Title: Logistic Regression

Business Analytics Module: Author: Reuben Kariuki

****

Let's get some hands-on experience using logistic regression. Logistic regression is a valuable classification-type machine learning algorithm. In this project, we will use the algorithm to predict whether a transaction is from a loyalty customer or not. We will use the TECA dataset.

TECA desires to increase their loyalty customers and the number of products a loyalty customer buys, because loyalty customers create more business and more profitable business for TECA.

First, let's bring in the data and the needed packages.

Bring in packages and data

```
library(tidyverse)

df1_even_kp <-readRDS("C:/Users/User/OneDrive/Documents/projects/logistic1.rds")
```

Next, run the following lines of code. In these lines I have created a very basic function that will help us evaluate the quality of our logistic regression model, later on.

We will use this code a few times, so, it made sense to put it into a function that can be reused, without having to cut and paste all of this code every time we want to use it.

Create function for confusion matrix, to be used later

```
# Make reusable 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_negativ
e + 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_negat
ive),
                  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 Positi
ve Rate (How many positives did the model get right? TP/(TP+FN))", sensitivit
y_recall),
                  sprintf("%1.4f = Specificity, Selectivity, True Negative Ra
te (How many negatives did the model get right? TN/(TN+FP))", specificity_sel
ectivity),
                  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 th
e model's negative predictions? TN/(TN+FN)", neg_pred_value)
  )
  return(my_list)
}
```

Now, let's go through our six steps of for using a machine learning algorithm. Recall that these steps are the following:

1. **Data scrubbing.**
2. **Algorithm selection.**
3. **Model training or fitting.**
4. **Model evaluation.**
5. **Model improvement.**
6. **Deploy the model**

We have done all of the data scrubbing, so number 1 is already complete. For number 2, we have selected logistic regression as our classification algorithm. So, let's move to number 3.

We will break this step down into multiple parts. Our goal is to train a univariate logistic regression model using our data. That is, we will predict whether a particular transaction is from a loyalty customer or not.

First, we need to select which independent/input variable we are going to use to predict out dependent variable. Let's look at the data to remind ourselves what it looks like.

View data

```
slice_sample(df1_even_kp, n=10)
```

```
##           loyalty                 category quarter    state
## 1565972 not loyal Cold Dispensed Beverage       4  Alabama
## 369220      loyal                     Fuel       3  Alabama
## 174459      loyal    Breakfast Sandwiches       3  Alabama
## 837593  not loyal                   Bakery       1 Oklahoma
## 39243       loyal                Pop (587)       3 Arkansas
## 600962  not loyal               Cigarettes       2 Missouri
## 70732       loyal                   Bakery       1     Iowa
## 238928      loyal                   Cigars       3  Alabama
## 700720  not loyal             Salty Snacks       3     Iowa
## 821186  not loyal                     Fuel       3  Alabama
```

`loyalty` is our dependent/output/target variable. We need to pick one of the other three to be our independent/input variable. Since TECA is concerned about predicting which transactions will be loyalty transactions, it probably makes the most sense to use `category` as our independent variable.

Before we train our model, we have a few more things to do to get ready.

First, let's look at the baseline percentage of the occurrence of `loyalty`. First, we will use the function `table()`. This function uses the levels of categorical variables to build a contingency table of the counts at each combination of variable levels. We would like to build a table of how many loyal and no loyal transactions we have and then calculate a percentage of loyal transactions.

Let's describe what is done here. First, the `table()` function is used to create a table from the `loyalty` variable. Second, we printed that table. Next, we printed the percentage of loyal transactions divided by the total transactions. This is done by indexing the table and using indexing to select individual parts of the table with the square brackets `[]`. Thus, for example, `loyal_table[2]` = 519744.

