# Title: KNN

Course: MBABusiness Analytics Module: 07 Author: Kariuki \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Run this code before you start.

# load needed packages  
library(tidyverse)

## ── Attaching packages ─────────────────────────────────────── tidyverse 1.3.2 ──  
## ✔ ggplot2 3.3.6 ✔ purrr 0.3.5   
## ✔ tibble 3.1.8 ✔ dplyr 1.0.10  
## ✔ tidyr 1.2.1 ✔ stringr 1.4.1   
## ✔ readr 2.1.3 ✔ forcats 0.5.2   
## ── Conflicts ────────────────────────────────────────── tidyverse\_conflicts() ──  
## ✖ dplyr::filter() masks stats::filter()  
## ✖ dplyr::lag() masks stats::lag()

#install.packages('e1071')  
library(e1071)  
# Confusion Matrix function  
my\_confusion\_matrix <- function(cf\_table) {  
 true\_positive <- cf\_table[4]  
 true\_negative <- cf\_table[1]  
 false\_positive <- cf\_table[2]  
 false\_negative <- cf\_table[3]  
 accuracy <- (true\_positive + true\_negative) / (true\_positive + true\_negative + false\_positive + false\_negative)  
 sensitivity\_recall <- true\_positive / (true\_positive + false\_negative)   
 specificity\_selectivity <- true\_negative / (true\_negative + false\_positive)  
 precision <- true\_positive / (true\_positive + false\_positive)   
 neg\_pred\_value <- true\_negative/(true\_negative + false\_negative)  
 print(cf\_table)  
 my\_list <- list(sprintf("%1.0f = True Positive (TP), Hit", true\_positive),  
 sprintf("%1.0f = True Negative (TN), Rejection", true\_negative),  
 sprintf("%1.0f = False Positive (FP), Type 1 Error", false\_positive),  
 sprintf("%1.0f = False Negative (FN), Type 2 Error", false\_negative),  
 sprintf("%1.4f = Accuracy (TP+TN/(TP+TN+FP+FN))", accuracy),   
 sprintf("%1.4f = Sensitivity, Recall, Hit Rate, True Positive Rate (How many positives did the model get right? TP/(TP+FN))", sensitivity\_recall),  
 sprintf("%1.4f = Specificity, Selectivity, True Negative Rate (How many negatives did the model get right? TN/(TN+FP))", specificity\_selectivity),   
 sprintf("%1.4f = Precision, Positive Predictive Value (How good are the model's positive predictions? TP/(TP+FP))", precision),  
 sprintf("%1.4f = Negative Predictive Value (How good are the model's negative predictions? TN/(TN+FN)", neg\_pred\_value)  
 )  
 return(my\_list)  
}

We will now us the KNN algorithm to address our business problem.

TECA is planning for the future and would like to set up their business so they are not so reliant on selling gas. One way to do that is to increase the sales of profitable products. Thus, TECA would like to predict when a transaction is a going to be a sale of a high gross profit margin product. Profit margin is the percentage of gross profit to revenue, or revenue minus costs divided by revenue. It is a percentage of how much profit each product makes or what percentage of profit is earned for each dollar of revenue.

The first thing we need to do is bring in the dataset we are going to work with. This is TECA data that has been transformed to be used in our KNN analysis. Let’s load it and examine it a bit. We first notice that the dataset has 20,000 rows and 38 features. This data is aggregated at the level of a purchase. Thus, each example or row is the purchase of one item. The first feature is the target feature or the dependent variable. This is the variable we are trying to predict. It is called high\_gpm. It is a factor that is either low or high. Next, we see revenue. This variable is the amount of revenue TECA makes for each purchase. Revenue is a continuous variable. Next, we have four variables related to quarter. These are one-hot encoded or dummy variables for the quarter of the year. For example, if the purchase happened during the first quarter of the year, quarter.1 would have a 1 value and the other three quarters’ variables would have a 0. In using the KNN algorithm we must all of the features, except the target feature, as numbers. Thus, to we created these four dummy variables from a one feature that called quarter that had four different types of entries–‘quarter 1’, ‘quarter 2’, ‘quarter 3’, and ‘quarter 4’. Next, income, bachelors\_degree, and population list these averages for the location of the store where the purchase took place. Next, we have another set of dummy variables–for each of the states TECA operates in. Next, we have num\_trans, which indicates how many purchases were made by this same person as part of this transaction. basket.no and basket.yes measure whether the purchase was part of a multi-part transaction. refill.no and refill.yes indicate whether or not the purchase was a refill of fountain soda. The next 11 dummy variables that all start with area, indicate the area of the store the purchase was made, as follows: \* area.alcohol: products in this area include, for example, wine and beer; \* area.cooler: products in this area include, for example, energy drinks, canned soda, and juice; \* area.dispensed: products in this area include, for example, cold and hot dispensed (fountain) drinks; \* area.fresh: products in this area include, for example, pizza, hot sandwiches, salads, and roller grill items; \* area.fuel: products in this area include, for example, gas; \* area.grocery: products in this area include, for example, milk, eggs, and cheese; \* area.lottery: products in this area include, for example, lottery tickets; \* area.miscellaneous: products in this area include, for example, store services and coupons; \* area.nongrocery: products in this area include, for example, clothing, magazines, medicine, and newspapers; \* area.snacks: products in this area include, for example, candy, gum, chips, and salty snacks; and \* area.tobacco: products in this area include, for example, cigarettes and chewing tobacco.

