Glass ID Machine Learning Classification

Sam Dummer

10/25/2021

Abstract

There are many functions of glass all around the world. Each of these different functions requires a different method of production. Through these methods, we are able to achieve stronger glass, translucent glass, or even flexible glass. On top of that, each pane of glass comes with a different refraction index. In this data set we will be analyzing the chemical makeup, function, and refraction index of many glasses. Some of these uses include building windows both float processed and not, vehicle windows float processed, containers, table ware, and headlamps. Float processing is a certain step taken when producing glass that entails

Introduction

In this paper, we will be using the k-NN algorithm also known as the k-nearest neighbors algorithm. This algorithm pairs new values based on the neighbors. It takes the average or in some cases most common values from the number of neighbors. This number of neighbors is specified through the variable **k**. There are many ways to calculate or find the predicted value. For continuous values, Euclidean distance is used and the mean is found, but for discrete values, the overlap method is generally used. This method find the k number of closest variables and then assigns the mode of the k number of variables to the value being predicted. There are some ways to select a good value of k, but we decided to just try again and again to find a value that seemed to work best for the set.

To test the algorithm the data set is split into two sets: a training set and a testing set. Generally, the sets are split by 60-70% of the data for the training set and 30-40% of the data for the testing set. The training set is used to train the algorithm and then the algorithm checks itself with the testing set. To check itself, it will take the values from the testing set and use the algorithm to predict what values in a variable will be.

In our specific data set, we will be predicting the type of glass. We chose to split the data set up by 70% and 30%.

Within this data set there are 214 different observations of glass production. Of these 214 observations, there are 11 variables to describe each of them. These being the ID number, the refraction index, the percent chemical makeup(Na, Mg, Al, Si, K, Ca, Ba, Fe), and the function/type of glass.

Methodology

To analyze this data, we start by performing the basic tasks to set everything up such as cleaning the environment and setting the directory. Then we quickly look at the structure and summary. We realized there were no headers so we created some then we created a new column that helped to describe what each type of glass was. Then we plotted some scatter plots of the chemical makeup and type of glass Then we plotted some pairwise correlation plots. Then, since we are using the algorithm to predict the the type of glass, we took a look at the distribution of the type of glass in the data set. Then we set everything up to run the data set then ran it. We then merged the predicted values into the

Results

Cleaning Environment, Setting Up Directory, and Loading Libraries

To help get us ready for the script, we need to clean up the environment from any previous scripts. Then we set the directory and load in any necessary librries. Some of these libraries include, tidyverse, cluster, and factoextra. The last two help with the the cluster analysis.

```
# Sam Dummer
# October 26, 2021
# WK10L1Dummer.R
rm(list=ls())
setwd("C:/Users/isabe/Desktop/RFLoder")
library(class)
library(knitr)
library(ggvis)
library(gmodels)
library(tidyverse)
library(caret)
library(GGally)
library(gridExtra)
```

Reading in Data Set

Now that everything is set up and ready for the data set, we can read in the data. In this case we use the read.delim2() function.

```
glass <- read.csv("glass.csv", header = F)
```

