Introduction to Machine Learning

Nicolas Attalides

Welcome to all!

- VIENNA<-R
- LONDON R
- BRIGHTON R
- R GLASGOW
- PORTSMOUTH R USER GROUP MEETUP
- WARWICK R USER GROUP (R PROGRAMMING LANGUAGE)
- CARDIFF - THE CARDIFF R USER GROUP
- R-LADIES BELGRADE
- AALBORGRUG
- LUCERNE R USER GROUP
- BELGRADER
- ATHENSR
- R-LADIES BUCHAREST
- BERLIN R USERS GROUP
- TURKISH COMMUNITY OF R
- Other groups that I might have missed ☺ (apologies this was the list from meetup.com)
- Anyone from anywhere! ☺

Who am I?

- **Name:** Nicolas Attalides
- **Coding in R since:** 2005 (yes that's before RStudio!)
- **Profession:** Senior Data Scientist and trainer (6+ yrs.)
- **Education:** PhD in Statistical Science from UCL (2015)
- **R Status:** A never-ending evolving R dinosaur
- **Hobbies:** Tennis and coding (not at the same time)

Workshop Setup:

Wi-Fi

- Network Name: N/A
- Password: N/A

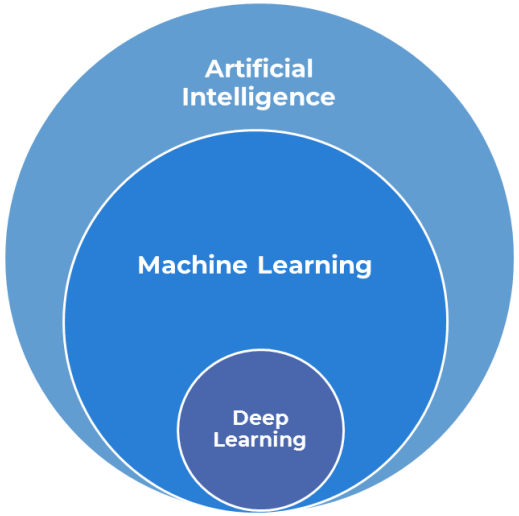
Resources

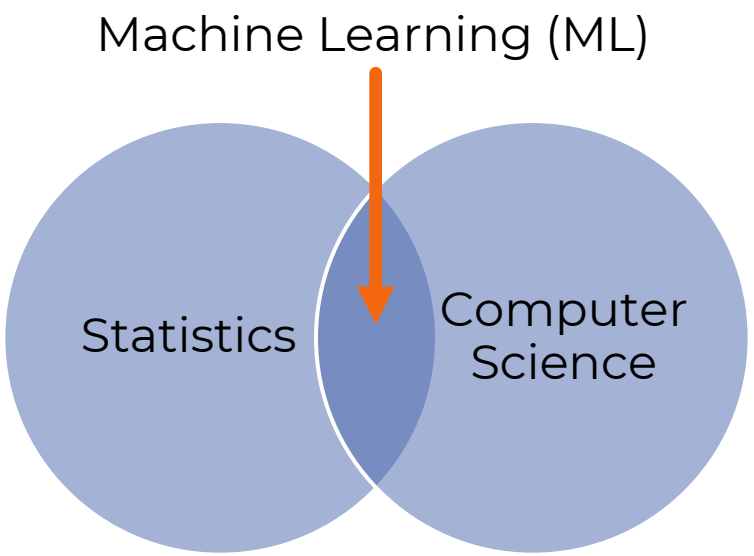
- R (version 3.6.3)
- RStudio (version 1.3.959)

Packages

- tidyverse (version 1.3.0)
- tidymodels (version 0.1.2)
- rpart (version 4.1-15)
- randomForest (version 4.6-14)
- xgboost (version 1.3.1.1)
- + some optional packages for visualisations

What is Machine Learning?





Some breakthroughs in this area are machine vision and reinforcement learning (also known as deep learning) with some exciting examples such as DeepMind's AlphaGo.

*Image reference: https://www.lexalytics.com/

Machine Learning & R

There are many R packages dedicated to machine learning that you can install from CRAN. Some popular ones are **{rpart}**, **{randomForest}** and **{xgboost}** to name a few.

In this workshop we are going to use **{tidymodels}** which is a **framework package** aiming to streamline ML tasks and unify the interface of the various algorithms. It also follows the tidyverse principles.

Check out: https://cran.r-project.org/web/views/MachineLearning.html

Topics

- Workshop aim:

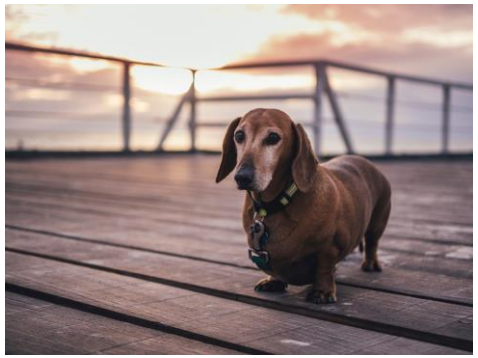
Learn how to design, fit and evaluate a machine learning model to solve a specific problem.

- Topics:
  - Define the problem and evaluation metrics
  - Load, prepare and split the data to train and test sets
  - Design the formula for the model
  - Choose an algorithm and fit a model
  - Predict and evaluate a fitted model

Define the problem

Machine Learning is commonly used to solve two types of problems:

- **Regression –** This is when… what you are trying to predict (the **target variable**) is **numeric**, for example the number of units sold of a product.
- **Classification –** This is when… what you are trying to predict (the **target variable**) is **categorical** (or a **class**), for example "hot dog or not hot dog".