Next, items\_sold indicate how many of these items were purchased. Finally, the two loyalty dummies indicate whether the purchase was made by a loyalty customer (a customer who scanned their loyalty card) or not. Note, that the data was adjusted so that about half of the purchases in this dataset were made by loyalty customers, since TECA is particularly interested in these customers.

# Explore the data

knn\_input <- knn <- readRDS("C:/Users/User/OneDrive/Documents/projects/knn.rds")  
str(knn\_input)

## 'data.frame': 20000 obs. of 38 variables:  
## $ high\_gpm : Factor w/ 2 levels "low","high": 2 1 1 2 2 2 1 1 1 1 ...  
## $ revenue : num 0.99 27.25 2.49 2.26 3 ...  
## $ quarter.1 : int 1 0 1 1 0 0 0 0 0 0 ...  
## $ quarter.2 : int 0 1 0 0 1 1 1 0 0 0 ...  
## $ quarter.3 : int 0 0 0 0 0 0 0 0 1 0 ...  
## $ quarter.4 : int 0 0 0 0 0 0 0 1 0 1 ...  
## $ income : int 65268 58854 61115 45874 43547 41827 45874 66250 70963 61115 ...  
## $ bachelors\_degree : int 118 68 264 2488 502 169 2488 22 1959 264 ...  
## $ population : int 1003 734 2048 30327 10615 2379 30327 567 16052 2048 ...  
## $ state\_province.Alabama : int 0 0 0 0 1 0 0 0 1 0 ...  
## $ state\_province.Arkansas : int 0 0 0 1 0 0 1 0 0 0 ...  
## $ state\_province.Colorado : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ state\_province.Iowa : int 1 0 0 0 0 0 0 1 0 0 ...  
## $ state\_province.Minnesota : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ state\_province.Missouri : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ state\_province.Nebraska : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ state\_province.Oklahoma : int 0 0 1 0 0 1 0 0 0 1 ...  
## $ state\_province.South Dakota: int 0 1 0 0 0 0 0 0 0 0 ...  
## $ state\_province.Wyoming : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ num\_trans : int 1 1 1 1 1 1 1 1 1 1 ...  
## $ basket.no : int 1 1 1 1 1 1 1 1 1 1 ...  
## $ basket.yes : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ refill.no : int 1 1 1 1 1 1 1 1 1 1 ...  
## $ refill.yes : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ area.alcohol : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ area.cooler : int 0 0 1 0 0 0 0 0 0 0 ...  
## $ area.dispensed : int 0 0 0 0 0 1 0 0 0 0 ...  
## $ area.fresh : int 1 0 0 1 1 0 0 0 0 0 ...  
## $ area.fuel : int 0 1 0 0 0 0 1 1 0 1 ...  
## $ area.grocery : int 0 0 0 0 0 0 0 0 1 0 ...  
## $ area.lottery : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ area.miscellaneous : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ area.nongrocery : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ area.snacks : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ area.tobacco : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ items\_sold : num 1 1 1 1 3 1 1 1 1 1 ...  
## $ loyalty2.not loyal : int 1 1 1 1 1 1 1 1 1 1 ...  
## $ loyalty2.loyal : int 0 0 0 0 0 0 0 0 0 0 ...  
## - attr(\*, "dummies")=List of 6  
## ..$ quarter : int [1:4] 3 4 5 6  
## ..$ state\_province: int [1:10] 10 11 12 13 14 15 16 17 18 19  
## ..$ basket : int [1:2] 21 22  
## ..$ refill : int [1:2] 23 24  
## ..$ area : int [1:11] 25 26 27 28 29 30 31 32 33 34 ...  
## ..$ loyalty2 : int [1:2] 37 38

slice\_sample(knn\_input, n=10)

