COSC 5557: Practical Machine Learning

Exploratory Data Analysis- Wine Data

Abiodun Awosola

2024-02-21

Note: Not all lines of code are displayed.

Loading the Primary Tumor Data

```
#Imports Data

wine_data <- read.table("winequality-white.csv", sep = ";",
                        check.names = TRUE, header=T)
```

Exploring the Data

As a first step, we explore the data and look for simple problems such as constant or duplicated features. This can be done quite efficiently with a package like `DataExplorer` or `skimr` which can be used to create a large number of informative plots.

Below we summarize the most important findings for data cleaning, but we only consider this aspect in a cursory manner:

Data Attributes

```
library(tidyverse)
library(knitr)


library(xtable)
#Rotates the table
wine_data1 <- xtable(t(wine_data))

#First 10 rows of data
kable(wine_data1[,1:10],
      caption = "First 10 Rows of White Wine Data")
```

Table 1: First 10 Rows of White Wine Data

	1	2	3	4	5	6	7	8	9	10
fixed.acidity	7.000	6.300	8.1000	7.2000	7.2000	8.1000	6.2000	7.000	6.300	8.1000
volatile.acidity	0.270	0.300	0.2800	0.2300	0.2300	0.2800	0.3200	0.270	0.300	0.2200
citric.acid	0.360	0.340	0.4000	0.3200	0.3200	0.4000	0.1600	0.360	0.340	0.4300
residual.sugar	20.700	1.600	6.9000	8.5000	8.5000	6.9000	7.0000	20.700	1.600	1.5000

	1	2	3	4	5	6	7	8	9	10
chlorides	0.045	0.049	0.0500	0.0580	0.0580	0.0500	0.0450	0.045	0.049	0.0440
free.sulfur.dioxide	45.000	14.000	30.0000	47.0000	47.0000	30.0000	30.0000	45.000	14.000	28.0000
total.sulfur.dioxide	170.000	132.000	97.0000	186.0000	186.0000	97.0000	136.0000	170.000	132.000	129.0000
density	1.001	0.994	0.9951	0.9956	0.9956	0.9951	0.9949	1.001	0.994	0.9938
pH	3.000	3.300	3.2600	3.1900	3.1900	3.2600	3.1800	3.000	3.300	3.2200
sulphates	0.450	0.490	0.4400	0.4000	0.4000	0.4400	0.4700	0.450	0.490	0.4500
alcohol	8.800	9.500	10.1000	9.9000	9.9000	10.1000	9.6000	8.800	9.500	11.0000
quality	6.000	6.000	6.0000	6.0000	6.0000	6.0000	6.0000	6.000	6.000	6.0000

```
wine_data_summary <- xtable(t(summary(wine_data)))


kable(wine_data_summary,
      caption = "Summary of White Wine Data")
```

Table 2: Summary of White Wine Data

	V1	V2	V3	V4	V5	V6
fixed.acidity	Min. : 3.800	1st Qu.: 6.300	Median : 6.800	Mean : 6.855	3rd Qu.: 7.300	Max. :14.200
volatile.acidity	Min. :0.0800	1st Qu.:0.2100	Median :0.2600	Mean :0.2782	3rd Qu.:0.3200	Max. :1.1000
citric.acid	Min. :0.0000	1st Qu.:0.2700	Median :0.3200	Mean :0.3342	3rd Qu.:0.3900	Max. :1.6600
residual.sugar	Min. : 0.600	1st Qu.: 1.700	Median : 5.200	Mean : 6.391	3rd Qu.: 9.900	Max. :65.800
chlorides	Min. :0.00900	1st Qu.:0.03600	Median :0.04300	Mean :0.04577	3rd Qu.:0.05000	Max. :0.34600
free.sulfur.dioxide	Min. : 2.00	1st Qu.: 23.00	Median : 34.00	Mean : 35.31	3rd Qu.: 46.00	Max. :289.00
total.sulfur.dioxide	Min. : 9.0	1st Qu.:108.0	Median :134.0	Mean :138.4	3rd Qu.:167.0	Max. :440.0
density	Min. :0.9871	1st Qu.:0.9917	Median :0.9937	Mean :0.9940	3rd Qu.:0.9961	Max. :1.0390
pH	Min. :2.720	1st Qu.:3.090	Median :3.180	Mean :3.188	3rd Qu.:3.280	Max. :3.820
sulphates	Min. :0.2200	1st Qu.:0.4100	Median :0.4700	Mean :0.4898	3rd Qu.:0.5500	Max. :1.0800
alcohol	Min. : 8.00	1st Qu.: 9.50	Median :10.40	Mean :10.51	3rd Qu.:11.40	Max. :14.20
quality	Min. :3.000	1st Qu.:5.000	Median :6.000	Mean :5.878	3rd Qu.:6.000	Max. :9.000

```
# Details the data attributes
 result <- skimr::skim(wine_data)

 result[,c(1:3)]
```