Baseline occurrence of `loyalty`

```
loyal_table <- table(df1_even_kp$loyalty)
print(loyal_table)
```

```
##
## not loyal     loyal
##    520000    519744
```

```
print(loyal_table[2]/(loyal_table[1]+loyal_table[2]))
```

```
##     loyal
## 0.4998769
```

Next, I want to double check that the levels of `loyalty` are in the order I think they are. Logistic regression and my confusion matrix function are going to use factors in the order they are in, so I need to understand what that order is so that I can interpret the output of the model correctly. The `contrasts()` function will check that for me. I would like loyal to be coded as 1.

```
contrasts(df1_even_kp$loyalty)
```

```
##           loyal
## not loyal     0
## loyal         1
```

This is correct, so we can move on to our next step. The next step in training our data is to split it into training and testing proportions. Recall, that we want to train our model on one set of data, but then make sure it works on other data, so that we are not overfitting our model. I will use the `caret` package to split the data, though there are myriad ways of doing this. However, as you continue to explore and work on classification models it is likely you will use the `caret` package.

The second line sets a seed so we can get the same split again. The next line uses the `caret` package to select 75% of the entries in the column `loyalty`. We could have used a different percentage–like 80% or 70%. This creates a matrix of numbers that we can uses to select the rows from our dataset that will become the training data. That is what we have done in the fourth line. Here, we create a dataset called `data_train` by taking the original dataset and telling it to only keep the rows we randomly selected in `partition`. The square brackets here are again using indexing to tell R which rows to take. We put nothing after the comma, which tells R to take all of the columns. We could have instead selected only some subset of the columns. Next, we select the opposite columns, using the `-` symbol in front of `partition` to make the testing dataset, `data_test`. Finally, we print the percentage of rows that are in the training data to make sure it is close to 75%.

Split data

```
library(caret)

set.seed(77)
partition <- caret::createDataPartition(y=df1_even_kp$loyalty, p=.75, list=FA
LSE)
data_train <- df1_even_kp[partition,]
data_test <- df1_even_kp[-partition,]
print(nrow(data_train)/(nrow(data_test)+nrow(data_train)))
```

```
## [1] 0.75
```

With our data split, we are ready to train our model. Recall that an algorithm is a set of rules for how to make a model. A model takes our data and options

we select to create new data/information and additional rules for how to use the model. Below, we create our model, called `model_train` and then summarize it to view the output. We use the `glm` function with the `family=binomial` argument to select logistic regression. The syntax is really simple here. Our dependent variable, `loyalty` goes first. Then we use ~ to act like an equals sign followed by our independent variables, `category` in this case. Finally, we list the data set. We are, of course, using the training data, since we are training the model.

There is a lot of output here, so let's pick out only the important stuff. Let's focus on the Coefficients area. There we have coefficients listed for out variable under the Estimate column. A ways over we have the z value and the p value (Pr(>|z|)). What does all of this mean? The coefficients tell us the effect of that variable on the dependent variable, `loyalty`. The `glm` function is smart enough to know that `category` is a categorical variable and not a continuous variable. Thus, `glm` automatically did something called "one-hot encoding" or, equivalently, "dummy coding", to our variable. That is, it took the variable and essentially created individual "dummy variables" for each level of `category`. Thus, instead of having on `category` variable, we now have 19 dummy variables for each level. Thus, `categoryBeer` is a new variable that has a 1 if that row is the level Beer and a 0 otherwise. That is an example of a dummy variable. Likewise, `categoryPizza` has a 1 if that level row has pizza in the `category` column and 0 otherwise. Hopefully you notice an issue, however. We have 20 levels of `category` but only 19 dummy variable coefficients. That is because `glm` put one of our levels into the intercept term. Specifically, Bakery is in the intercept. This is done to avoid over-specifying the model. It is important to know which level is not in the model because it affects how we interpret the results. Specifically, the effect of the other coefficients is in reference to the variable in the intercept.

