Machine learning on Wine data using CLASSIFICATION

Harrie

22/05/2021

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --

## v ggplot2 3.3.3     v purrr   0.3.4
## v tibble  3.1.1     v dplyr   1.0.5
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.0

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(tidymodels)
```

```
## -- Attaching packages --------------------------------------- tidymodels 0.1.2 --

## v broom     0.7.3      v recipes   0.1.15
## v dials     0.0.9      v rsample   0.0.8
## v infer     0.5.3      v tune      0.1.2
## v modeldata 0.1.0      v workflows 0.2.1
## v parsnip   0.1.4      v yardstick 0.0.7

## -- Conflicts ------------------------------------------ tidymodels_conflicts() --
## x scales::discard() masks purrr::discard()
## x dplyr::filter()   masks stats::filter()
## x recipes::fixed()  masks stringr::fixed()
## x dplyr::lag()      masks stats::lag()
## x yardstick::spec() masks readr::spec()
## x recipes::step()   masks stats::step()
```

```
library(caret)
```

```
## Loading required package: lattice

##
## Attaching package: 'caret'
```

```
## The following objects are masked from 'package:yardstick':
##
##     precision, recall, sensitivity, specificity

## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(rpart)
```

```
##
## Attaching package: 'rpart'

## The following object is masked from 'package:dials':
##
##     prune
```

```
library(randomForest)
```

```
## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:dplyr':
##
##     combine

## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
library(visdat)
library(GGally)
```

```
## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
```

```
library(gt)
library(skimr)
library(corrplot)
```

```
## corrplot 0.84 loaded
```

```
library(visNetwork)
library(sparkline)
```

INTRODUCTION

`Tidymodels` is the successor to the `caret` package which is used during the *Introduction to Data Science*-course of the Harvard University (Kuhn & Johnson, 2013; Irizarry, 2020). `Tidymodels` is a collection of modeling packages that, like the `tidyverse`, have consistent API and are designed to work together specifically to support predictive analytics and machine learning. Different books (Kuhn & Silge, 2021; Kuhn en Johnson, 2019) blogs (Lendway, 2020; Roamiar (2021); Ruiz (2019), Barter (2019; Seyedia (2021) and couses/video's (Lewis, 2020; Silge, 2021; Silge 2020) I followed and looked at. I tried to learn this new system and wrote different blogs in Dutch on this (Jonkman, 2021).

Core tidymodel packages include: parsnip, recipes, rsample and tune. Collectively, these packages provide a grammar for modeling that makes things a lot easier and provide a unified modeling and analysis interface to seamlessly access several model varieteis in R.

- `tidymodels` is a meta-package that installs and load the core packages listed below that you need for modeling and machine learning;
- `recipes` is tidy interface for to data pre-processing tools for feature/variables engineering;
- `rsample` provides infrastucture for efficient data splitting and resampling;
- `parsnip` is a tidy, unified interface to models that can be used to try a range of models without getting bagged down in the syntactical minutae of the underlying packages;
- `tune` helps you optimize the hyperparameters of your model and chose pre-processing steps;
- `yardstick` measures the effectiveness of models during performance metrics;
- `workflow` bundles your pre-processing modeling and post-processing together;
- `dials` creates and manages tuning parameters and parameters grids;
- `brooms` convert the information in common statistical R objects into user-friendly predictable formats.

Last months I tried to learn working with `tidymodels`. I tried to finish the Capstone course with the use of this packages.

PROBLEM DEFINITION

Data mining approaches are use to predict human wine taste preferences that are based on easily available analytical tests at certification step. A dataset on red wine from Portugal is used here to research quality of the wine and different predictors (Cortez et al., 2009) Supervised machine learning support us in this. In this world two kind of algorithms are often used. One is called regression and the other is called classification. In this study we use classification for predicting quality of wine (high,low) based on the predictors.

The wine data used here contains the following independent and dependent variables

Independent variables: (symbol I) - I1 *Fixed acidity* (g(tartaric acid/dm3) - I2 *Volatile acidity* (g(acetic acid)/dm3) - I3 *Citric acid* (g/dm3) - I4 *Residual sugar* (g/dm3) - I5 *Chlorides* (g(sodium chloride)/dm3)

- I6 *Free sulfar dioxide* (mg/dm3) - I7 *Total sulfar dioxide* (mg/dm3) - I8 *Density* (g/cm3) - I9 *pH* - I10 *Sulphates* (g(potassium sulphate)/dm3) - I11 *Alcohol* (vol%)

Dependent variable: (symbol D) - D1 *Quality*

DATA LOADING AND PREPROCESSING

Let us first load the dataset.

```
wf<-readRDS("wine.rds")
```

Let us give a summary of the data frame.

```
glimpse(wf)
```

```
## Rows: 1,599
## Columns: 12
## $ `fixed acidity`        <dbl> 7.4, 7.8, 7.8, 11.2, 7.4, 7.4, 7.9, 7.3, 7.8, 7~
## $ `volatile acidity`     <dbl> 0.700, NA, 0.760, 0.280, 0.700, NA, 0.600, 0.65~
## $ `citric acid`          <dbl> 0.00, 0.00, 0.04, 0.56, 0.00, 0.00, 0.06, 0.00,~
## $ `residual sugar`       <dbl> 1.9, 2.6, 2.3, 1.9, 1.9, 1.8, 1.6, 1.2, 2.0, 6.~
## $ chlorides              <dbl> 0.076, 0.098, 0.092, 0.075, 0.076, 0.075, 0.069~
## $ `free sulfur dioxide`  <dbl> 11, 25, 15, 17, 11, 13, 15, 15, 9, 17, 15, 17, ~
## $ `total sulfur dioxide` <dbl> 34, 67, 54, 60, 34, 40, 59, 21, 18, 102, 65, 10~
## $ density                <dbl> 0.9978, 0.9968, 0.9970, 0.9980, 0.9978, 0.9978,~
## $ pH                     <dbl> 3.51, 3.20, 3.26, 3.16, 3.51, 3.51, 3.30, 3.39,~
## $ sulphates              <dbl> 0.56, 0.68, 0.65, 0.58, 0.56, 0.56, 0.46, 0.47,~
## $ alcohol                <dbl> 9.4, 9.8, 9.8, 9.8, 9.4, 9.4, 9.4, NA, 9.5, 10.~
## $ quality                <dbl> 5, 5, 5, 6, 5, 5, 5, 7, 7, 5, 5, 5, 5, 5, 5, 5,~
```

2.2 Clean the data

To get a first impression of the data we take a look at the top 4 rows:

```
library(gt)

wf %>%
  slice_head(n = 4) %>%
  gt() # print output using gt
```

fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density
7.4	0.70	0.00	1.9	0.076	11	34	0.9978
7.8	NA	0.00	2.6	0.098	25	67	0.9968
7.8	0.76	0.04	2.3	0.092	15	54	0.9970
11.2	0.28	0.56	1.9	0.075	17	60	0.9980

2.3 Format data

Next, we take a look at the data structure and check wether all data formats are correct:

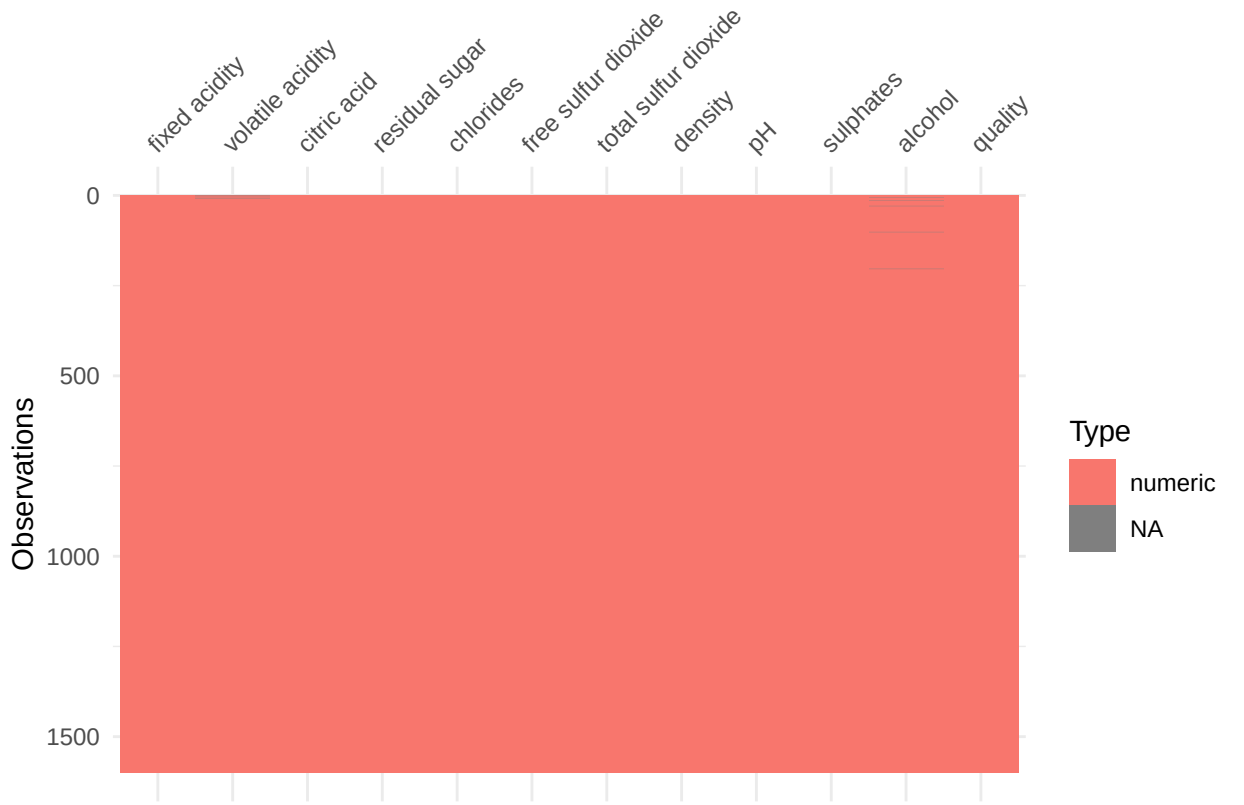
- Numeric variables should be formatted as integers (int) or double precision floating point numbers (dbl).
- Categorical (nominal and ordinal) variables should usually be formatted as factors (fct) and not characters (chr). Especially, if they don't have many levels.

