K NEAREST NEIGHBORS

This PDF version is not interactive. For interactivity, use the html version instead

Carsten Lange
clange@cpp.edu
Cal Poly, Pomona

Textbook

BEFORE WE BEGIN LET US DO A THOUGHT EXPERIMENT

I need three volunteers. Everybody else, please follow along on one computer in groups of three.

Imagine the three of you are moving into a shared apartment.

There are two bedrooms: two people will share a big room, and one person will have a small room alone.

To minimize conflicts, we want the two most similar people to share the two rooms.

Please rate yourselves on a **scale from 1 to 10** for the two criteria below:

1. How tidy you are (**1=messy, 10= tidy**)
2. How social you are at home (**1=quiet, 10=very social**)

We will treat your answers as data points and use Euclidean distance to decide who should share.

Textbook

THOUGHT EXPERIMENT: DATA ENTRY

For `Student1`, `Student2`, and `Student3`, let us enter the names of the three volunteer student. Then we replace the sample scores (e.g., `Tidy = c(1, 8, 3)`) with the scores from the volunteer students (in the correct order of `Student1`, `Student2`, and `Student3`).

Afterwards, click Run Code (note, you need to run this code in order to work with the following examples).

R Code Start Over Run Code

```r
library(kableExtra)
DataRoomMates <- data.frame(
  name = c("Student 1", "Student 2", "Student 3"),
  Tidy = c(1, 8, 3),   # 1 = messy, 10 = very tidy
  Social = c(7, 2,5 )     # 1 = quiet, 10 = very social
)
kable(DataRoomMates)
```

THOUGHT EXPERIMENT: PLOTTING THE DATA

R Code   Start Over   Run Code

```r
library(ggplot2)
ggplot(DataRoomMates, aes(x = Tidy, y = Social, label = name)) +
  geom_point(size = 4) +
  geom_text(vjust = -1) +
  scale_x_continuous(limits = c(0.5, 10.5), breaks=seq(1:10)) +
  scale_y_continuous(limits = c(0.5, 10.5), breaks=seq(1:10)) +
  coord_fixed(ratio = 1) +   # <-- makes the plot square
  labs(
    title = "Roommate Matching Using Euclidean Distance",
    x = "Tidy (1 = messy, 10 = tidy)",
    y = "Social at Home (1 = quiet, 10 = social)"
  )
```

$$EucDist^2 = DiffTidy^2 + DiffSocial^2$$

$$EucDist = sqrt(DiffTidy^2 + DiffSocial^2)$$

Textbook

COMPUTE EUCLIDEAN DISTANCES

R Code Start Over Run Code

```r
MatrixDistance <- dist(DataRoomMates[, c("Tidy", "Social")], method = "euclidean")
print(as.matrix(MatrixDistance))
```

Testing the distance between the first and the second student:

$$EucDist^2 = DiffTidy^2 + DiffSocial^2$$

$$EucDist = sqrt(DiffTidy^2 + DiffSocial^2)$$

R Code Start Over Run Code

```r
DiffTidy=DataRoomMates[[1,2]]-DataRoomMates[[2,2]]
cat("The difference in Tidy between the first and the second student is:",DiffTidy)
DiffSocial=DataRoomMates[[1,3]]-DataRoomMates[[2,3]]
cat("The difference in Social between the first and the second student is:",DiffSocial)
EucDist=sqrt((DiffTidy^2+DiffSocial^2))
cat("Euclidean Distance=",EucDist)
```

Textbook

USE THE SAME SCALE FOR ALL PREDICTOR VARIABLES

IT IS VERY IMPORTANT THAT ALL PREDICTOR VARIABLES HAVE THE SAME SCALE

Like here, both variables are in a range from 1 to 10.

WHAT HAPPENS IF WE SCALE `Social` FROM 10 TO 100 INSTEAD FROM 1 TO 10

R Code Start Over Run Code

```r
DataRoomMatesScaled=DataRoomMates |>
                    mutate(Social=Social*10)

kable(DataRoomMatesScaled)
```

Textbook

THE INFLUENCE OF SOCIAL COMPARED TO TIDY BECOMES VERY SMALL

R Code Start Over Run Code

```r
library(ggplot2)
ggplot(DataRoomMatesScaled, aes(x = Tidy, y = Social, label = name)) +
  geom_point(size = 4) +
  geom_text(vjust = -1) +
  scale_x_continuous(limits = c(1, 10), breaks=seq(1,10,9)) +
  scale_y_continuous(limits = c(10, 100), breaks=seq(10,100,10)) +
  coord_fixed(ratio = 1) +   # <-- makes the plot square
  labs(
    title = "Roommate Matching Using Euclidean Distance",
    x = "Tidy (1 = messy, 10 = tidy)",
    y = "Social at Home (1 = quiet, 10 = social)"
  )
```

The Euclidean Distance is almost the same as `DistSocial`, which makes `DistTidy` almost irrelevant.

OVERVIEW

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

Textbook

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.

1. Cortez, Paulo, António Cerdeira, Fernando Almeida, Telmo Matos, and José Reis. 2009. “Modeling Wine Preferences by Data Mining from Physicochemical Properties.” Decision Support Systems 47 (4): 547–53. https://doi.org/10.1016/j.dss.2009.05.016.

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
18     white    5.90           0.260       0.24          2.40     0.046
19     white    6.80           0.180       0.28          9.80     0.039
20     white    6.80           0.250       0.30         11.80     0.043
21     white    6.90           0.190       0.35          1.70     0.036
22     white    6.50           0.310       0.14          7.50     0.044
```

Textbook

OBSERVATIONS FROM WINE DATASET FOR SELECTED VARIABLES

SULFUR DIOXIDE AND ACIDITY

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