Types of learning

In Machine Learning there are different types of learning that can be done … this depends on the available data and the outcome:

Type	Description	Example
Supervised	Target variable is **known**	Predict which customer is going to cancel their subscription service
Unsupervised	Target variable is **unknown**	Group “similar” customers into categories
Semi-supervised	Target variable is **partially known**	Detect credit card fraud
Reinforcement	Maximise a reward by taking actions	Win a game of chess

Metrics for model performance

There are many ways that you can measure the performance of your model. Below is a list of some of the typical metrics used for regression and classification problems:

- **Regression**
  - MSE – Mean Squared Error
  - RMSE – Root Mean Squared Error
  - MAE – Mean Absolute Error
  - R^2 – A measure that is related to MSE and is scaled between 0 and 1
- **Classification**
  - Accuracy
  - Precision
  - Recall
  - AUC – Area Under the Curve

You can always create your own custom metric!

Load and prepare data

One of the most essential tasks (and usually the most time consuming – but sometimes can feel like a relaxing activity but can be equally frustrating) is … **data cleaning**! 😱

It is important that you **understand the data**! What are the types? Any missing values? Is there correlation in your features?

A good Exploratory Data Analysis (EDA) is the best starting point, otherwise…

💩 → 💻 → 💩

Data – Used for live coding examples

The data that we will use during this workshop are:

Red wine quality of the Portuguese "Vinho Verde" wine which includes:

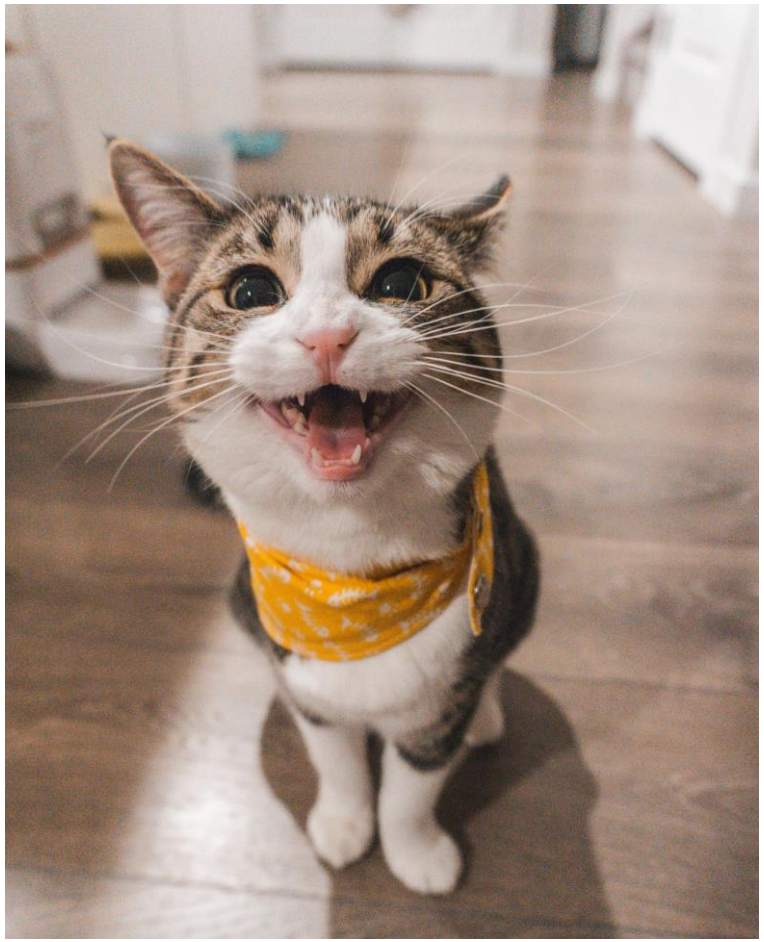
Physicochemical test results (such as PH) and quality assessment graded by experts - 0 (very bad) and 10 (very excellent)

DATASET CITATION: P. Cortez, A. Cerdeira, F. Almeida, T. Matos and J. Reis.
Modeling wine preferences by data mining from physicochemical properties.
In Decision Support Systems, Elsevier, 47(4):547-553. ISSN: 0167-9236.

Data Dictionary

- fixed acidity (tartaric acid - g / dm^3)
- volatile acidity (acetic acid - g / dm^3)
- citric acid (g / dm^3)
- residual sugar (g / dm^3)
- chlorides (sodium chloride - g / dm^3)
- free sulfur dioxide (mg / dm^3)
- total sulfur dioxide (mg / dm^3)
- density (g / cm^3)
- pH
- sulphates (potassium sulphate - g / dm^3)
- alcohol (% by volume)
- quality (score between 0 and 10)

Live Coding Example 1



Download the wine quality dataset from:
https://github.com/nattalides/BarcelonaR_workshop_Introduction_to_Machine_Learning/blob/master/data/data.rds

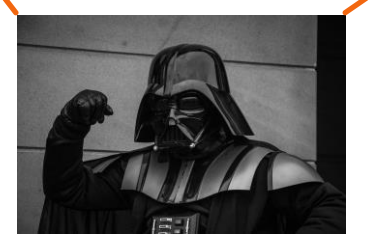
1. Load and view the data.
2. Do a quick exploratory data analysis (EDA).
3. Fix column names.
4. Remove any missing values.
5. Split the data into:
   a) Train set
   b) Test set

For more practice datasets go to:
https://archive.ics.uci.edu/ml/index.php

Live Coding Example 1

column names

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
1	7.4	0.700	0.00	1.90	0.076	11	34	0.9978	3.51	0.56	9.4	5
2	7.8	NA	0.00	2.60	0.098	25	67	0.9968	3.20	0.68	9.8	5
3	7.8	0.760	0.04	2.30	0.092	15	54	0.9970	3.26	0.65	9.8	5
4	11.2	0.280	0.56	1.90	0.075	17	60	0.9980	3.16	0.58	9.8	6
5	7.4	0.700	0.00	1.90	0.076	11	34	0.9978	3.51	0.56	9.4	5
6	7.4	NA	0.00	1.80	0.075	13	40	0.9978	3.51	0.56	9.4	5
7	7.9	0.600	0.06	1.60	0.069	15	59	0.9964	3.30	0.46	9.4	5
8	7.3	0.650	0.00	1.20	0.065	15	21	0.9946	3.39	0.47	NA	7
9	7.8	0.580	0.02	2.00	0.073	9	18	0.9968	3.36	0.57	9.5	7
10	7.5	NA	0.36	6.10	0.071	17	102	0.9978	3.35	0.80	10.5	5
11	6.7	0.580	0.08	1.80	0.097	15	65	0.9959	3.28	0.54	9.2	5
12	7.5	0.500	0.36	6.10	0.071	17	102	0.9978	3.35	0.80	10.5	5
13	5.6	0.615	0.00	1.60	0.089	16	59	0.9943	3.58	0.52	9.9	5
14	7.8	0.610	0.29	1.60	0.114	9	29	0.9974	3.26	1.56	9.1	5