Summary and Structure

Looking at the summary and structure of the data is the most important thing to do once the data has been loaded. Doing so helps to find any problems when reading in the data and helps to get a general sense of the data set. The functions we used to observe the structure and summary were head(), tail(), str(), glimpse(), and summary(). Through this we were able to recognize that the headers were not labeled and therefore should rename them.

```
head(glass)
```

```
##   V1      V2    V3   V4   V5    V6   V7   V8 V9  V10 V11
## 1  1 1.52101 13.64 4.49 1.10 71.78 0.06 8.75  0 0.00   1
## 2  2 1.51761 13.89 3.60 1.36 72.73 0.48 7.83  0 0.00   1
## 3  3 1.51618 13.53 3.55 1.54 72.99 0.39 7.78  0 0.00   1
## 4  4 1.51766 13.21 3.69 1.29 72.61 0.57 8.22  0 0.00   1
## 5  5 1.51742 13.27 3.62 1.24 73.08 0.55 8.07  0 0.00   1
## 6  6 1.51596 12.79 3.61 1.62 72.97 0.64 8.07  0 0.26   1
```

```
tail(glass)
```

```
##      V1      V2    V3 V4   V5    V6   V7   V8   V9 V10 V11
## 209 209 1.51640 14.37  0 2.74 72.85 0.00 9.45 0.54   0   7
## 210 210 1.51623 14.14  0 2.88 72.61 0.08 9.18 1.06   0   7
## 211 211 1.51685 14.92  0 1.99 73.06 0.00 8.40 1.59   0   7
## 212 212 1.52065 14.36  0 2.02 73.42 0.00 8.44 1.64   0   7
## 213 213 1.51651 14.38  0 1.94 73.61 0.00 8.48 1.57   0   7
## 214 214 1.51711 14.23  0 2.08 73.36 0.00 8.62 1.67   0   7
```

```
head(glass)
```

```
##   V1      V2    V3   V4   V5    V6   V7   V8 V9  V10 V11
## 1  1 1.52101 13.64 4.49 1.10 71.78 0.06 8.75  0 0.00   1
## 2  2 1.51761 13.89 3.60 1.36 72.73 0.48 7.83  0 0.00   1
## 3  3 1.51618 13.53 3.55 1.54 72.99 0.39 7.78  0 0.00   1
## 4  4 1.51766 13.21 3.69 1.29 72.61 0.57 8.22  0 0.00   1
## 5  5 1.51742 13.27 3.62 1.24 73.08 0.55 8.07  0 0.00   1
## 6  6 1.51596 12.79 3.61 1.62 72.97 0.64 8.07  0 0.26   1
```

```
str(glass)
```

```
## 'data.frame':    214 obs. of  11 variables:
##  $ V1 : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ V2 : num  1.52 1.52 1.52 1.52 1.52 ...
##  $ V3 : num  13.6 13.9 13.5 13.2 13.3 ...
##  $ V4 : num  4.49 3.6 3.55 3.69 3.62 3.61 3.6 3.61 3.58 3.6 ...
##  $ V5 : num  1.1 1.36 1.54 1.29 1.24 1.62 1.14 1.05 1.37 1.36 ...
##  $ V6 : num  71.8 72.7 73 72.6 73.1 ...
##  $ V7 : num  0.06 0.48 0.39 0.57 0.55 0.64 0.58 0.57 0.56 0.57 ...
##  $ V8 : num  8.75 7.83 7.78 8.22 8.07 8.07 8.17 8.24 8.3 8.4 ...
##  $ V9 : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ V10: num  0 0 0 0 0 0.26 0 0 0 0.11 ...
##  $ V11: int  1 1 1 1 1 1 1 1 1 1 ...
```

```
glimpse(glass)
```

```
## Rows: 214
## Columns: 11
## $ V1  <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,~
## $ V2  <dbl> 1.52101, 1.51761, 1.51618, 1.51766, 1.51742, 1.51596, 1.51743, 1.5~
## $ V3  <dbl> 13.64, 13.89, 13.53, 13.21, 13.27, 12.79, 13.30, 13.15, 14.04, 13.~
## $ V4  <dbl> 4.49, 3.60, 3.55, 3.69, 3.62, 3.61, 3.60, 3.61, 3.58, 3.60, 3.46, ~
## $ V5  <dbl> 1.10, 1.36, 1.54, 1.29, 1.24, 1.62, 1.14, 1.05, 1.37, 1.36, 1.56, ~
## $ V6  <dbl> 71.78, 72.73, 72.99, 72.61, 73.08, 72.97, 73.09, 73.24, 72.08, 72.~
## $ V7  <dbl> 0.06, 0.48, 0.39, 0.57, 0.55, 0.64, 0.58, 0.57, 0.56, 0.57, 0.67, ~
## $ V8  <dbl> 8.75, 7.83, 7.78, 8.22, 8.07, 8.07, 8.17, 8.24, 8.30, 8.40, 8.09, ~
## $ V9  <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ V10 <dbl> 0.00, 0.00, 0.00, 0.00, 0.00, 0.26, 0.00, 0.00, 0.00, 0.11, 0.24, ~
## $ V11 <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ~
```

```
summary(glass)
```

```
##        V1               V2              V3              V4
##  Min.   :  1.00   Min.   :1.511   Min.   :10.73   Min.   :0.000
##  1st Qu.: 54.25   1st Qu.:1.517   1st Qu.:12.91   1st Qu.:2.115
##  Median :107.50   Median :1.518   Median :13.30   Median :3.480
##  Mean   :107.50   Mean   :1.518   Mean   :13.41   Mean   :2.685
##  3rd Qu.:160.75   3rd Qu.:1.519   3rd Qu.:13.82   3rd Qu.:3.600
##  Max.   :214.00   Max.   :1.534   Max.   :17.38   Max.   :4.490
##        V5              V6              V7              V8
##  Min.   :0.290   Min.   :69.81   Min.   :0.0000   Min.   : 5.430
##  1st Qu.:1.190   1st Qu.:72.28   1st Qu.:0.1225   1st Qu.: 8.240
##  Median :1.360   Median :72.79   Median :0.5550   Median : 8.600
##  Mean   :1.445   Mean   :72.65   Mean   :0.4971   Mean   : 8.957
##  3rd Qu.:1.630   3rd Qu.:73.09   3rd Qu.:0.6100   3rd Qu.: 9.172
##  Max.   :3.500   Max.   :75.41   Max.   :6.2100   Max.   :16.190
##        V9             V10              V11
##  Min.   :0.000   Min.   :0.00000   Min.   :1.00
##  1st Qu.:0.000   1st Qu.:0.00000   1st Qu.:1.00
##  Median :0.000   Median :0.00000   Median :2.00
##  Mean   :0.175   Mean   :0.05701   Mean   :2.78
##  3rd Qu.:0.000   3rd Qu.:0.10000   3rd Qu.:3.00
##  Max.   :3.150   Max.   :0.51000   Max.   :7.00
```