```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
18       white  132.0    5.90
19       white  113.0    6.80
20       white  133.0    6.80
21       white  101.0    6.90
```

Textbook

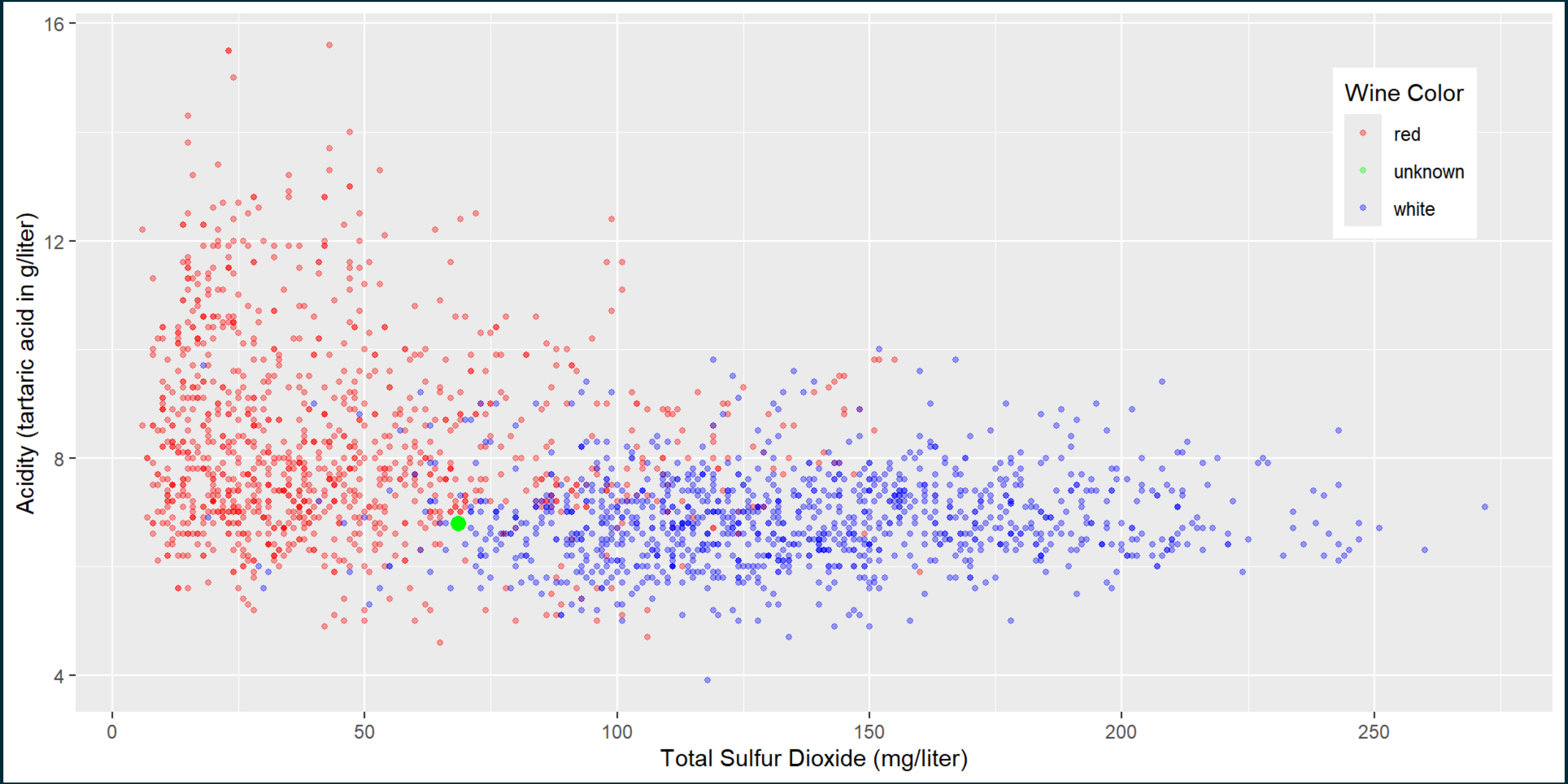
CONCEPTUAL DETOUR: BEFORE STARTING WITH K NEAREST NEIGHBORS

INTRODUCING A FEW OTHER MACHINE LEARNING MODELS WITH DIAGRAMS