missing values

Live Coding Example 1

A summary of the data frame

```
 fixed acidity   volatile acidity  citric acid    residual sugar     chlorides       free sulfur dioxide
 Min.   : 4.60   Min.   :0.1200   Min.   :0.000   Min.   : 0.900   Min.   :0.01200   Min.   : 1.00
 1st Qu.: 7.10   1st Qu.:0.3900   1st Qu.:0.090   1st Qu.: 1.900   1st Qu.:0.07000   1st Qu.: 7.00
 Median : 7.90   Median :0.5200   Median :0.260   Median : 2.200   Median :0.07900   Median :14.00
 Mean   : 8.32   Mean   :0.5275   Mean   :0.271   Mean   : 2.539   Mean   :0.08747   Mean   :15.87
 3rd Qu.: 9.20   3rd Qu.:0.6400   3rd Qu.:0.420   3rd Qu.: 2.600   3rd Qu.:0.09000   3rd Qu.:21.00
 Max.   :15.90   Max.   :1.5800   Max.   :1.000   Max.   :15.500   Max.   :0.61100   Max.   :72.00
                 NA's   :3
 total sulfur dioxide    density             pH          sulphates         alcohol         quality
 Min.   :  6.00       Min.   :0.9901   Min.   :2.740   Min.   :0.3300   Min.   : 8.40   Min.   :3.000
 1st Qu.: 22.00       1st Qu.:0.9956   1st Qu.:3.210   1st Qu.:0.5500   1st Qu.: 9.50   1st Qu.:5.000
 Median : 38.00       Median :0.9968   Median :3.310   Median :0.6200   Median :10.20   Median :6.000
 Mean   : 46.47       Mean   :0.9967   Mean   :3.311   Mean   :0.6581   Mean   :10.43   Mean   :5.636
 3rd Qu.: 62.00       3rd Qu.:0.9978   3rd Qu.:3.400   3rd Qu.:0.7300   3rd Qu.:11.10   3rd Qu.:6.000
 Max.   :289.00       Max.   :1.0037   Max.   :4.010   Max.   :2.0000   Max.   :14.90   Max.   :8.000
                                                                        NA's   :5
```

missing values

Live Coding Example 1

A correlation matrix of the data

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH
fixed acidity	1.00000000	NA	0.67170343	0.114776724	0.093705186	-0.153794193	-0.11318144	0.66804729	-0.68297819
volatile acidity	NA	1	NA	NA	NA	NA	NA	NA	NA
citric acid	0.67170343	NA	1.00000000	0.143577162	0.203822914	-0.060978129	0.03553302	0.36494718	-0.54190414
residual sugar	0.11477672	NA	0.14357716	1.000000000	0.055609535	0.187048995	0.20302788	0.35528337	-0.08565242
chlorides	0.09370519	NA	0.20382291	0.055609535	1.000000000	0.005562147	0.04740047	0.20063233	-0.26502613
free sulfur dioxide	-0.15379419	NA	-0.06097813	0.187048995	0.005562147	1.000000000	0.66766645	-0.02194583	0.07037750
total sulfur dioxide	-0.11318144	NA	0.03553302	0.203027882	0.047400468	0.667666450	1.00000000	0.07126948	-0.06649456
density	0.66804729	NA	0.36494718	0.355283371	0.200632327	-0.021945831	0.07126948	1.00000000	-0.34169933
pH	-0.68297819	NA	-0.54190414	-0.085652422	-0.265026131	0.070377499	-0.06649456	-0.34169933	1.00000000
sulphates	0.18300566	NA	0.31277004	0.005527121	0.371260481	0.051657572	0.04294684	0.14850641	-0.19664760
alcohol	NA	NA	NA	NA	NA	NA	NA	NA	NA
quality	0.12405165	NA	0.22637251	0.013731637	-0.128906560	-0.050656057	-0.18510029	-0.17491923	-0.05773139

	sulphates	alcohol	quality
fixed acidity	0.183005664	NA	0.12405165
volatile acidity	NA	NA	NA
citric acid	0.312770044	NA	0.22637251
residual sugar	0.005527121	NA	0.01373164
chlorides	0.371260481	NA	-0.12890656
free sulfur dioxide	0.051657572	NA	-0.05065606
total sulfur dioxide	0.042946836	NA	-0.18510029
density	0.148506412	NA	-0.17491923
pH	-0.196647602	NA	-0.05773139
sulphates	1.000000000	NA	0.25139708
alcohol	NA	1	NA
quality	0.251397079	NA	1.00000000