```
glimpse(wf)
```

```
## Rows: 1,599
## Columns: 12
## $ `fixed acidity`        <dbl> 7.4, 7.8, 7.8, 11.2, 7.4, 7.4, 7.9, 7.3, 7.8, 7~
## $ `volatile acidity`     <dbl> 0.700, NA, 0.760, 0.280, 0.700, NA, 0.600, 0.65~
## $ `citric acid`          <dbl> 0.00, 0.00, 0.04, 0.56, 0.00, 0.00, 0.06, 0.00,~
## $ `residual sugar`       <dbl> 1.9, 2.6, 2.3, 1.9, 1.9, 1.8, 1.6, 1.2, 2.0, 6.~
## $ chlorides              <dbl> 0.076, 0.098, 0.092, 0.075, 0.076, 0.075, 0.069~
## $ `free sulfur dioxide`  <dbl> 11, 25, 15, 17, 11, 13, 15, 15, 9, 17, 15, 17, ~
## $ `total sulfur dioxide` <dbl> 34, 67, 54, 60, 34, 40, 59, 21, 18, 102, 65, 10~
## $ density                <dbl> 0.9978, 0.9968, 0.9970, 0.9980, 0.9978, 0.9978,~
## $ pH                     <dbl> 3.51, 3.20, 3.26, 3.16, 3.51, 3.51, 3.30, 3.39,~
## $ sulphates              <dbl> 0.56, 0.68, 0.65, 0.58, 0.56, 0.56, 0.46, 0.47,~
## $ alcohol                <dbl> 9.4, 9.8, 9.8, 9.8, 9.4, 9.4, 9.4, NA, 9.5, 10.~
## $ quality                <dbl> 5, 5, 5, 6, 5, 5, 5, 7, 7, 5, 5, 5, 5, 5, 5, 5,~
```

The package `visdat` helps us to explore the data class structure visually:

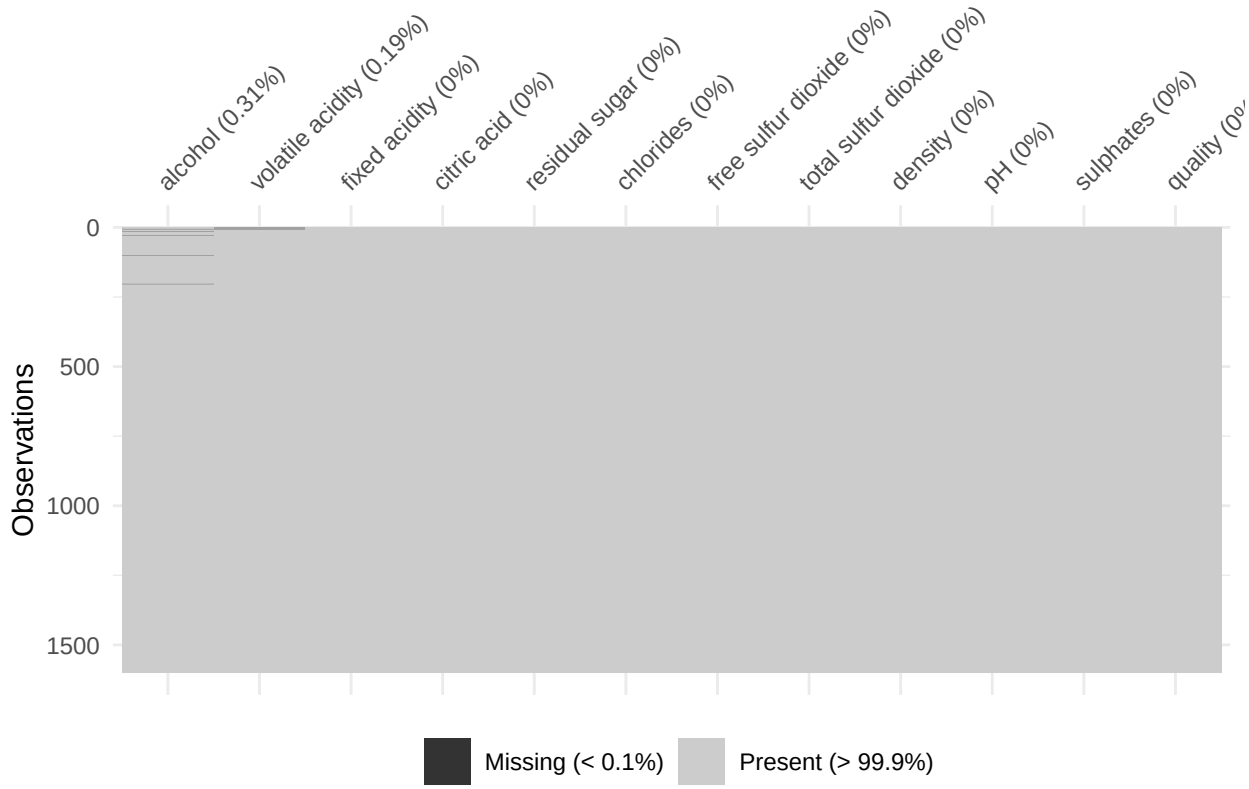
```
library(visdat)

vis_dat(wf)
```



2.4 Missing data

Now let's turn our attention to missing data. Missing data can be viewed with the function vis_miss from the package visdat. We arrange the data by columns with most missingness:

```
vis_miss(wf, sort_miss = TRUE)
```



Create new variables One very important thing you may want to do at the beginning of your data science project is to create new variable combinations. For example here I want to work with classification and change the contiunues variable `quality` to categorical variable `quality_two`

```
percentage <- prop.table(table(wf$quality)) * 100
cbind(freq=table(wf$quality), percentage=percentage)
```

```
##   freq percentage
## 3   10  0.6253909
## 4   53  3.3145716
## 5  681 42.5891182
## 6  638 39.8999375
## 7  199 12.4452783
## 8   18  1.1257036
```

With `recipe` package we can do a lot at the same time

```
set.seed(123)

wdNA<-
  wf %>%
  mutate(
    # Convert quality to a factor
    quality = ifelse(quality >5, "high", "low"),
    quality = factor(quality)
)
```

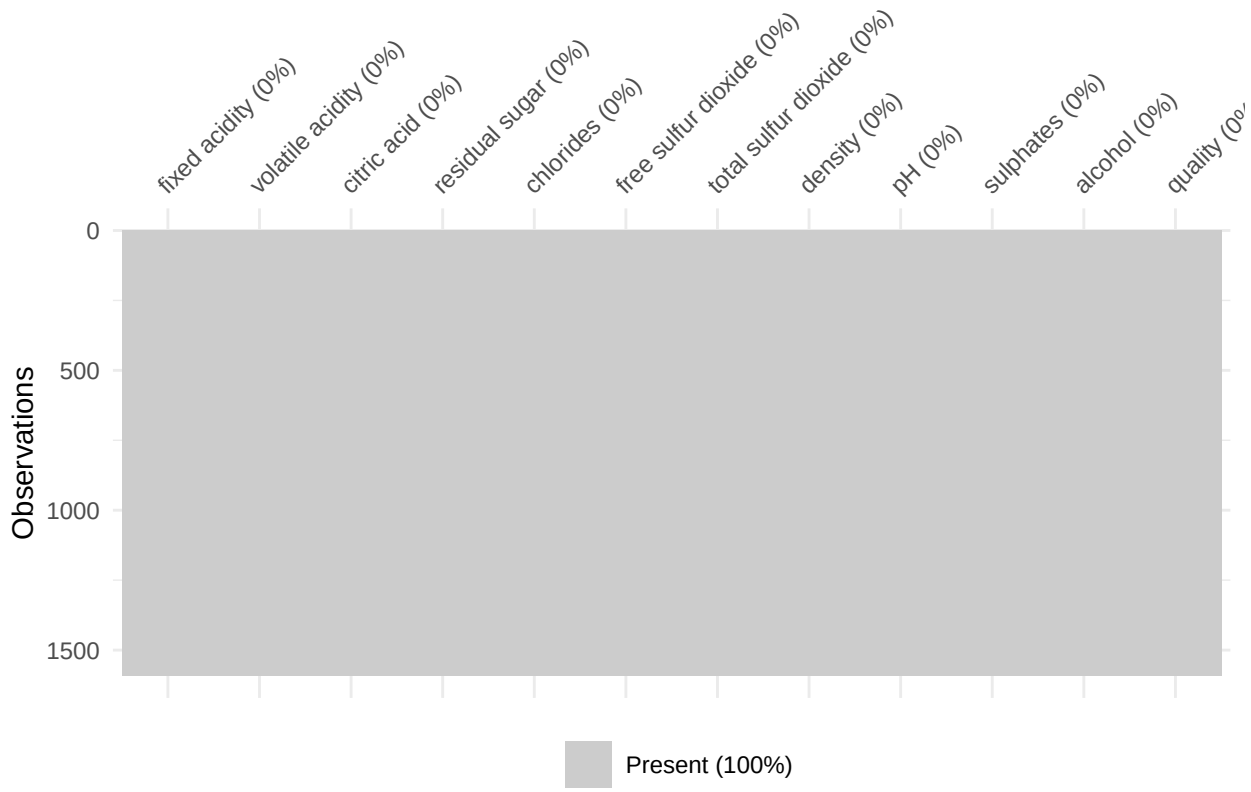
Take out missing data.