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 DATA

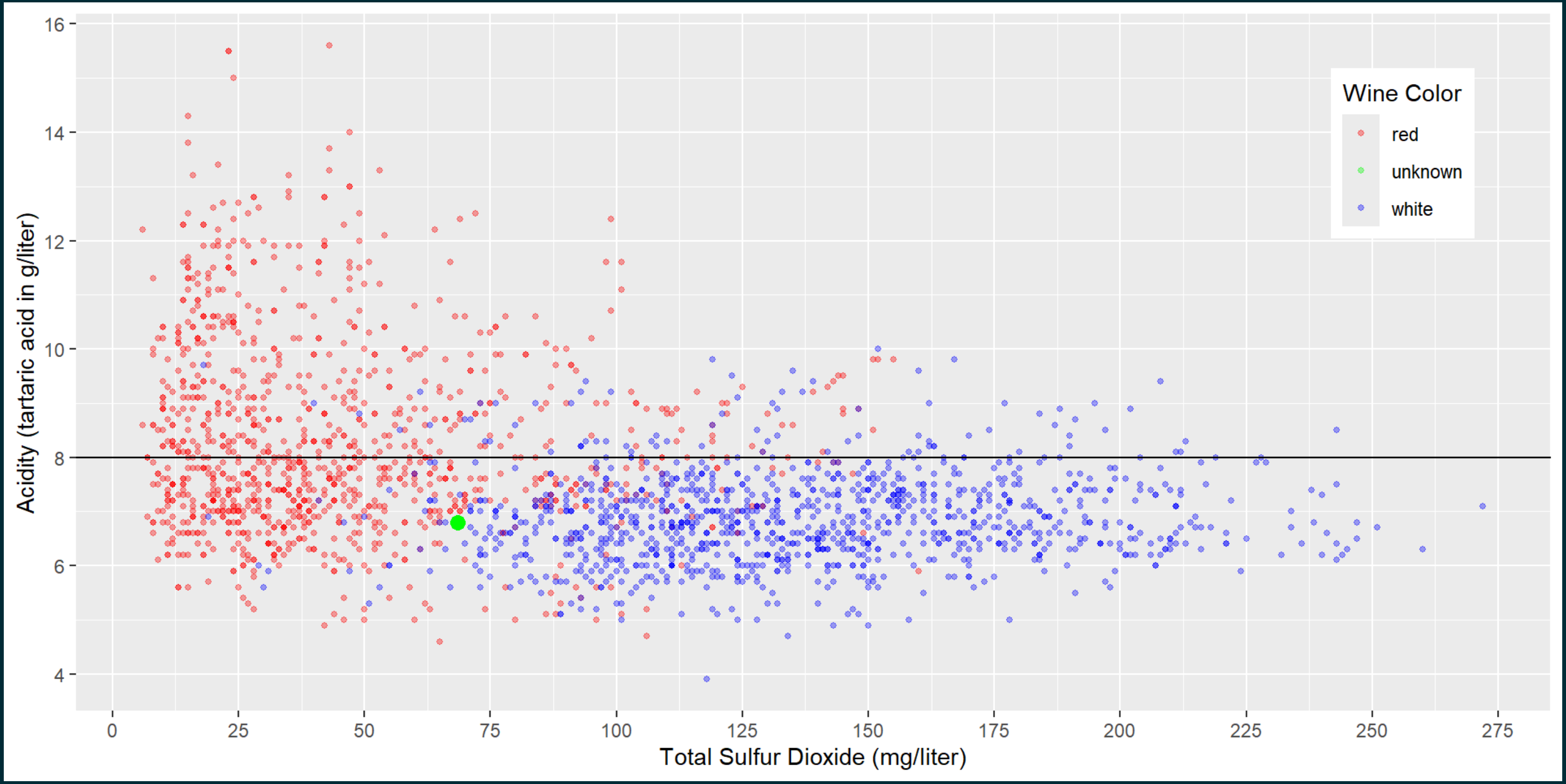


Acidity and Total Sulfur Dioxide Related to Wine Color

Textbook

EYE BALLING TECHNIQUES TO IDENTIFY RED AND WHITE WINES

HORIZONTAL BOUNDARY



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

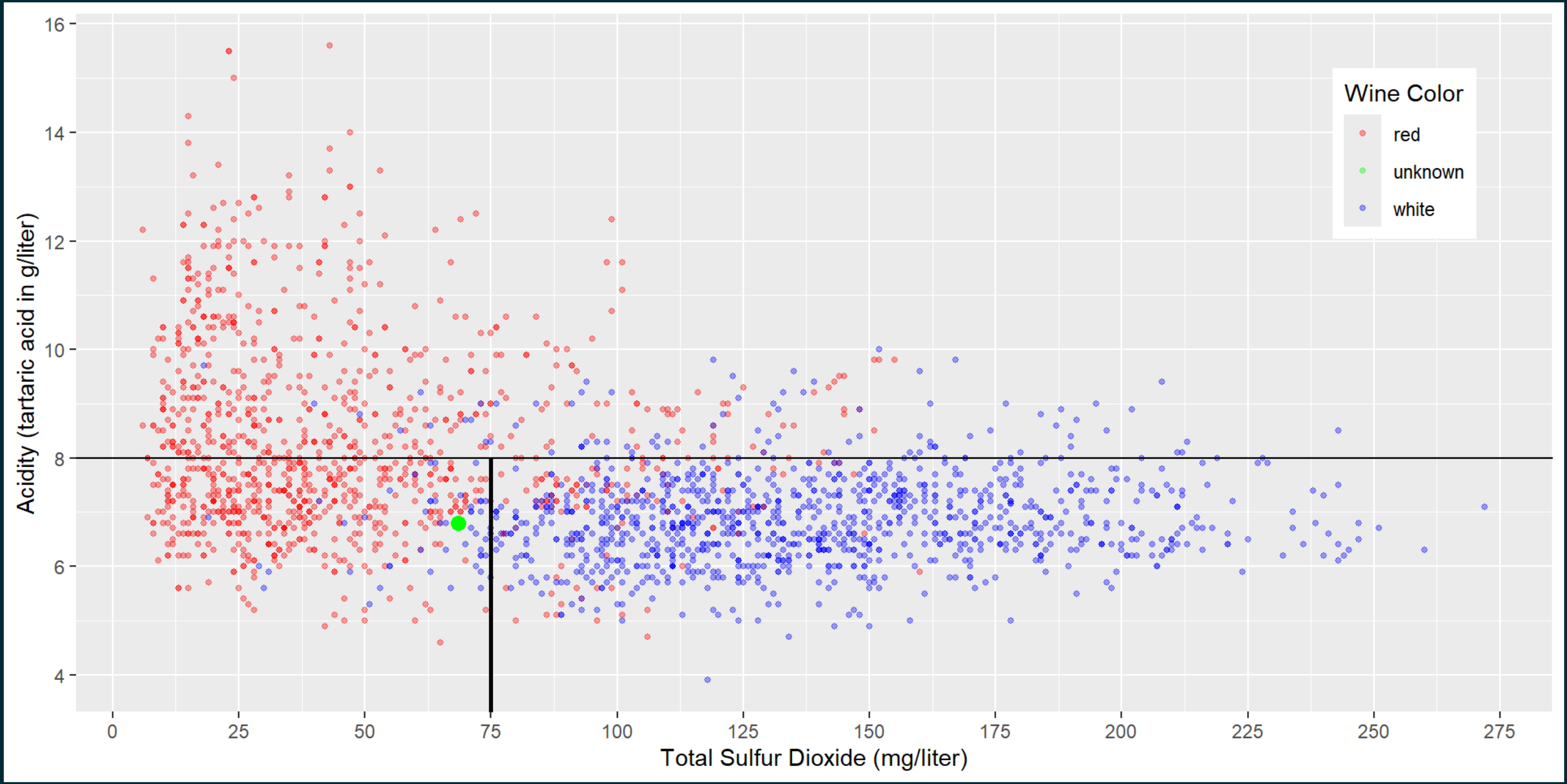
CONFUSION MATRIX