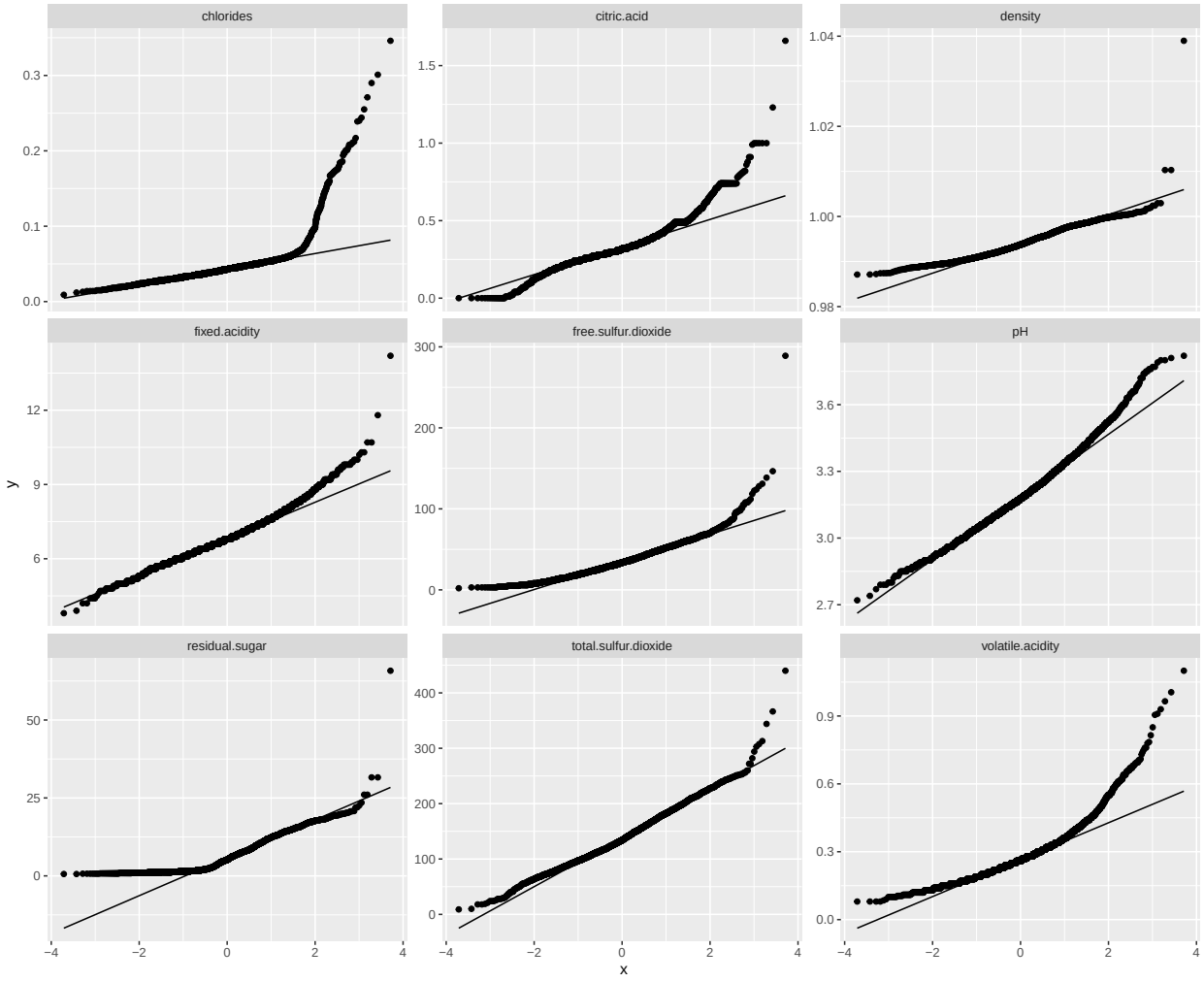
Table 3: Data summary

Name	wine_data
Number of rows	4898
Number of columns	12
Column type frequency:	
numeric	12
Group variables	None