```
wdNA<-na.omit(wdNA)
```

‘

Check the missings

```
vis_miss(wdNA, sort_miss = TRUE)
```

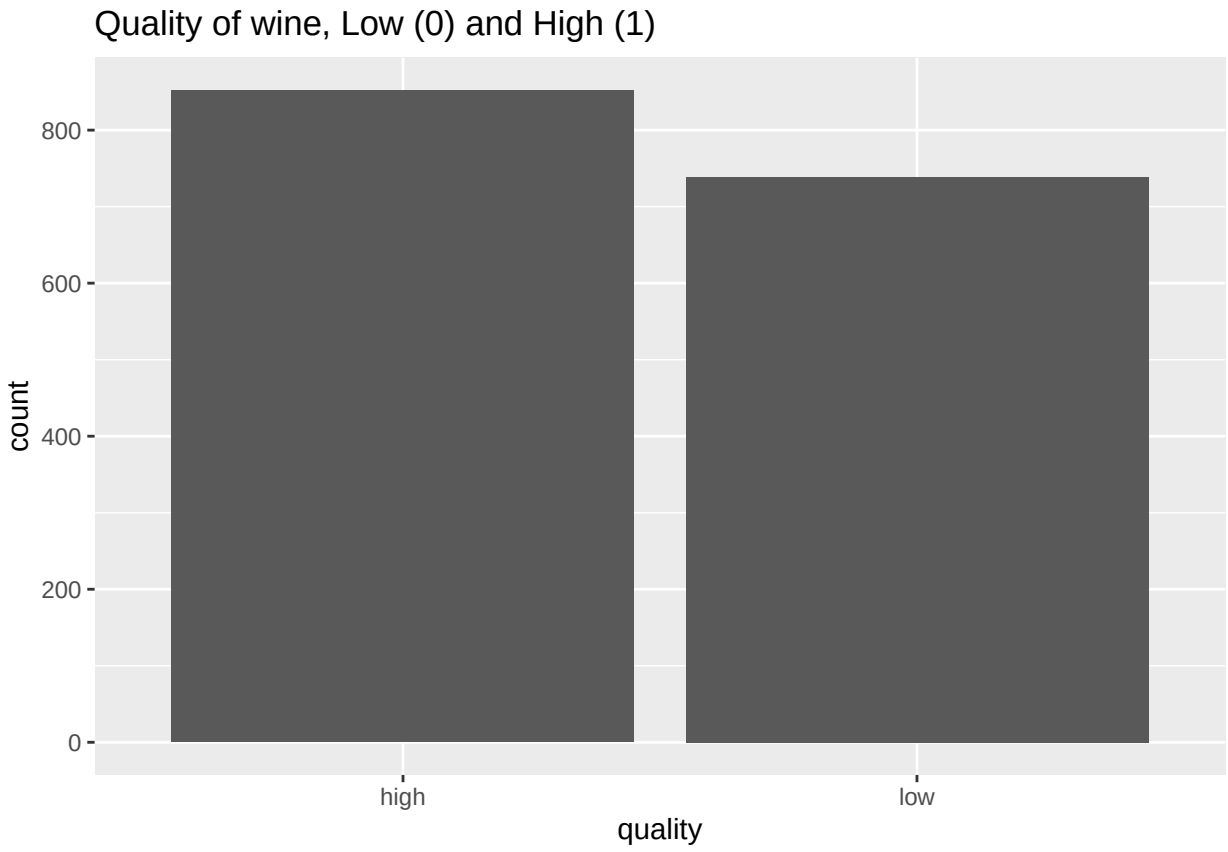


Let us look at the dependent variable closely.

```
percentage <- prop.table(table(wdNA$quality)) * 100
cbind(freq=table(wdNA$quality), percentage=percentage)
```

```
##      freq percentage
## high  852   53.55123
## low   739   46.44877
```

```
wdNA %>%
  ggplot(aes(quality)) +
  geom_bar() +
  ggtitle("Quality of wine, Low (0) and High (1)")
```



2.6 Fix column names

```
colnames(wdNA) <- wdNA %>%
  colnames() %>% str_replace_all(pattern = " ", replacement = "_")
colnames(wdNA)
```

```
##  [1] "fixed_acidity"        "volatile_acidity"     "citric_acid"
##  [4] "residual_sugar"       "chlorides"            "free_sulfur_dioxide"
##  [7] "total_sulfur_dioxide" "density"              "pH"
## [10] "sulphates"            "alcohol"              "quality"
```

2.7 Data overview

