the dealer could select a wine from the bottom of the plot. The dealer could furthermore ask the customer if they would prefer an acidic or a base wine. Based on the customers preference, the dealer would either select a wine from the bottom left or bottom right.

10 Self-Organizing Maps (SOMs)

Purpose: Data visualization of high-dimensional data. How: Mapping from a higher-dimensional input space to a lower-dimensional map space with competitive learning. Neural network uses competetive learning.

10.1 Calculate SOMs and Show Metrics

Available metrics in the returend PCA object

- **unit.classif:** winning units for all data objects
- **distances:** distances of objects to their corresponding winning unit
- **grid:** number of neurons in grid (two dimensional map)
- **rlen:** Number of iterations, the number of times the complete data set will be presented to the network
- **alpha:** Learning rate, a vector of two numbers indicating the amount of change. Default is to decline linearly from 0.05 to 0.01 over rlen updates

More metrics can be found with ?som.

Central Caveats:

- Influence/choice of parameters not very clear
- Not building generative model (not ???understanding??? the data, cannot reproduce similar ones) mapping

```
# preprocess data for model
df_unique <- unique(df)  # remove duplicates, here 200 data rows
glimpse(df_unique)
```

```
## Observations: 1,359
## Variables: 12
## $ fixed.acidity        <dbl> 7.4, 7.8, 7.8, 11.2, 7.4, 7.9, 7.3, 7.8, ...
## $ volatile.acidity     <dbl> 0.700, 0.880, 0.760, 0.280, 0.660, 0.600,...
## $ citric.acid          <dbl> 0.00, 0.00, 0.04, 0.56, 0.00, 0.06, 0.00,...
## $ residual.sugar       <dbl> 1.9, 2.6, 2.3, 1.9, 1.8, 1.6, 1.2, 2.0, 6...
## $ chlorides            <dbl> 0.076, 0.098, 0.092, 0.075, 0.075, 0.069,...
## $ free.sulfur.dioxide  <dbl> 11, 25, 15, 17, 13, 15, 15, 9, 17, 15, 16...
## $ total.sulfur.dioxide <dbl> 34, 67, 54, 60, 40, 59, 21, 18, 102, 65, ...
## $ density              <dbl> 0.9978, 0.9968, 0.9970, 0.9980, 0.9978, 0...
## $ pH                   <dbl> 3.51, 3.20, 3.26, 3.16, 3.51, 3.30, 3.39,...
## $ sulphates            <dbl> 0.56, 0.68, 0.65, 0.58, 0.56, 0.46, 0.47,...
## $ alcohol              <dbl> 9.4, 9.8, 9.8, 9.8, 9.4, 9.4, 10.0, 9.5, ...
## $ quality              <int> 5, 5, 5, 6, 5, 5, 7, 7, 5, 5, 5, 5, 5, 5,...
```

```
data <-
  as.matrix(scale(df_unique[, 1:11]))  # scale: mean = 0, sd = 1

# show feature names
dimnames(data)[2]  # label successfully removed
```

```
## [[1]]
##  [1] "fixed.acidity"        "volatile.acidity"     "citric.acid"
##  [4] "residual.sugar"       "chlorides"            "free.sulfur.dioxide"
##  [7] "total.sulfur.dioxide" "density"              "pH"
## [10] "sulphates"            "alcohol"
```

```
# For plotting evaluation against colorcode
# category (~ classification solution)
row_label <- as.factor(rownames(data))
# qualities <- as.character(df$quality)
colors <- c("empty", "empty", "violet", "green", "blue", "red", "orange", "black")
colors <- colors[df$quality]

data_train_matrix <- as.matrix(scale(data))
```

Removed wine quality label and transformed to scaled matrix which serves as input for the SOM model.

