Wine Quality

Tolga Tabanli

2025-09-16

We can understand and summarize a general machine learning and prediction task with the following order of steps one should take:

- Define the task at hand
- Determine 'predictor' and 'predictee'
- Explore your data
- Feature engineering / feature extraction
- Clean and complete* the data
- Modelling
- Performance evaluation
- Presenting the model

Predicting Wine Quality

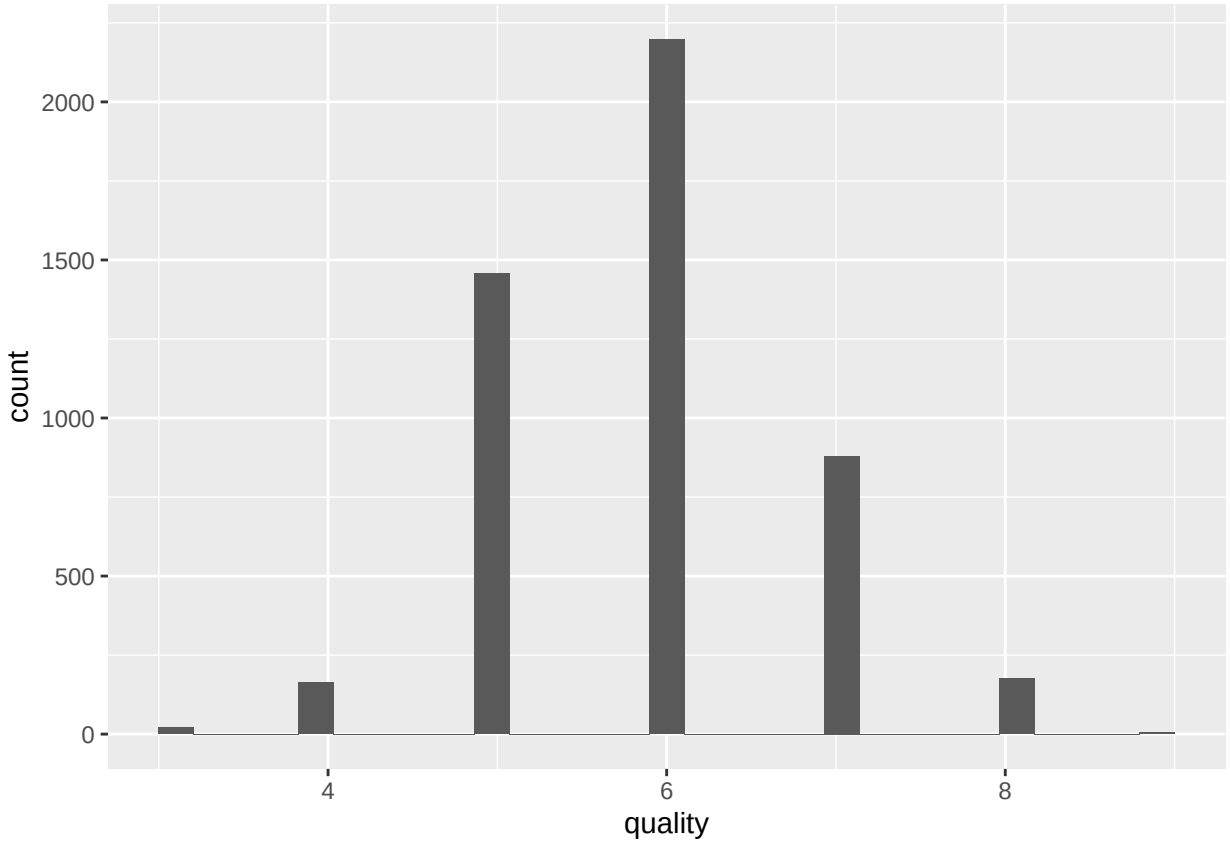
Certain chemical properties of wine that can also be quantified give its flavour and charm. Its acidity, alcohol, sugar content and consistency blend together in different proportions in different kinds. **Can we predict the "quality" of wine depending on these chemical properties?**

```
library(readr)
library(dplyr)
library(tidyr)
library(ggplot2)

wine_white <- read_delim("data/winequality-white.csv", delim = ";")
head(wine_white)
```

```
## # A tibble: 6 x 12
##   `fixed acidity` `volatile acidity` `citric acid` `residual sugar` chlorides
##             <dbl>              <dbl>         <dbl>            <dbl>     <dbl>
## 1             7                0.27          0.36             20.7     0.045
## 2             6.3              0.3           0.34              1.6     0.049
## 3             8.1              0.28          0.4               6.9     0.05
## 4             7.2              0.23          0.32              8.5     0.058
## 5             7.2              0.23          0.32              8.5     0.058
## 6             8.1              0.28          0.4               6.9     0.05
## # i 7 more variables: `free sulfur dioxide` <dbl>,
## #   `total sulfur dioxide` <dbl>, density <dbl>, pH <dbl>, sulphates <dbl>,
## #   alcohol <dbl>, quality <dbl>
```