```
library(skimr)
skim(wdNA)
```

Data summary

Name	wdNA
Number of rows	1591
Number of columns	12

Data summary	
Column type frequency:	
factor	1
numeric	11
Group variables	None

Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
quality	0	1	FALSE	2	hig: 852, low: 739

Variable type: numeric

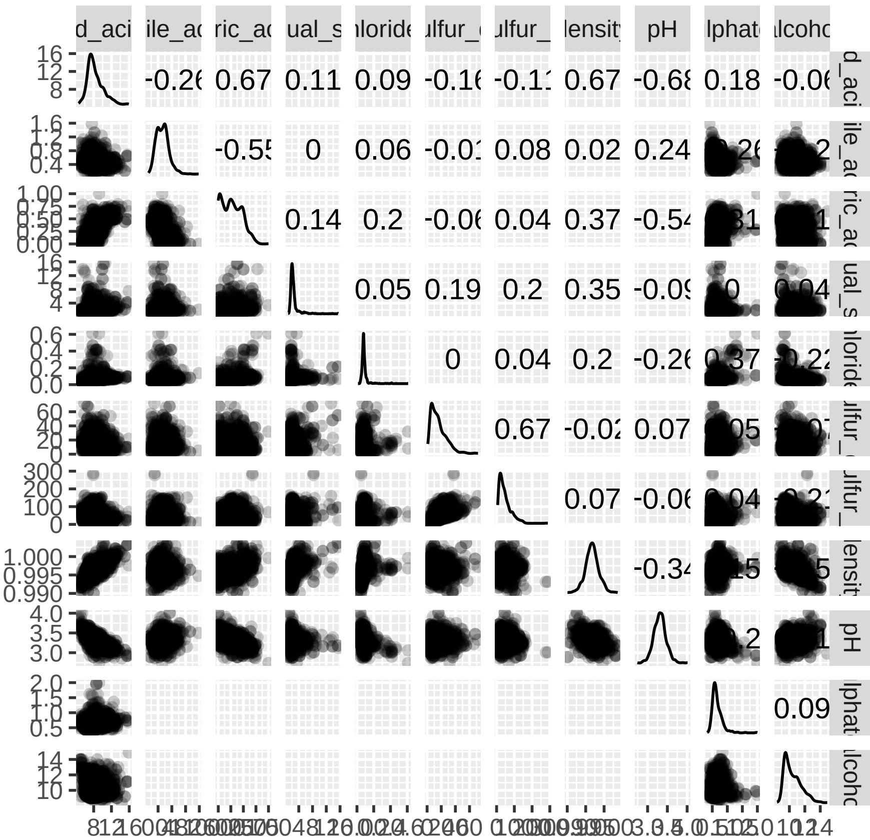
skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
fixed_acidity	0	1	8.32	1.74	4.60	7.10	7.90	9.20	15.90	
volatile_acidity	0	1	0.53	0.18	0.12	0.39	0.52	0.64	1.58	
citric_acid	0	1	0.27	0.19	0.00	0.09	0.26	0.42	1.00	
residual_sugar	0	1	2.54	1.41	0.90	1.90	2.20	2.60	15.50	
chlorides	0	1	0.09	0.05	0.01	0.07	0.08	0.09	0.61	
free_sulfur_dioxide	0	1	15.85	10.45	1.00	7.00	14.00	21.00	72.00	
total_sulfur_dioxide	0	1	46.37	32.83	6.00	22.00	38.00	62.00	289.00	
density	0	1	1.00	0.00	0.99	1.00	1.00	1.00	1.00	
pH	0	1	3.31	0.15	2.74	3.21	3.31	3.40	4.01	
sulphates	0	1	0.66	0.17	0.33	0.55	0.62	0.73	2.00	
alcohol	0	1	10.43	1.07	8.40	9.50	10.20	11.10	14.90	

We have:

-

```
library(GGally)
wdNA %>%
  ggscatmat(alpha=0.2)
```

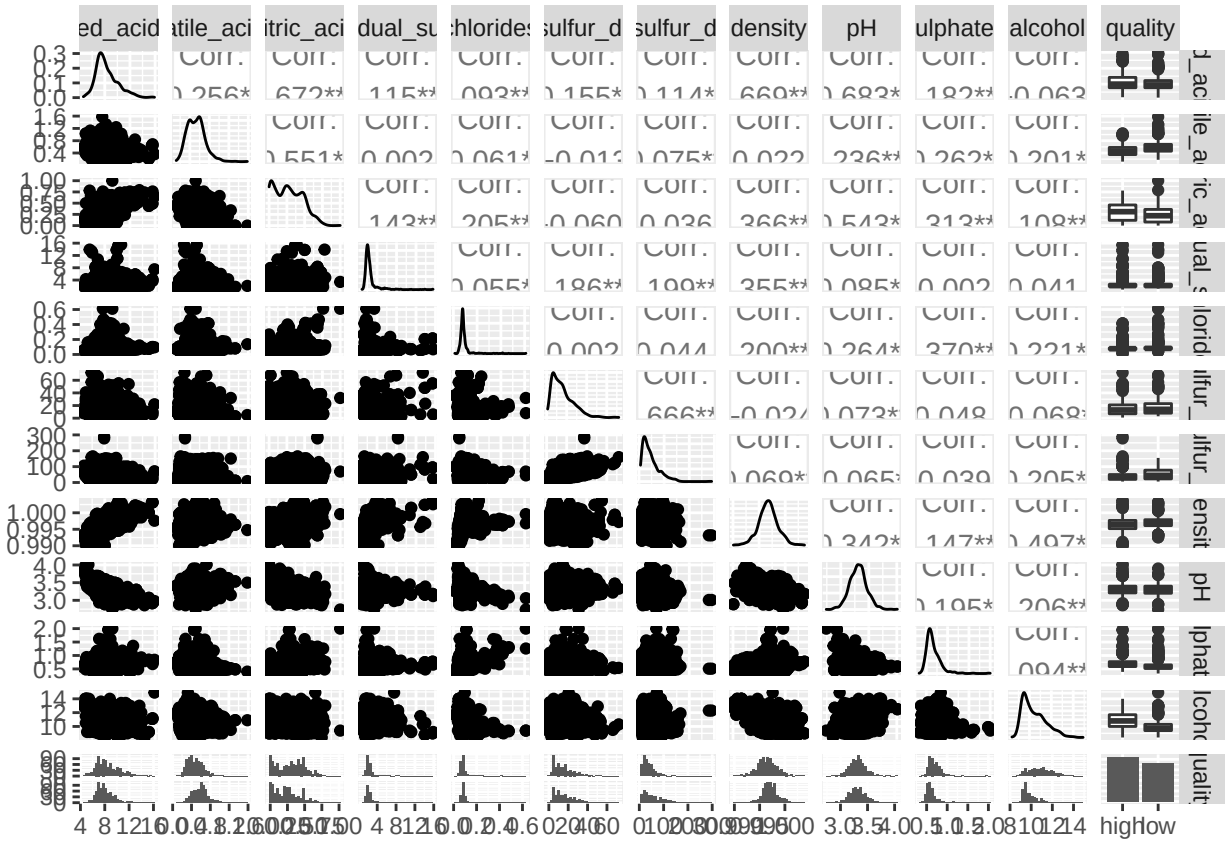
```
## Warning in ggscatmat(., alpha = 0.2): Factor variables are omitted in plot
```



Another option is to use ggpairs:

```
wdNA %>%
  ggpairs()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



A summary of the data frame

```
summary(wdNA)
```

```
##  fixed_acidity    volatile_acidity  citric_acid     residual_sugar
##  Min.   : 4.600   Min.   :0.1200   Min.   :0.0000   Min.   : 0.900
##  1st Qu.: 7.100   1st Qu.:0.3900   1st Qu.:0.0900   1st Qu.: 1.900
##  Median : 7.900   Median :0.5200   Median :0.2600   Median : 2.200
##  Mean   : 8.323   Mean   :0.5274   Mean   :0.2716   Mean   : 2.538
##  3rd Qu.: 9.200   3rd Qu.:0.6400   3rd Qu.:0.4250   3rd Qu.: 2.600
##  Max.   :15.900   Max.   :1.5800   Max.   :1.0000   Max.   :15.500
##    chlorides       free_sulfur_dioxide total_sulfur_dioxide    density
##  Min.   :0.01200   Min.   : 1.00       Min.   :  6.00       Min.   :0.9901
##  1st Qu.:0.07000   1st Qu.: 7.00       1st Qu.: 22.00       1st Qu.:0.9956
##  Median :0.07900   Median :14.00       Median : 38.00       Median :0.9968
##  Mean   :0.08744   Mean   :15.85       Mean   : 46.37       Mean   :0.9967
##  3rd Qu.:0.09000   3rd Qu.:21.00       3rd Qu.: 62.00       3rd Qu.:0.9978
##  Max.   :0.61100   Max.   :72.00       Max.   :289.00       Max.   :1.0037
##        pH          sulphates         alcohol      quality
##  Min.   :2.740   Min.   :0.3300   Min.   : 8.40   high:852
##  1st Qu.:3.210   1st Qu.:0.5500   1st Qu.: 9.50   low :739
##  Median :3.310   Median :0.6200   Median :10.20
##  Mean   :3.311   Mean   :0.6582   Mean   :10.43
```

```
##  3rd Qu.:3.400   3rd Qu.:0.7300   3rd Qu.:11.10
##  Max.   :4.010   Max.   :2.0000   Max.   :14.90
```

```
glimpse(wdNA)
```

```
## Rows: 1,591
## Columns: 12
## $ fixed_acidity        <dbl> 7.4, 7.8, 11.2, 7.4, 7.9, 7.8, 6.7, 7.5, 5.6, 7.8~
## $ volatile_acidity     <dbl> 0.700, 0.760, 0.280, 0.700, 0.600, 0.580, 0.580, ~
## $ citric_acid          <dbl> 0.00, 0.04, 0.56, 0.00, 0.06, 0.02, 0.08, 0.36, 0~
## $ residual_sugar       <dbl> 1.9, 2.3, 1.9, 1.9, 1.6, 2.0, 1.8, 6.1, 1.6, 1.6,~
## $ chlorides            <dbl> 0.076, 0.092, 0.075, 0.076, 0.069, 0.073, 0.097, ~
## $ free_sulfur_dioxide  <dbl> 11, 15, 17, 11, 15, 9, 15, 17, 16, 9, 52, 35, 16,~
## $ total_sulfur_dioxide <dbl> 34, 54, 60, 34, 59, 18, 65, 102, 59, 29, 145, 103~
## $ density              <dbl> 0.9978, 0.9970, 0.9980, 0.9978, 0.9964, 0.9968, 0~
## $ pH                   <dbl> 3.51, 3.26, 3.16, 3.51, 3.30, 3.36, 3.28, 3.35, 3~
## $ sulphates            <dbl> 0.56, 0.65, 0.58, 0.56, 0.46, 0.57, 0.54, 0.80, 0~
## $ alcohol              <dbl> 9.4, 9.8, 9.8, 9.4, 9.4, 9.5, 9.2, 10.5, 9.9, 9.1~
## $ quality              <fct> low, low, high, low, low, high, low, low, low, lo~
```

EXPLORATIVE DATA ANALYSIS (EDA).

```
library(corrplot)
wdNA %>%
   select(-quality) %>%
  cor() %>%
  corrplot.mixed(upper = "circle",
                 tl.cex = 1,
                 tl.pos = 'lt',
                 number.cex = 0.75)
```

Split the data into: a) Train set, b) Test set

All functions below come from the `rsample` package

```
set.seed(12345) # to fix randomisation by setting the seed (reproducibility)

data_split <- initial_split(wdNA, prop = 0.8) # Use 80% of the data for training

train_data <- training(data_split)

test_data  <- testing(data_split)
```

Validation set

Remember that we already partitioned our data set into a training set and test set. This lets us judge whether a given model will generalize well to new data. However, using only two partitions may be insufficient when doing many rounds of hyperparameter tuning (which we don't perform in this tutorial but it is always recommended to use a validation set).

Therefore, it is usually a good idea to create a so called validation set. Watch this short video from Google's Machine Learning crash course to learn more about the value of a validation set.

We use k-fold crossvalidation to build a set of 5 validation folds with the function vfold_cv. We also use stratified sampling:

```r
set.seed(100)

wine_cv <-
 vfold_cv(train_data,
          v = 5,
          strata = quality)
```

We will come back to the validation set after we specified our models

MODELING AND DATA ANALYSIS

Prepare models

```r
wdNA_rec <-
  recipe(quality ~ .,
         data=train_data) %>%
         prep()
```

```r
wdNA_rec
```

```
## Data Recipe
##
## Inputs:
##
##       role #variables
##    outcome          1
##  predictor         11
##
## Training data contained 1273 data points and no missing data.
```