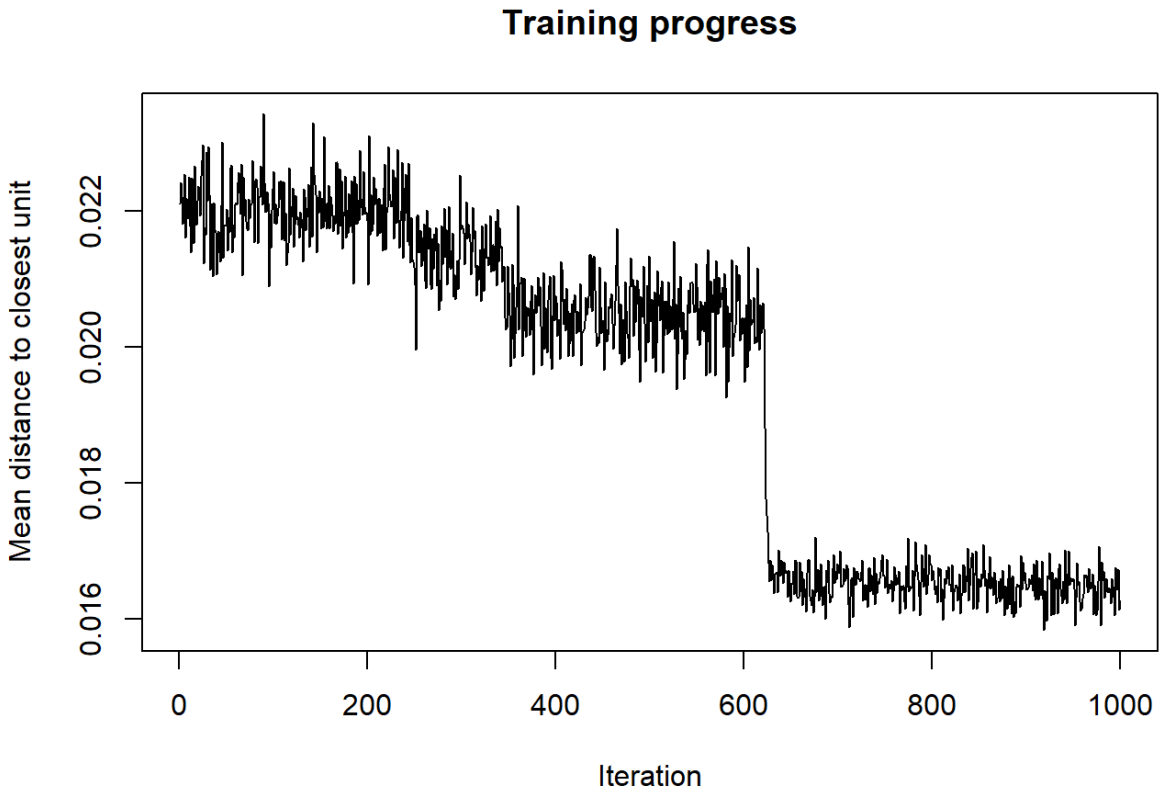
General tipps and hints regarding usage of SOMs:

- Hyperparameter tuning: try & error
- Codes plot shows feature importance per neuron
- Mapping plot shows logical cluster which could contain several labels/classes

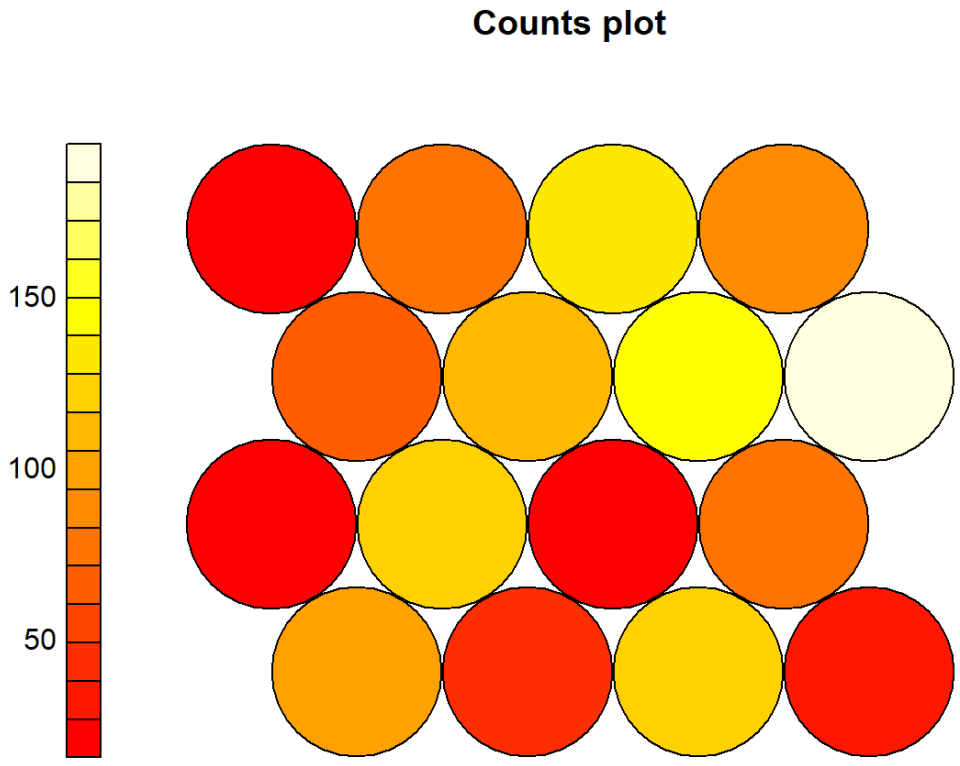
```
# set up model
set.seed(22)  # for reproducible results

# get heuristic number of neurons for the following grid
# use this heuristic from Alexander Maier https://www.researchgate.net/post/How_many_nodes_for_self-organizing_maps

neurons <- 5*sqrt(nrow(df))
neuron_per_grid <- round(sqrt(neurons))

# Define the neuronal grid
som_grid <- somgrid(xdim = neuron_per_grid, ydim = neuron_per_grid,
                    topo = "hexagonal")

# this grid is not really usable because the interpretation is not really possible, so one would choose a really low level for the neurons

# Redefine the neuronal grid
som_grid <- somgrid(xdim = 4, ydim = 4,
                    topo = "hexagonal")

# Train the model
som_model <- som(
  data_train_matrix,
  grid = som_grid,
  rlen = 1000,  # number of iterations, tried with 10'000 and 100'000 but with no significant better result. Always approches minimum after two third of the iterations
  alpha = c(0.05, 0.01),  # learning rate
  keep.data = TRUE
)

summary(som_model)
```

```
## SOM of size 4x4 with a hexagonal topology and a bubble neighbourhood function.
## The number of data layers is 1.
## Distance measure(s) used: sumofsquares.
## Training data included: 1359 objects.
## Mean distance to the closest unit in the map: 4.076.
```