Before modelling, although it's not always necessary, it's a good idea to check the distribution of our target variable to see if it's skewed, multimodal or, in any way, differs from a normal distribution. This is helpful, especially if we are going to work with parametric models, if we want to take precautions to handle evaluation and objective metrics, or if we want to explore the characteristics of what we are dealing with.

```
ggplot(wine_white, aes(x = quality)) +
  geom_histogram()
```



As a basic histogram shows, the quality follows a relatively good normal distribution. The next step would be to see any missing values in our data set, and if there are, whether the missingness has a particular pattern and how it should be handled. In real data sets, missingness will be very important, since it will appear almost everywhere. The model's performance and reliability depend on how these missing pieces are interpreted, removed and/or filled.

```
# in more complex data sets with many variables, geom_raster() can be used to visualize missingness
# however, in our case, we opt for a basic NA check:

any(is.na(wine_white))
```

```
## [1] FALSE
```

R's vectorised logic quickly shows us that we do not have any missing values here. It is important to note that different data sets might have other ways of denoting missing values such as dash ('-') characters, "NA" as a string entry (instead of any missing value between two commas), etc.

If we had to deal with missingness, we would basically have three options:

- either remove the row for missing values that are sparse and not column-dependent
- remove the column, if it mostly consists of missing values
- impute, if we can 'predict' a *sufficiently reasonable* value to fill in with.

Imputation has its own advantages and disadvantages, and depends on the characteristics of the data. There are many ways of imputing. For instance, the missing value could be replaced by the column mean or can be 'predicted' by algorithms like k-nearest neighbours (KNN).

Before moving on, a last thing to consider is the scale at which our predictors live. If we use linear models or models that use linear functions intrinsically, standardisation, that is, centring and scaling, is important for interpretation and mechanisms of algorithms. It gives each variable *a fair chance to explain itself.*

Simple Linear Regression

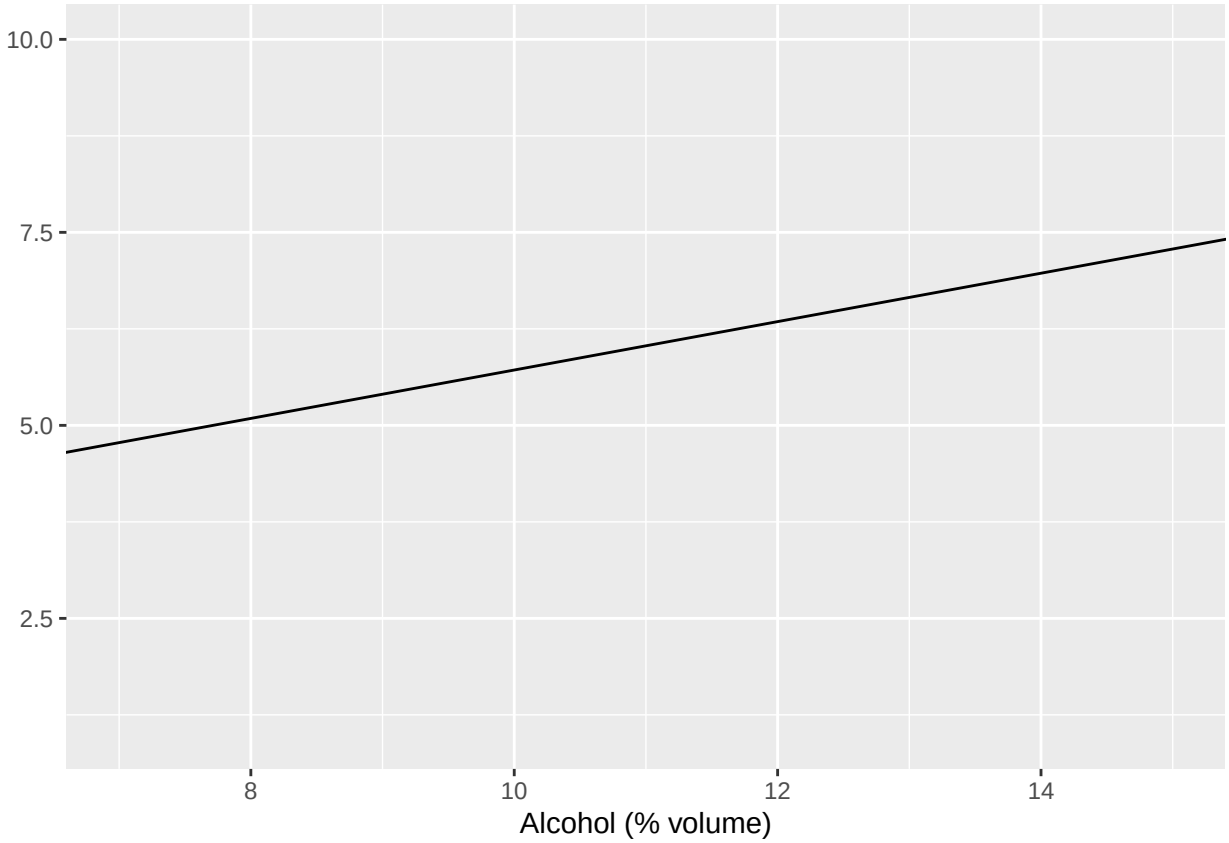
Let's start by implementing a simple linear regressor using a single variable to predict the quality of wine. We choose the alcohol content here. You're encouraged to try out other predictors as well.

```
model_alcohol <- lm(quality ~ alcohol, data = wine_white)
summary(model_alcohol)
```

```
##
## Call:
## lm(formula = quality ~ alcohol, data = wine_white)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.5317 -0.5286  0.0012  0.4996  3.1579
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 2.582009   0.098008   26.34   <2e-16 ***
## alcohol     0.313469   0.009258   33.86   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7973 on 4896 degrees of freedom
## Multiple R-squared:  0.1897, Adjusted R-squared:  0.1896
## F-statistic:  1146 on 1 and 4896 DF,  p-value: < 2.2e-16
```

