K NEAREST NEIGHBORS

BEFORE WE BEGIN: LET US RUN AN EXPERIMENT

I need two volunteers who are willing to reveal their political opinion and for which party they would vote, if we had elections today (ideally one Democrat and one Republican).

The survey questions are below (wait for instructions before you take the survey and record your answers from the digital survey **also on a piece of paper**):

- Survey for all students (except the two volunteers): https://tinyurl.com/KNearNeighTrain
- Survey for the two volunteers https://tinyurl.com/KNearNeighTest

IDEAS ON HOW TO IDENTIFY THE 2 VOLUNTEERS POLITICAL AFFILIATION?

- CL: Ask volunteers what their answers were and find nearest neighbor(s) (volunteers: do not reveal political affiliation yet).

- What were the shortcomings of the survey

- CL: Run R Script

OVERWIEW

In this session you will learn:

1. What is the underlying **idea of k-Nearest Neighbors**
2. How similarity can be measured with **Euclidean distance**
3. Why **scaling predictor variables** is important for some machine learning models
4. Why the **tidymodels package** makes it easy to work with machine learning models
5. How you can define a **recipe** to pre-process data with the `tidymodels` package
6. How you can define a **model-design** with the `tidymodels` package
7. How you can create a machine learning **workflow** with the `tidymodels` package
8. How **metrics** derived from a **confusion matrix** can be used to asses prediction quality
9. Why you have to be careful when interpreting *accuracy*, when you work with **unbalanced observations**

0. How a machine learning model can **process images** and how OCR (Optical Character Recognition) works»

ABOUT THE WINE DATASET

We will work with a publicly available ’ wine dataset[1] containing 3,198 observations about different wines and their chemical properties.

Our goal is to develop a k-Nearest Neighbors model that can predict if a wine is red or white based on the wine’s chemical properties.»

RAW OBSERVATIONS FROM WINE DATASET

```r
library(rio)
DataWine=import("https://ai.lange-analytics.com/data/WineData.rds")
print(DataWine)
```

```
     wineColor acidity volatileAcidity citricAcid residualSugar Chlorides
1          red   10.80           0.320       0.44          1.60     0.063
2        white    6.40           0.310       0.39          7.50     0.040
3        white    9.40           0.280       0.30          1.60     0.045
4        white    8.20           0.220       0.36          6.80     0.034
5        white    6.40           0.290       0.44          3.60     0.197
6          red    6.70           0.855       0.02          1.90     0.064
7          red   11.80           0.380       0.55          2.10     0.071
8        white    6.70           0.250       0.23          7.20     0.038
9          red    7.50           0.380       0.57          2.30     0.106
10         red    7.10           0.270       0.60          2.10     0.074
11       white    6.40           0.270       0.19          1.90     0.085
12         red    7.80           0.600       0.26          2.00     0.080
13         red    8.00           0.580       0.28          3.20     0.066
14       white    7.00           0.360       0.35          2.50     0.048
15         red    9.90           0.440       0.46          2.20     0.091
16       white    7.80           0.280       0.31          2.10     0.046
17         red    7.60           0.400       0.29          1.90     0.078
```

»

OBSERVATIONS FROM WINE DATASET FOR SELECTED VARIABLES

SULFOR DIOXIDE AND ACIDITY

Note we use `clean_names("upper_camel")` from the `janitor` package to change all column (variable) names to UpperCamel.

```
library(tidyverse); library(rio);library(janitor)
DataWine=import("https://ai.lange-analytics.com/data/WineData.rds") %>%
  clean_names("upper_camel") %>%
  select(WineColor,Sulfur=TotalSulfurDioxide,Acidity) %>%
  mutate(WineColor=as.factor(WineColor))
print(DataWine)
```

```
     WineColor Sulfur Acidity
1          red   37.0   10.80
2        white  213.0    6.40
3        white  139.0    9.40
4        white   90.0    8.20
5        white  183.0    6.40
6          red   38.0    6.70
7          red   19.0   11.80
8        white  220.0    6.70
9          red   12.0    7.50
10         red   25.0    7.10
11       white  196.0    6.40
12         red  131.0    7.80
```

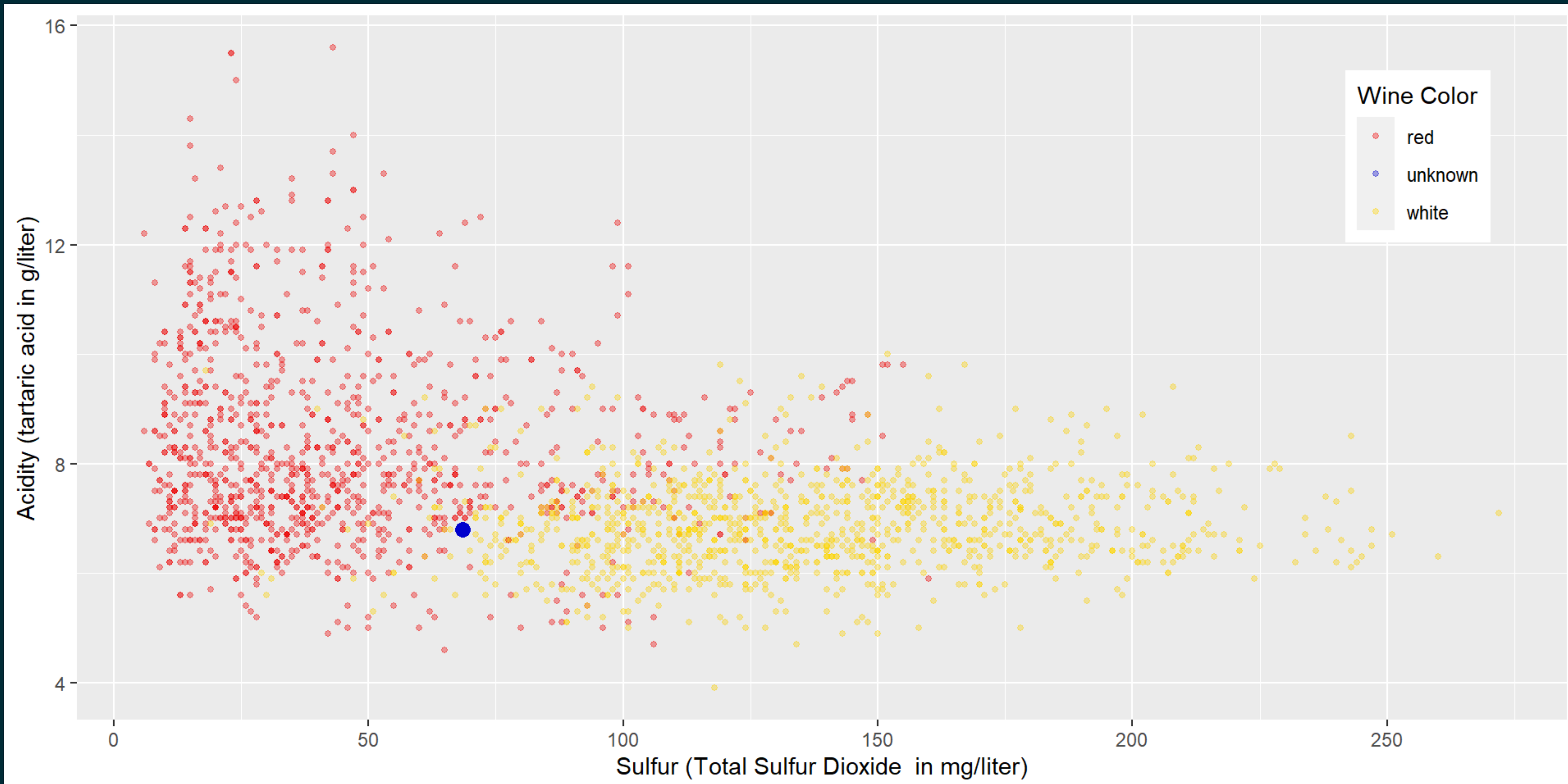
```
13           red  114.0    8.00
14         white  161.0    7.00
15           red   41.0    9.90
16         white  208.0    7.80
```