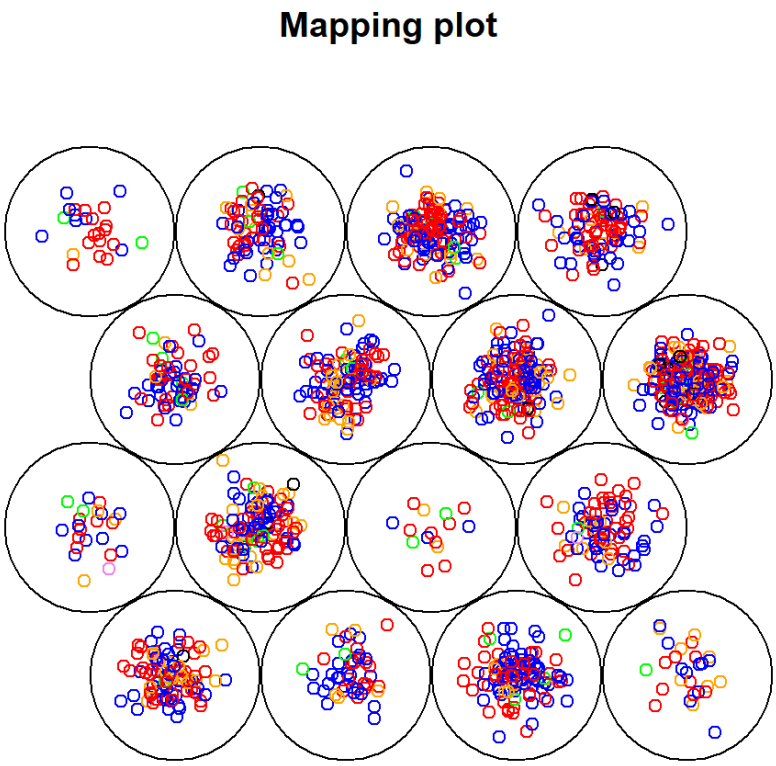
```
# Check training progress
options(scipen = 999)  # for better reading
plot(som_model, type = "changes")
```