```
            Truth
Prediction   Red Wine   White Wine
  Red Wine   TP: 'half' FP: 'few'
  White Wine FN: 'half' TN: 'most'
```

Textbook

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

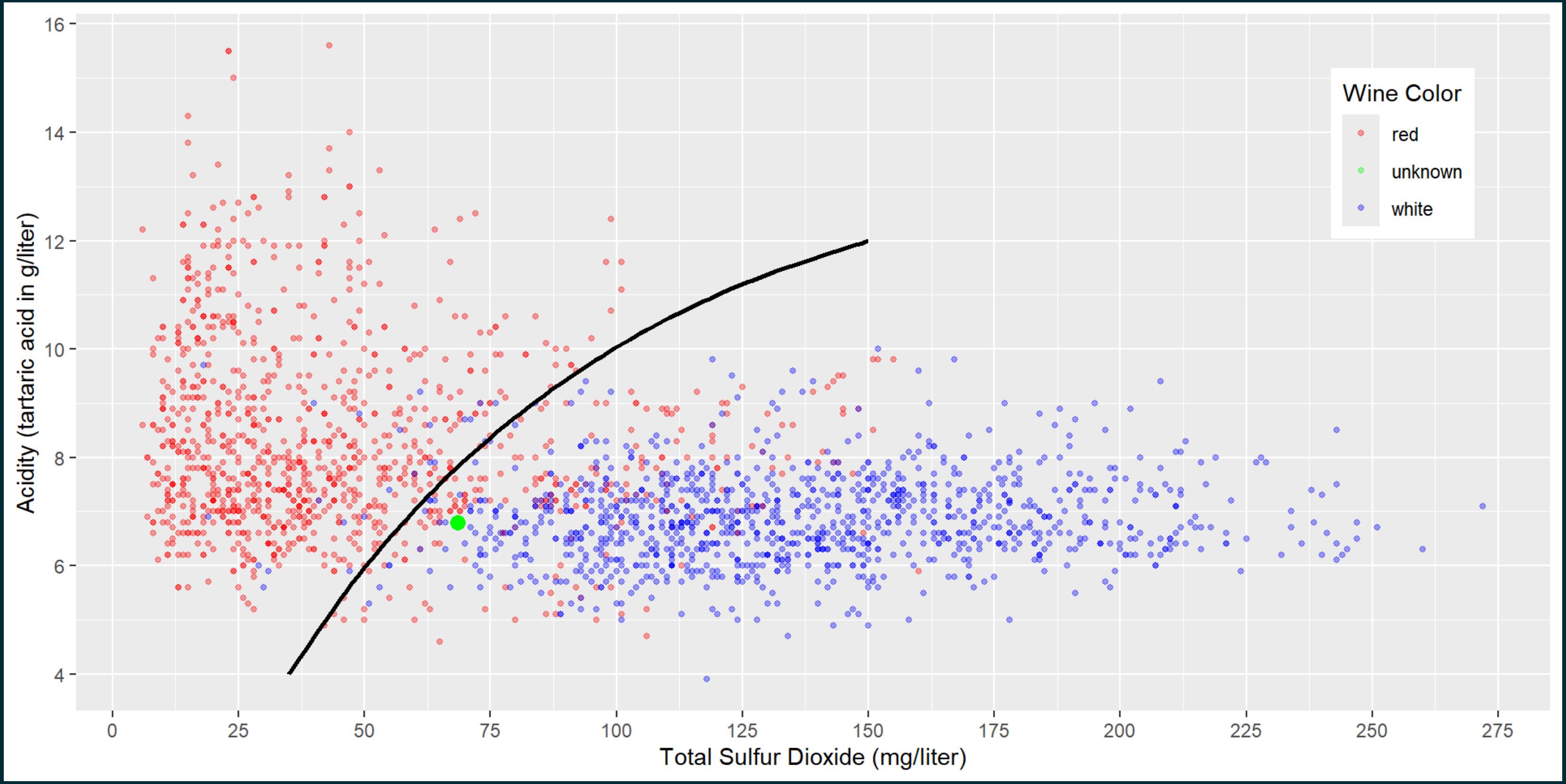
CONFUSION MATRIX