BEFORE STARTING WITH K NEAREST NEIGHBORS

LET US FIND SOME EYEBALLING TECHNIQUES THAT ARE RELATED TO VARIOUS MACHINE LEARNING MODELS»

EYE BALLING TECHNIQUES TO IDENTIFY RED AND WHITE WINES

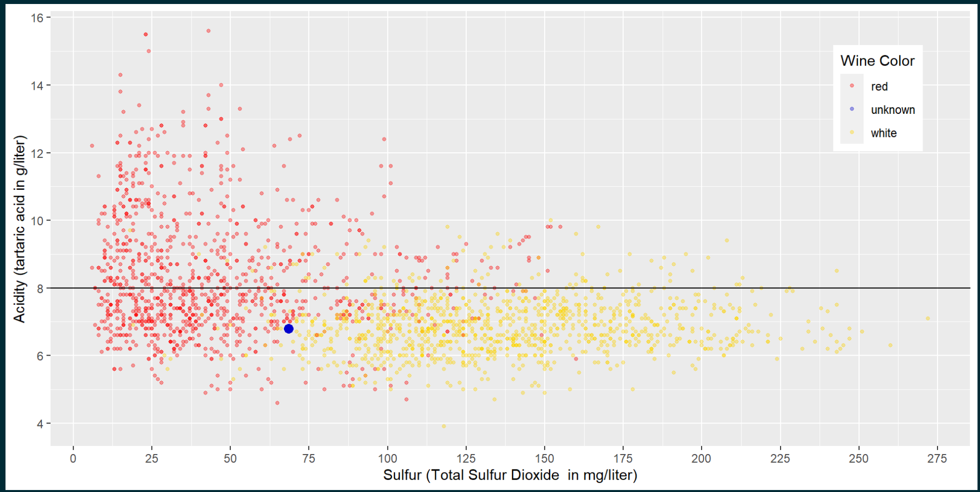
TRY EYEBALLING THE TRAINING DATA ($N = 2238$)



Acidity and Total Sulfur Dioxide Related to Wine Color

EYE BALLING TECHNIQUES TO IDENTIFY RED AND WHITE WINES

HORIZONTAL BOUNDARY



Horizontal Decision Boundary for Acidity and Total Sulfur Dioxide Related to Wine Color

CONFUSION MATRIX

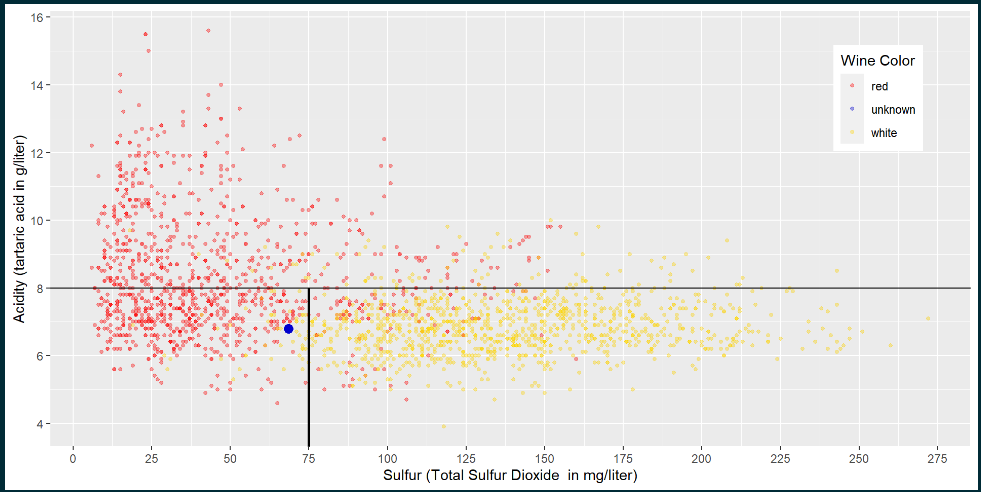
```
          Truth
Prediction  red white
     red    510    80
     white  609  1039
```

ACCURACY

https://ai.lange-analytics.com/

EYEBALLING TECHNIQUES TO IDENTIFY RED AND WHITE WINES

CREATING SUBSPACES LIKE SIMILAR TO A DECISION TREE



Sub-Space Boundaries for Acidity and Total Sulfur Dioxide Related to Wine Color

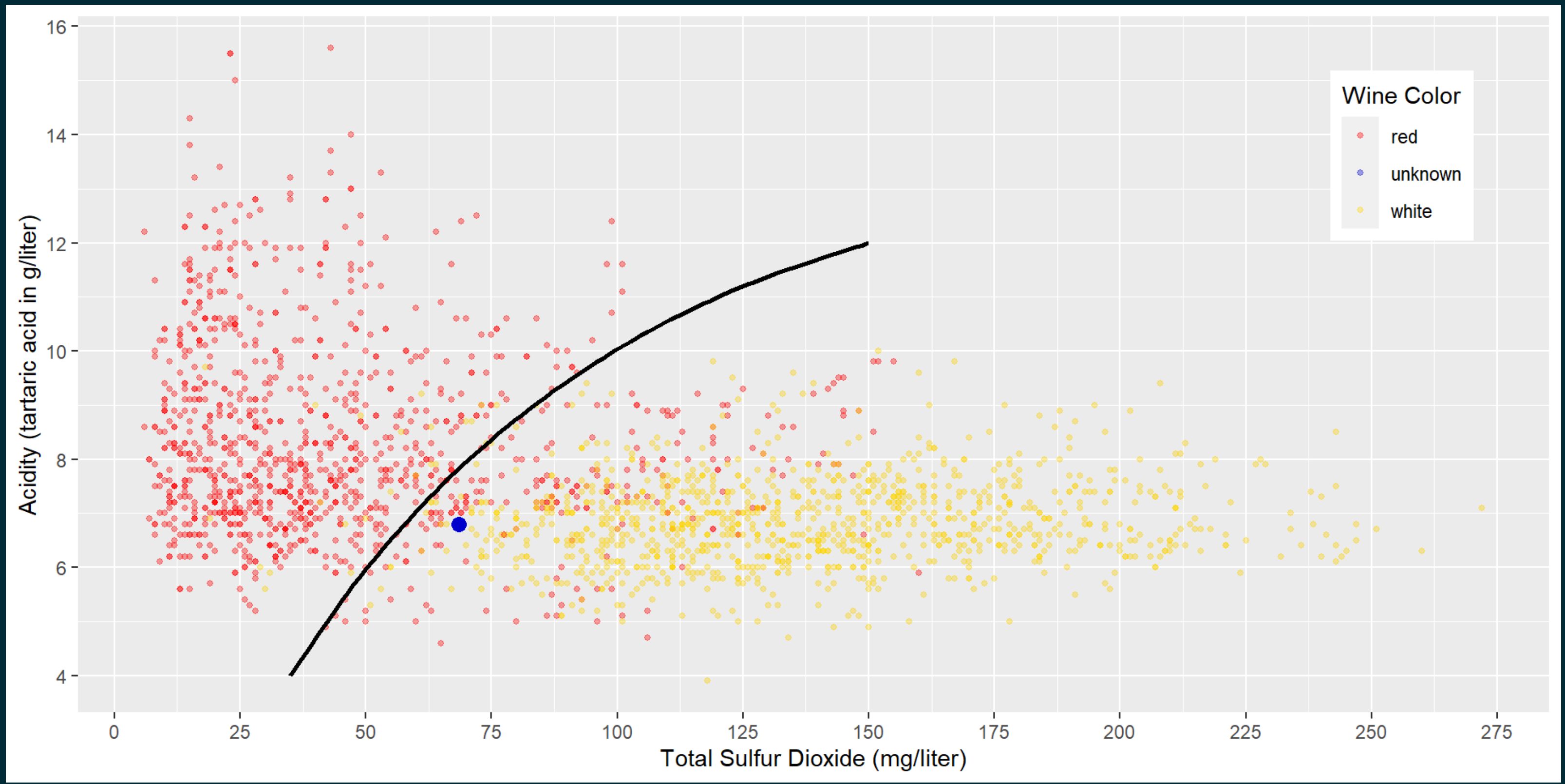
CONFUSION MATRIX

```
          Truth
Prediction  red white
     red   1016   145
     white  103   974
```