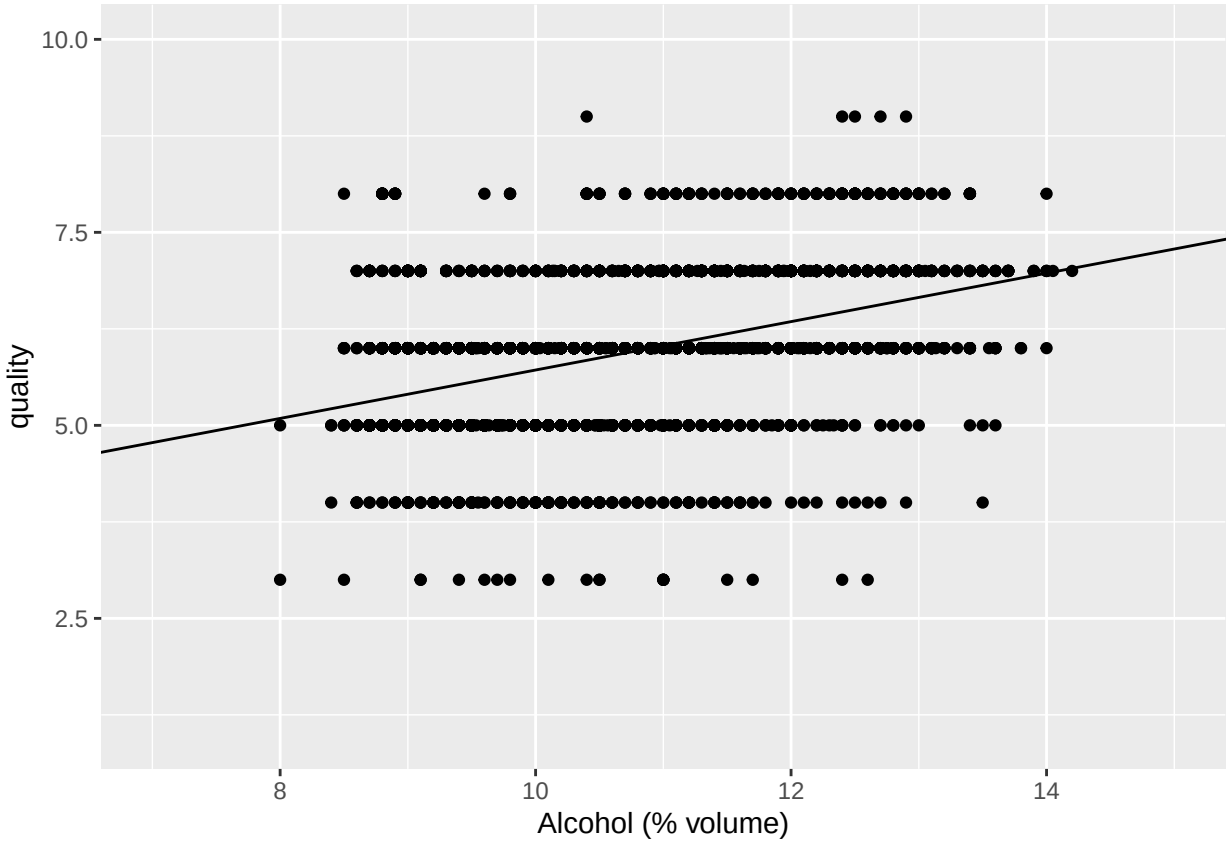
How would this appear on 2D?

```
# linear model is a + bx
a <- model_alcohol$coefficients[1]
b <- model_alcohol$coefficients[2]

p <- ggplot() +
  geom_abline(intercept = a, slope = b) +
  xlim(7, 15) +
  ylim(1, 10) +
  labs(x = "Alcohol (% volume)")
p
```