## high\_gpm revenue quarter.1 quarter.2 quarter.3 quarter.4 income  
## 136885 low 10.00 1 0 0 0 89196  
## 167781 low 41.02 0 1 0 0 52500  
## 313052 high 1.39 1 0 0 0 37130  
## 555424 high 1.00 0 0 0 1 50020  
## 1364432 low 31.50 0 1 0 0 62508  
## 499690 high 0.90 0 0 1 0 65861  
## 386152 high 1.00 1 0 0 0 45705  
## 495690 high 1.00 0 0 1 0 50473  
## 474543 high 1.65 1 0 0 0 34152  
## 375996 low 10.00 0 1 0 0 37917  
## bachelors\_degree population state\_province.Alabama  
## 136885 90 423 0  
## 167781 95 1192 0  
## 313052 1238 15153 0  
## 555424 1806 20238 1  
## 1364432 2139 17311 1  
## 499690 164 1730 0  
## 386152 136 1408 0  
## 495690 589 7344 1  
## 474543 32 1904 0  
## 375996 10 546 0  
## state\_province.Arkansas state\_province.Colorado state\_province.Iowa  
## 136885 0 0 0  
## 167781 0 0 0  
## 313052 0 1 0  
## 555424 0 0 0  
## 1364432 0 0 0  
## 499690 0 0 1  
## 386152 0 0 0  
## 495690 0 0 0  
## 474543 0 0 0  
## 375996 1 0 0  
## state\_province.Minnesota state\_province.Missouri  
## 136885 0 0  
## 167781 0 0  
## 313052 0 0  
## 555424 0 0  
## 1364432 0 0  
## 499690 0 0  
## 386152 0 0  
## 495690 0 0  
## 474543 0 1  
## 375996 0 0  
## state\_province.Nebraska state\_province.Oklahoma  
## 136885 0 1  
## 167781 1 0  
## 313052 0 0  
## 555424 0 0  
## 1364432 0 0  
## 499690 0 0  
## 386152 0 1  
## 495690 0 0  
## 474543 0 0  
## 375996 0 0  
## state\_province.South Dakota state\_province.Wyoming num\_trans basket.no  
## 136885 0 0 1 1  
## 167781 0 0 1 1  
## 313052 0 0 1 1  
## 555424 0 0 1 1  
## 1364432 0 0 1 1  
## 499690 0 0 1 1  
## 386152 0 0 1 1  
## 495690 0 0 1 1  
## 474543 0 0 1 1  
## 375996 0 0 1 1  
## basket.yes refill.no refill.yes area.alcohol area.cooler area.dispensed  
## 136885 0 1 0 0 0 0  
## 167781 0 1 0 0 0 0  
## 313052 0 1 0 0 0 1  
## 555424 0 1 0 0 0 1  
## 1364432 0 1 0 0 0 0  
## 499690 0 1 0 0 0 1  
## 386152 0 1 0 0 0 1  
## 495690 0 1 0 0 0 1  
## 474543 0 1 0 0 0 0  
## 375996 0 1 0 0 0 0  
## area.fresh area.fuel area.grocery area.lottery area.miscellaneous  
## 136885 0 1 0 0 0  
## 167781 0 1 0 0 0  
## 313052 0 0 0 0 0  
## 555424 0 0 0 0 0  
## 1364432 0 1 0 0 0  
## 499690 0 0 0 0 0  
## 386152 0 0 0 0 0  
## 495690 0 0 0 0 0  
## 474543 1 0 0 0 0  
## 375996 0 0 0 1 0  
## area.nongrocery area.snacks area.tobacco items\_sold loyalty2.not loyal  
## 136885 0 0 0 1 1  
## 167781 0 0 0 1 1  
## 313052 0 0 0 1 0  
## 555424 0 0 0 1 0  
## 1364432 0 0 0 1 1  
## 499690 0 0 0 1 0  
## 386152 0 0 0 1 0  
## 495690 0 0 0 1 0  
## 474543 0 0 0 2 0  
## 375996 0 0 0 1 1  
## loyalty2.loyal  
## 136885 0  
## 167781 0  
## 313052 1  
## 555424 1  
## 1364432 0  
## 499690 1  
## 386152 1  
## 495690 1  
## 474543 1  
## 375996 0

Let’s look more in depth at the target feature. For KNN analysis, the target feature is a categorical variable. In this implementation we can leave it as a factor. About 44% of these purchases are for high gross profit items.