We can now interpret. The intercept lists the effect of bakery items on whether a transaction is from a loyalty customer or not. The coefficient is positive, which means that selling a bakery item increases the likelihood that a transaction is a loyalty transaction. More precisely, purchasing this item increases the log odds of a loyalty purchase happening by 0.30575. We won't go into log odds, but it suffices to say that buying a bakery item increases that likelihood of the transaction being from a loyalty customer. Let's look at some more categories. Fuel actually decreases the chance of the transaction being from a loyalty customer relative to a bakery item, as can be seen by its negative coefficient. More specifically, the coefficient is still negative even relative to the intercept, which we remember is Bakery items. That is, Fuel's coefficient plus the intercept is still negative meaning Fuel decreases the chance that a loyalty customer makes a purchase. This makes sense since a bakery item is one you go into the store for, which is more likely to happen for a loyalty customer, whereas anyone can drive by and purchase gas. Next, fountain soda, Cold Dispensed Beverage, increases the chance the purchase was from a loyalty customer, even more than purchasing a bakery item. Thus, fountain soda seems to be a big draw for loyalty customers and could be a way TECA could bring increase loyalty purchases. Finally, what effect do Bread and Cakes have? This is a small negative coefficient. If we add it to the intercept, we see that the difference is positive. Thus, Bread and Cakes have a significant

effect on the chance a purchase is a loyalty purchase. We have to compare to the intercept, though, since all of the coefficients are relative to Bakery.

Next, it is not enough to see what the sign of the coefficient is, but we need to see whether the coefficient has a significant effect on `loyalty`. We see this in at least two ways. First, we can look at the p-level (Pr(>|z|)). This is the probability that the coefficient is equal to 0. This is not the technical definition, but the point to remember is that a low p-value means the coefficient is not 0 and is statistically significant. Additionally, *** indicate significance, with more stars meaning it is less likely the coefficient is 0. We have a lot of data here so even small coefficients are likely to be significant. Sure enough, only energy drinks and Hot Sandwiches & Chicken are not significant relative to bakery. So, overall, what do we learn here? We know now that several categories, such as fountain sodas, bakery items, and breakfast sandwiches help increase the chance a customer will be a loyalty customer. TECA could promote these products to try to draw in more loyalty customers.

Train logistic regression model

```
model_train <- glm(loyalty ~ category, family=binomial, data=data_train)
summary(model_train)

##
## Call:
## glm(formula = loyalty ~ category, family = binomial, data = data_train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.5152  -1.1350  -0.8436   1.1812   1.5530
##
## Coefficients:
##                                  Estimate Std. Error z value Pr(>|z|)
## (Intercept)                       0.30575    0.01539  19.870  < 2e-16 ***
## categoryBeer                     -1.15590    0.02101 -55.009  < 2e-16 ***
## categoryBread & Cakes            -0.11000    0.02173  -5.063 4.14e-07 ***
## categoryBreakfast Sandwiches      0.22179    0.02156  10.288  < 2e-16 ***
## categoryCandy/Gum                -0.33491    0.01816 -18.442  < 2e-16 ***
## categoryChips                    -0.20758    0.02392  -8.677  < 2e-16 ***
## categoryCigarettes               -0.48665    0.01769 -27.517  < 2e-16 ***
## categoryCigars                   -0.95134    0.02544 -37.395  < 2e-16 ***
## categoryCold Dispensed Beverage   0.46045    0.01706  26.991  < 2e-16 ***
## categoryEnergy                    0.02773    0.01797   1.544  0.12269
## categoryFuel                     -0.63600    0.01622 -39.211  < 2e-16 ***
## categoryHot Dispensed Beverage   -0.01130    0.01866  -0.605  0.54487
## categoryHot Sandwiches & Chicken -0.06145    0.02298  -2.674  0.00749 **
## categoryJuice/tonics             -0.40630    0.01752 -23.196  < 2e-16 ***
## categoryLottery                  -0.91991    0.01865 -49.313  < 2e-16 ***
## categoryPizza                     0.10643    0.02130   4.996 5.84e-07 ***
## categoryPop (587)                -0.31478    0.01706 -18.448  < 2e-16 ***
## categoryRoller Grill             -0.18406    0.02183  -8.430  < 2e-16 ***
## categorySalty Snacks             -0.28467    0.01997 -14.252  < 2e-16 ***
```

```
## categorySmokeless (951)          -0.67115    0.02315 -28.989  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1081043  on 779807  degrees of freedom
## Residual deviance: 1050668  on 779788  degrees of freedom
## AIC: 1050708
##
## Number of Fisher Scoring iterations: 4
```