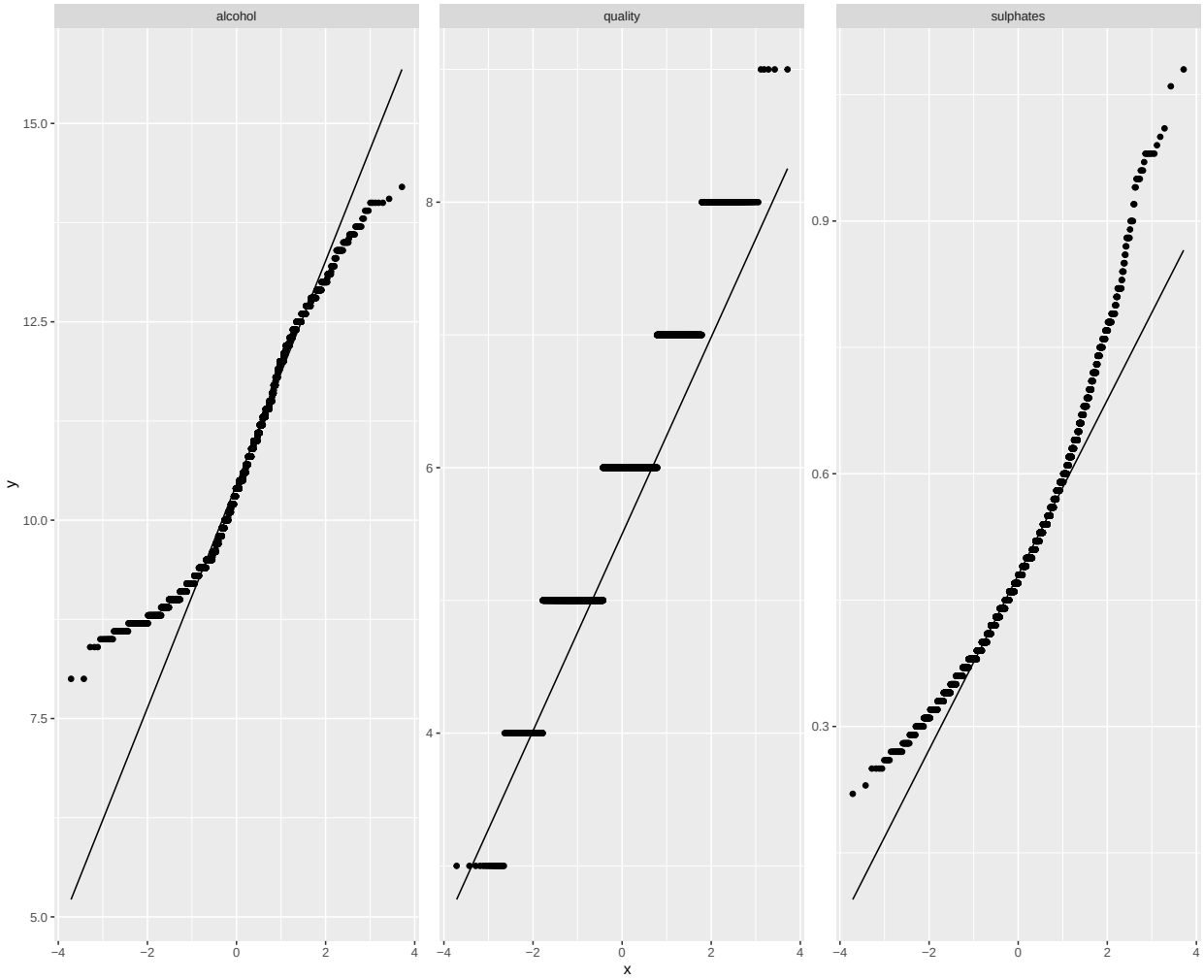
Variable type: numeric

skim_variable	n_missing
fixed.acidity	0
volatile.acidity	0
citric.acid	0
residual.sugar	0
chlorides	0
free.sulfur.dioxide	0
total.sulfur.dioxide	0
density	0
pH	0
sulphates	0
alcohol	0
quality	0

```
DataExplorer::plot_qq(wine_data) # normal distribution check
```