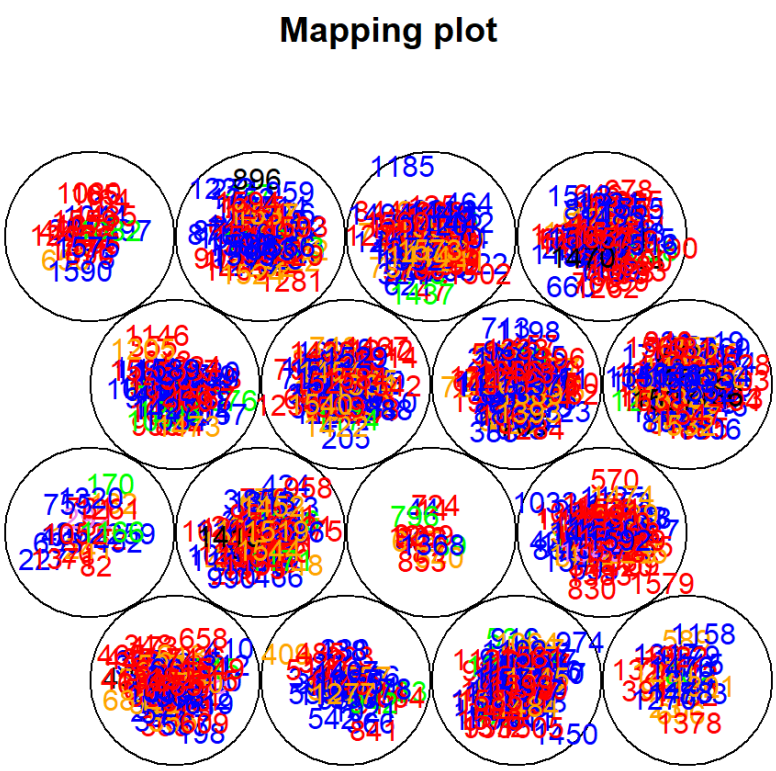

```
# Check how many samples are mapped to each
# node on the map. (5-10 samples per node)

# Explore training results
plot(som_model, type = "count")  # how many datapoints are in a neuron
```



```
plot(som_model, type = "mapping",  # show datapoints per neuron, 5 - 10 datapoints
     col = colors[row_label])      # per neuron is the target range
```




```
# one color per wine quality

plot(  # show sample of datapoints, wine quality and number of row
  som_model,
  type = "mapping",
  labels = (rownames(data)),
  col = colors[row_label]
)
```



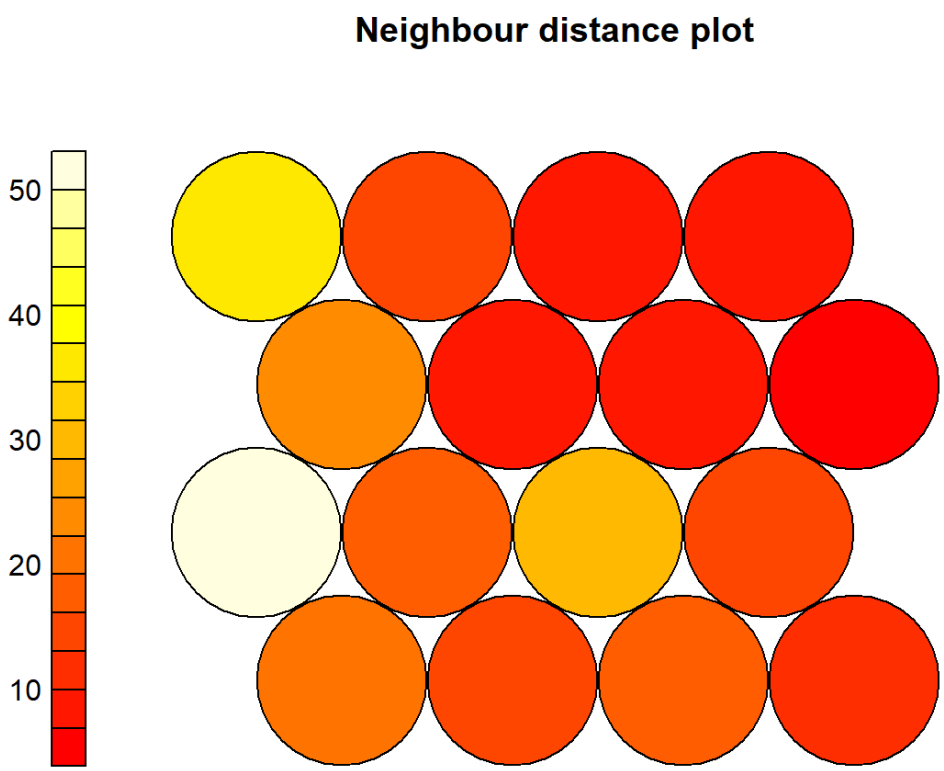
10.2 Results interpretation part one

Model description: SOM of size 4x4 with a hexagonal topology and a bubble neighbourhood function. The number of data layers is 1. Distance measure(s) used: sumofsquares. Training data included: 1359 objects. Mean distance to the closest unit in the map: 4.076.