ACCURACY

EYEBALLING TECHNIQUES TO IDENTIFY RED AND WHITE WINES

USING A NON-LINEAR DECISION BOUNDARY LIKE A NEURAL NETWORK



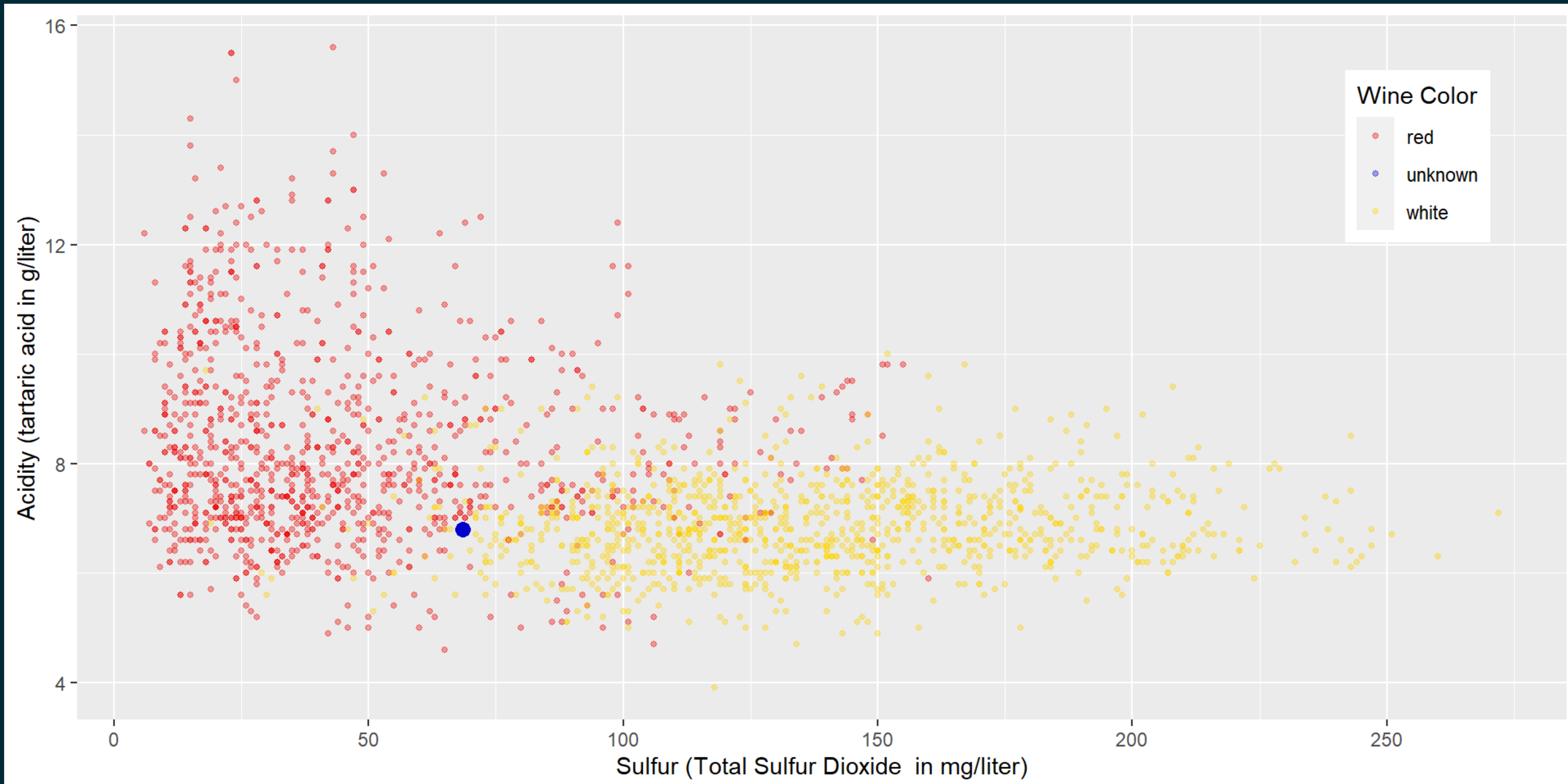
Curved Decision Boundary for Acidity and Total Sulfur Dioxide Related to Wine Color

CONFUSION MATRIX AND ACCURACY

Similar as previous one.