Adding the real data, we get...

```
p + geom_point(aes(x = alcohol, y = quality), data = wine_white)
```



What is the problem here? Using a single predictor does not quite capture all the variance in the data. Let's quantify our **error.** We'll use mean absolute error (MAE) and root mean squared error (RMSE).

```
residuals <- model_alcohol$residuals

# mean absolute error
mae <- mean(abs(residuals))

# root mean squared error
rmse <- sqrt(mean(residuals^2))

c(MAE = mae, RMSE = rmse)
```

```
##       MAE      RMSE
## 0.6276288 0.7971285
```

Let's choose MAE as out performance metric. In other words, in our modelling we will try to minimize the MAE. For this, let's include all the data coming from other variables as well:

```
model_all <- lm(quality ~ ., data = wine_white)
summary(model_all)
```

```
##
## Call:
## lm(formula = quality ~ ., data = wine_white)
```

```
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.8348 -0.4934 -0.0379  0.4637  3.1143
##
## Coefficients:
##                          Estimate Std. Error t value Pr(>|t|)
## (Intercept)             1.502e+02  1.880e+01   7.987 1.71e-15 ***
## `fixed acidity`         6.552e-02  2.087e-02   3.139  0.00171 **
## `volatile acidity`     -1.863e+00  1.138e-01 -16.373  < 2e-16 ***
## `citric acid`           2.209e-02  9.577e-02   0.231  0.81759
## `residual sugar`        8.148e-02  7.527e-03  10.825  < 2e-16 ***
## chlorides              -2.473e-01  5.465e-01  -0.452  0.65097
## `free sulfur dioxide`   3.733e-03  8.441e-04   4.422 9.99e-06 ***
## `total sulfur dioxide` -2.857e-04  3.781e-04  -0.756  0.44979
## density                -1.503e+02  1.907e+01  -7.879 4.04e-15 ***
## pH                      6.863e-01  1.054e-01   6.513 8.10e-11 ***
## sulphates               6.315e-01  1.004e-01   6.291 3.44e-10 ***
## alcohol                 1.935e-01  2.422e-02   7.988 1.70e-15 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7514 on 4886 degrees of freedom
## Multiple R-squared:  0.2819, Adjusted R-squared:  0.2803
## F-statistic: 174.3 on 11 and 4886 DF,  p-value: < 2.2e-16
```