```
            Truth
Prediction   Red Wine   White Wine
  Red Wine   TP: 'most' FP: 'few'
  White Wine FN: 'few'  TN: 'most'
```

Textbook

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

```
             Truth
Prediction   Red Wine   White Wine
  Red Wine   TP: 'most' FP: 'few'
  White Wine FN: 'few'  TN: 'most'
```

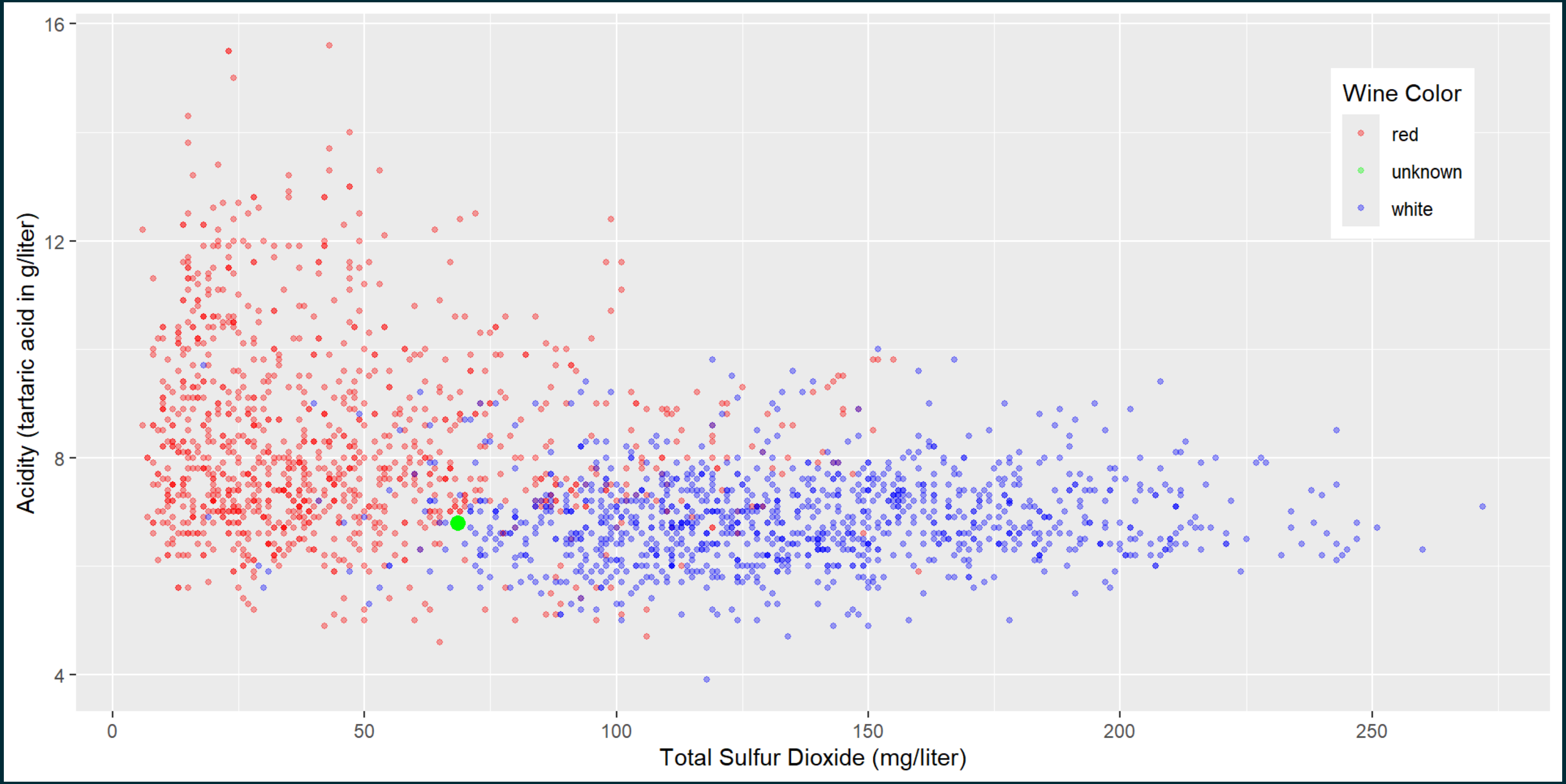
Textbook

SO, HOW DOES K NEAREST NEIGHBORS WORK?

Textbook

K NEAREST NEIGHBORS K=1

(FIND THE CLOSEST DATAPOINT TO THE ONE WE WANT TO PREDICT)

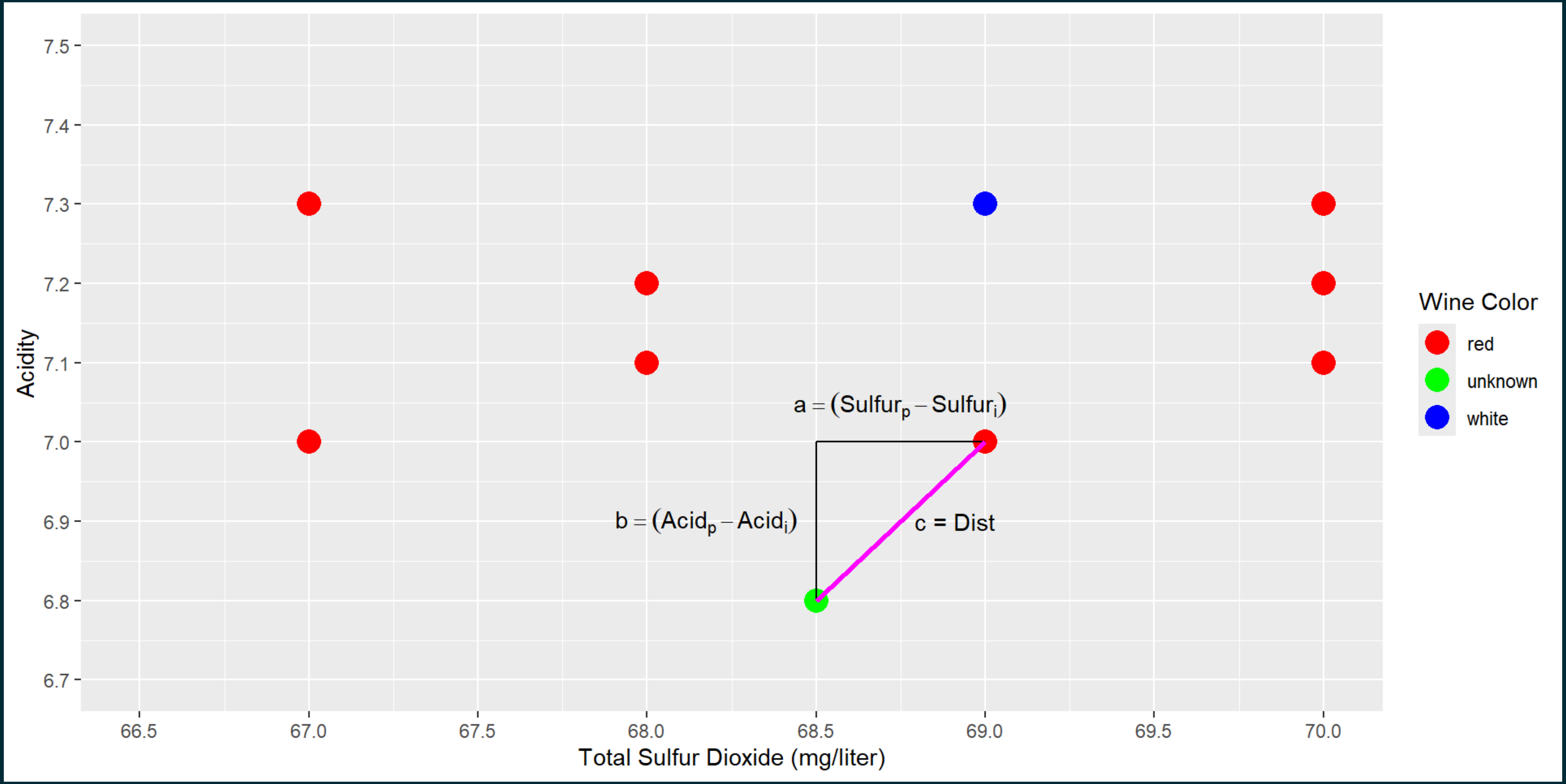


Acidity and Total Sulfur Dioxide Related to Wine Color

Textbook

K NEAREST NEIGHBORS K=1

(FIND THE CLOSEST DATAPOINT TO THE ONE WE WANT TO PREDICT)



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

Textbook

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):