Quick Cleaning of Data Set

After reading in the data, there were a few problems we found. This included untitled headers and unlabeled factors. To fix the headers, we simply renamed them and then we created a new variable that included the specific classification/label of each observation in the "type" variable.

```
names(glass) <- c("ID", "refrac", "Na", "Mg", "Al", "Si", "K", "Ca", "Ba", "Fe", "type")
names(glass)
```

```
##  [1] "ID"     "refrac" "Na"     "Mg"     "Al"     "Si"     "K"      "Ca"
##  [9] "Ba"     "Fe"     "type"
```

```
# 1 building_windows_float_processed
# 2 building_windows_non_float_processed
# 3 vehicle_windows_float_processed
# 4 vehicle_windows_non_float_processed (none in this database)
# 5 containers
# 6 tableware
# 7 headlamps
glass$type_label  <- factor(glass$type,
levels=c(1,2,3,4,5,6,7),
labels = c("building_windows_float_processed",
"building_windows_non_float_processed",
"vehicle_windows_float_processed",
"vehicle_windows_non_float_processed", "containers",
"tableware", "headlamps"))
head(glass)
```

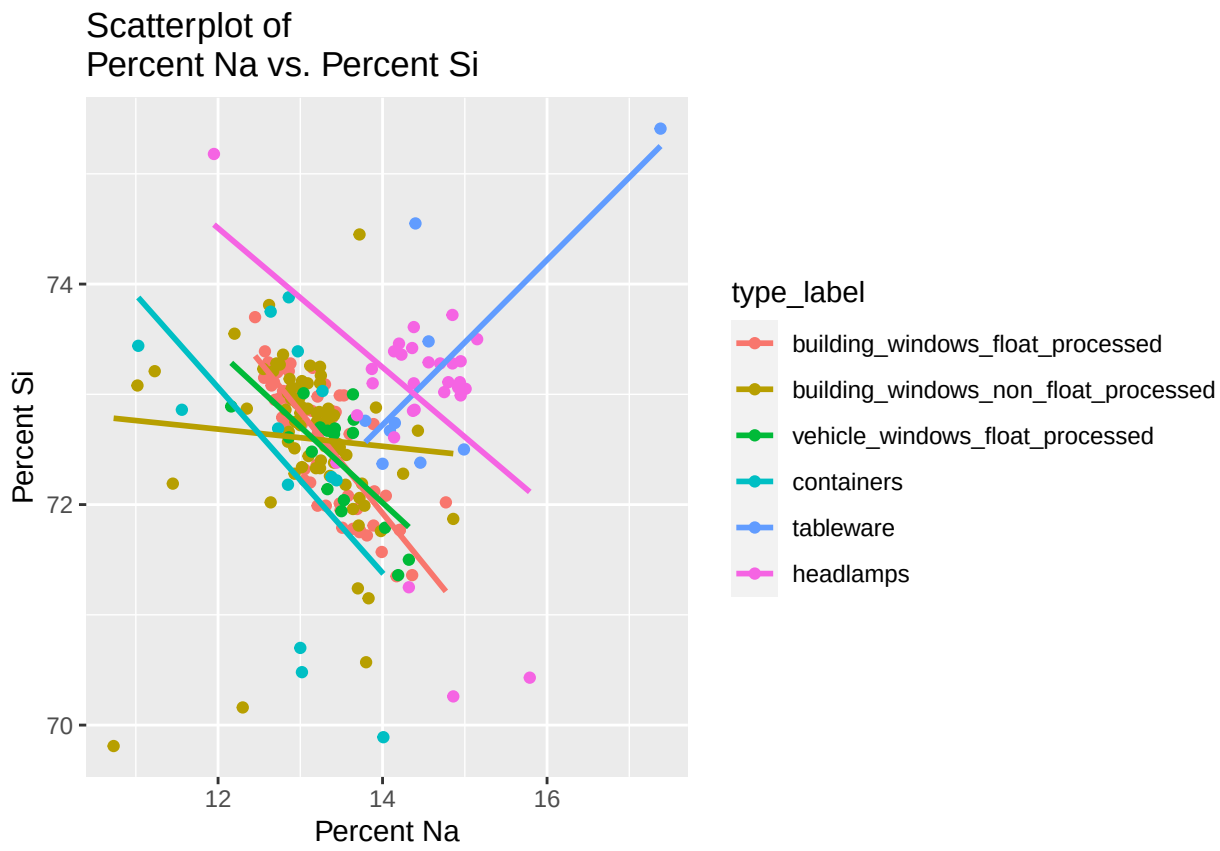
```
##   ID  refrac    Na   Mg   Al    Si    K   Ca Ba   Fe type
## 1  1 1.52101 13.64 4.49 1.10 71.78 0.06 8.75  0 0.00    1
```

```
## 2  2 1.51761 13.89 3.60 1.36 72.73 0.48 7.83  0 0.00    1
## 3  3 1.51618 13.53 3.55 1.54 72.99 0.39 7.78  0 0.00    1
## 4  4 1.51766 13.21 3.69 1.29 72.61 0.57 8.22  0 0.00    1
## 5  5 1.51742 13.27 3.62 1.24 73.08 0.55 8.07  0 0.00    1
## 6  6 1.51596 12.79 3.61 1.62 72.97 0.64 8.07  0 0.26    1
##                         type_label
## 1 building_windows_float_processed
## 2 building_windows_float_processed
## 3 building_windows_float_processed
## 4 building_windows_float_processed
## 5 building_windows_float_processed
## 6 building_windows_float_processed
```