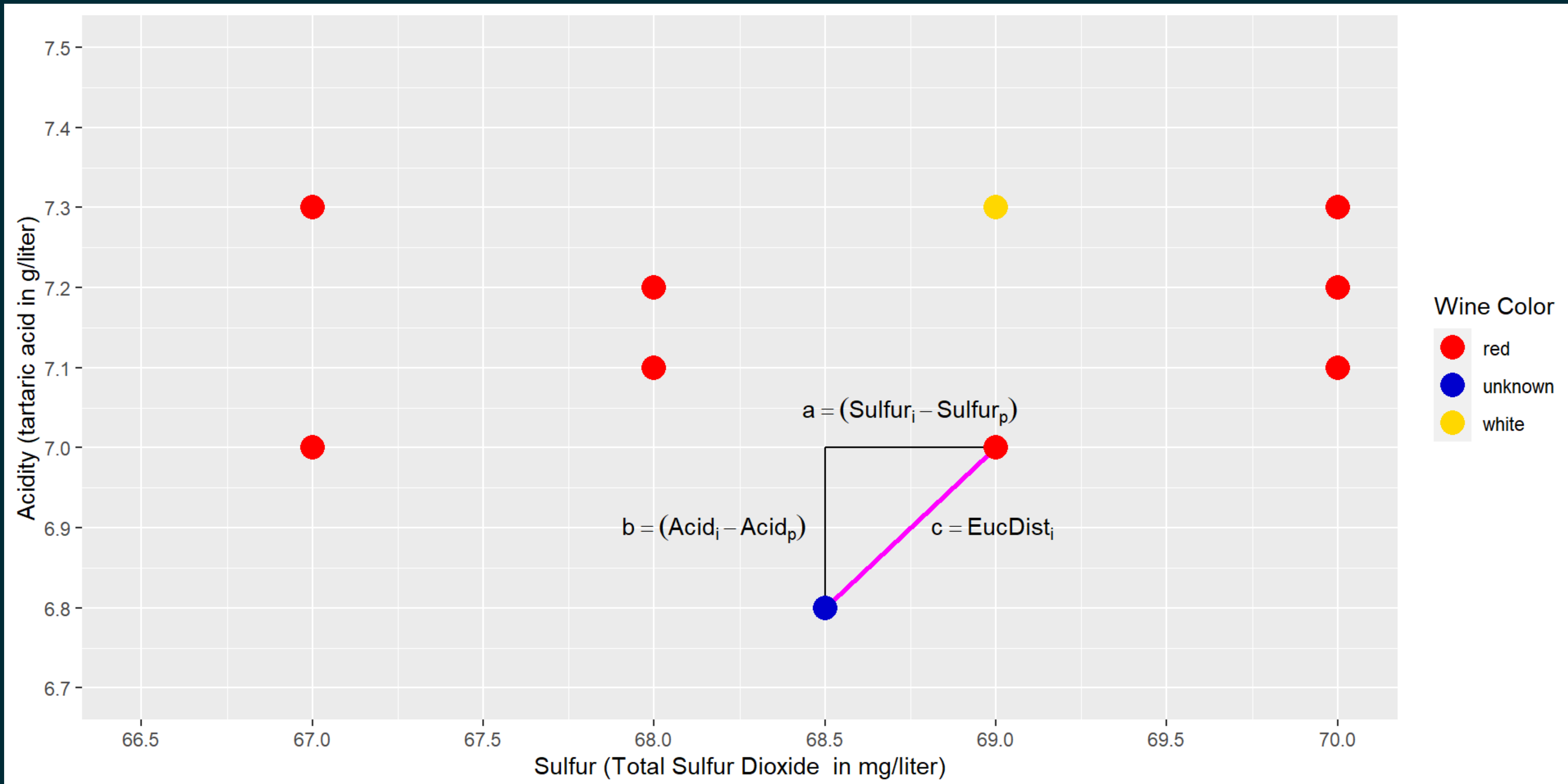
SO, HOW DOES K NEAREST NEIGHBORS WORK?

K NEAREST NEIGHBORS K=1



Acidity and Total Sulfur Dioxide Related to Wine Color»

K NEAREST NEIGHBORS K=1



Predicting Wine Color with k-Nearest Neighbors (k=1)

HOW TO CALCULATE EUCLIDEAN DISTANCE FOR TWO VARIABLES

Assume our observations have **two predictor variables** x and y. We compare the unknown point p to one of the points from the training data (e,g., point i):

$$Dist_i = \sqrt{(x_p - x_i)^2 + (y_p - y_i)^2}$$

»

HOW TO CALCULATE EUCLIDEAN DISTANCE FOR THREE VARIABLES

Assume our observations have **three predictor variables** x, y, and z. We compare the unknown point p to one of the points from the training data (e,g., point i):

$$Dist_i = \sqrt{(x_p - x_i)^2 + (y_p - y_i)^2 + (z_p - z_i)^2}$$

»

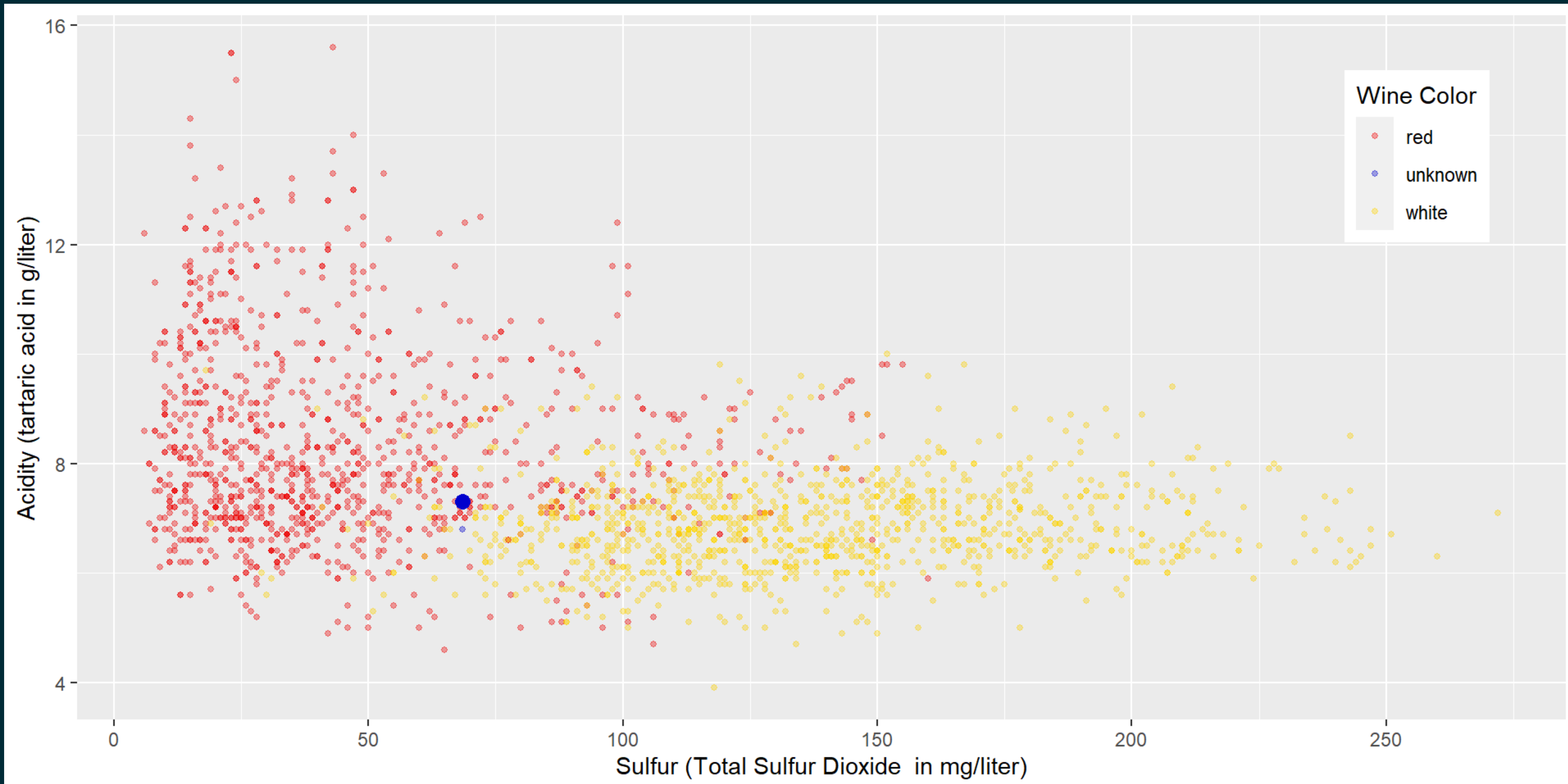
HOW TO CALCULATE EUCLIDEAN DISTANCE FOR N VARIABLES