Our next step is to evaluate the model we have created by predicting the probabilities of each row and examining how accurate the model is at predicting a loyalty purchase. Recall that logistic regression uses the independent variables to create a probability for each row that maximized that likelihood that that row is close to its true category–`loyal` or `not loyal`. We should predict those probabilities and examine their accuracy, since we have labeled data and know the true category for each row.

First, let's predict that probabilities. We really want to do this with the testing data, but we will do this on the training data first, and then compare to the real data, the testing data, next.

Let's examine this code line by line. The first line is the prediction of the probabilities. We are predicting using the model we just created, `model_train`. Next, we are using the training data, `data_train`. The argument `type='response'` gives the predicted probabilities rather than the log odds, which are more difficult to interpret. Next, we print a summary of the probabilities, just so we can get a sense of what they look like. To further evaluate the model, we take the probabilities, `predict_train`, and put them into the training data as a new column, `data_train$prediction`. Finally, we take a look at a snapshot of the data. You can see that the probabilities here make sense, generally. All of these first 10 rows are from loyalty customers, and they are all relatively high, considering our highest probability is only 68%.

Predict the probabilities of each row on the training data

```
predict_train <- predict(model_train, newdata=data_train, type='response')
print(summary(predict_train))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.2994  0.4182  0.4927  0.4999  0.5731  0.6827
```

```
data_train$prediction <- predict_train
head(data_train, n=10)
```

```
##    loyalty      category quarter    state prediction
## 1    loyal Bread & Cakes       3 Colorado  0.5487819
```

```
## 2    loyal     Candy/Gum       1  Wyoming  0.4927105
## 3    loyal     Pop (587)       3  Wyoming  0.4977430
## 4    loyal     Pop (587)       4 Colorado  0.4977430
## 5    loyal         Chips       1 Oklahoma  0.5245229
## 6    loyal  Juice/tonics       2 Colorado  0.4748839
## 7    loyal  Juice/tonics       3  Wyoming  0.4748839
## 9    loyal          Beer       3  Alabama  0.2994030
## 10   loyal         Chips       2 Oklahoma  0.5245229
## 11   loyal    Cigarettes       1 Nebraska  0.4548983
```

Next, let's evaluate our model in another way, by looking at how accurate the model is at making correct predictions. There are a lot of ways to think about accuracy of this model. You can think of how accurate the model was a detecting loyalty, but you can also think about how accurate the model was at detecting not loyalty. Both of these types of accuracy to us, since we want an overall accurate model. A typical way to visualize accuracy is by using a confusion or classification matrix. Again, we are using the training data just so we can compare to the testing data next.

Let's again look at this code line by line. The first line creates a table using the `table` function. This table creates the probability predictions from above, `predict_train`, and writes "FALSE" if the probability is less than 0.5 and "TRUE" if it is more than 0.5. These are our predictions using the training data. You can see the FALSE and TRUE categories on the left of the table. It then compares these to the truth, which is the `loyalty` column from the `data_train` column. The second line then runs the functions we pasted above. This function just takes the four cells of this table and labels them and manipulates them to create various types of accuracy, displayed below.

Let's explore all of this. The table shows the following output: * When a transaction is actually/truthfully a loyalty transaction, the model correctly classifies it as such-by saying "TRUE", 185131 times. This is called a True Positive (TP), Hit.

- When a transaction is truthfully a non-loyalty transaction, the model correctly classifies it as such–by saying "FALSE", 265829 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 loyalty transaction but the model incorrectly says it is, this is called a False Positive (FP), Type 1 Error. It happens 124171 times.
- Finally, when a transaction is a loyalty transaction and the model says it is not, which here happens 204677, 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 0.578296.