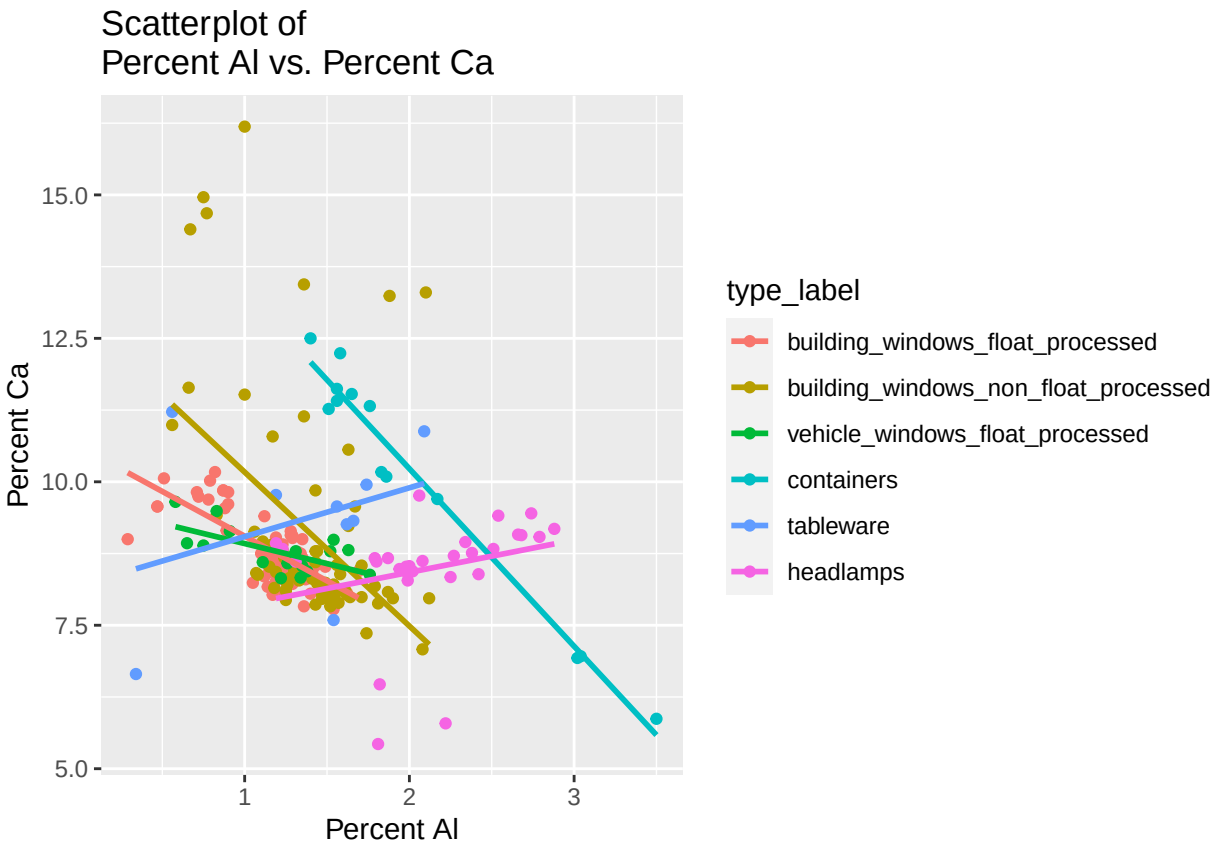
Plotting the Data

Next we decided to perform some Exploratory Data Analysis(EDA) this was done by plotting some functions to help summarize the data. For the first plots, we plotted some scatter plots colored by their type and then a linear regression was plotted for each type. In this, we are able to observe that there was a negative correlation between the percent composition of sodium and silicon in every type except headlamps, were there is a steep positive correlation. Additionally, there don't seem to be