missing values

Live Coding Example 1

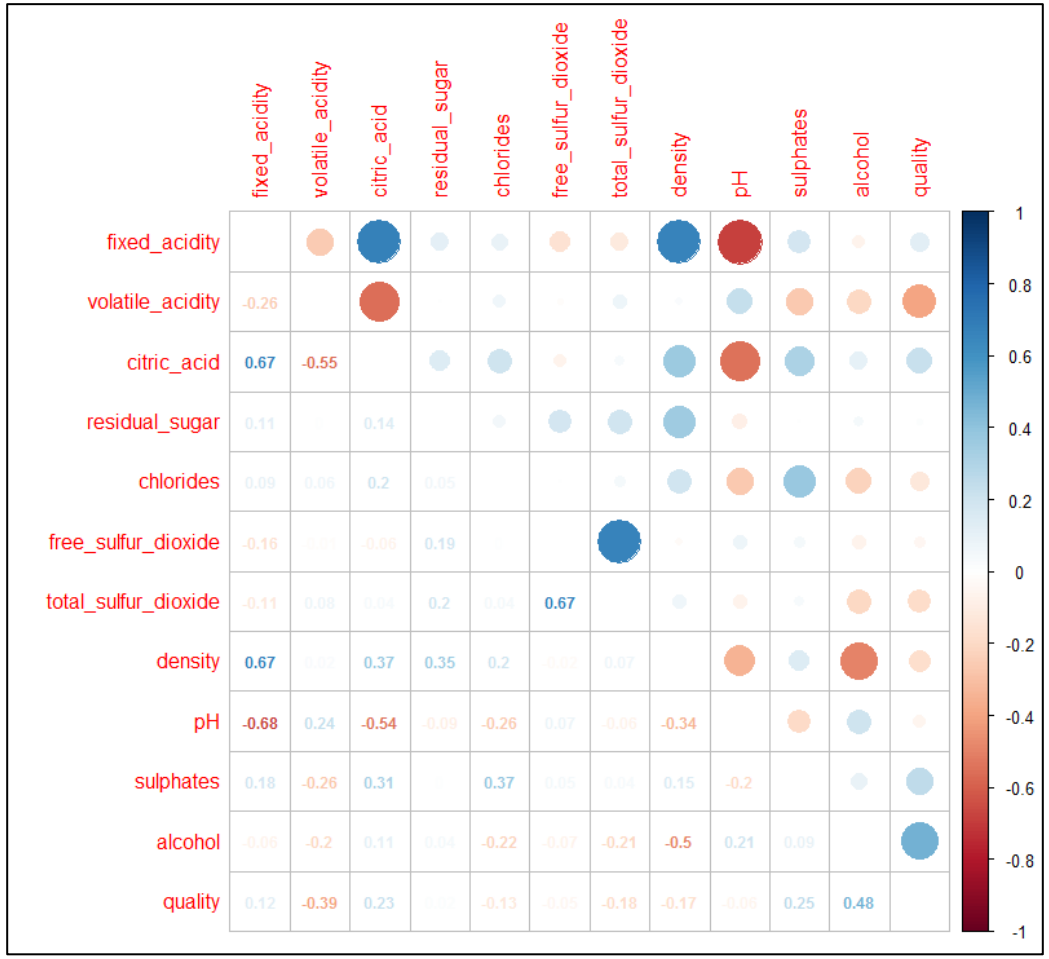
```
library(tidyverse)
library(tidymodels)

# Example 1

# Load and view the data.
df <- readRDS("data/data.rds")

view(df)

# Do some exploratory data analysis (EDA).
# 1. A summary of the data frame
df %>% summary

# 2. A correlation plot of the data
df %>% cor()
```

Live Coding Example 1

```r
# 3. Fix column names
colnames(df) <- df %>%
  colnames() %>% str_replace_all(pattern = " ", replacement = "_")

# 4. Remove any missing values
df <- df %>% drop_na()

df %>% summary()
```

```
fixed_acidity    volatile_acidity  citric_acid     residual_sugar     chlorides       free_sulfur_dioxide total_sulfur_dioxide
Min.   : 4.600   Min.   :0.1200   Min.   :0.0000   Min.   : 0.900   Min.   :0.01200   Min.   : 1.00       Min.   :  6.00
1st Qu.: 7.100   1st Qu.:0.3900   1st Qu.:0.0900   1st Qu.: 1.900   1st Qu.:0.07000   1st Qu.: 7.00       1st Qu.: 22.00
Median : 7.900   Median :0.5200   Median :0.2600   Median : 2.200   Median :0.07900   Median :14.00       Median : 38.00
Mean   : 8.323   Mean   :0.5274   Mean   :0.2716   Mean   : 2.538   Mean   :0.08744   Mean   :15.85       Mean   : 46.37
3rd Qu.: 9.200   3rd Qu.:0.6400   3rd Qu.:0.4250   3rd Qu.: 2.600   3rd Qu.:0.09000   3rd Qu.:21.00       3rd Qu.: 62.00
Max.   :15.900   Max.   :1.5800   Max.   :1.0000   Max.   :15.500   Max.   :0.61100   Max.   :72.00       Max.   :289.00
   density            pH          sulphates         alcohol         quality
Min.   :0.9901   Min.   :2.740   Min.   :0.3300   Min.   : 8.40   Min.   :3.000
1st Qu.:0.9956   1st Qu.:3.210   1st Qu.:0.5500   1st Qu.: 9.50   1st Qu.:5.000
Median :0.9968   Median :3.310   Median :0.6200   Median :10.20   Median :6.000
Mean   :0.9967   Mean   :3.311   Mean   :0.6582   Mean   :10.43   Mean   :5.637
3rd Qu.:0.9978   3rd Qu.:3.400   3rd Qu.:0.7300   3rd Qu.:11.10   3rd Qu.:6.000
Max.   :1.0037   Max.   :4.010   Max.   :2.0000   Max.   :14.90   Max.   :8.000
```

Live Coding Example 1




```r
# A nice way to visualise correlation
library(corrplot)
df %>% cor() %>%
  corrplot.mixed(upper = "circle",
                 tl.cex = 1,
                 tl.pos = 'lt',
                 number.cex = 0.75)
```

Live Coding Example 1

```
# 5. Split the data into: a) Train set, b) Test set

set.seed(12345) # Fix randomisation by setting the seed (reproducibility)

# All functions below come from the {rsample} package
data_split <- initial_split(df, prop = 0.8) # Use 80% of the data for training

train_data <- training(data_split)

test_data  <- testing(data_split)
```

df	1591 obs. of 12 variables
test_data	318 obs. of 12 variables
train_data	1273 obs. of 12 variables