- Sensitivity, Recall, Hit Rate, True Positive Rate (How many positives did the model get right? TP/(TP+FN)), is 0.474929.
- Specificity, Selectivity, True Negative Rate (How many negatives did the model get right? TN/(TN+FP)) is 0.681613.
- Precision, Positive Predictive Value (How good are the model's positive predictions? TP/(TP+FP)) is 0.598544.
- Negative Predictive Value (How good are the model's negative predictions? TN/(TN+FN) is 0.564985.

Thus, overall, our model predicts better than chance. Recall, that loyalty transactions happen about 50% of the time. Thus, if we had no model at all and just flipped a coin every time to guess if a transaction was going to be a loyalty transaction, we would get it right 50% of the time. Our model helps us to get it right about 58% of the time, which is not super high, but is better than chance. Where the model really fails right now is in sensitivity. That is, the model is pretty good at predicting `not loyal`, 68%, but bad at predicting the loyal, getting only 47% correct: worse than flipping a coin. So, our next step is to improve the model.

Confusion/classification matrix for training data

```
table1 <- table(predict_train>0.5, data_train$loyalty) #prediction on left and truth on top
my_confusion_matrix(table1)
```

```
##
##         not loyal  loyal
##   FALSE    265829 204677
##   TRUE     124171 185131

## [[1]]
## [1] "185131 = True Positive (TP), Hit"
##
## [[2]]
## [1] "265829 = True Negative (TN), Rejection"
##
## [[3]]
## [1] "124171 = False Positive (FP), Type 1 Error"
##
## [[4]]
## [1] "204677 = False Negative (FN), Type 2 Error"
##
## [[5]]
## [1] "0.5783 = Accuracy (TP+TN/(TP+TN+FP+FN))"
##
```

```
## [[6]]
## [1] "0.4749 = Sensitivity, Recall, Hit Rate, True Positive Rate (How many
positives did the model get right? TP/(TP+FN))"
##
## [[7]]
## [1] "0.6816 = Specificity, Selectivity, True Negative Rate (How many negat
ives did the model get right? TN/(TN+FP))"
##
## [[8]]
## [1] "0.5985 = Precision, Positive Predictive Value (How good are the model
's positive predictions? TP/(TP+FP))"
##
## [[9]]
## [1] "0.5650 = Negative Predictive Value (How good are the model's negative
predictions? TN/(TN+FN)"
```

However, first, let's take our trained model and predict/run it on the testing data. Recall, that we train on the training data and test on the testing data. We want to use new data to test on so that we aren't just trying to memorize the training data, but rather are creating a model that can work on any data.

This, of course, looks very similar to the code above with the **key** exception that the data being used is the test data, `data_test`.

How does our model perform on the new data? It actually does a little bit better than the training data. While accuracy is very similar, our problem from before, the low sensitivity, is a tiny bit improved.