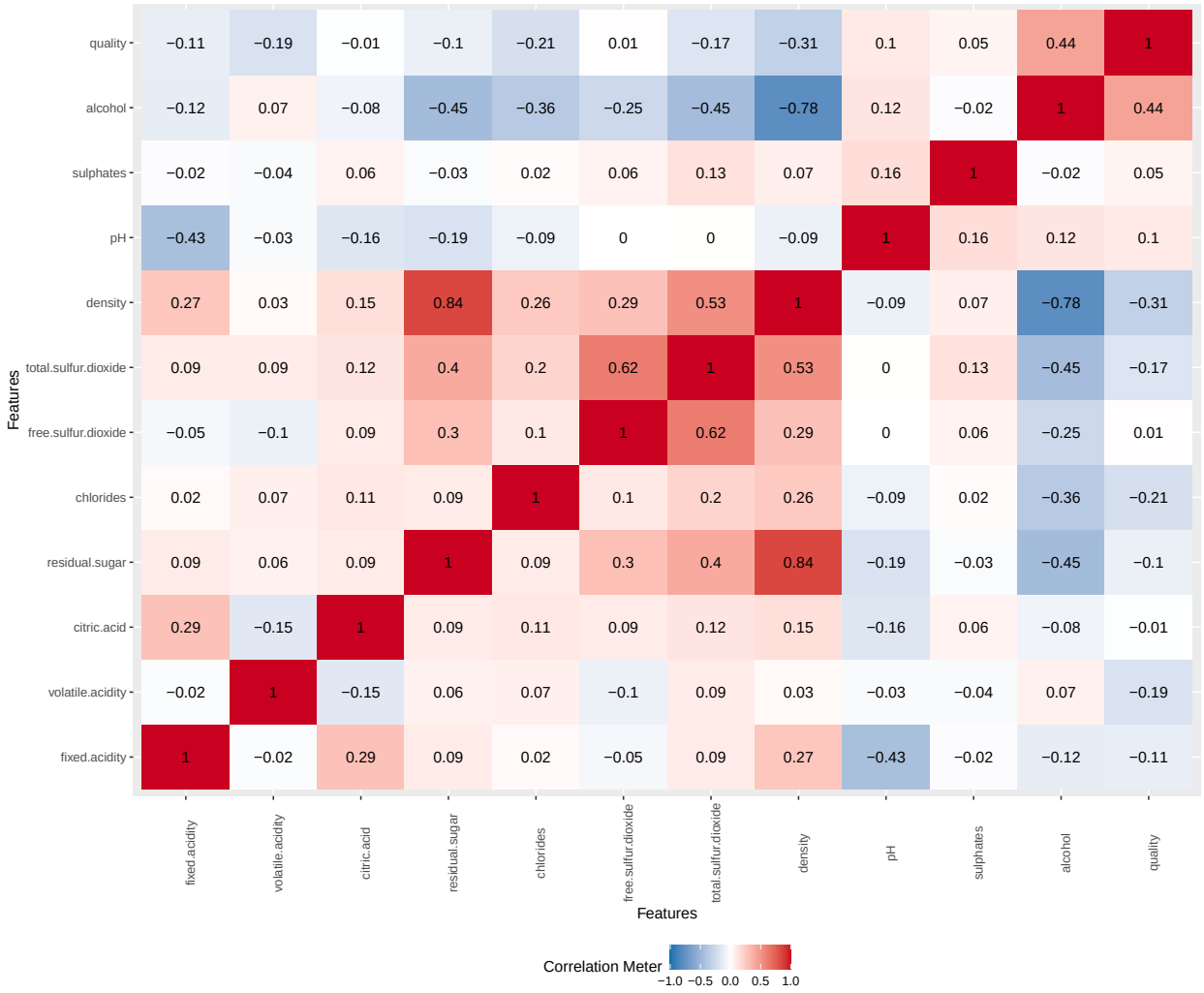

chlorides
citric.acid
density
fixed.acidity
free.sulfur.dioxide
pH
residual.sugar
total.sulfur.dioxide
volatile.acidity
y
x

Page 1

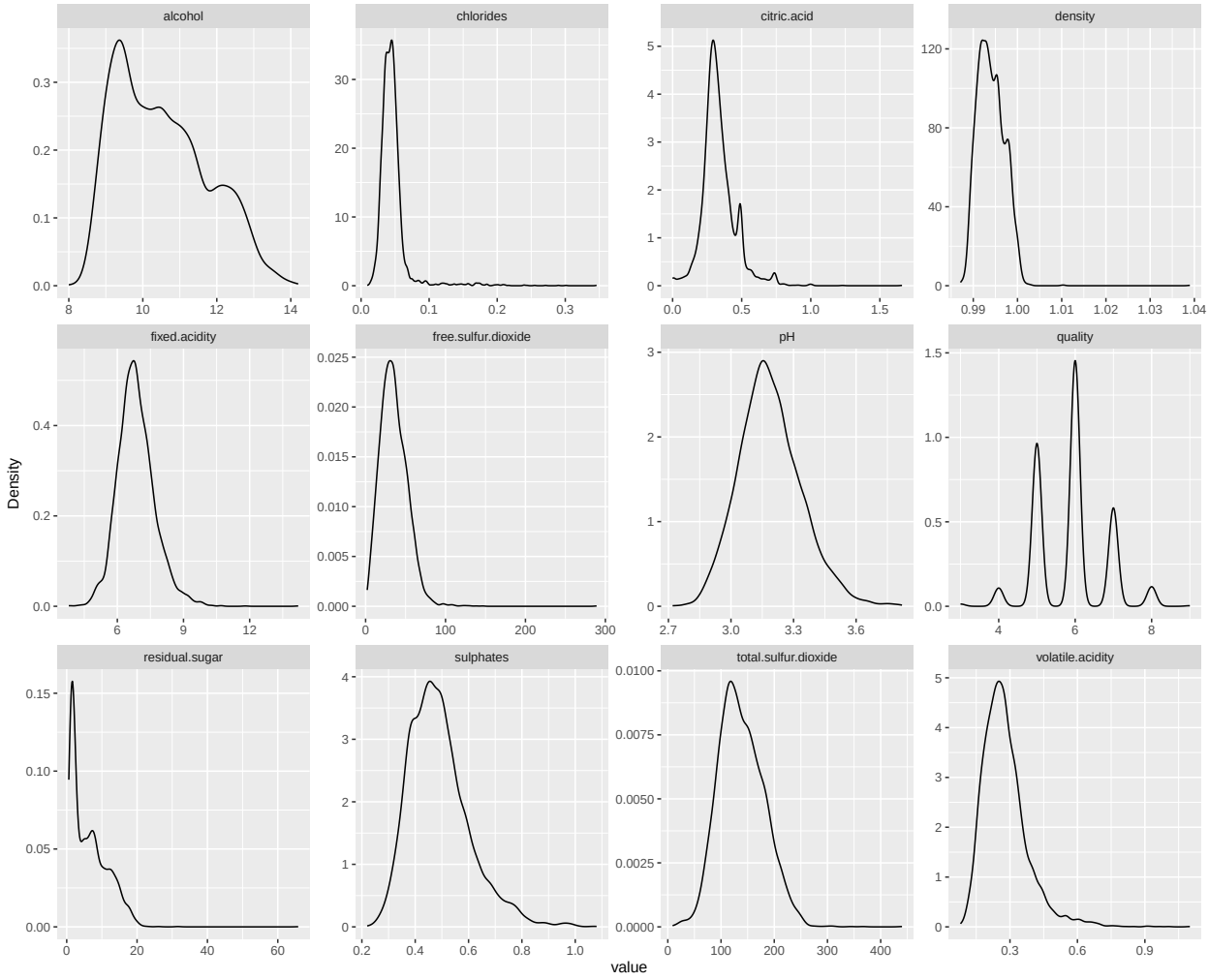


Page 2

```
DataExplorer::plot_correlation(wine_data)
```




```
DataExplorer::plot_density(wine_data)
```



The next thing is to clean up the data by fixing those anomalies.