many linear regression with a strong correlation except for headlamps and "building_windows_float_processed."

```
ggplot(glass, aes(Na, Si, color = type_label)) + geom_point() +
labs(x = "Percent Na", y = "Percent Si",
title = "Scatterplot of
Percent Na vs. Percent Si") +
geom_smooth(method = "lm", se = F)
```



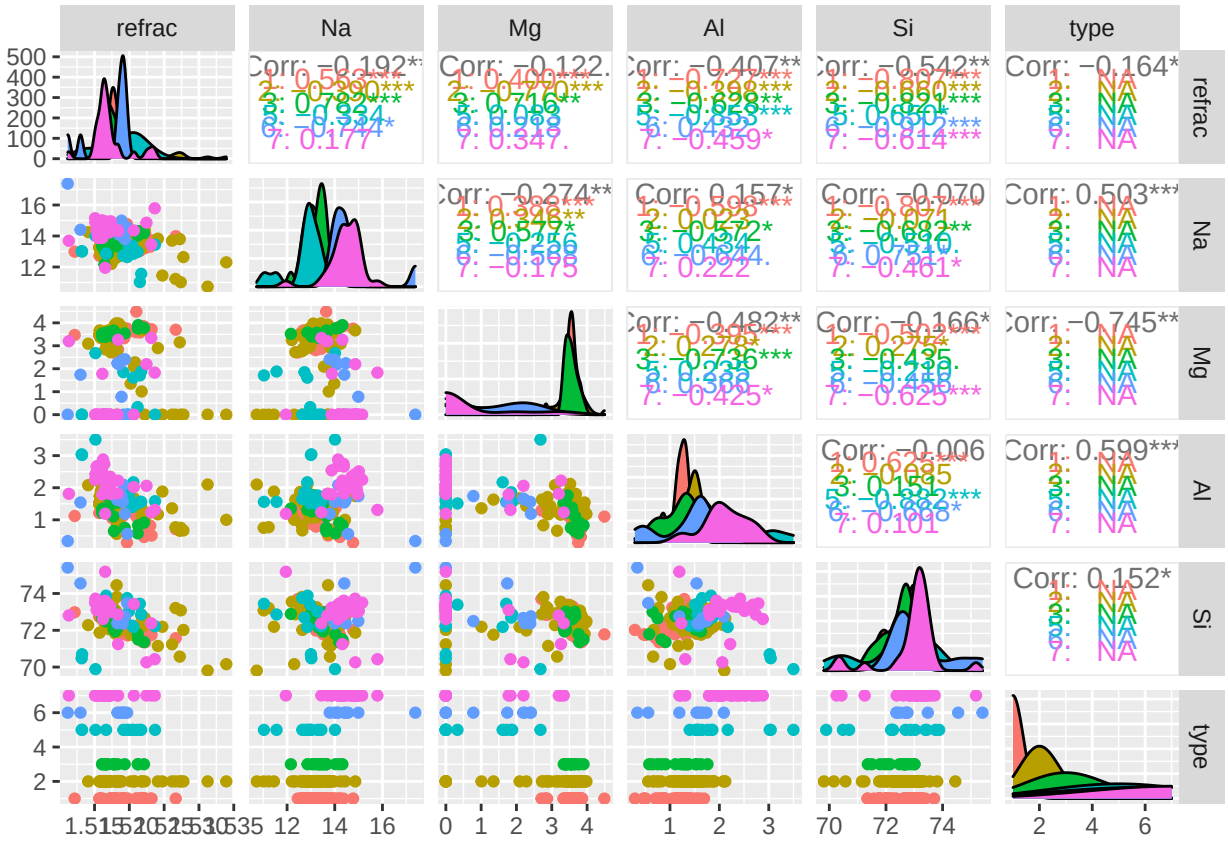

```
ggplot(glass, aes(Al, Ca, color = type_label)) + geom_point() +
labs(x = "Percent Al", y = "Percent Ca",
title = "Scatterplot of
Percent Al vs. Percent Ca") +
geom_smooth(method = "lm", se = F)
```