Predict and evaluate the model on the test data

```
predict_test <- predict(model_train, newdata=data_test, type='response')
print(summary(predict_test))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.2994  0.4182  0.4927  0.5003  0.5731  0.6827
```

```
data_test$prediction <- predict_test
head(data_test, n=10)
```

```
##    loyalty               category quarter    state prediction
## 8    loyal          Bread & Cakes       2  Alabama  0.5487819
## 13   loyal              Pop (587)       4     Iowa  0.4977430
## 16   loyal           Salty Snacks       3 Colorado  0.5052708
## 20   loyal Cold Dispensed Beverage       1 Missouri  0.6826999
## 21   loyal                   Fuel       2  Alabama  0.4181812
## 24   loyal        Smokeless (951)       1  Wyoming  0.4096534
## 26   loyal           Juice/tonics       4  Wyoming  0.4748839
## 34   loyal              Candy/Gum       1 Oklahoma  0.4927105
```

```
## 36   loyal            Juice/tonics       1  Alabama  0.4748839
## 37   loyal                  Energy       4 Oklahoma  0.5826080
```

```
table2 <- table(predict_test>.5, data_test$loyalty) #prediction on left and t
ruth on top
my_confusion_matrix(table2)
```

```
##
##         not loyal loyal
##   FALSE     88248 67847
##   TRUE      41752 62089
```

```
## [[1]]
## [1] "62089 = True Positive (TP), Hit"
##
## [[2]]
## [1] "88248 = True Negative (TN), Rejection"
##
## [[3]]
## [1] "41752 = False Positive (FP), Type 1 Error"
##
## [[4]]
## [1] "67847 = False Negative (FN), Type 2 Error"
##
## [[5]]
## [1] "0.5784 = Accuracy (TP+TN/(TP+TN+FP+FN))"
##
## [[6]]
## [1] "0.4778 = Sensitivity, Recall, Hit Rate, True Positive Rate (How many
positives did the model get right? TP/(TP+FN))"
##
## [[7]]
## [1] "0.6788 = Specificity, Selectivity, True Negative Rate (How many negat
ives did the model get right? TN/(TN+FP))"
##
## [[8]]
## [1] "0.5979 = Precision, Positive Predictive Value (How good are the model
's positive predictions? TP/(TP+FP))"
##
## [[9]]
## [1] "0.5653 = Negative Predictive Value (How good are the model's negative
predictions? TN/(TN+FN)"
```

Last time, we predicted loyalty using a single variable, `category`. This time we will use all of our variables. We will see if this improves the prediction accuracy of our model.

- Of our six steps of for using a machine learning algorithm, we are now focused on step 5. 1.

- Data scrubbing.
- Algorithm selection.
- Model training or fitting.
- Model evaluation.
- Model improvement.
- Deploy the model

With our data split up above, we are ready to train our new model. Recall that an algorithm is a set of rules for how to make a model. A model takes our data and options we select to create new data/information and additional rules for how to use the model. Below, we create our model, called `model_train` and then summarize it to view the output. We use the `glm` function with the `family=binomial` argument to select logistic regression. Our dependent variable will continue to be `loyalty`, but this time we will include all of our variables in the model. We are, of course, using the training data since we are training the model.

Last time we examined the impact of `category` on `loyalty`. Let's look at `quarter`. The quarter is just the quarter of the year. I make it a factor so that R knows it is a category. There is some debate about whether we could leave that as a continuous variable, since the difference between one quarter of the year and another does have meaning, but we will leave it as a factor. The first quarter of the year is the reference level of these multiple dummy variables. Thus, when we evaluate the effects of these quarters it is in relationship to the effect of quarter 1. Looking at the signs of the coefficients and the p-values, we see that each quarter significantly increases the probability that a transaction will be a loyalty transaction. For example, the third quarter of the year has a positive coefficient of 0.216750 and a very low p-value. This means that relative to quarter 1, quarter 3 has more transactions that are from loyalty customers. With this knowledge, TECA could consider focusing on the beginning of the year in an effort to increase the number of their loyalty customers. Additionally, TECA could focus on summer and fall as key times when loyalty customers are active to promote to them and increases their purchases even more.

Next, let's look at how each state affects the occurrence of loyalty transactions. The reference state is Alabama. Relative to Alabama, the states that are most likely to generate loyalty transactions are Colorado, Missouri, and Wyoming. Missouri, in particular has a strong effect, as can be seen by their relatively large coefficient. TECA should investigate key stores in this state to see what is going right.