Data Cleaning

Not deemed needed.

```
library(mlr3verse)
```

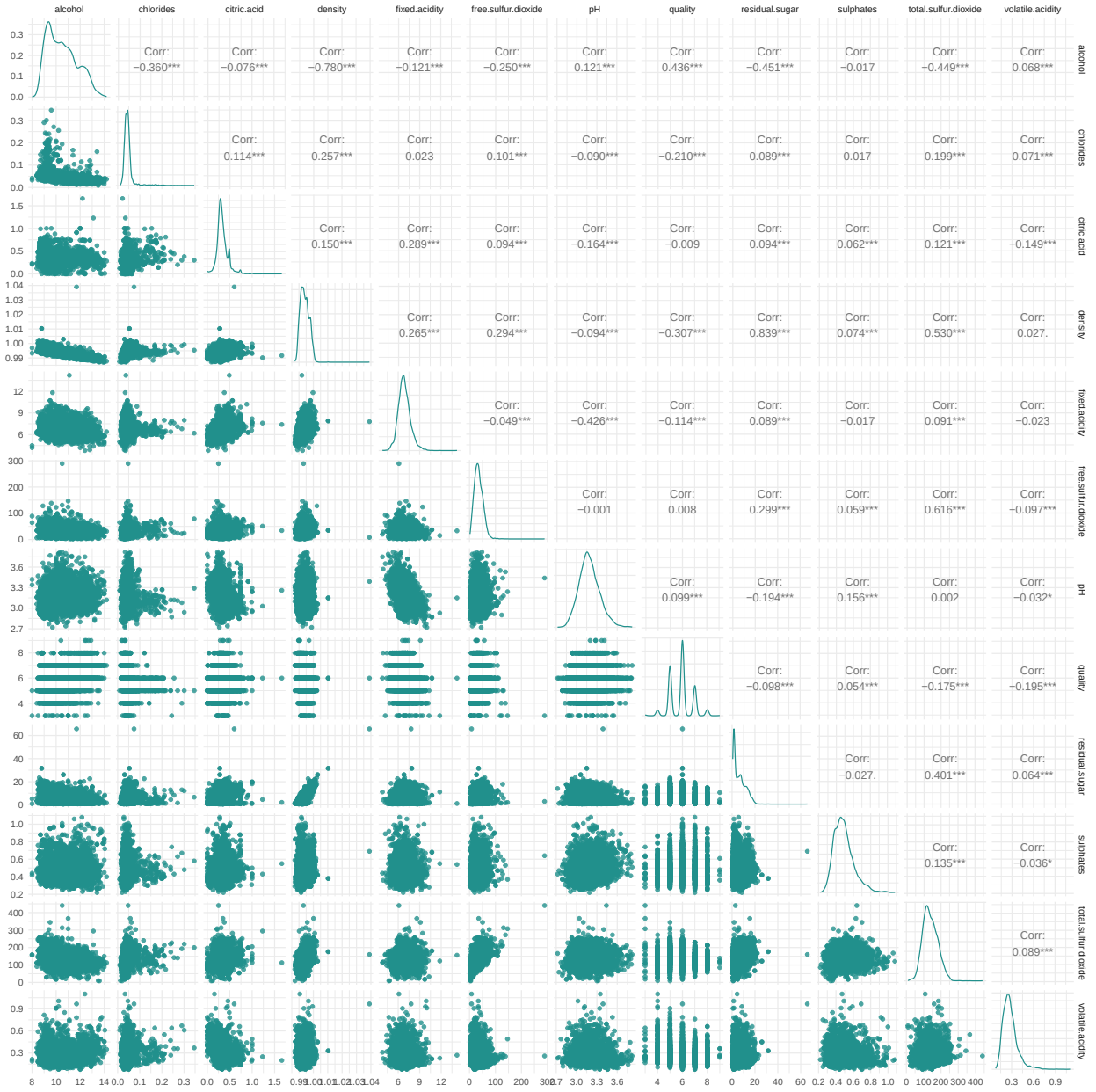
```
## Loading required package: mlr3
```

```
tsk_wine_alc = as_task_regr(wine_data, target = "alcohol",
id = "White Wine Alcohol Content")

tsk_wine_alc
```

```
## <TaskRegr:White Wine Alcohol Content> (4898 x 12)
## * Target: alcohol
## * Properties: -
## * Features (11):
##   - dbl (10): chlorides, citric.acid, density, fixed.acidity,
##     free.sulfur.dioxide, pH, residual.sugar, sulphates,
##     total.sulfur.dioxide, volatile.acidity
```

```
##   - int (1): quality
```



*** Frequency Distribution of the Variable Levels**

Task Mutators

Used when some variables need be dropped.

	x
Features1	chlorides
Features2	citric.acid
Features3	density
Features4	fixed.acidity
Features5	free.sulfur.dioxide

	x
Features6	pH
Features7	quality
Features8	residual.sugar
Features9	sulphates
Features10	total.sulfur.dioxide
Features11	volatile.acidity
Target	alcohol

Training

In the simplest use case, models are trained by passing a task to a learner with the `$train()` method:

After training, the fitted model is stored in the `$model` field for future inspection and prediction:

Partitioning Data

When assessing the quality of a model's predictions, one will likely want to partition the dataset to get a fair and unbiased estimate of a model's generalization error.