Assume our observations have N predictor variables v_j with $j = 1...N$. We compare the unknown point p to one of the points from the training data (e,g., point i):

$$Dist_i = \sqrt{\sum_{j=1}^{N}(v_{p,j} - v_{i,j})^2}$$

»

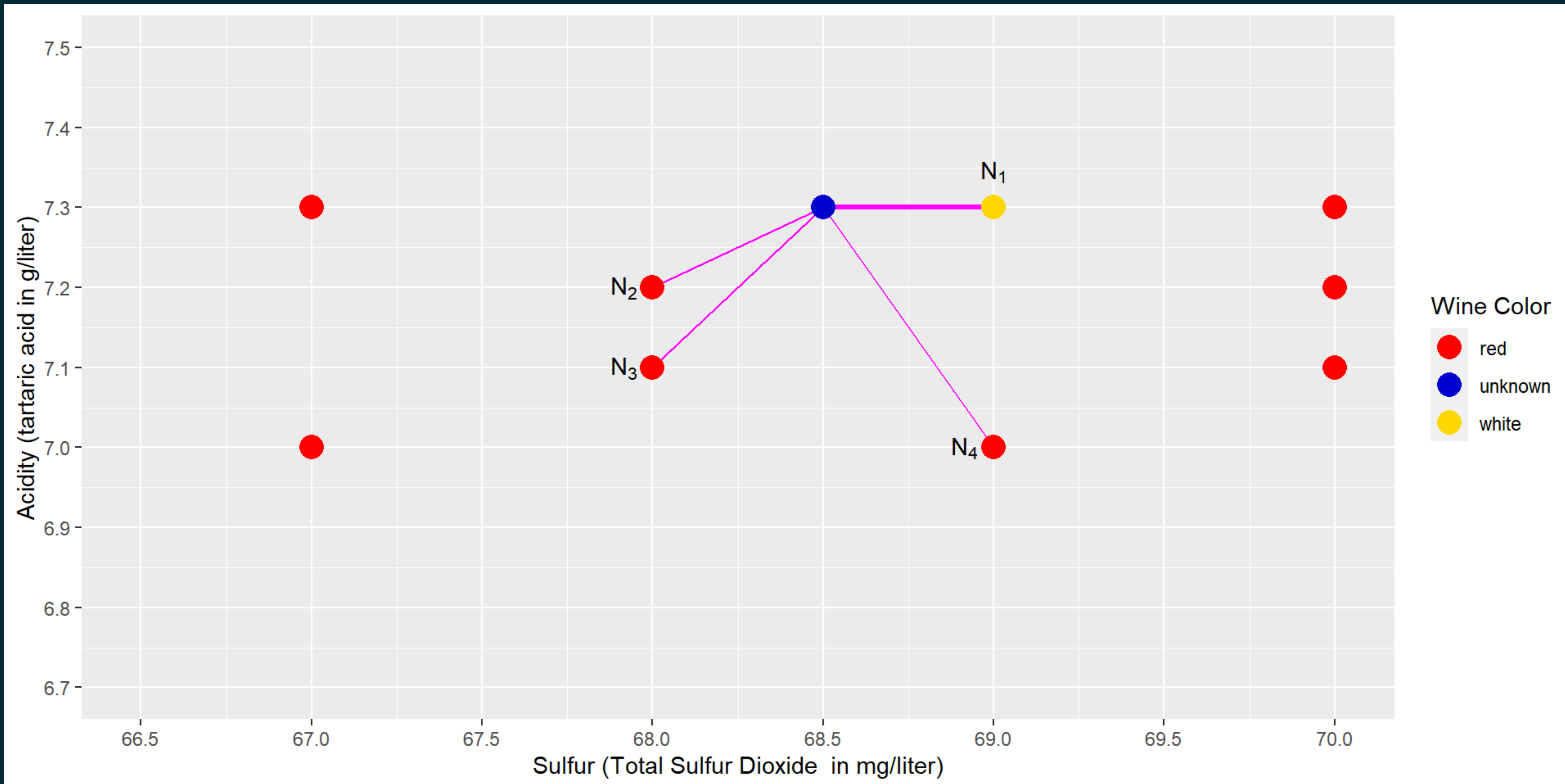
K NEAREST NEIGHBORS K=4 (FOR A DIFFERENT UNKNOWN WINE)



Acidity and Total Sulfur Dioxide Related to Wine Color

K NEAREST NEIGHBORS K=4 (FOR A DIFFERENT UNKNOWN WINE)