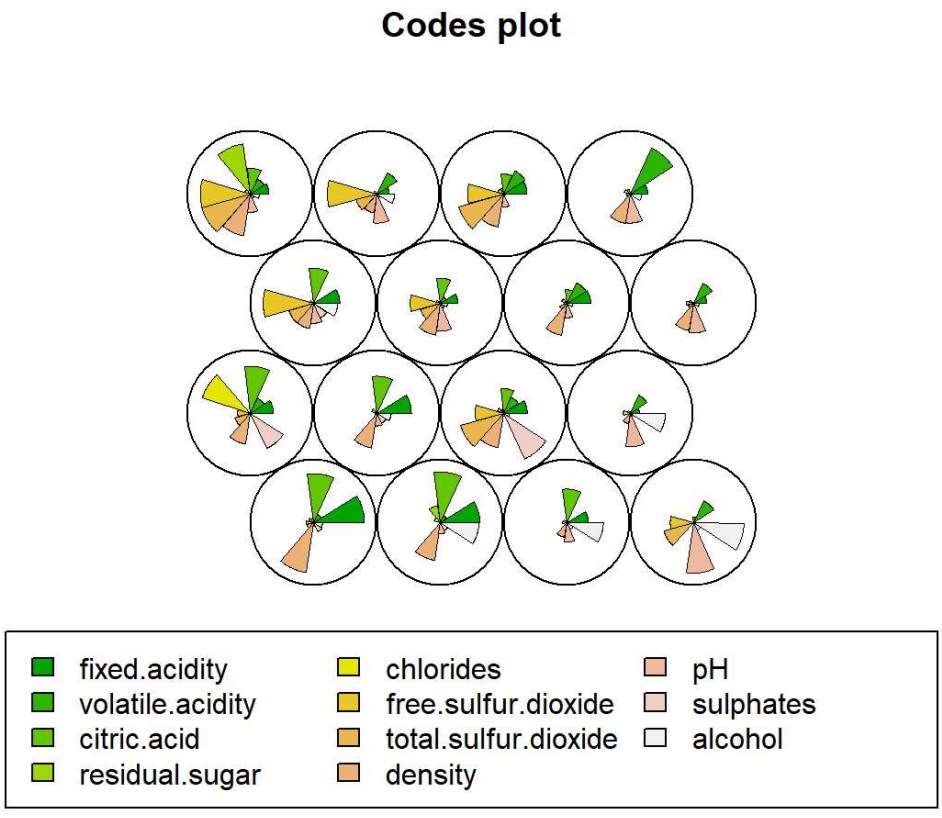
The Counts and Mapping plot show two cluster of neurons: One with a lot of data points (n > 75) and the other with few data points (n <= 75). Wine quality legend for the colors:

- 3: “violet”
- 4: “green”
- 5: “blue”
- 6:“red”
- 7: “orange”
- 8: “black”