When training we will tell the model to only use the training data by passing the row IDs from `partition` to the `row_ids` argument of `$train()`:

```
lrn_rpart$train(tsk_wine_alc, row_ids = splits$train)
```

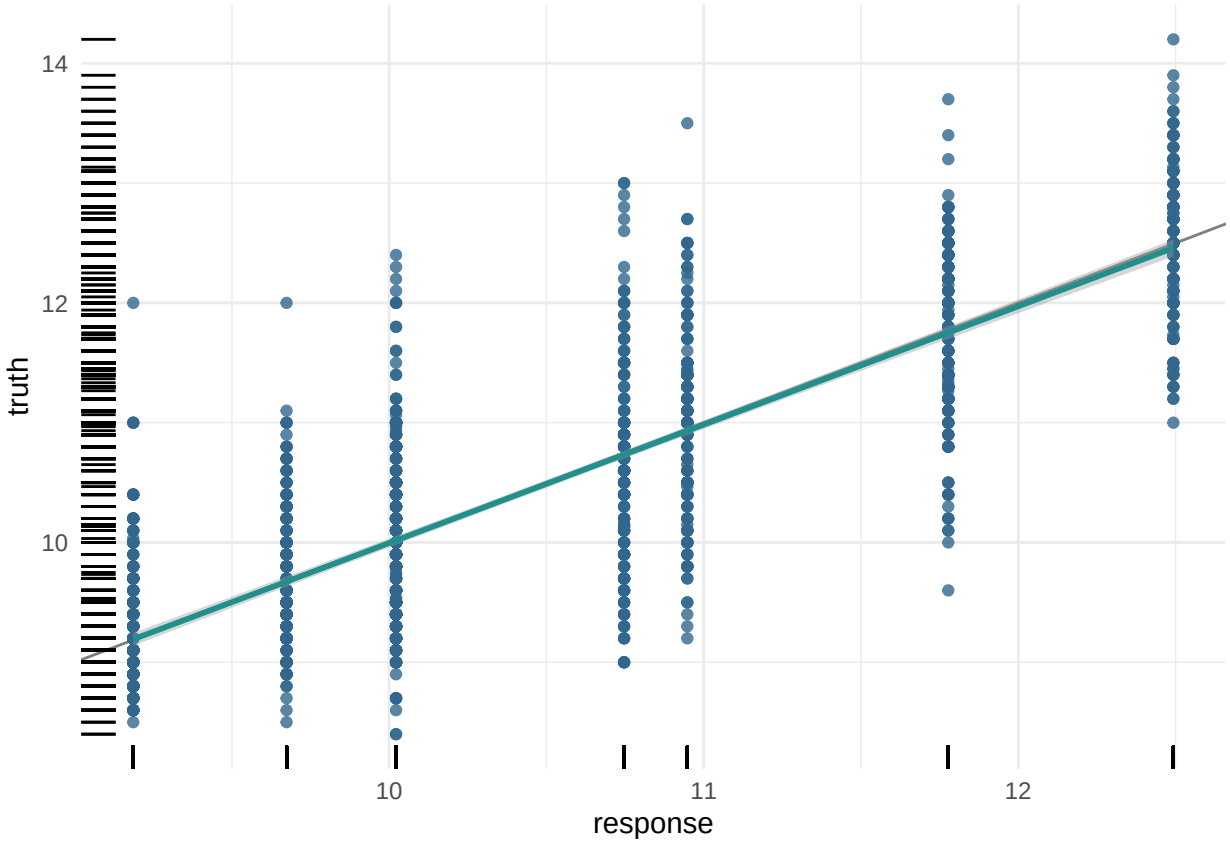
Predicting

Predicting from trained models is as simple as passing your data as a Task to the `$predict()` method of the trained Learner.

Carrying straight on from our last example, we will call the `$predict()` method of our trained learner and again will use the `row_ids` argument, but this time to pass the IDs of our test set:

The `$predict()` method returns an object inheriting from `Prediction`, in this case `PredictionRegr` as this is a regression task.

Similarly to plotting `Tasks`, `mlr3viz` provides an `autoplot()` method for `Prediction` objects.



Changing the Prediction Type

While predicting a single numeric quantity is the most common prediction type in regression, it is not the only prediction type. Several regression models can also predict standard errors. To predict this, the `$predict_type` field of a `LearnerRegr` must be changed from "response" (the default) to `"se"` before training. The `"rpart"` learner we used above does not support predicting standard errors, so in the example below we will use a linear regression model (`lrn("regr.lm")`).

```
library(mlr3learners)
```

```
## Warning: package 'mlr3learners' was built under R version 4.3.2
```

```
lrn_lm = lrn("regr.lm", predict_type = "se")
lrn_lm$train(tsk_wine_alc, splits$train)
lrn_lm$predict(tsk_wine_alc, splits$test)
```

```
## <PredictionRegr> for 1616 observations:
##     row_ids truth  response         se
##           8  8.80  8.778956 0.02599166
##          37 10.00 10.573302 0.01966323
##          40  8.60  9.327223 0.02656098
## ---
##        4866 13.60 12.785176 0.02451457
##        4887 12.15 12.139407 0.04774855
##        4897 12.80 12.555868 0.02417406
```