You can also specify stratified sampling (for class imbalance)

Design the formula for the model

A **formula** is an important element of machine learning because it is "a symbolic description of the model to be fitted" (taken from `?lm()` help). It allows us to specify what the **target** variable is and what **features** we will use which we separate by a tilde symbol (~).

```
target ~ features
```

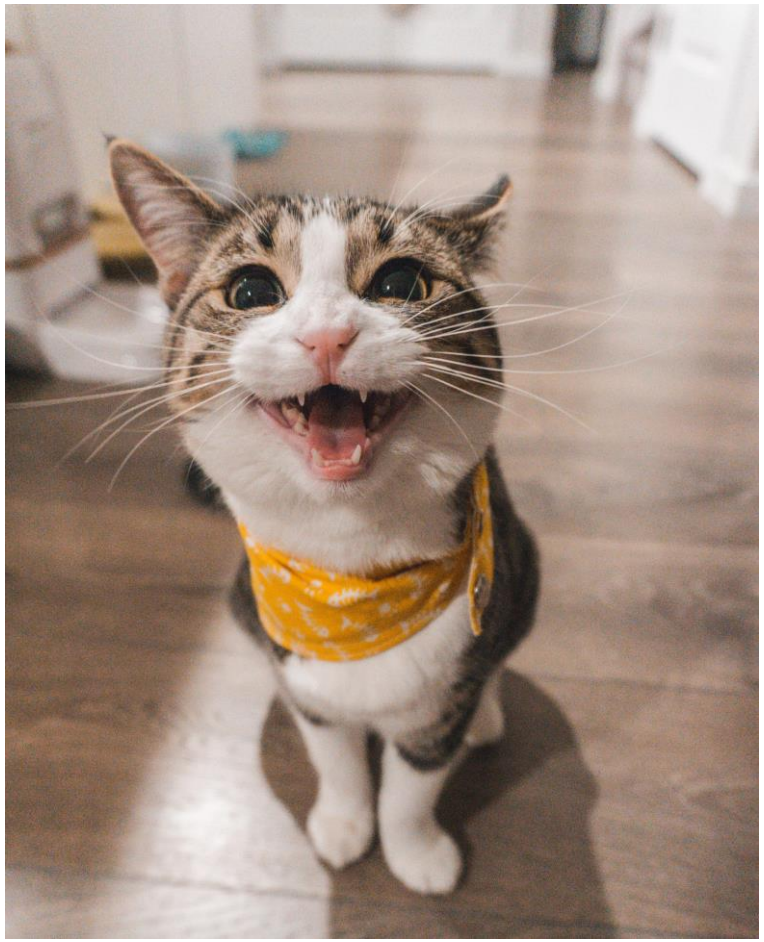
For more details check out `?formula`

Design the formula for the model

Target	Features
This is the "thing" you are trying to predict. Depending on the problem this is going to be a ***number*** or a ***category*** (class). In maths or statistics this is also known as ***"y"*** or ***"response variable"***	These are your ***variables*** that you have available to fit a model in order to predict the target variable. These can also be numbers or categories. In maths or statistics these are also known as ***"x"*** or ***"covariates"*** or ***"explanatory variables"***

Feature engineering is a method of adding or creating more features to your formula in the hope of better predictions and model performance.

Live Coding Example 2



For the below tasks, please store each formula in a different R object.

1. Using the loaded data what is/are:
   a) The target variable (is it numeric or a class?)
   b) The features of the model
2. Design a simple formula to predict the target variable.
3. Get creative with the features and design other formulas!

Live Coding Example 2

```
# 2. Design a simple formula to predict the target variable.

# Formula that uses all available features
fmla1 <- formula(quality ~ fixed_acidity + volatile_acidity + citric_acid +
                   residual_sugar + chlorides + free_sulfur_dioxide +
                   total_sulfur_dioxide + density + pH + sulphates + alcohol)

# Or the same as above but in shorter format
fmla1 <- formula(quality ~ . ) # The "." says use all available features

# 3. Get creative and engineer some features to design other formulas!

# Remove some of the correlated features
fmla2 <- formula(quality ~ fixed_acidity + volatile_acidity + residual_sugar +
                   chlorides + free_sulfur_dioxide + pH + sulphates + alcohol)

# Engineer some new features
fmla3 <- formula(quality ~ log(volatile_acidity) + log(alcohol))
```

Choose an algorithm and fit a model

A challenging task when building machine learning models is choosing which **algorithm** to use. There is a huge variety of options to select from! 😱

Unfortunately there is no right or wrong answer for this choice, however it is often common for this decision to be influenced by the model's **explainability**, **interpretability** and overall model **performance**.

- **Explainability –** literally explain exactly what is happening with the model and the predictions it generates
- **Interpretability –** able to find out the mechanics of the model and the predictions it generates but without necessarily knowing why

Fit a linear regression model

A typical starting place for a regression type problem is to fit a **linear regression model**. We demonstrate here how this can easily be done within **{tidymodels}** by using the functionality of the **{parsnip}** package. In the following example we explore how we can use other algorithms.

```
# Fit a linear regression model to the data

lm_fit <- # Create the object that will store the model fit
  linear_reg() %>% # Model type: Linear Regression
  set_mode("regression") %>% # Model mode: regression
  set_engine("lm") %>% # Computational engine: lm
  fit(fmla1, data = train_data) # Supply formula & train data and fit model
```