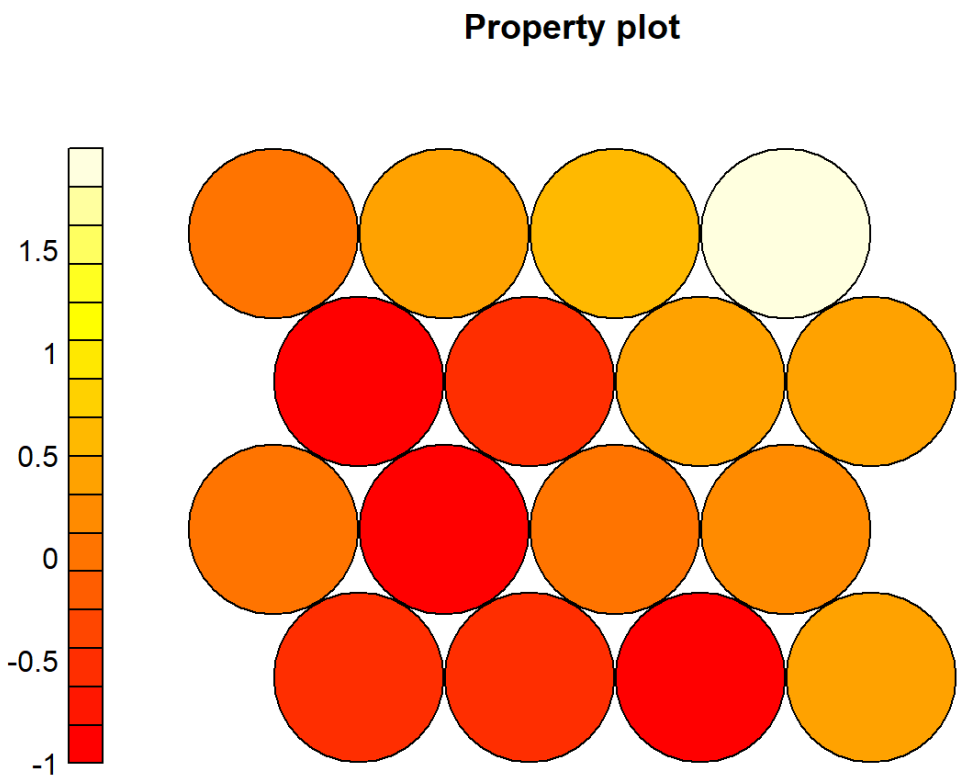
```
# U-Matrix: measure of distance between each node and its neighbours.
# (Euclidean distance between weight vectors of neighboring neurons)
# Can be used to identify clusters/boundaries within the SOM map.
# Areas of low neighbour distance ~ groups of nodes that are similar.
plot(som_model, type = "dist.neighbours")
```



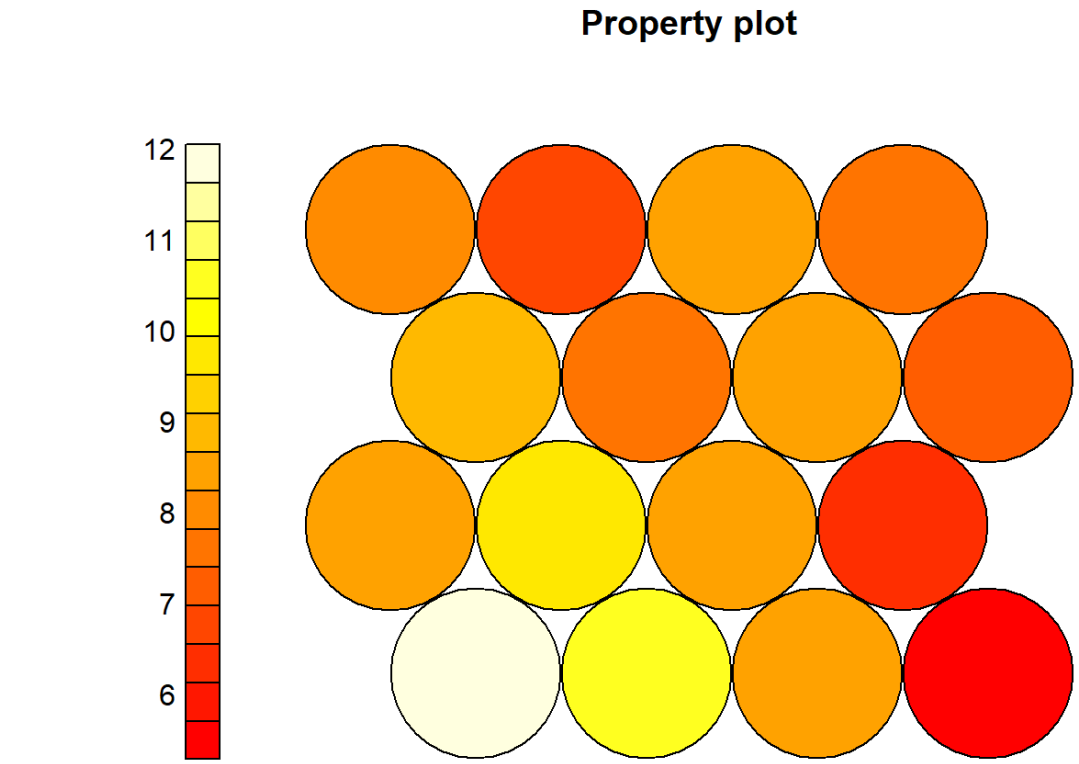

```
# Codes / Weight vectors: representative of the samples mapped to a node.
# highlights patterns in the distribution of samples and variables.
plot(som_model, type = "codes")
```