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

Textbook

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):

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

Textbook

HOW TO CALCULATE EUCLIDEAN DISTANCE FOR N VARIABLES

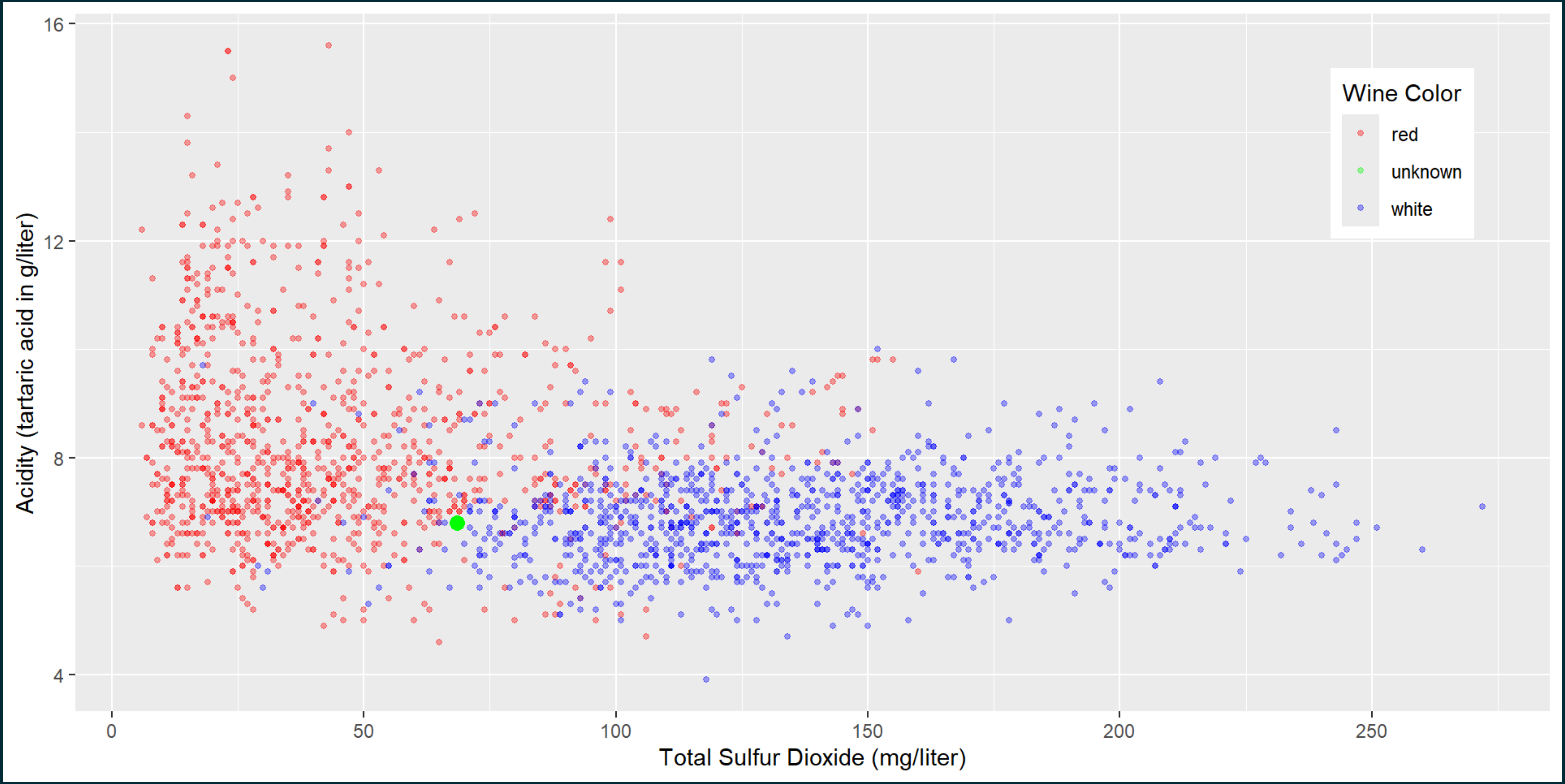
Assume our observations have N predictor variables v_j with $j = 1 \ldots N$. We compare the unknown point p to one of the points from the training data (e,g., point i):

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

Textbook

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

(FIND THE CLOSEST 4 DATAPOINT TO THE ONE WE WANT TO PREDICT)



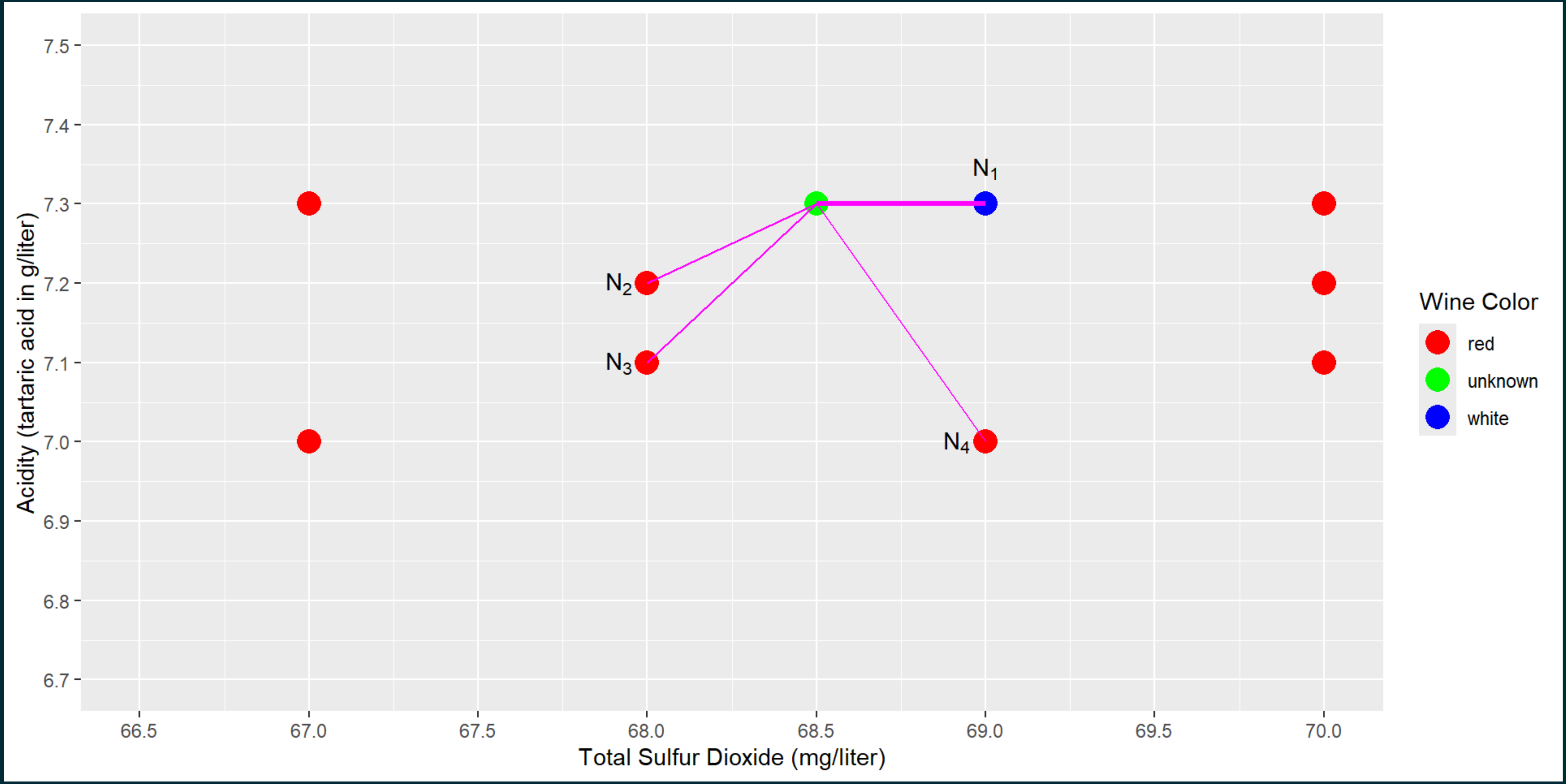
Acidity and Total Sulfur Dioxide Related to Wine Color

Textbook

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

(FIND THE CLOSEST 4 DATAPOINT TO THE ONE WE WANT TO PREDICT)