4 NEAREST NEIGHBORS VOTE ON "RED" VS. "WHITE"

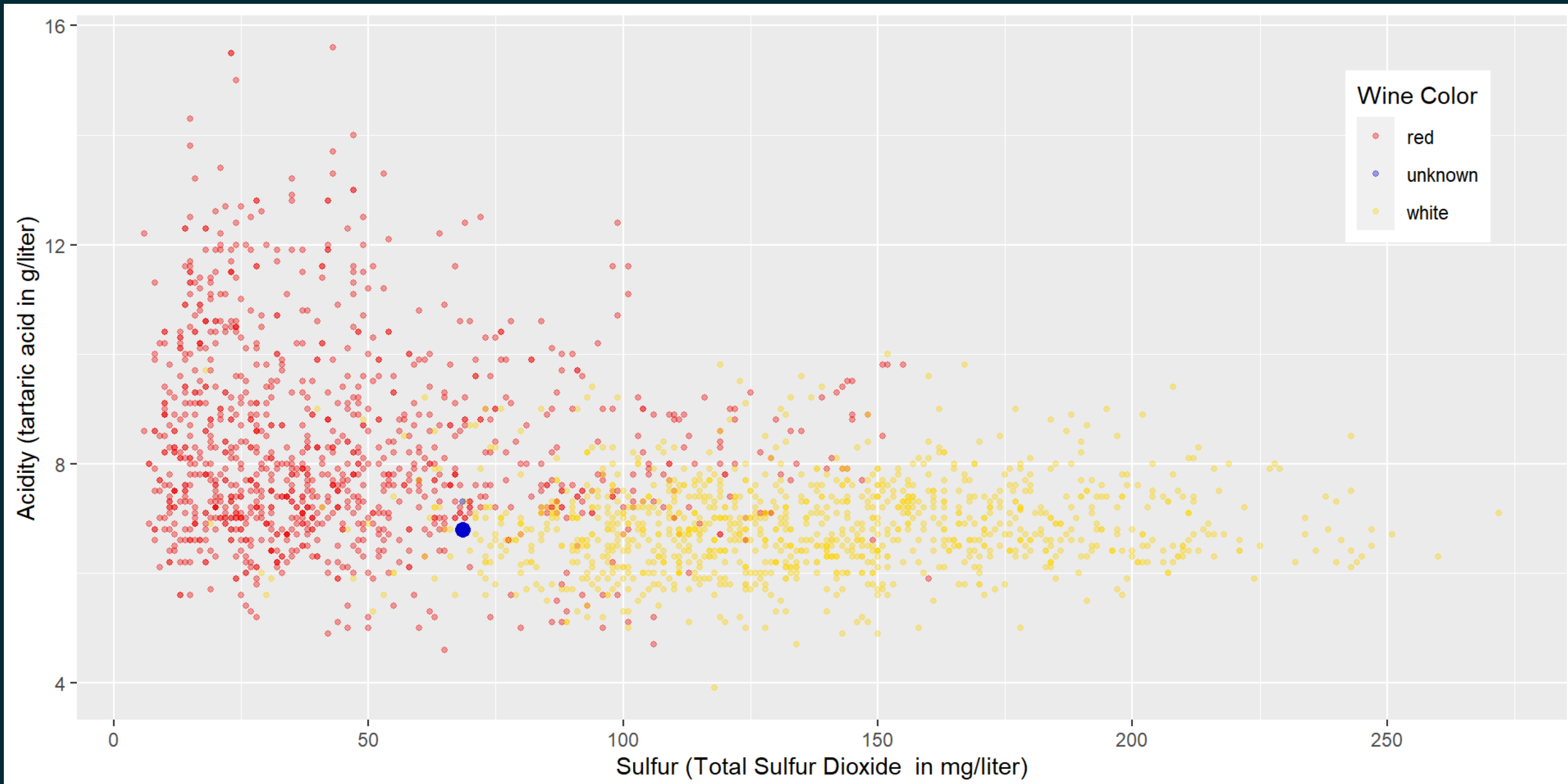


Predicting Wine Color with k-Nearest Neighbors (k=4)

https://ai.lange-analytics.com/

K NEAREST NEIGHBORS K=4 (FOR A DIFFERENT UNKNOWN WINE)

WATCH THE SCALE: G/LITER VS. MG/LITER. THAT DOES NOT LOOK RIGHT!



Acidity and Total Sulfur Dioxide Related to Wine Color »

https://ai.lange-analytics.com/

A FEW COMMON SCALING OPTIONS

- **Convert from sub-units to major units or vice versa by multiplying or deviding**

  For example, if one variable is in feet and one in inches, you can multiply feet by 12 to get inches or devide inches by 12 to get feet. This is often not possible (e.g., `Height` and `Weight`).

- **Rescaling** (*R* command `range()`)

  Generates a variable y that is scaled to a range between 0 and 1 based on the original variable's value x, its minimum x_{min} and its maximum x_{max}:

$$y = \frac{x - x_{min}}{x_{max} - x_{min}}$$

- **Z-Score Normalization**

  Z-score normalization uses the mean ($\overline{x}$) and the standard deviation (s) of a variable to scale the variable x to the variable z:

$$z = \frac{x - \overline{x}}{s}$$

»

TIME TO RUN K-NEAREST NEIGHBORS

LOADING DATA AND SELECTING VARIABLES

```r
library(tidyverse); library(rio);library(janitor)
DataWine=import("https://ai.lange-analytics.com/data/WineData.rds") %>%
          clean_names("upper_camel") %>%
          select(WineColor,Sulfur=TotalSulfurDioxide,Acidity) %>%
  mutate(WineColor=as.factor(WineColor))
print(DataWine)
```

```
     WineColor Sulfur Acidity
1          red   37.0   10.80
2        white  213.0    6.40
3        white  139.0    9.40
4        white   90.0    8.20
5        white  183.0    6.40
6          red   38.0    6.70
7          red   19.0   11.80
8        white  220.0    6.70
9          red   12.0    7.50
10         red   25.0    7.10
11       white  196.0    6.40
12         red  131.0    7.80
13         red  114.0    8.00
14       white  161.0    7.00
15         red   41.0    9.90
16       white  208.0    7.80
17         red   66.0    7.60
```

TIME TO RUN K-NEAREST NEIGHBORS

Generate Training and Testing Data (Splitting):

```r
library(tidymodels);
set.seed(876)
Split7030=initial_split(DataWine,prop=0.7,strata = WineColor)

DataTrain=training(Split7030)
DataTest=testing(Split7030)
print(DataTrain)
```

```
     WineColor Sulfur Acidity
1          red   37.0   10.80
2          red   38.0    6.70
3          red   12.0    7.50
4          red   25.0    7.10
5          red  114.0    8.00
6          red   66.0    7.60
7          red   49.0    6.80
8          red  110.0    7.00
9          red   44.0    6.50
10         red   10.0   10.40
11         red  149.0    6.60
12         red   31.0    6.60
13         red   49.0    6.60
14         red   17.0    9.10
15         red   32.0    6.10
16         red   30.0    8.50
17         red   17.0   10.20
```

```python
print(DataTest)
```

```
   WineColor Sulfur Acidity
1      white   90.0    8.20
2        red   19.0   11.80
3      white  220.0    6.70
4        red  131.0    7.80
5      white  161.0    7.00
6        red   41.0    9.90
7      white  156.0    7.80
8      white  150.0    6.50
9        red  102.0    7.90
10       red   11.0    5.80
11     white   64.0    5.90
12     white  178.0    6.90
13       red   41.0   12.60
14       red   19.0   12.70
15     white  206.0    6.20
16     white  149.0    6.90
17     white  182.0    7.80
```

TIME TO RUN K-NEAREST NEIGHBORS

CLICK HERE TO FIND A REFERENCE LIST FOR VARIOUS Step_ COMMANDS

Recipe: Prepare Data for Analysis:

```
RecipeWine=recipe(WineColor~Acidity+Sulfur, data = DataTrain) %>%
              step_naomit() %>%
              step_normalize(all_predictors())
# or

RecipeWine=recipe(WineColor~., data = DataTrain) %>%
              step_naomit() %>%
              step_normalize(all_predictors())
print(RecipeWine)
```

TIME TO RUN K-NEAREST NEIGHBORS

CLICK HERE TO FIND A REFERENCE LIST FOR VARIOUS `Step_` COMMANDS

Creating a Model Design:

```
ModelDesignKNN=nearest_neighbor(neighbors = 4, weight_func = "rectangular") %>%
                set_engine("kknn") %>%
                set_mode("classification")
print(ModelDesignKNN)
```

```
K-Nearest Neighbor Model Specification (classification)

Main Arguments:
  neighbors = 4
  weight_func = rectangular

Computational engine: kknn
```

TIME TO RUN K-NEAREST NEIGHBORS

Putting it all together in a **fitted** `workflow`:

```
WFModelWine=workflow() %>%
              add_recipe(RecipeWine) %>%
              add_model(ModelDesignKNN) %>%
              fit(DataTrain)
print(WFModelWine)
```

```
══ Workflow [trained] ══════════════════════════════════════════════════════
Preprocessor: Recipe
Model: nearest_neighbor()

── Preprocessor ────────────────────────────────────────────────────────────
2 Recipe Steps

• step_naomit()
• step_normalize()

── Model ───────────────────────────────────────────────────────────────────

Call:
kknn::train.kknn(formula = ..y ~ ., data = data, ks = min_rows(4,     data, 5), kernel =
~"rectangular")

Type of response variable: nominal
Minimal misclassification: 0.1000894
```

TIME TO RUN K-NEAREST NEIGHBORS

How to use the **fitted** `workflow` to predict the wine color for the wines in the testing dataset:

1. Start with observation $i = 1$ from `DataTest` (the first observation).
2. Take observation i from `DataTest` and use `Acidity` and `Sulfur` to calculate the Euclidean distance to **each** of the observations of `DataTrain`.
3. Isolate the 4 observations with the smallest Euclidean distance and use the majority of their wine color as a prediction for observation i from `DataTest` (in case of a par, decide randomly).
4. Increase i by one (i.e., take the next observation from `DataTest`) and go to step 2 (until all `DataTest` observations are processed).