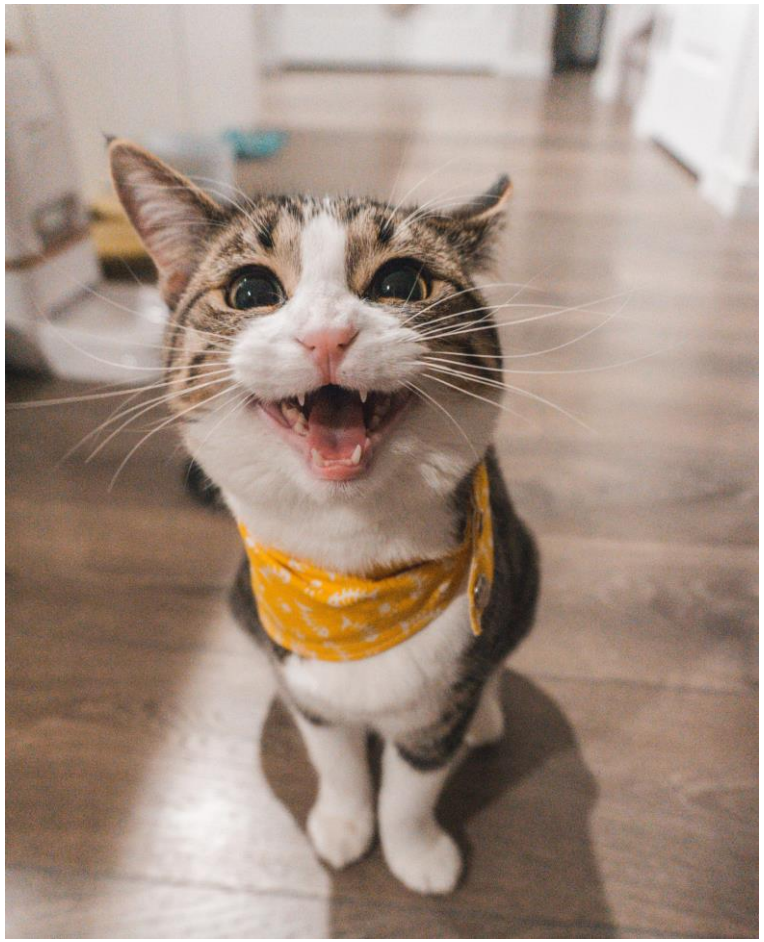
Some algorithms can be used for both regression or classification problems… that is why you should specify the type of problem with the function `set_mode()`

Fit a linear regression model

```
print(lm_fit$fit)
```

```
Call:
stats::lm(formula = formula, data = data)

Coefficients:
         (Intercept)        fixed_acidity     volatile_acidity          citric_acid       residual_sugar            chlorides
           18.274260             0.010860            -1.027444            -0.078096             0.010779            -1.791911
 free_sulfur_dioxide  total_sulfur_dioxide              density                   pH            sulphates              alcohol
             0.006088            -0.003397           -14.013720            -0.433262             0.971165             0.267353
```

Live Coding Example 3



For this example, select one of the formulas you designed. You can always switch to another formula very easily.

1. Fit a model using the following algorithms:
   a) Decision Tree
   b) Random Forest
   c) Xgboost

and store the model fit for each one in different R objects.

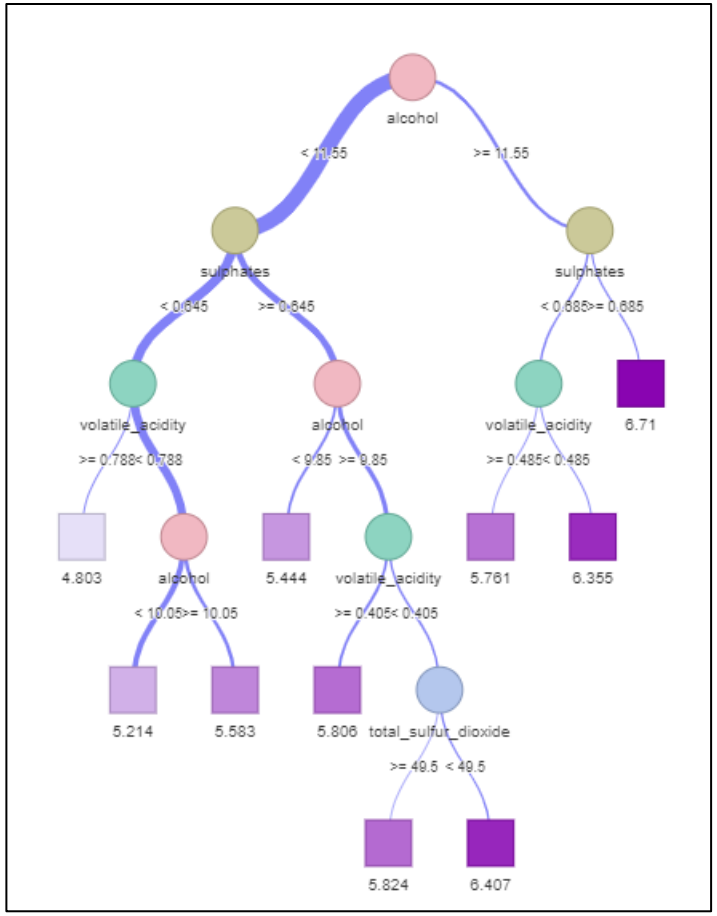
Don't forget to install the necessary packages for the algorithms!

Live Coding Example 3

```
# Example 3

# 1 a) Decision Tree
# You need to install {rpart}

dt_fit <-
  decision_tree() %>%
  set_mode("regression") %>%
  set_engine("rpart") %>%
  fit(fmla1, data = train_data)

print(dt_fit$fit)

# Nice way to visualise a decision tree ...
# Need to install {visNetwork} and {sparkline}
library(visNetwork)
library(sparkline)
visTree(dt_fit$fit)
```