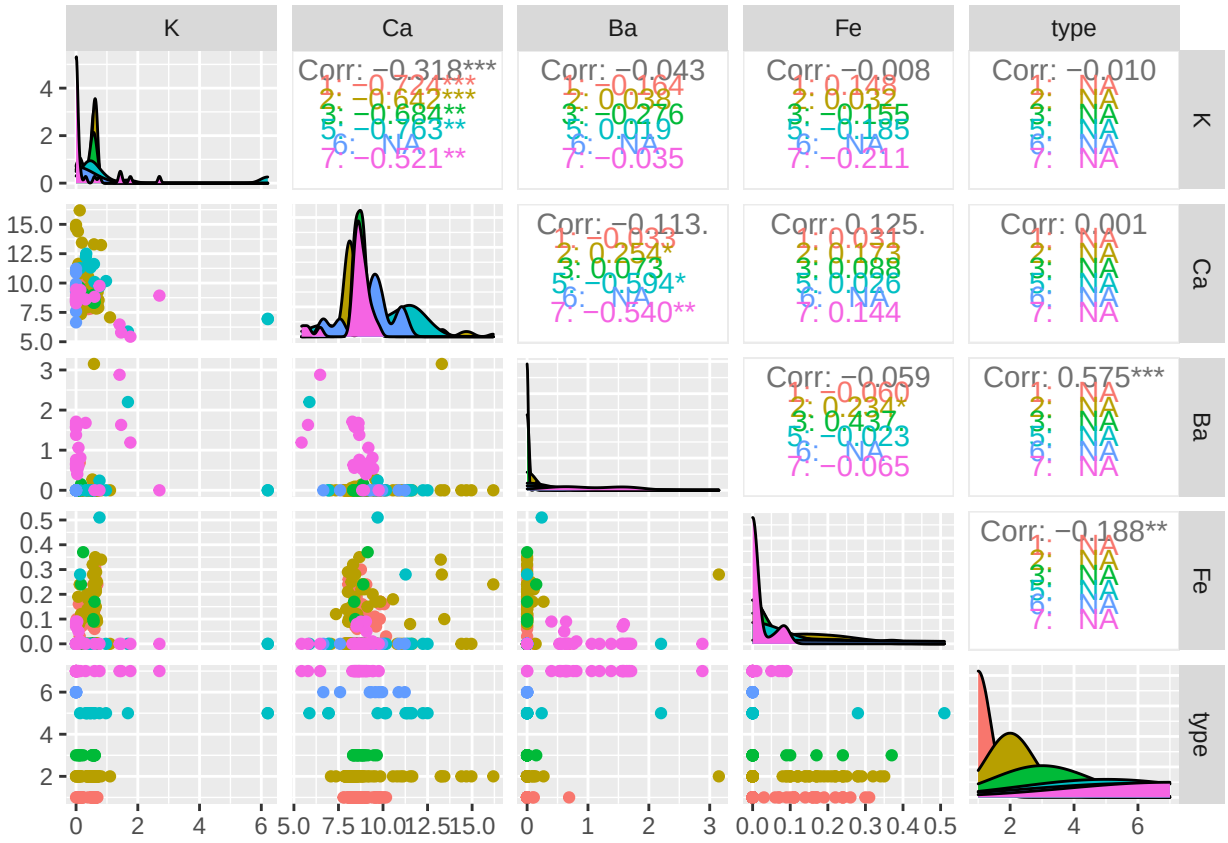
Pairwise Plots

The other form of graph that we are analyzing are pairwise plots. This are very helpful is visualizing the correlation between different variables. In this we are able to observe that there are not many correlations over .5 in the positive or negative. The largest correlation we observed was between Magnesium and the type of glass.

```
short1 <- glass[, c(2:6,11)]
short2 <- glass[, c(7:10,11)]
ggpairs(short1, aes(color = as.character(type)))
```




```
ggpairs(short2, aes(color = as.character(type)))
```