TIME TO RUN K-NEAREST NEIGHBORS

Predicting with the fitted workflow using `predict()` (not exactly helpful!):

```
predict(WFModelWine, DataTest)
```

```
# A tibble: 960 × 1
   .pred_class
   <fct>
 1 white
 2 red
 3 white
 4 white
 5 white
 6 red
 7 white
 8 white
 9 red
10 red
# ℹ 950 more rows
```

TIME TO RUN K-NEAREST NEIGHBORS

Predicting with the fitted workflow using `augment()` which *augments* `DataTest` with the predictions:

```
DataPredWithTestData=augment(WFModelWine, DataTest)
head(DataPredWithTestData)
```

```
# A tibble: 6 × 6
  WineColor Sulfur Acidity .pred_class .pred_red .pred_white
  <fct>      <dbl>   <dbl> <fct>           <dbl>       <dbl>
1 white         90     8.2 white            0.25        0.75
2 red           19    11.8 red              1           0
3 white        220     6.7 white            0           1
4 red          131     7.8 white            0.25        0.75
5 white        161     7   white            0           1
6 red           41     9.9 red              1           0
```

HAVING A DATA FRAME WITH `truth` AND `esimate` WE CAN CALCULATE PERFORMANCE METRICS

Confusion Matrix:

```
ConfMatrixWine=conf_mat(DataPredWithTestData, truth = WineColor, estimate = .pred_clas
print(ConfMatrixWine)
```

```
          Truth
Prediction red white
     red   436    46
     white  44   434
```

READING THE CONFUSION MATRIX

```
            Truth
Prediction   Red Wine White Wine
  Red Wine   TP: 436  FP: 46
  White Wine FN: 44   TN: 434
```

- The **positive class** (wine is “red”) is in the **first column**. 436 of the positives are classified correctly (TR: true positives), and 44 positives are incorrectly classified (FN: false negatives).
- The **negative class** (wine is “white”) is in the **second column**. 44 negatives are incorrectly classified (FP: false positives), and 434 negatives are classified correctly (TN: true negatives).

Accuracy: Number of wines on diagonal/number of all wines:

```
1  accuracy(DataPredWithTestData, truth = WineColor, estimate = .pred_class)
```

```
# A tibble: 1 × 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary         0.906
```

»

WARNING: BE CAREFUL WITH THE ACCURACY RATE

THE STORY OF DR. NEBULOUS'S GAMBLERS SYSTEM

Dr. Nebulous offers a **97% Machine Learning Gambling Prediction**. Here is how it works: Gamblers can buy a prediction for a fee of $5. Dr. Nebulous will then run his famous machine learning model and send a closed envelope with the prediction. The gambler is supposed to open the envelope in the casino, right before placing a bet of $100 on a number in roulette. The envelope contains a message that states either "You will win" or "You will lose", which allows the gambler to act accordingly by either bet or not bet.

Dr. Nebulous claims that a "clinical trial" of 1000 volunteers, who opened the envelope after they had bet on a number in roulette, shows an accuracy of 97.3%.

How could Dr. Nebulous have such a precise model?

WARNING: BE CAREFUL WITH THETHE ACCURACY RATE

THE STORY OF DR. NEBULOUS'S GAMBLERS SYSTEM

The trick is Dr. Nebulous's machine learning model uses the *naive prognosis*: It always predicts "You will lose".

Here is the confusion matrix from the 1,000 volunteers trial:

```
          Truth
Prediction Win Lose
      Win    0    0
      Lose  27  997
```

Roulette has 37 numbers to bet on. Chance to win is: $\frac{1}{37} = 0.027$.

Out of the 1000 volunteers, 27 are expected to win, and 973 are expected to lose.

$$Accuracy = \frac{0 + 973}{1000} = 0.973$$

WARNING: BE CAREFUL WITH THE ACCURACY RATE

THE STORY OF DR. NEBULOUS'S GAMBLERS SYSTEM

```
          Truth
Prediction Win Lose
      Win    0    0
      Lose  27  997
```

However, when we look at the correct positive and the correct negative rate separately, we see that Dr. Nebulous' accuracy rate (although correct) makes little sense.

- The correct negative rate (**specificity**) is 100%
- The correct positive rate (**sensitivity**) is zero (out of the 27 winners, all were falsely predicted as "You will lose").

This example shows: When interpreting the confusion matrix, you must look at accuracy, sensitivity, and specificity simultaneously

TIME TO RUN K-NEAREST NEIGHBORS 🤓

`accuracy()`, `sensitivity()` and `specificity()` for the wine data:

```
accuracy(DataPredWithTestData, truth = WineColor, estimate = .pred_class)
```

```
# A tibble: 1 × 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary         0.906
```

```
sensitivity(DataPredWithTestData, truth = WineColor, estimate = .pred_class)
```

```
# A tibble: 1 × 3
  .metric     .estimator .estimate
  <chr>       <chr>          <dbl>
1 sensitivity binary         0.908
```

```
specificity(DataPredWithTestData, truth = WineColor, estimate = .pred_class)
```

```
# A tibble: 1 × 3
  .metric     .estimator .estimate
  <chr>       <chr>          <dbl>
1 specificity binary         0.904
```

Can we improve by using all predictors.»

PROJECT: DESIGN A MACHINE LEARNING WORKFLOW FOR OPTICAL CHARACTER RECOGNITION »