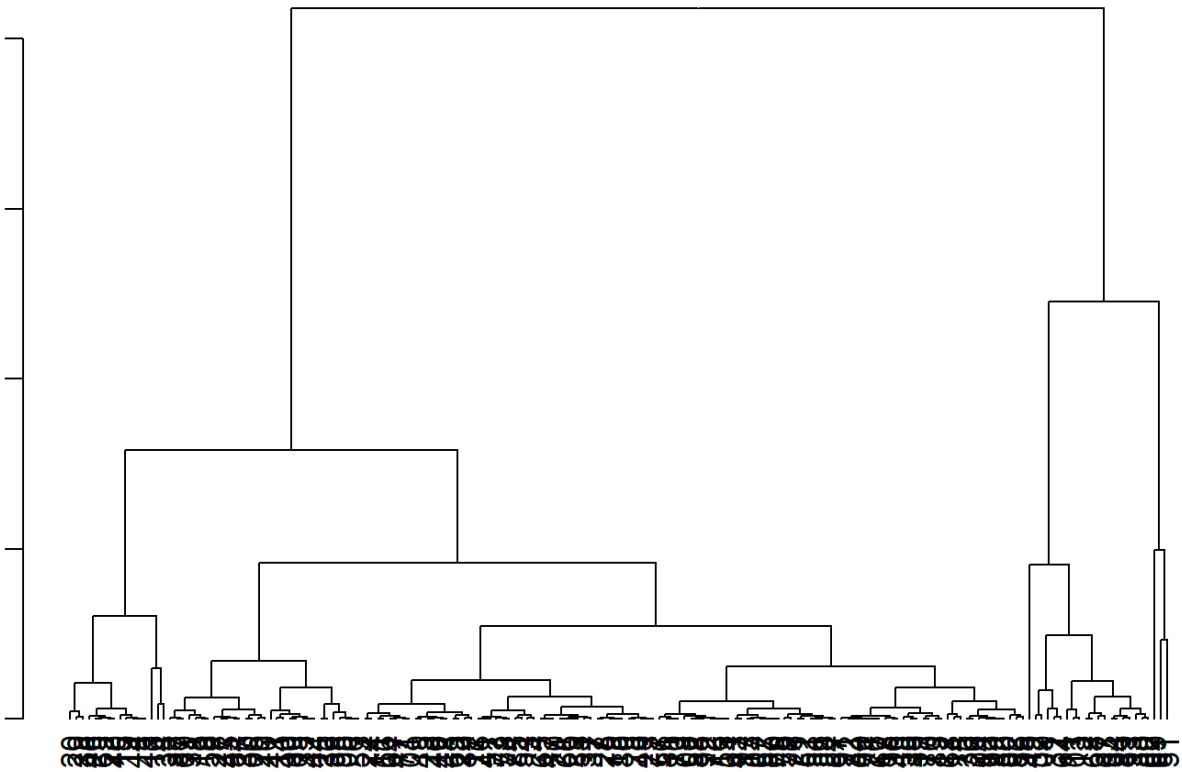
```
# Heatmaps: identify interesting areas on the map.
# Visualise the distribution of a single variable (defined in [,x]) across the map
plot(som_model,
     type = "property",
     property = getCodes(som_model, 1)[, 2])
```