# Explore the target feature

freq <- table(knn\_input$high\_gpm)  
freq[2]/(freq[1]+freq[2])

## high   
## 0.43695

contrasts(knn\_input$high\_gpm)

## high  
## low 0  
## high 1

Before using the algorithm, we need to prepare the data. The first line below loads the caret package. The next line sets the seed for the randomization that will be used for the algorithm. The caret::createDataPartition() function uses the caret package to split the data. Basically, it creates a number for p amount, in this case 0.75, of the target feature and lists these numbers in a matrix (since list is FALSE). Next, the training data and testing data are created. These use the numbers from the partition matrix that we just created. data\_train retains each of the rows with numbers in partition while data\_test takes the numbers not in partition, i.e., -partition.

# Partition the data

library(caret)

## Loading required package: lattice

##   
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':  
##   
## lift

set.seed(77)  
partition <- caret::createDataPartition(y=knn\_input$high\_gpm, p=.75, list=FALSE)  
data\_train <- knn\_input[partition, ]  
data\_test <- knn\_input[-partition, ]

Next, we need to remove the dependent variable, high\_gpm to create testing data and training data without the target variable, y and training and testing data that is just the dependent variable.

# Separate the target variable

X\_train <- data\_train %>% select(-high\_gpm)  
X\_test <- data\_test %>% select(-high\_gpm)   
y\_train <- data\_train$high\_gpm  
y\_test <- data\_test$high\_gpm

Next, let’s standardize our data. Recall, that KNN uses a distance function to determine its nearest neighbor. Thus, we need all of the variables to be on similar scales. This function creates matrices that have variables that have been standardized with z-score standardization. The mean of each feature is subtracted from each individual feature and then divided by that featureâ€™s standard deviation. This transformation rescales the feature such it has a mean of zero and a standard deviation of one. Thus, it is measured in how many standard deviations it falls above or below its mean.

# z-score standardization

X\_train <- scale(X\_train)  
X\_test <- scale(X\_test)

Let’s just double check that our training data is the correct percentage and that the size of the training input variables and the training output variables are the same.

# Double check sizes

nrow(X\_train)/(nrow(X\_test)+nrow(X\_train))

## [1] 0.75005

dim(X\_train)

## [1] 15001 37

length(y\_train)

## [1] 15001

Finally, let’s run the model. Please remember that this algorithm works differently from some of the others you will see. It is what we call lazy because it does not create a model, but rather stores all of the training data and then uses it on the test data all in one step. Thus, the training step and the prediction step are combined into one step.

To implement KNN we use one function that accepts four arguments: \* The training data with the label’s column/target variable removed \* The testing data with the label’s column/target variable removed \* The class labels (cl)/target variable for the training data (but not the testing data, since that is what is being predicted), and \* The K that we select.

How did I select this k? There is no one right way to do this, but I just took the square root of the total number of rows.

knn1 is a feature vector of predicted labels for each of the rows/examples in the test data.

# Run the model

library(class)  
knn1 = class::knn(train=X\_train, test=X\_test, cl=y\_train, k=141)

Finally, let’s check to see how accurate our model is. We will use the function created at the beginning of the notebook. Overall, our model is very accurate. The model makes the correct prediction about 83% of the time! Recall, that KNN has used the training data to predict on the testing data. Thus, there is some assurance that the model would predict well on new data.

Let’s explore some details of the accuracy of this model. The table shows the following output: \* When a transaction is actually/truthfully a high gross profit margin transaction, the model correctly classifies it as such-by saying “high”, 1700 times. This is called a True Positive (TP), Hit.

* When a transaction is truthfully a low gross profit margin transaction, the model correctly classifies it as such–by saying “low”, 2448 times. This is a True Negative (TN), Rejection.
* On the other hand, the model makes two kinds of errors. When the model transaction is not a high gross profit margin transaction, but the model incorrectly says it is, this is called a False Positive (FP), Type 1 Error. It happens 367 times.
* Finally, when a transaction is a high gross profit margin transaction and the model says it is not, which here happens 484, it is called a False Negative (FN), Type 2 Error.