Taking a Look Before Running Algorithm

Before we run the prediction, it is best to see the distribution of the types throughout the data set to get a good overview. We looked at a table and a proportional table. Most of the glass distribution seems to be building windows.

```
table(glass$type)
```

```
##
##  1  2  3  5  6  7
## 70 76 17 13  9 29
```

```
round(prop.table(table(glass$type)) * 100, digits = 1)
```

```
##
##    1    2    3    5    6    7
## 32.7 35.5  7.9  6.1  4.2 13.6
```

Prearing for the k-NN Algorithm

Before we are able to run the k-NN algorithm, there are a few tasks needed to be done. Firstly, we must set a seed since this algorithm is seed-based. Then, we split the data set into two sides one which is 70% of the data set and the other which is 30%. Then we assign all values in the 70% to a training set data set

and the other 30% to the test set. Then we create two more values which are labels for the training and testing sets. Once this is all done we are ready to run the k-NN algorithm.

```
set.seed(5816497)
split <- sample(2, nrow(glass), replace=TRUE, prob=c(0.7, 0.3))
split
```

```
##   [1] 1 1 1 1 2 2 1 1 2 1 1 1 1 2 2 2 2 1 1 1 1 2 1 1 1 1 1 1 2 1 1 2 1 2 1 2 1
##  [38] 1 1 2 1 1 1 1 1 2 1 1 1 2 1 1 1 2 1 1 1 1 1 1 1 2 1 1 2 1 1 1 1 2 2 1 1 1
##  [75] 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 2 1 2 1 1 1 1 2 1 1 1 2
## [112] 1 1 1 1 2 1 1 1 1 1 1 1 2 1 1 2 1 1 1 2 1 1 1 1 2 1 1 2 1 1 2 2 1 1 1 2 1
## [149] 1 1 1 1 1 1 1 2 1 1 2 1 1 1 1 2 1 1 1 1 1 2 2 1 1 1 2 2 1 1 1 2 1 1 2 1 1 1
## [186] 1 1 1 2 1 1 1 2 1 1 1 2 1 2 1 1 2 1 2 1 1 2 1 1 1 1 1 1 2
```

```
glass.training <- glass[split==1, 1:11]
head(glass.training)
```

```
##   ID  refrac    Na   Mg   Al    Si    K   Ca Ba Fe type
## 1  1 1.52101 13.64 4.49 1.10 71.78 0.06 8.75  0  0    1
## 2  2 1.51761 13.89 3.60 1.36 72.73 0.48 7.83  0  0    1
## 3  3 1.51618 13.53 3.55 1.54 72.99 0.39 7.78  0  0    1
## 4  4 1.51766 13.21 3.69 1.29 72.61 0.57 8.22  0  0    1
## 7  7 1.51743 13.30 3.60 1.14 73.09 0.58 8.17  0  0    1
## 8  8 1.51756 13.15 3.61 1.05 73.24 0.57 8.24  0  0    1
```

```
glass.test <- glass[split==2, 1:11]
head(glass.training)
```

```
##   ID  refrac    Na   Mg   Al    Si    K   Ca Ba Fe type
## 1  1 1.52101 13.64 4.49 1.10 71.78 0.06 8.75  0  0    1
## 2  2 1.51761 13.89 3.60 1.36 72.73 0.48 7.83  0  0    1
## 3  3 1.51618 13.53 3.55 1.54 72.99 0.39 7.78  0  0    1
## 4  4 1.51766 13.21 3.69 1.29 72.61 0.57 8.22  0  0    1
## 7  7 1.51743 13.30 3.60 1.14 73.09 0.58 8.17  0  0    1
## 8  8 1.51756 13.15 3.61 1.05 73.24 0.57 8.24  0  0    1
```

```
glass.trainingLabels <- glass[split==1, 11]
glass.testLabels <- glass[split==2, 11]
```

Running and Checking the Algorithm with the Test Set

Now that everything is set up an ready for the algorithm to run it is time to run. In the function, "cl" is the tru classifications of the training set. Next, we need to set the k value which with some trial and error we found to be 14. Many other values were very good, but this specific circumstance created a perfect result when testing it. For this specific (or really any) data set, this is very good since it means that the algorithm can perfectly predict values which can be very helpful.

```
glassPred <- knn(train = glass.training, test = glass.test, cl = glass.trainingLabels, k = 14)
glassPred
```

```
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3
## [39] 3 5 5 5 5 6 6 7 7 7 7 7 7 7 7
## Levels: 1 2 3 5 6 7
```

```
glass.testLabels
```

```
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3
## [39] 5 5 5 5 5 6 6 7 7 7 7 7 7 7 7
```