Evaluation

Perhaps the most important step of the applied machine learning workflow is evaluating model performance. Without this, we would have no way to know if our trained model makes very accurate predictions, is worse than randomly guessing, or somewhere in between. We will continue with our decision tree example to establish if the quality of our predictions is 'good', first we will rerun the above code so it is easier to follow along.

```
lrn_rpart = lrn("regr.rpart")
tsk_wine_alc = as_task_regr(wine_data, target = "alcohol",
id = "White Wine Alcohol Content")
splits = partition(tsk_wine_alc)
lrn_rpart$train(tsk_wine_alc, splits$train)
prediction = lrn_rpart$predict(tsk_wine_alc, splits$test)
```

Measures

The quality of predictions is evaluated using measures that compare them to the ground truth data for supervised learning tasks. Similarly to Tasks and Learners, the available measures in `mlr3` are stored in a dictionary called `mlr_measures` and can be accessed with `msr()`:

```
## <MeasureRegrSimple:regr.mae>: Mean Absolute Error
## * Packages: mlr3, mlr3measures
## * Range: [0, Inf]
## * Minimize: TRUE
## * Average: macro
## * Parameters: list()
## * Properties: -
## * Predict type: response
```

This measure compares the absolute difference ('error') between true and predicted values: Lower values are considered better (`Minimize: TRUE`), which is intuitive as we would like the true values, to be identical (or as close as possible) in value to the predicted values. We can see that the range of possible values the learner can take is from (`Range: [0, Inf]`), it has no special properties (`Properties:-`), it evaluates `response` type predictions for regression models (`Predict type: response`), and it has no control parameters (`Parameters: list()`).

Now let us see how to use this measure for scoring our predictions.

Scoring Predictions

Usually, supervised learning measures compare the difference between predicted values and the ground truth. `mlr3` simplifies the process of bringing these quantities together by storing the predictions and true outcomes in the `Prediction` object as we have already seen.

```
prediction
```

```
## <PredictionRegr> for 1616 observations:
##     row_ids truth  response
##           2  9.50  9.955302
##           7  9.60  9.955302
##           9  9.50  9.955302
## ---
##        4864 13.30 12.551593
##        4868 13.00 12.551593
##        4887 12.15 12.297192
```

To calculate model performance, we simply call the `$score()` method of a Prediction object and pass as a single argument the measure that we want to compute:

```
## regr.mae
## 0.497749
```

Note that all task types have default measures that are used if the argument to `$score()` is omitted, for regression this is the mean squared error (`msr("regr.mse")`), which is the squared difference between true and predicted values averaged over the test set.

It is possible to calculate multiple measures at the same time by passing multiple measures to `$score()`. For example, below we compute performance for mean squared error (`"regr.mse"`) and mean absolute error (`"regr.mae"`) – note we use `msrs()` to load multiple measures at once.

```
measures = msrs(c("regr.mse", "regr.mae"))
prediction$score(measures)
```

```
##  regr.mse  regr.mae
## 0.4069196 0.4977490
```

Technical Measures

This section covers advanced ML or technical details. `mlr3` also provides measures that do not quantify the quality of the predictions of a model, but instead provide 'meta'-information about the model. These include:

`msr("time_train")` – The time taken to train a model. `msr("time_predict")` – The time taken for the model to make predictions. `msr("time_both")` – The total time taken to train the model and then make predictions. `msr("selected_features")` – The number of features selected by a model, which can only be used if the model has the "selected_features" property.

For example, we could score our decision tree to see how many seconds it took to train the model and make predictions:

```
measures = msrs(c("time_train", "time_predict", "time_both"))
prediction$score(measures, learner = lrn_rpart)
```

```
##   time_train time_predict    time_both
##         0.02         0.02         0.04
```

Notice a few key properties of these measures:

1. `time_both` is simply the sum of `time_train` and `time_predict`.
2. We had to pass `learner = lrn_rpart` to `$score()` as these measures have the `requires_learner` property:

The `selected_features` measure calculates how many features were used in the fitted model.

Filter Data

```
tsk_wine_alc = as_task_regr(wine_data, target = "alcohol",
id = "White Wine Alcohol Content")
tsk_wine_alc$select(c("citric.acid", "fixed.acidity", "pH", "sulphates",
"total.sulfur.dioxide", "volatile.acidity")) # keeps only these features

# retrieve all data
tsk_wine_alc$data()
```