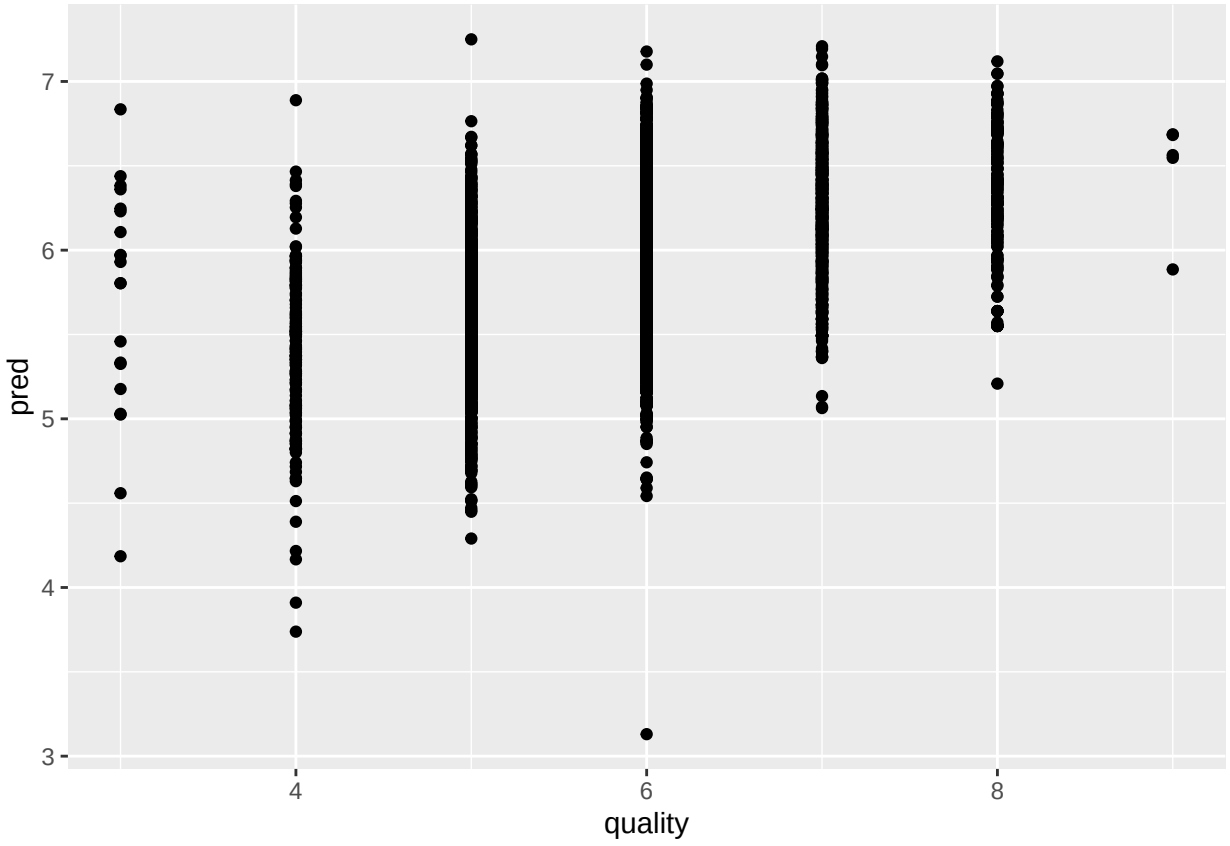
Now, we get an MAE of...

```r
residuals <- model_all$residuals

# mean absolute error
mae <- mean(abs(residuals))
mae
```

```
## [1] 0.583635
```

How does the predicted and real quality values compare?

```r
# first add predicted values as a new column
wine_white_with_pred <- cbind(wine_white, pred = predict(model_all))

# plotting
ggplot(wine_white_with_pred, aes(x = quality, y = pred)) +
  geom_point()
```



Summarizing with Caret

An important concept in machine learning is model generalizability. This concept is closely related to **overfitting** and **underfitting**. In short, underfitting is when a model fails to capture the trends and patterns in the training dataset. This causes a low performance in both the training and test sets. On the other hand, a model is said to overfit (the training set) when it fits its parameters to capture the noise, that is, the non-explanatory variance, beyond meaningful information in the training dataset. This shows the symptom of a high performance in the training set with a low performance in the test set.

We are going to revise our linear modelling process using cross-validation and compare the two models. Here, caret provides a consistent toolset.

```
library(caret)

# Model with alcohol as predictor
set.seed(2025)
model_cv_alcohol <- train(
  form = quality ~ alcohol,
  data = wine_white,
  method = "lm",
  trControl = trainControl(method = "cv", number = 10)
)

# Model with all variables as predictor
set.seed(2025)
```

```r
model_cv_all <- train(
  form = quality ~ .,
  data = wine_white,
  method = "lm",
  trControl = trainControl(method = "cv", number = 10)
)
```

Now, we can compare the two models' performance on cross-validation. For this, we need to extract the resampled values, re-form the data frame into the format we want for plotting (model in one column and the MAE in another) and show the results in boxplots:

```r
cv_perf_results <- resamples(list(
  with_alcohol = model_cv_alcohol,
  with_all = model_cv_all
))$values

cv_perf_results %>% pivot_longer(cols = !Resample,
                                 names_to = "model~metric",
                                 values_to = "value") %>%
  separate("model~metric", sep = "~", into = c("model", "metric")) %>%
  filter(metric == "MAE") %>%
  ggplot(aes(x = model, y = value)) +
    geom_boxplot() +
    labs(x = "Model", y = "MAE")
```