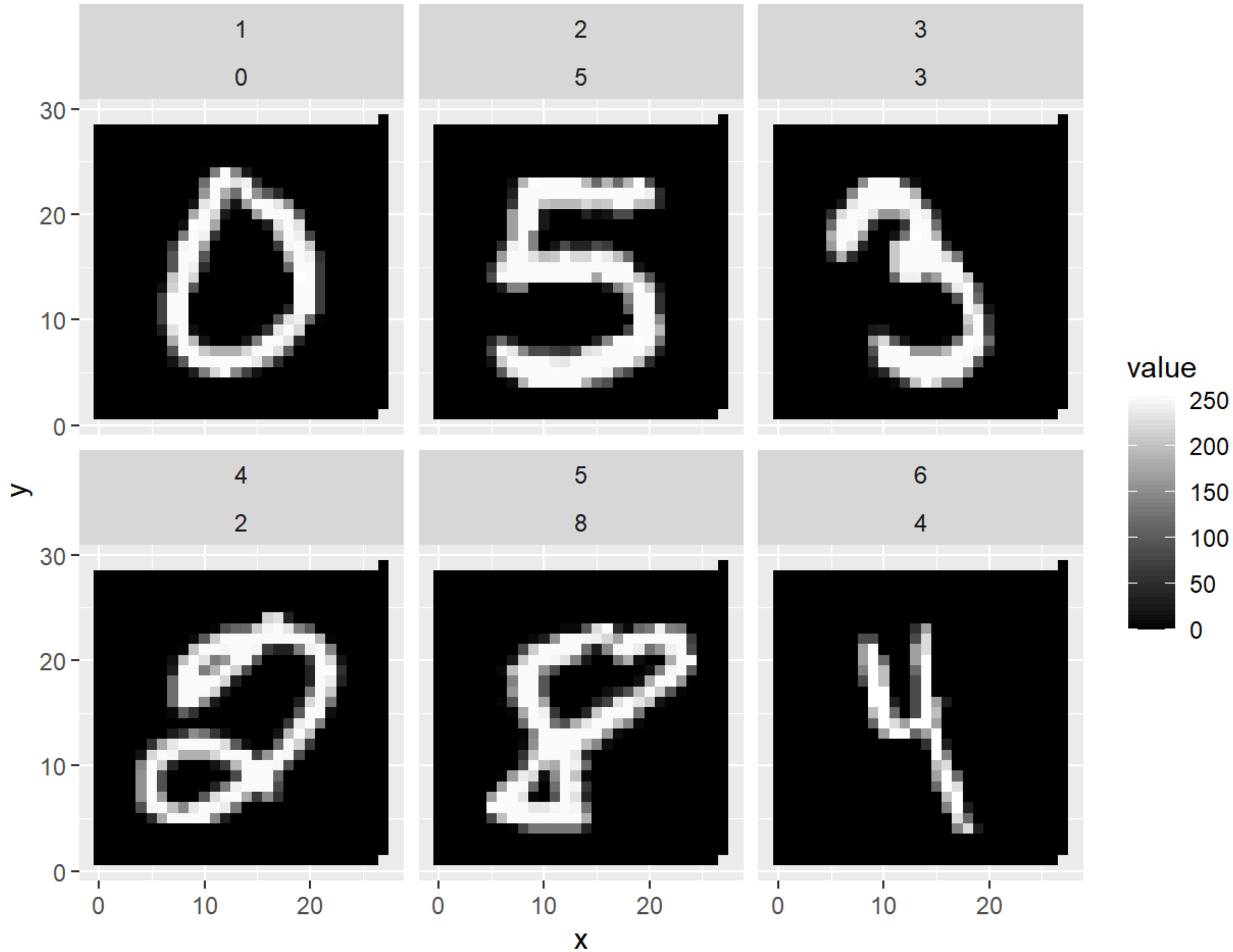
MNIST DATA SET

You will develop a machine learning model based on k-Nearest Neighbors to recognize handwritten digits from images.

You will use the MNIST dataset, a standard dataset for image recognition in machine learning (60,000 images for training and 10,000 images for testing). Developed by LeCun, Cortes, and Burges (2010) based on two datasets from handwritten digits obtained from Census workers and high school students.

We will use only the first 500 images of the original MNIST dataset to speed up the *k-Nearest Neighbors* model's training time.

VISUALIZATION OF THE FIRST SIX IMAGES FROM THE MNIST DATA SET



HOW A IMAGE IS STORED IN THE MNIST DATASET

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	49	138	243	255	154	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	9	197	252	252	253	236	161	50	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	130	252	227	130	190	252	252	227	48	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	155	236	227	50	0	5	83	252	252	137	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	53	253	252	130	0	0	0	26	221	252	242	158	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	127	255	228	32	0	0	0	0	185	253	253	231	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	230	253	132	0	0	0	0	0	111	252	252	230	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	230	253	92	0	0	0	0	0	153	252	252	199	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	230	253	92	0	0	0	0	17	209	252	252	74	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	126	253	196	0	0	0	0	99	252	252	252	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	255	253	222	97	24	87	212	253	253	190	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	159	252	252	252	252	253	252	252	252	137	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	7	141	183	183	246	215	196	252	252	137	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	42	21	47	252	252	137	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	47	252	252	137	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	47	253	253	137	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	47	252	252	137	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	47	252	252	137	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	47	252	252	137	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	137	252	85	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Image of a Handwritten Nine

The image has 28 rows and 28 columns. Each of the 784 cells (pixels) holds a value between 0 (black) and 255 (white)

https://ai.lange-analytics.com/

HOW A IMAGE IS STORED IN THE MNIST DATASET

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	49	138	243	255	154	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	9	197	252	252	253	236	161	50	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	130	252	227	130	190	252	252	227	48	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	155	236	227	50	0	5	83	252	252	137	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	53	253	252	130	0	0	0	26	221	252	242	158	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	127	255	228	32	0	0	0	0	185	253	253	231	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	230	253	132	0	0	0	0	0	111	252	252	230	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	230	253	92	0	0	0	0	0	153	252	252	199	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	230	253	92	0	0	0	0	17	209	252	252	74	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	126	253	196	0	0	0	0	99	252	252	252	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	255	253	222	97	24	87	212	253	253	190	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	159	252	252	252	252	253	252	252	252	137	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	7	141	183	183	246	215	196	252	252	137	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	42	21	47	252	252	137	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	47	252	252	137	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	47	253	253	137	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	47	252	252	137	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	47	252	252	137	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	47	252	252	137	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	137	252	85	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Image of a Handwritten Nine

- Pixel values for a single image are not stored in a table. Ohterwise we would end-up with a table containing tables.
- Pixel values are stored as one row for each image.
- Concatenating the 28 rows of an image into one row with 28*28=784 cells (pixels)

THREE ROWS FROM THE DATA FRAME OF THE MNIST DATASET

```
1  print(Mnist4PlotAndTable[1:3,1:784])
```

```
  Label Pix1 Pix2 Pix3 Pix4 Pix5 Pix6 Pix7 Pix8 Pix9 Pix10 Pix11 Pix12 Pix13
1     0    0    0    0    0    0    0    0    0    0     0     0     0     0
2     5    0    0    0    0    0    0    0    0    0     0     0     0     0
3     3    0    0    0    0    0    0    0    0    0     0     0     0     0
  Pix14 Pix15 Pix16 Pix17 Pix18 Pix19 Pix20 Pix21 Pix22 Pix23 Pix24 Pix25 Pix26
1     0     0     0     0     0     0     0     0     0     0     0     0     0
2     0     0     0     0     0     0     0     0     0     0     0     0     0
3     0     0     0     0     0     0     0     0     0     0     0     0     0
  Pix27 Pix28 Pix29 Pix30 Pix31 Pix32 Pix33 Pix34 Pix35 Pix36 Pix37 Pix38 Pix39
1     0     0     0     0     0     0     0     0     0     0     0     0     0
2     0     0     0     0     0     0     0     0     0     0     0     0     0
3     0     0     0     0     0     0     0     0     0     0     0     0     0
  Pix40 Pix41 Pix42 Pix43 Pix44 Pix45 Pix46 Pix47 Pix48 Pix49 Pix50 Pix51 Pix52
1     0     0     0     0     0     0     0     0     0     0     0     0     0
2     0     0     0     0     0     0     0     0     0     0     0     0     0
3     0     0     0     0     0     0     0     0     0     0     0     0     0
  Pix53 Pix54 Pix55 Pix56 Pix57 Pix58 Pix59 Pix60 Pix61 Pix62 Pix63 Pix64 Pix65
1     0     0     0     0     0     0     0     0     0     0     0     0     0
2     0     0     0     0     0     0     0     0     0     0     0     0     0
3     0     0     0     0     0     0     0     0     0     0     0     0     0
  Pix66 Pix67 Pix68 Pix69 Pix70 Pix71 Pix72 Pix73 Pix74 Pix75 Pix76 Pix77 Pix78
1     0     0     0     0     0     0     0     0     0     0     0     0     0
2     0     0     0     0     0     0     0     0     0     0     0     0     0
3     0     0     0     0     0     0     0     0     0     0     0     0     0
  Pix79 Pix80 Pix81 Pix82 Pix83 Pix84 Pix85 Pix86 Pix87 Pix88 Pix89 Pix90 Pix91
```

GO TO PROJECT IN BOOK 🤓

BUILD YOUR OWN OCR SYSTEM.»