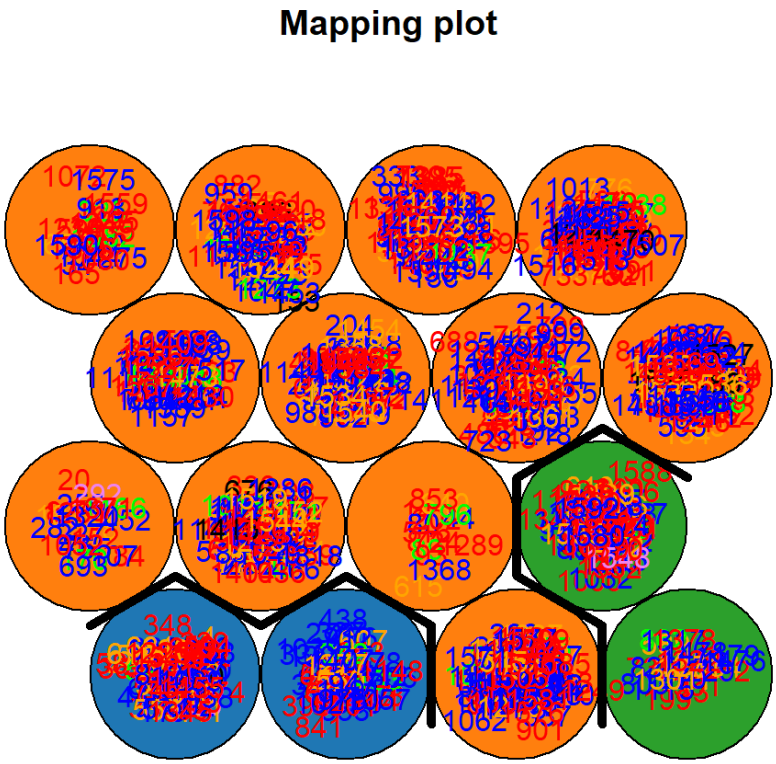

```
# Same as above but with original, unscaled data (can also be useful)
var_unscaled <- aggregate(
  as.numeric(df_unique[, 1]),
  by = list(som_model$unit.classif),
  FUN = mean,
  simplify = TRUE
)[, 2]
plot(som_model, type = "property", property = var_unscaled)
```




```
# Clustering: isolate groups of samples with similar metrics
tree <-
  as.dendrogram(hclust(dist(as.numeric(
    unlist(som_model$codes)
  ))))
plot(tree, ylab = "Height (h)")
```



```
# Cut the tree somewhere based on the above tree
som_cluster <-
  cutree(hclust(dist(as.numeric(
    unlist(som_model$codes)
  ))),
  h = 2) # k groups or at h hight

# Visualize mapping based on HC
pretty_palette <- c("#1f77b4",
                    '#ff7f0e',
                    '#2ca02c',
                    '#d62728',
                    '#9467bd',
                    '#8c564b',
                    '#e377c2')
plot(
  som_model,
  type = "mapping",
  labels = (rownames(data)),
  bgcol = pretty_palette[som_cluster],
  col = colors[row_label]
)
add.cluster.boundaries(som_model, som_cluster)
```



10.3 Final interpretation: what are possible wine clusters?

The Codes plot shows the importance of each feature within a single neuron. The closer a value is to the edge of the circle, the more distinctive the feature is. There seems to be just a few neurons with high feature values (~6). The dendogramm contains too many data points and is therefore not suited for analytical purposes.

The mapping plot indicates how many clusters are reasonable given the trained model. There could be several classes per cluster. Three new possible clusters can be proposed:

- The blue cluster shows 12.5% of all neurons. It contains low data points (see “Counts plot”) with strong characteristics for citric acid and density (check “Codes plot”). This cluster contains wines with a lot of “freshness” and flavor.
- The green cluster shows 12.5% of all neurons as well. It contains low data points as well (see “Counts plot”) with strong values for the variables in pH and alcohol (check “Codes plot”). These wines tend to be base and “palatable”.
- The orange cluster contains 75% of all neurons. It containsmore than 80% (check “Codes plot”) of all data points with partially strong characteristics for fixed acidity, residual sugar, chlorides and sulphates (check “Codes plot”). These wines seem pretty sweet and salty at the same time. The sweetness should remain a long time thanks to the high sulphate values.

> **outlook:** The wine dealer could wish to get more details about the single neurons within the three clusters. One could test more grid sizes, respectively the number of neurons as input for the SOM and try to get even more insights.

11 Conclusion

For the wine dealer there are following take-aways concerning the pre-selection of his wines. In the Hierarchical Clustering Transposed Dendogram plot the most surprising finding is that the quality and alcohol content of wine seem to be close. It seems like wines of higher quality contain more alcohol than wines of lower quality. Although this does not imply any causality, this could be an important finding for our wine dealer.
Additionally, the dealer could use the Principal Component Analysis (PCA) biplot as a visual instrument to select high quality wines that are either acidic or base (alkaline). Also, if a customer prefers a wine with high alcohol content, the dealer could select a wine from the bottom of the plot. The dealer could furthermore ask the customers if they would prefer an acidic or base wine. Based on the customers preference, the dealer would either select a wine from the bottom left or bottom right of the PCA biplot. The wine dealer could further cluster his wine collection according to the Self-Organising Maps (SOM). The following wine segments could be created: “Fresh and flavorable” wines corresponding to wines with strong citric acid and density characteristics (blue SOM cluster). “Intense and soothing” for wines with high pH and alcohol levels (green SOM cluster). “Rich and long-lasting wines” for wines with partially strong characteristics for fixed acidity, residual sugar, chlorides and sulphates (orange SOM cluster).