Execute pre-processing

The testing data can now be transformed using the exact same steps, weights, and categorization used to pre-process the training data. To do this, another function with a cooking term is used: bake(). Notice that the testing() function is used in order to extract the appropriate data set.

```r
wdNATEST <-wdNA_rec %>%
  bake(testing(data_split))
```

```r
glimpse(wdNATEST)
```

```
## Rows: 318
## Columns: 12
## $ fixed_acidity        <dbl> 7.4, 8.5, 8.1, 7.4, 7.9, 6.3, 8.3, 8.8, 7.5, 8.1,~
## $ volatile_acidity     <dbl> 0.700, 0.280, 0.560, 0.590, 0.430, 0.390, 0.655, ~
## $ citric_acid          <dbl> 0.00, 0.56, 0.28, 0.08, 0.21, 0.16, 0.12, 0.30, 0~
## $ residual_sugar       <dbl> 1.9, 1.8, 1.7, 4.4, 1.6, 1.4, 2.3, 2.8, 2.6, 2.2,~
## $ chlorides            <dbl> 0.076, 0.092, 0.368, 0.086, 0.106, 0.080, 0.083, ~
```

```
## $ free_sulfur_dioxide  <dbl> 11, 35, 16, 6, 10, 11, 15, 17, 8, 9, 12, 6, 4, 20~
## $ total_sulfur_dioxide <dbl> 34, 103, 56, 29, 37, 23, 113, 46, 14, 23, 96, 14,~
## $ density              <dbl> 0.9978, 0.9969, 0.9968, 0.9974, 0.9966, 0.9955, 0~
## $ pH                   <dbl> 3.51, 3.30, 3.11, 3.38, 3.17, 3.34, 3.17, 3.26, 3~
## $ sulphates            <dbl> 0.56, 0.75, 1.28, 0.50, 0.91, 0.56, 0.66, 0.51, 0~
## $ alcohol              <dbl> 9.4, 10.5, 9.3, 9.0, 9.5, 9.3, 9.8, 9.3, 10.5, 10~
## $ quality              <fct> low, high, low, low, low, low, low, low, high, lo~
```

Performing the same operation over the training data is redundant, because that data has already been prepped. To load the prepared training data into a variable, we use juice(). It will extract the data from the iris_recipe object.

```
wdNATRAING <- juice(wdNA_rec)
glimpse(wdNATRAING)
```

```
## Rows: 1,273
## Columns: 12
## $ fixed_acidity        <dbl> 7.8, 11.2, 7.4, 7.9, 7.8, 6.7, 7.5, 5.6, 7.8, 8.9~
## $ volatile_acidity     <dbl> 0.760, 0.280, 0.700, 0.600, 0.580, 0.580, 0.500, ~
## $ citric_acid          <dbl> 0.04, 0.56, 0.00, 0.06, 0.02, 0.08, 0.36, 0.00, 0~
## $ residual_sugar       <dbl> 2.3, 1.9, 1.9, 1.6, 2.0, 1.8, 6.1, 1.6, 1.6, 3.8,~
## $ chlorides            <dbl> 0.092, 0.075, 0.076, 0.069, 0.073, 0.097, 0.071, ~
## $ free_sulfur_dioxide  <dbl> 15, 17, 11, 15, 9, 15, 17, 16, 9, 52, 17, 29, 23,~
## $ total_sulfur_dioxide <dbl> 54, 60, 34, 59, 18, 65, 102, 59, 29, 145, 56, 60,~
## $ density              <dbl> 0.9970, 0.9980, 0.9978, 0.9964, 0.9968, 0.9959, 0~
## $ pH                   <dbl> 3.26, 3.16, 3.51, 3.30, 3.36, 3.28, 3.35, 3.58, 3~
## $ sulphates            <dbl> 0.65, 0.58, 0.56, 0.46, 0.57, 0.54, 0.80, 0.52, 1~
## $ alcohol              <dbl> 9.8, 9.8, 9.4, 9.4, 9.5, 9.2, 10.5, 9.9, 9.1, 9.2~
## $ quality              <fct> low, high, low, low, high, low, low, low, low, lo~
```

Model training

The process of specifying our models is always as follows:

Pick a model type
- set the engine
- Set the mode: regression or classification
- You can choose the model type and engine from this list.

Logistic regression

We run the model and show the estimates.

```
log_spec <- logistic_reg() %>%
  set_engine(engine = "glm") %>%
  set_mode("classification") %>%
  fit(quality~., data=train_data)

  tidy(log_spec)
```

```
## # A tibble: 12 x 5
```

```
##    term                 estimate std.error statistic  p.value
##    <chr>                   <dbl>     <dbl>     <dbl>    <dbl>
##  1 (Intercept)          -37.0     89.6        -0.413 6.80e- 1
##  2 fixed_acidity         -0.0980   0.111      -0.882 3.78e- 1
##  3 volatile_acidity       2.94     0.530       5.54  2.99e- 8
##  4 citric_acid            0.808    0.611       1.32  1.86e- 1
##  5 residual_sugar        -0.0528   0.0585     -0.903 3.66e- 1
##  6 chlorides              3.97     1.79        2.22  2.67e- 2
##  7 free_sulfur_dioxide   -0.0275   0.00917    -3.00  2.73e- 3
##  8 total_sulfur_dioxide   0.0172   0.00316     5.44  5.22e- 8
##  9 density               44.4     91.4         0.485 6.27e- 1
## 10 pH                     0.520    0.798       0.652 5.14e- 1
## 11 sulphates             -3.03     0.510      -5.95  2.64e- 9
## 12 alcohol               -0.838    0.118      -7.12  1.08e-12
```

Here we write down the results in Odds-ratio's which are better to understand.

```
tidy(log_spec, exponentiate=TRUE)
```

```
## # A tibble: 12 x 5
##    term                 estimate std.error statistic  p.value
##    <chr>                   <dbl>     <dbl>     <dbl>    <dbl>
##  1 (Intercept)          8.71e-17  89.6        -0.413 6.80e- 1
##  2 fixed_acidity        9.07e- 1   0.111      -0.882 3.78e- 1
##  3 volatile_acidity     1.89e+ 1   0.530       5.54  2.99e- 8
##  4 citric_acid          2.24e+ 0   0.611       1.32  1.86e- 1
##  5 residual_sugar       9.49e- 1   0.0585     -0.903 3.66e- 1
##  6 chlorides            5.32e+ 1   1.79        2.22  2.67e- 2
##  7 free_sulfur_dioxide  9.73e- 1   0.00917    -3.00  2.73e- 3
##  8 total_sulfur_dioxide 1.02e+ 0   0.00316     5.44  5.22e- 8
##  9 density              1.84e+19  91.4         0.485 6.27e- 1
## 10 pH                   1.68e+ 0   0.798       0.652 5.14e- 1
## 11 sulphates            4.81e- 2   0.510      -5.95  2.64e- 9
## 12 alcohol              4.32e- 1   0.118      -7.12  1.08e-12
```