Merging the Values in Data Set

Now that we have officially run and tested the algorithm, we are able to merge this into the data set for it to be easily viewed. This can be don using the cbind() function and binding the testing set and new data together. Then new names are given the the new data. This being "Observed" and "Predicted."

```
glassmerge <- data.frame(glass.testLabels, glassPred)
names <- colnames(glass.test)
finalglass <- cbind(glass.test, glassmerge)
names(finalglass) <- c(names, "Oberved", "Predicted")
```

Creating a Crosstable of the Data

Once all our tasks are done, we are able to create a crosstable of all the data. In our case, we were able to achieve a perfect result. This can be seen in the table since all the data is diagonally through the middle of the table. This means that all the predicted and observed values are the same. This can also be see in the proportion under the number of variables which shows the proportion of the column and row which are all 100% for each.

```
CrossTable(x = glass.testLabels, y = glassPred, prop.chisq = F)
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  53
##
##
##                  | glassPred
## glass.testLabels |         1 |         2 |         3 |         5 |         6 |         7 | Row Total
## -----------------|-----------|-----------|-----------|-----------|-----------|-----------|----------
##                1 |        18 |         1 |         0 |         0 |         0 |         0 |        19
##                  |     0.947 |     0.053 |     0.000 |     0.000 |     0.000 |     0.000 |     0.358
##                  |     1.000 |     0.059 |     0.000 |     0.000 |     0.000 |     0.000 |
##                  |     0.340 |     0.019 |     0.000 |     0.000 |     0.000 |     0.000 |
```

```
## -----------------|----------|----------|----------|----------|----------|----------|----------
##                2 |        0 |       16 |        0 |        0 |        0 |        0 |       16
##                  |    0.000 |    1.000 |    0.000 |    0.000 |    0.000 |    0.000 |    0.302
##                  |    0.000 |    0.941 |    0.000 |    0.000 |    0.000 |    0.000 |
##                  |    0.000 |    0.302 |    0.000 |    0.000 |    0.000 |    0.000 |
## -----------------|----------|----------|----------|----------|----------|----------|----------
##                3 |        0 |        0 |        3 |        0 |        0 |        0 |        3
##                  |    0.000 |    0.000 |    1.000 |    0.000 |    0.000 |    0.000 |    0.057
##                  |    0.000 |    0.000 |    0.750 |    0.000 |    0.000 |    0.000 |
##                  |    0.000 |    0.000 |    0.057 |    0.000 |    0.000 |    0.000 |
## -----------------|----------|----------|----------|----------|----------|----------|----------
##                5 |        0 |        0 |        1 |        4 |        0 |        0 |        5
##                  |    0.000 |    0.000 |    0.200 |    0.800 |    0.000 |    0.000 |    0.094
##                  |    0.000 |    0.000 |    0.250 |    1.000 |    0.000 |    0.000 |
##                  |    0.000 |    0.000 |    0.019 |    0.075 |    0.000 |    0.000 |
## -----------------|----------|----------|----------|----------|----------|----------|----------
##                6 |        0 |        0 |        0 |        0 |        2 |        0 |        2
##                  |    0.000 |    0.000 |    0.000 |    0.000 |    1.000 |    0.000 |    0.038
##                  |    0.000 |    0.000 |    0.000 |    0.000 |    1.000 |    0.000 |
##                  |    0.000 |    0.000 |    0.000 |    0.000 |    0.038 |    0.000 |
## -----------------|----------|----------|----------|----------|----------|----------|----------
##                7 |        0 |        0 |        0 |        0 |        0 |        8 |        8
##                  |    0.000 |    0.000 |    0.000 |    0.000 |    0.000 |    1.000 |    0.151
##                  |    0.000 |    0.000 |    0.000 |    0.000 |    0.000 |    1.000 |
##                  |    0.000 |    0.000 |    0.000 |    0.000 |    0.000 |    0.151 |
## -----------------|----------|----------|----------|----------|----------|----------|----------
##     Column Total |       18 |       17 |        4 |        4 |        2 |        8 |       53
##                  |    0.340 |    0.321 |    0.075 |    0.075 |    0.038 |    0.151 |
## -----------------|----------|----------|----------|----------|----------|----------|----------
##
##
```

Conclusion

Overall, this paper looked over the cleaning, EDA, and running of the k-NN algorithm on a data set that went over many different variables describing different glass panes. This k-NN algorithm was a great success, in predicting the values in the test set. This may not always be true in the end, but it does help to show that it is very accurate in predicting the type of glass. This would be very helpful especially if someone found some glass and studied it to find all the chemical components and refracive index. This algorithm could help them find out what type of glass it was.