These numbers can then be manipulated to create different measures of accuracy, as follows: \* Overall accuracy (TP+TN/(TP+TN+FP+FN)) is 0.8298.

* Sensitivity, Recall, Hit Rate, True Positive Rate (How many positives did the model get right? TP/(TP+FN)), is 0.7784.
* Specificity, Selectivity, True Negative Rate (How many negatives did the model get right? TN/(TN+FP)) is 0.8696.
* Precision, Positive Predictive Value (How good are the model’s positive predictions? TP/(TP+FP)) is 0.8224.
* Negative Predictive Value (How good are the model’s negative predictions? TN/(TN+FN) is 0.8349.

But what does this all tell us? Overall, our model does a great job at predicting when a purchase will be high profit margin versus low profit margin and gets it right 83% of the time. The aspect of the model that is the least effective is its sensitivity and the aspect that is the most effective is the specificity. This means that the models is very good at classifying the low margin purchases successfully (specificity), but slightly worse at classifying the high margin purchases correctly (sensitivity). Thus, while the model is quite good overall, it particularly excels at identifying bad transactions.

# Confusion matrix - checking accuracy

table2 <- table(knn1, y\_test) #prediction on left and truth on top  
my\_confusion\_matrix(table2)

## y\_test  
## knn1 low high  
## low 2448 484  
## high 367 1700

## [[1]]  
## [1] "1700 = True Positive (TP), Hit"  
##   
## [[2]]  
## [1] "2448 = True Negative (TN), Rejection"  
##   
## [[3]]  
## [1] "367 = False Positive (FP), Type 1 Error"  
##   
## [[4]]  
## [1] "484 = False Negative (FN), Type 2 Error"  
##   
## [[5]]  
## [1] "0.8298 = Accuracy (TP+TN/(TP+TN+FP+FN))"  
##   
## [[6]]  
## [1] "0.7784 = Sensitivity, Recall, Hit Rate, True Positive Rate (How many positives did the model get right? TP/(TP+FN))"  
##   
## [[7]]  
## [1] "0.8696 = Specificity, Selectivity, True Negative Rate (How many negatives did the model get right? TN/(TN+FP))"  
##   
## [[8]]  
## [1] "0.8224 = Precision, Positive Predictive Value (How good are the model's positive predictions? TP/(TP+FP))"  
##   
## [[9]]  
## [1] "0.8349 = Negative Predictive Value (How good are the model's negative predictions? TN/(TN+FN)"

The above confusion matrix was made from our function above. It is helpful because I put in some extra text that helps interpret the results. R, of course, has several packages that will build the confusion matrix for you. Here is one. Note, that we did need to specify which of our levels is the “Positive Class”, since this normally takes the first level of the variable, which is our case is low, which is the “negative” class, or the wrong thing, rather than the thing we are predicting. One thing that is provided here is a significance test of accuracy. As you can see from the very low P-Value and Mcnemar's Test P-Value, our model is significant.

# Pre-programmed confusion matrix

caret::confusionMatrix(knn1, y\_test, positive='high')

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction low high  
## low 2448 484  
## high 367 1700  
##   
## Accuracy : 0.8298   
## 95% CI : (0.8191, 0.8401)  
## No Information Rate : 0.5631   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.6519   
##   
## Mcnemar's Test P-Value : 6.996e-05   
##   
## Sensitivity : 0.7784   
## Specificity : 0.8696   
## Pos Pred Value : 0.8224   
## Neg Pred Value : 0.8349   
## Prevalence : 0.4369   
## Detection Rate : 0.3401   
## Detection Prevalence : 0.4135   
## Balanced Accuracy : 0.8240   
##   
## 'Positive' Class : high   
##

Finally, we can put the prediction back into the test data and compare when our model is and is not accurate. The first line of the code below adds the predicted low or high gross profit margin back to the test data. The second line creates a new variable called correct that takes on the value TRUE when the model was correct and FALSE when the model was incorrect. Printing out a sample of this dataframe and scrolling through it gives us the ability to investigate where the model went wrong.

It also gives us some insight into TECA’s problem. Just doing a quick scan, it seems like promoting certain areas, such as fresh and dispensed might help increase the sale of high profit margin products, while other areas, such as lottery, might not help. However, are efforts here were to make a prediction model. To examine the relevance of the factors we would need to employee another method, such as logistic regression or decision trees.