Live Coding Example 3

```
n= 1273

node), split, n, deviance, yval
      * denotes terminal node

 1) root 1273 816.65670 5.641791
   2) alcohol< 11.55 1072 573.99160 5.502799
     4) sulphates< 0.645 636 270.01730 5.294025
       8) volatile_acidity>=0.7875 66  40.43939 4.803030 *
       9) volatile_acidity< 0.7875 570 211.82460 5.350877
        18) alcohol< 10.05 359 100.48470 5.214485 *
        19) alcohol>=10.05 211  93.29858 5.582938 *
     5) sulphates>=0.645 436 235.81650 5.807339
      10) alcohol< 9.85 151  51.27152 5.443709 *
      11) alcohol>=9.85 285 154.00000 6.000000
        22) volatile_acidity>=0.405 160  74.99375 5.806250 *
        23) volatile_acidity< 0.405 125  65.31200 6.248000
          46) total_sulfur_dioxide>=49.5 34  14.94118 5.823529 *
          47) total_sulfur_dioxide< 49.5 91  41.95604 6.406593 *
   3) alcohol>=11.55 201 111.50250 6.383085
     6) sulphates< 0.685 108  53.87963 6.101852
      12) volatile_acidity>=0.485 46  22.36957 5.760870 *
      13) volatile_acidity< 0.485 62  22.19355 6.354839 *
     7) sulphates>=0.685 93  39.16129 6.709677 *
```



Live Coding Example 3

```
# 1 b) Random Forest
# You need to install {randomForest}

rf_fit <-
  rand_forest() %>%
  set_mode("regression") %>%
  set_engine("randomForest") %>%
  fit(fmla1, data = train_data)

print(rf_fit$fit)

# 1 c) Xgboost
# You need to install {xgboost}

xgboost_fit <-
  boost_tree() %>%
  set_mode("regression") %>%
  set_engine("xgboost") %>%
  fit(fmla1, data = train_data)

print(xgboost_fit$fit)
```

Live Coding Example 3

Random Forest

```
Call:
 randomForest(x = as.data.frame(x), y = y)
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 3

          Mean of squared residuals: 0.3369284
                    % Var explained: 47.48
```

xgboost

```
##### xgb.Booster
raw: 38.8 Kb
call:
  xgboost::xgb.train(params = list(eta = 0.3, max_depth = 6, gamma = 0,
    colsample_bytree = 1, min_child_weight = 1, subsample = 1),
    data = x, nrounds = 15, verbose = 0, objective = "reg:linear",
    nthread = 1)
params (as set within xgb.train):
  eta = "0.3", max_depth = "6", gamma = "0", colsample_bytree = "1", min_child_weight = "1", subsample = "1", objective = "reg:linear", nthread = "1", silent = "1"
xgb.attributes:
  niter
# of features: 11
niter: 15
nfeatures : 11
```

Predict and evaluate a model fit

Questions:

- How well can this model predict our target variable?
- How can we measure the performance of the model fit so that we can compare it with other models?

This is where the **test set** comes into action! It is important to note that the fitted (or trained) model has never ever ever ever ever… ever seen the test set.

We use the feature values of the test set to **predict** the target variable.

Predict and evaluate a model fit

The `predict()` function requires us to supply a **model fit** and the **test set** in order to generate predictions for the target variable. These get automatically stored in the column `.pred`

```
lm_pred <- test_data %>%
  bind_cols(predict(object = lm_fit, new_data = test_data))
view(lm_pred)
```

	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_dioxide	total_sulfur_dioxide	density	pH	sulphates	alcohol	quality	.pred
1	7.4	0.700	0.00	1.90	0.076	11.0	34	0.99780	3.51	0.56	9.4	5	5.024527
2	8.5	0.280	0.56	1.80	0.092	35.0	103	0.99690	3.30	0.75	10.5	7	5.888475
3	8.1	0.560	0.28	1.70	0.368	16.0	56	0.99680	3.11	1.28	9.3	5	5.444247
4	7.4	0.590	0.08	4.40	0.086	6.0	29	0.99740	3.38	0.50	9.0	4	5.023587

Why is this happening?!

Predict and evaluate a model fit

For this specific dataset we know that the target variable is in fact an **integer** and when we inspect our predictions we can see that these are **numeric** (decimal). We can solve this issue by simply rounding the predictions to the nearest integer.

```
lm_pred <- test_data %>%
  bind_cols(predict(object = lm_fit, new_data = test_data)) %>%
  mutate(pred = round(.pred, 0))
```

Spoiler alert! This “issue” should make you think about the problem definition…

Predict and evaluate a model fit

Since this is **supervised learning** (i.e. we have the actual observations of the target variable) we calculate a metric such as the **Mean Squared Error** (MSE) – *the lower the better* – in order to measure how good or bad these predictions are in comparison to other model fits.

```
lm_mse <- lm_pred %>%
  summarise(type = "lm",
            MSE = round(mean((pred - quality)^2), 4))

view(lm_mse)
```

	type	MSE
1	lm	0.4843

A **residual** (also referred to as **error**) is the difference between the observed outcome (truth) and the predicted outcome (estimate) of the target variable

Predict and evaluate a model fit

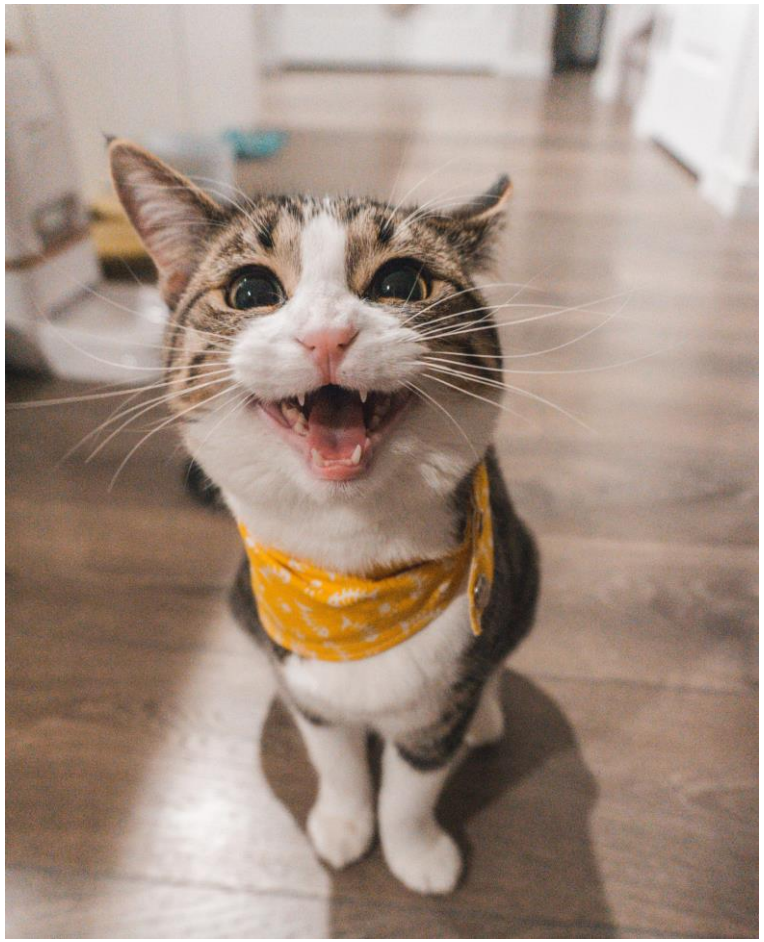
We have seen how to calculate the **Mean Squared Error** metric in an "old-school" fashion. This helps to understand the maths behind the metric.