Let us show the same results but only the signficant predictors on quality.

```
tidy(log_spec, exponentiate=TRUE) %>%
  filter(p.value < 0.05)
```

```
## # A tibble: 6 x 5
##   term                estimate std.error statistic  p.value
##   <chr>                  <dbl>     <dbl>     <dbl>    <dbl>
## 1 volatile_acidity      18.9      0.530       5.54 2.99e- 8
## 2 chlorides             53.2      1.79        2.22 2.67e- 2
## 3 free_sulfur_dioxide    0.973    0.00917    -3.00 2.73e- 3
## 4 total_sulfur_dioxide   1.02     0.00316     5.44 5.22e- 8
## 5 sulphates              0.0481   0.510      -5.95 2.64e- 9
## 6 alcohol                0.432    0.118      -7.12 1.08e-12
```

Model prediction

Now it is time to use the test data. Let us show the class predictions for the first 5 cases.

```
pred_class<-predict(log_spec,
                      new_data=test_data,
                      type="class")

pred_class[1:5,]
```

```
## # A tibble: 5 x 1
##   .pred_class
##   <fct>
## 1 low
## 2 high
## 3 low
## 4 low
## 5 high
```

Test data class probabilities (the predicted probabilities for the first five cases)

```
pred_proba<-predict(log_spec,
                    new_data=test_data,
                    type="prob")


pred_proba[1:5,]
```

```
## # A tibble: 5 x 2
##   .pred_high .pred_low
##        <dbl>     <dbl>
## 1      0.206     0.794
## 2      0.644     0.356
## 3      0.462     0.538
## 4      0.185     0.815
## 5      0.620     0.380
```

Here we see the final data preparation for model evaluation

```
quality_results<-test_data %>%
  select(quality) %>%
  bind_cols(pred_class, pred_proba)

quality_results[1:5,]
```

```
## # A tibble: 5 x 4
##   quality .pred_class .pred_high .pred_low
##   <fct>   <fct>            <dbl>     <dbl>
## 1 low     low              0.206     0.794
## 2 high    high             0.644     0.356
## 3 low     low              0.462     0.538
## 4 low     low              0.185     0.815
## 5 low     high             0.620     0.380
```

Model evaluation

Let us evaluate the defined model. For this we use the confusion matrix. 120 case were high and predicted as such and 116 cases were low and predicted as such. 37 wre low but predicted as high and 45 had a true high score but predicted as low.

```
conf_mat(quality_results, truth = quality,
         estimate = .pred_class)
```

```
##           Truth
## Prediction high low
##       high  120  37
##       low    45 116
```

Let us now research the accuracy of the model.

```
accuracy(quality_results, truth=quality,
         estimate=.pred_class)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.742
```

What can we say about the sensitivity of the model.

```
sens(quality_results, truth=quality, estimate = .pred_class)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 sens    binary         0.727
```

What about the specificity of the model.

```
spec(quality_results, truth=quality, estimate = .pred_class)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 spec    binary         0.758
```

Now these three scores together.

```
custom_metrics<-metric_set(accuracy, sens, spec)

custom_metrics(quality_results,
               truth=quality,
               estimate=.pred_class)
```

```
## # A tibble: 3 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.742
## 2 sens     binary         0.727
## 3 spec     binary         0.758
```

RESULTS SUMMARIZED

Let us summarize the results. And it is easy to use the good and old `caret` package.

```
library(caret)

confusionMatrix(quality_results$.pred_class,
                quality_results$quality,
                pos="high")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction high low
##       high  120  37
##       low    45 116
##
##                Accuracy : 0.7421
##                  95% CI : (0.6904, 0.7893)
##     No Information Rate : 0.5189
##     P-Value [Acc > NIR] : 2.752e-16
##
##                   Kappa : 0.4845
##
##  Mcnemar's Test P-Value : 0.4395
##
##             Sensitivity : 0.7273
##             Specificity : 0.7582
##          Pos Pred Value : 0.7643
##          Neg Pred Value : 0.7205
##              Prevalence : 0.5189
##          Detection Rate : 0.3774
##    Detection Prevalence : 0.4937
##       Balanced Accuracy : 0.7427
##
##        'Positive' Class : high
##
```

References

Attalides, N. (2020). Introduction to machine learning. Barcelona. Presentation

Barter, R.(2020). Tidymodels: tidy machine learning in R

Baumer, B. Kaplan, D.T., Horton, N.J. (2017). *Modern data science with R.* CRCPress: Boca Raton.

Boehmke, B. & Greenwell, B. (2020). *Hands on machine learning with R.* Bookdown version

Cortez, P., Cerdeira, A., Almeida, F., Matos, T. & Reis, J. (2009). Modeling wine prefernces by data mining from physicochemical properties. *Decision Support Systems, 47*, 547-533.

Hartie, T., Tibskirani, R. & Friedmann, J. (2009). *The elements of statistical learning. Data mining, inference and prediction.* 2nd edition. Springer: New York.

Irizarry, R.A. (2020). *Introduction to data science. Data analysis and prediction algorithms with R.* CRC Press: Boca Raton.

James, S., Witten, D., Hastie, T.. & Tibskirani, R. (2013). *An introduction to statistical learning with application in R.*

Jonkman, H. (2019-2021). Harrie's hoekje, his website with different blogs on machine learning in Dutch

Kuhn, M. 7 Johnson, K. (2013). *Applied predictive modeling.* Springer: New York.

Kuhn, M. & Johnson, K. (2019). Feature engineering and selection: A practical approach for predictive models

Lendway, L. (2020). 2020_north-tidymodels.Introduction on github

Lewis, J.E. (2020). Coding machine learning models

Raoniar, R. (2021). Modeling binary logistic regression using tidymodels library in R (Part 1). Towards data science

Ruiz, E. (2019). A gentle introduction to tidymodels

Seyedian, A. (2021). Medical cost personal datasets. Insurance forcast by using linear regression

Silge, J. (2020). Get started with tidymodels and #TidyTuesdag Palmer penguins

Silge, J. (2021). Supervised machine learning case studies in R

Tidymodels. Tidymodels website