# Evaluate the data

data\_test$prediction <- knn1  
data\_test <- data\_test %>% mutate(correct = high\_gpm==prediction)  
slice\_sample(data\_test, n=20)

## high\_gpm revenue quarter.1 quarter.2 quarter.3 quarter.4 income  
## 209931 high 1.79 0 0 1 0 34210  
## 1403545 low 26.91 0 0 0 1 29091  
## 1226812 low 10.00 1 0 0 0 42772  
## 428400 low 5.00 1 0 0 0 66250  
## 417907 low 21.00 1 0 0 0 76071  
## 38464 high 0.99 0 1 0 0 52500  
## 1612610 high 1.00 0 0 0 1 29091  
## 1224527 low 45.25 0 0 0 1 70557  
## 879815 high 1.98 0 0 1 0 66160  
## 318760 low 7.03 0 1 0 0 37130  
## 1242817 low 26.90 0 0 0 1 50819  
## 382899 high 1.69 0 0 1 0 66250  
## 118951 low 13.98 1 0 0 0 41827  
## 855741 low 4.75 0 1 0 0 70963  
## 1551687 high 2.69 0 0 1 0 66250  
## 1201431 low 20.00 0 0 0 1 51461  
## 471531 high 7.56 0 0 1 0 63673  
## 15110 low 2.13 0 0 0 1 53026  
## 1555083 high 1.30 1 0 0 0 132230  
## 417764 low 44.64 0 0 1 0 73148  
## bachelors\_degree population state\_province.Alabama  
## 209931 1632 25999 1  
## 1403545 3 181 0  
## 1226812 160 1876 0  
## 428400 22 567 0  
## 417907 93 493 0  
## 38464 95 1192 0  
## 1612610 3 181 0  
## 1224527 4048 51863 0  
## 879815 10178 57364 0  
## 318760 1238 15153 0  
## 1242817 958 8778 0  
## 382899 22 567 0  
## 118951 169 2379 0  
## 855741 1959 16052 1  
## 1551687 22 567 0  
## 1201431 2922 52172 0  
## 471531 1172 9046 0  
## 15110 19 557 0  
## 1555083 974 3294 0  
## 417764 139 852 0  
## state\_province.Arkansas state\_province.Colorado state\_province.Iowa  
## 209931 0 0 0  
## 1403545 0 0 0  
## 1226812 0 0 0  
## 428400 0 0 1  
## 417907 0 0 0  
## 38464 0 0 0  
## 1612610 0 0 0  
## 1224527 0 1 0  
## 879815 0 0 0  
## 318760 0 1 0  
## 1242817 0 0 0  
## 382899 0 0 1  
## 118951 0 0 0  
## 855741 0 0 0  
## 1551687 0 0 1  
## 1201431 0 1 0  
## 471531 0 0 1  
## 15110 0 0 1  
## 1555083 1 0 0  
## 417764 0 0 1  
## state\_province.Minnesota state\_province.Missouri  
## 209931 0 0  
## 1403545 0 0  
## 1226812 0 0  
## 428400 0 0  
## 417907 0 0  
## 38464 0 0  
## 1612610 0 0  
## 1224527 0 0  
## 879815 0 1  
## 318760 0 0  
## 1242817 1 0  
## 382899 0 0  
## 118951 0 0  
## 855741 0 0  
## 1551687 0 0  
## 1201431 0 0  
## 471531 0 0  
## 15110 0 0  
## 1555083 0 0  
## 417764 0 0  
## state\_province.Nebraska state\_province.Oklahoma  
## 209931 0 0  
## 1403545 0 1  
## 1226812 0 1  
## 428400 0 0  
## 417907 1 0  
## 38464 1 0  
## 1612610 0 1  
## 1224527 0 0  
## 879815 0 0  
## 318760 0 0  
## 1242817 0 0  
## 382899 0 0  
## 118951 0 1  
## 855741 0 0  
## 1551687 0 0  
## 1201431 0 0  
## 471531 0 0  
## 15110 0 0  
## 1555083 0 0  
## 417764 0 0  
## state\_province.South Dakota state\_province.Wyoming num\_trans basket.no  
## 209931 0 0 1 1  
## 