It is useful to know that we have other options! For example, we can use the `metrics()` function from the **{yardstick}** package to calculate directly some other performance metrics!

```
metrics(lm_pred, truth = quality, estimate = pred)
```

```
# A tibble: 3 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 rmse    standard       0.696
2 rsq     standard       0.324
3 mae     standard       0.428
```

Live Coding Example 4



1. Evaluate the MSE for each of the fitted models.
2. Which model fit achieved the lowest MSE?
3. Could this have been a classification type problem? Let's discuss!

Live Coding Example 4

```
# 1 a) MSE for: Decision Tree
dt_pred <- test_data %>%
  bind_cols(predict(object = dt_fit, new_data = test_data)) %>%
  rename(pred = .pred) %>%
  mutate(pred = round(pred, 0))

dt_mse <- dt_pred %>%
  summarise(type = "dt",
            MSE = round(mean((pred - quality)^2), 4))

# 1 b) MSE for: Random Forest
rf_pred <- test_data %>%
  bind_cols(predict(object = rf_fit, new_data = test_data)) %>%
  rename(pred = .pred) %>%
  mutate(pred = round(pred, 0))

rf_mse <- rf_pred %>%
  summarise(type = "rf",
            MSE = round(mean((pred - quality)^2), 4))
```

Live Coding Example 4

```
# 1 c) MSE for: xgboost

xgboost_pred <- test_data %>%
  bind_cols(predict(object = xgboost_fit, new_data = test_data)) %>%
  rename(pred = .pred) %>%
  mutate(pred = round(pred, 0))

xgboost_mse <- xgboost_pred %>%
  summarise(type = "xgboost",
            MSE = round(mean((pred - quality)^2), 4))

# Join all results together

res <- bind_rows(lm_mse, dt_mse, rf_mse, xgboost_mse)

View(res)
```

	type	MSE
1	lm	0.4843
2	dt	0.5314
3	rf	0.3805
4	xgboost	0.4371

Live Coding Example 4

View predictions for the test set

	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_dioxide	total_sulfur_dioxide	density	pH	sulphates	alcohol	quality	pred
1	7.4	0.700	0.00	1.90	0.076	11	34	0.99780	3.51	0.56	9.4	5	5
2	8.5	0.280	0.56	1.80	0.092	35	103	0.99690	3.30	0.75	10.5	7	6
3	8.1	0.560	0.28	1.70	0.368	16	56	0.99680	3.11	1.28	9.3	5	5
4	7.4	0.590	0.08	4.40	0.086	6	29	0.99740	3.38	0.50	9.0	4	5
5	7.9	0.430	0.21	1.60	0.106	10	37	0.99660	3.17	0.91	9.5	5	5
6	6.3	0.390	0.16	1.40	0.080	11	23	0.99550	3.34	0.56	9.3	5	5
7	8.3	0.655	0.12	2.30	0.083	15	113	0.99660	3.17	0.66	9.8	5	5
8	8.8	0.610	0.30	2.80	0.088	17	46	0.99760	3.26	0.51	9.3	4	5
9	7.5	0.490	0.20	2.60	0.332	8	14	0.99680	3.21	0.90	10.5	6	6
10	8.1	0.660	0.22	2.20	0.069	9	23	0.99680	3.30	1.20	10.3	5	6
11	5.6	0.310	0.37	1.40	0.074	12	96	0.99540	3.32	0.58	9.2	5	5
12	6.6	0.500	0.04	2.10	0.068	6	14	0.99550	3.39	0.64	9.4	6	6
13	10.2	0.420	0.57	3.40	0.070	4	10	0.99710	3.04	0.63	9.6	5	6
14	7.7	0.690	0.49	1.80	0.115	20	112	0.99680	3.21	0.71	9.3	5	5
15	7.5	0.520	0.16	1.90	0.085	12	35	0.99680	3.38	0.62	9.5	7	5

Other topics in Machine Learning

- Further steps to do data **pre-processing** (such as scale, centre, PCA). Check out the **{recipes}** package which is part of **{tidymodels}** and is designed to help you for these tasks before you fit a model!
- Fit a model with resampling such as **cross-validation**. Check out the **{rsample}** package which is part of **{tidymodels}** that helps you do this.
- Model **hyper-parameter tuning**. A model can depend on parameters which might require you to tune them in order to find "the best setup" and achieve better performance. Check out the **{tune}** package which is part of **{tidymodels}** and is designed for this specific task.
- **One-hot-encoding**: What if you have a categorical variable in your set of features? This is the process by which we convert a categorical variable into columns of 1's and 0's. This might be needed for some ML algorithms that require that **all** your features are numeric.

https://www.tidymodels.org/

Next online R event!

Bayesian Multilevel Modelling with {brms}

https://www.meetup.com/BarcelonaR/events/275208091/

Thank you to our sponsors and partners!

MANGO SOLUTIONS

RStudio®