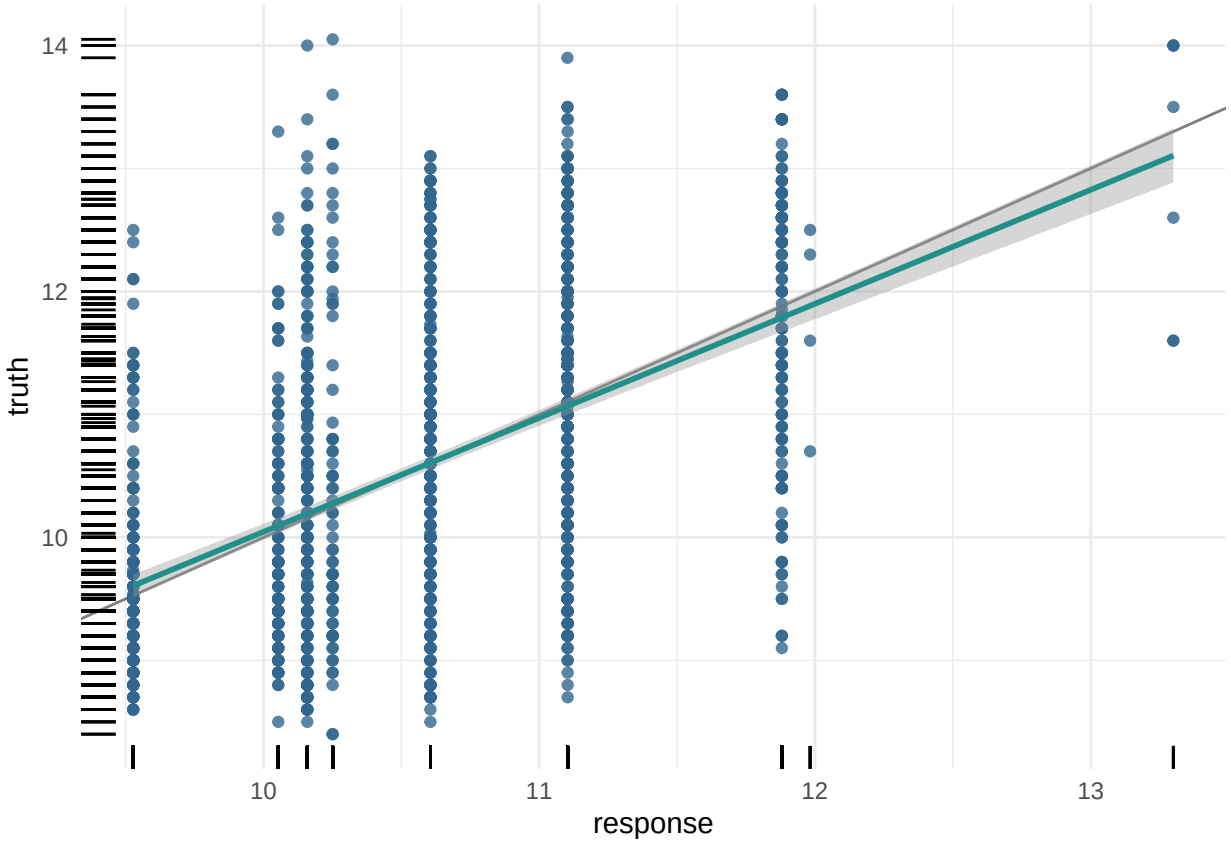
```
##       alcohol citric.acid fixed.acidity   pH sulphates total.sulfur.dioxide
##    1:     8.8        0.36           7.0 3.00      0.45                  170
##    2:     9.5        0.34           6.3 3.30      0.49                  132
##    3:    10.1        0.40           8.1 3.26      0.44                   97
##    4:     9.9        0.32           7.2 3.19      0.40                  186
##    5:     9.9        0.32           7.2 3.19      0.40                  186
##   ---
## 4894:    11.2        0.29           6.2 3.27      0.50                   92
## 4895:     9.6        0.36           6.6 3.15      0.46                  168
## 4896:     9.4        0.19           6.5 2.99      0.46                  111
## 4897:    12.8        0.30           5.5 3.34      0.38                  110
## 4898:    11.8        0.38           6.0 3.26      0.32                   98
##       volatile.acidity
##    1:             0.27
##    2:             0.30
##    3:             0.28
##    4:             0.23
##    5:             0.23
##   ---
## 4894:             0.21
## 4895:             0.32
## 4896:             0.24
## 4897:             0.29
## 4898:             0.21
```

```
tsk_wine_alc2 <- tsk_wine_alc
```

Used when some variables need be dropped.

	x
Features1	citric.acid
Features2	fixed.acidity
Features3	pH
Features4	sulphates
Features5	total.sulfur.dioxide
Features6	volatile.acidity
Target	alcohol

```
lrn_rpart$train(tsk_wine_alc2, row_ids = splits$train)
```




```
library(mlr3learners)
lrn_lm = lrn("regr.lm", predict_type = "se")
lrn_lm$train(tsk_wine_alc2, splits$train)

lrn_lm$predict(tsk_wine_alc2, splits$test)
```

```
## <PredictionRegr> for 1616 observations:
##     row_ids truth  response         se
##           4   9.9  9.755277 0.03639293
##           7   9.6 10.572062 0.03495069
##           8   8.8  9.867328 0.03449675
## ---
##        4767  12.4 10.662073 0.02505970
##        4803  12.8 10.874312 0.04490111
##        4866  13.6 11.096791 0.04157617
```

```
##  regr.mae
## 0.8559653
```

```
measures = msrs(c("regr.mse", "regr.mae"))
prediction2$score(measures)
```

```
##  regr.mse  regr.mae
## 1.1439437 0.8559653
```

Since a lower MAE indicates superior model accuracy, the model that used all the features is better.

```
measures = msrs(c("time_train", "time_predict", "time_both"))
prediction2$score(measures, learner = lrn_rpart)
```

```
##   time_train time_predict    time_both
##         0.00         0.01         0.01
```

```
measures = msrs(c("regr.mse", "regr.mae"))

cbind(c("Model 1", prediction$score(measures)), c("Model 2", prediction2$score(measures)))
```

```
##          [,1]                [,2]
##          "Model 1"           "Model 2"
## regr.mse "0.406919644301724" "1.14394370077004"
## regr.mae "0.497748976767655" "0.85596528015898"
```