1403545 0 0 1 1  
## 1226812 0 0 1 1  
## 428400 0 0 1 1  
## 417907 0 0 1 1  
## 38464 0 0 1 1  
## 1612610 0 0 1 1  
## 1224527 0 0 1 1  
## 879815 0 0 1 1  
## 318760 0 0 1 1  
## 1242817 0 0 1 1  
## 382899 0 0 1 1  
## 118951 0 0 1 1  
## 855741 0 0 1 1  
## 1551687 0 0 1 1  
## 1201431 0 0 1 1  
## 471531 0 0 1 1  
## 15110 0 0 1 1  
## 1555083 0 0 1 1  
## 417764 0 0 1 1  
## basket.yes refill.no refill.yes area.alcohol area.cooler area.dispensed  
## 209931 0 1 0 0 0 0  
## 1403545 0 1 0 0 0 0  
## 1226812 0 1 0 0 0 0  
## 428400 0 1 0 0 0 0  
## 417907 0 1 0 0 0 0  
## 38464 0 1 0 0 1 0  
## 1612610 0 1 0 0 0 1  
## 1224527 0 1 0 0 0 0  
## 879815 0 1 0 0 0 0  
## 318760 0 1 0 0 0 0  
## 1242817 0 1 0 0 0 0  
## 382899 0 1 0 0 0 1  
## 118951 0 1 0 0 0 0  
## 855741 0 1 0 0 0 0  
## 1551687 0 1 0 0 0 0  
## 1201431 0 1 0 0 0 0  
## 471531 0 1 0 0 1 0  
## 15110 0 1 0 0 1 0  
## 1555083 0 1 0 0 0 0  
## 417764 0 1 0 0 0 0  
## area.fresh area.fuel area.grocery area.lottery area.miscellaneous  
## 209931 1 0 0 0 0  
## 1403545 0 1 0 0 0  
## 1226812 0 1 0 0 0  
## 428400 0 1 0 0 0  
## 417907 0 1 0 0 0  
## 38464 0 0 0 0 0  
## 1612610 0 0 0 0 0  
## 1224527 0 1 0 0 0  
## 879815 0 0 0 0 0  
## 318760 0 0 0 0 0  
## 1242817 0 1 0 0 0  
## 382899 0 0 0 0 0  
## 118951 0 0 0 0 0  
## 855741 0 0 0 0 0  
## 1551687 1 0 0 0 0  
## 1201431 0 1 0 0 0  
## 471531 0 0 0 0 0  
## 15110 0 0 0 0 0  
## 1555083 1 0 0 0 0  
## 417764 0 1 0 0 0  
## area.nongrocery area.snacks area.tobacco items\_sold loyalty2.not loyal  
## 209931 0 0 0 1 1  
## 1403545 0 0 0 1 1  
## 1226812 0 0 0 1 1  
## 428400 0 0 0 1 1  
## 417907 0 0 0 1 0  
## 38464 0 0 0 1 0  
## 1612610 0 0 0 1 1  
## 1224527 0 0 0 1 1  
## 879815 0 1 0 2 1  
## 318760 0 0 1 1 0  
## 1242817 0 0 0 1 1  
## 382899 0 0 0 1 0  
## 118951 0 0 1 2 1  
## 855741 0 0 1 1 1  
## 1551687 0 0 0 2 1  
## 1201431 0 0 0 1 1  
## 471531 0 0 0 4 1  
## 15110 0 0 0 1 0  
## 1555083 0 0 0 1 1  
## 417764 0 0 0 1 0  
## loyalty2.loyal prediction correct  
## 209931 0 high TRUE  
## 1403545 0 low TRUE  
## 1226812 0 low TRUE  
## 428400 0 low TRUE  
## 417907 1 low TRUE  
## 38464 1 low FALSE  
## 1612610 0 high TRUE  
## 1224527 0 low TRUE  
## 879815 0 high TRUE  
## 318760 1 low TRUE  
## 1242817 0 low TRUE  
## 382899 1 high TRUE  
## 118951 0 low TRUE  
## 855741 0 low TRUE  
## 1551687 0 high TRUE  
## 1201431 0 low TRUE  
## 471531 0 low FALSE  
## 15110 1 low TRUE  
## 1555083 0 high TRUE  
## 417764 1 low TRUE

Finally, note that we are not covering advanced topics such as hyperparameter tuning. When you gain more experience, you might want to examine methods for picking k, cross validation, etc. Our goal is to provide you the framework to understand the algorithm and start working with it.