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

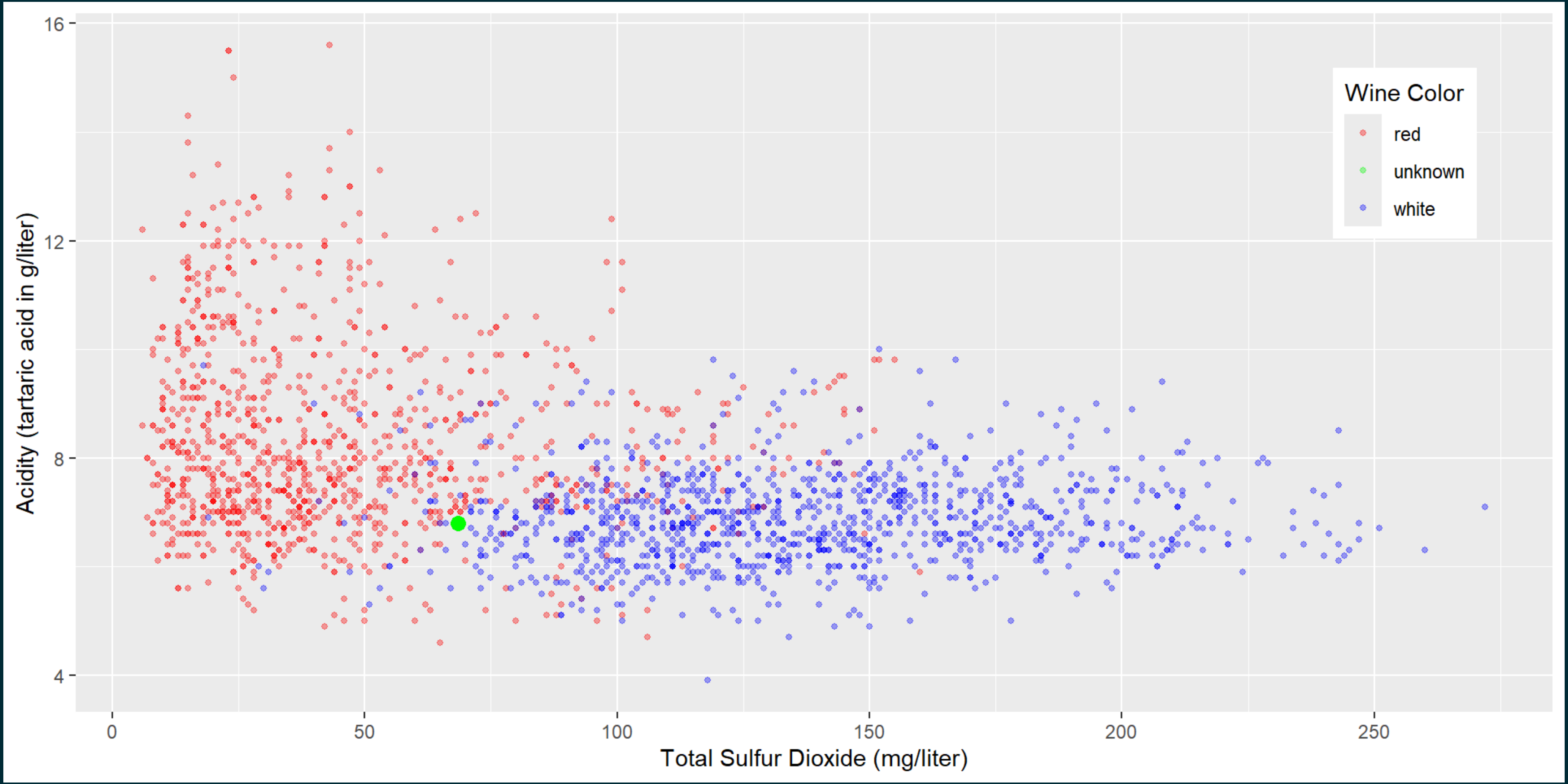


Predicting Wine Color with k-Nearest Neighbors (k=4)

Textbook

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

Textbook

A FEW COMMON SCALING OPTIONS

- **Same units**

  Divide or multiply to get the same units. This is often not possible (e.g., `Height` and `Weight`). Or it is not feasible (e.g. `Alcohol` and `StrawberryJuice` content in spiked strawberry drink))

- **Rescaling**

  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:

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

Textbook

TIME TO RUN K-NEAREST NEIGHBORS

The required library `tidymodels`, `kknn` and `janitor` are loaded in the background, and the Data we are using are already saved in the data frame `DataWine` (see below):

R Code Start Over Run Code

```r
library(kableExtra)
DataWine=readRDS(url("https://ai.lange-analytics.com/data/WineData.rds")) |>
          clean_names("upper_camel") |>
          select(WineColor,Sulfur=TotalSulfurDioxide,Acidity) |>
          mutate(WineColor=as.factor(WineColor))
kable(head(DataWine))
```

Textbook

GENERATE TRAINING AND TESTING DATA (SPLITTING):

R Code Start Over Run Code

```r
set.seed(876)
Split7030=initial_split(DataWine,prop=0.7,strata = WineColor)
DataTrain=training(Split7030)
DataTest=testing(Split7030)

Split7030$in_id[1:20] #only for output; usually not needed
```

`DataWine`

R Code Start Over Run Code

```r
head(Split7030$data, n=20)
```

`DataTrain`

R Code Start Over Run Code

```r
head(DataTrain)
```

`DataTest` (not original order)

R Code Start Over Run Code

```r
head(DataTest)
```

R Code Start Over Run Code

```r
cat("Proportion of red wines in DataTrain", mean(DataTrain$WineColor=="red"))
cat("Proportion of red wines in DataTest", mean(DataTest$WineColor=="red"))
```

Textbook

DEFINING THE RECIPE

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

Recipe: Prepare Data for Analysis:

R Code   Start Over   Run Code

```r
RecipeWine=recipe(WineColor~Acidity+Sulfur, data = DataTrain) |>
            step_naomit() |>
            step_normalize(all_predictors())
```

Or:

R Code   Start Over   Run Code

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

R Code   Start Over   Run Code

```r
print(RecipeWine) #red output does not mean error
```

Textbook

CREATING THE MODEL-DESIGN

CLICK HERE TO FIND A REFERENCE LIST FOR VARIOUS *MODEL- DESIGNS* COMMANDS

R Code Start Over Run Code

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

`weight_func = "rectangular"` is needed to use a *textbook version* of *k-Nearest-Neighbors*. It can be omitted for research.

ADDING RECIPE & MODEL-DESIGN TO WORKFLOW AND FIT() IT WITH THE TRAINING DATA

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

R Code Start Over Run Code

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

Textbook

THE PREDICTION PROCEDURE

How to use the **fitted** `workflow` which contains the training data (`DAtaTrain`)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).

Textbook

PREDICTING WINE COLOR OF TESTING DATA USING THE `predict()` COMMAND

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

R Code Start Over Run Code

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

Textbook

PREDICTING WINE COLOR OF TESTING DATA USING THE `augment()` COMMAND

Predicting with the fitted workflow using `augment()`. The `augment()` command *predicts* and then *augments* `DataTest` with the predictions:

R Code Start Over Run Code

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

Textbook

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

Confusion Matrix:

R Code Start Over Run Code

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

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 *predicted* `red`) is in the **first row** and shows all wines that are *predicted* `positive` (`red`). 436 are *predicted* correctly (**TP: True Positives**), and 46 are *predicted* incorrectly as `red` but are `white` (**FP: False Positives**).

- The **negative class** (wine is *predicted* `white`) is in the **second row** and shows all wines that are *predicted* `negative` (`white`). 44 are *predicted* incorrectly as `white` but are `red` (**FP: False Negatives**), and 434 are *predicted* correctly as `white`(**TP: True Negatives**).

- Remember:
  - in `tidymodels` *first class* is always *positive*
  - when determining `TP`, `FP`, `FN`, `TN`, approach matrix from the *prediction side*

Accuracy: Number of wines on diagonal/number of all wines ($\frac{TP+TN}{TP+TN+FP+FN}$):

R Code Start Over Run Code

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

Textbook

TIDYMODELS MODELING PIPELINE

1. Split the Data into *Training* and *Testing Data*

```r
set.seed(876)
Split7030=initial_split(DataToAnalyzeName, prop=0.7, strata=OutcomeVarName)
DataTrain=training(Split7030)
DataTest=testing(Split7030)
```

2. Create Recipe

```r
RecipeName=recipe(OutcomeVarName~Pred1VarName+Pred2VarName, data=DataTrain) |>
           step_naomit() |>
           step_Name()
```

3 Create Model-Design

```r
ModelDesignName=MachLearnModelName() |>
                set_engine("PackageName") |>
                set_mode("classification or regression")
```

4. Create Workflow and Fit to Training Data

```r
WFModelName=workflow() |>
            add_recipe(RecipeName) |>
            add_model(ModelDesignName) |>
            fit(DataTrain)
```

5. Predict with Workflow and Augment Testing Data with Predictions

```r
DataTestWithPred=augment(WFModelName, DataTest)
```

Textbook

6. Assess Predictive Quality with Metrics

```
MetricsCommand(DataTestWithPred, truth=OutcomeVarName,
                        estimate=.pred_class or estimate=.pred)
```

Textbook

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 THE 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  973
```

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 $\frac{1}{37} / cdot 1000 = 27$, and the others (1000-27=973) are expected to lose.

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

Textbook

WARNING: BE CAREFUL WITH THE ACCURACY RATE

THE STORY OF DR. NEBULOUS'S GAMBLERS SYSTEM

```
          Truth
Prediction Win Lose
      Win  0    0
      Lose 27   973
```

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

Textbook

USING ACCURACY(), SENSITIVITY() AND SPECIFICITY() TO ASSES PREDICTION QUALITY

accuracy()

R Code Start Over Run Code

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

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$$

- Remember:
  - TP and TN are the elements on the main diagonal of the confusion matrix

Textbook

USING ACCURACY(), SENSITIVITY() AND SPECIFICITY() TO ASSES PREDICTION QUALITY

`sensitivity()`

R Code Start Over Run Code

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

$Sensitivity = \frac{TP}{TP+FN}$ (first column)

- Remember:
  - in `tidymodels` *first class* is always *positive*, which is relevant for *Sensitivity*
  - when determining *Sensitivity* and *Specificity* approach matrix from the *Truth* side

USING ACCURACY(), SENSITIVITY() AND SPECIFICITY() TO ASSES PREDICTION QUALITY

`specificity()`

R Code Start Over Run Code

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

$Specificity = \frac{TN}{TN+FP}$ (second column)

- Remember:
  - in `tidymodels` *second class* is always *negative*, which is relevant for *Specificity*
  - when determining *Sensitivity* and *Specificity* approach matrix from the *Truth* side

Textbook

USING ALL VARIABLES IN THE WINE DATASET TO PREDICT WINE COLOR

Can we improve by using all predictors?

Try the Interactive Exercise: Chapter 4 – k-Nearest Neighbors: Predicting Wine Color

PROJECT: DESIGN A MACHINE LEARNING WORKFLOW FOR OPTICAL CHARACTER RECOGNITION

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.

Work with the Interactive Exercise: Chapter 4 – k-Nearest Neighbors Project: Read Images with Handwritten Digits