Train a multivariate model

```
model_train <- glm(loyalty ~ category + factor(quarter) + state, family=binom
ial, data=data_train)
summary(model_train)
```

```
##
## Call:
## glm(formula = loyalty ~ category + factor(quarter) + state, family = binom
ial,
##     data = data_train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.7136  -1.1285  -0.6539   1.1315   1.8153
##
## Coefficients:
##                                  Estimate Std. Error z value Pr(>|z|)
## (Intercept)                      0.254291   0.016633  15.289  < 2e-16 ***
## categoryBeer                    -1.096064   0.021205 -51.689  < 2e-16 ***
## categoryBread & Cakes           -0.114865   0.021934  -5.237 1.63e-07 ***
## categoryBreakfast Sandwiches     0.220034   0.021724  10.129  < 2e-16 ***
## categoryCandy/Gum               -0.341969   0.018316 -18.670  < 2e-16 ***
## categoryChips                   -0.204701   0.024131  -8.483  < 2e-16 ***
## categoryCigarettes              -0.511199   0.017849 -28.641  < 2e-16 ***
## categoryCigars                  -0.966302   0.025690 -37.614  < 2e-16 ***
## categoryCold Dispensed Beverage  0.447609   0.017227  25.982  < 2e-16 ***
## categoryEnergy                   0.015697   0.018120   0.866  0.38633
## categoryFuel                    -0.638023   0.016359 -39.002  < 2e-16 ***
## categoryHot Dispensed Beverage  -0.009300   0.018821  -0.494  0.62122
## categoryHot Sandwiches & Chicken -0.059721   0.023156  -2.579  0.00991 **
## categoryJuice/tonics            -0.433122   0.017665 -24.519  < 2e-16 ***
## categoryLottery                 -0.991808   0.018851 -52.612  < 2e-16 ***
## categoryPizza                    0.096907   0.021468   4.514 6.36e-06 ***
## categoryPop (587)               -0.279395   0.017215 -16.229  < 2e-16 ***
## categoryRoller Grill            -0.192345   0.022021  -8.735  < 2e-16 ***
## categorySalty Snacks            -0.281628   0.020151 -13.976  < 2e-16 ***
## categorySmokeless (951)         -0.691339   0.023350 -29.608  < 2e-16 ***
## factor(quarter)2                 0.194020   0.006887  28.170  < 2e-16 ***
## factor(quarter)3                 0.216750   0.006653  32.580  < 2e-16 ***
## factor(quarter)4                 0.228426   0.006624  34.486  < 2e-16 ***
## stateArkansas                   -0.341681   0.008364 -40.851  < 2e-16 ***
## stateColorado                    0.116604   0.007972  14.627  < 2e-16 ***
## stateIowa                       -0.285021   0.007312 -38.980  < 2e-16 ***
## stateMinnesota                  -0.285307   0.030645  -9.310  < 2e-16 ***
## stateMissouri                    0.276047   0.008295  33.281  < 2e-16 ***
## stateNebraska                   -0.182782   0.010315 -17.720  < 2e-16 ***
## stateOklahoma                   -0.592161   0.009407 -62.947  < 2e-16 ***
## stateSouth Dakota               -0.165378   0.021925  -7.543 4.60e-14 ***
## stateWyoming                     0.085997   0.013440   6.399 1.57e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1081043  on 779807  degrees of freedom
```

```
## Residual deviance: 1037616  on 779776  degrees of freedom
## AIC: 1037680
##
## Number of Fisher Scoring iterations: 4
```

Finally, let's examine our larger, more complicated model on our test data. Recall that our single-variable model has a relatively modest accuracy rate of about 57%. The more complex model has increased that accuracy slightly to about 60%. Perhaps more importantly, the sensitivity of the model is much improved. Recall that before, the sensitivity, or the accuracy of the model at predicting `not loyal`, was below 50%. This new model has increased that to 57%.

Overall, this is not a great model. We could add additional variables to make it more complex. Additionally, recall that this data is just a subset of the available data. Adding data could improve the model as well.

Predict and evaluate with test data

```
predict_test <- predict(model_train, newdata=data_test, type='response')
summary(predict_test)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.1925  0.4148  0.4903  0.5000  0.5788  0.7697
```

```
data_test$prediction <- predict_test
head(data_test, n=10)
```

```
##    loyalty                category quarter    state prediction
## 8    loyal           Bread & Cakes       2  Alabama  0.5825976
## 13   loyal               Pop (587)       4     Iowa  0.4795867
## 16   loyal            Salty Snacks       3 Colorado  0.5759126
## 20   loyal Cold Dispensed Beverage       1 Missouri  0.7267008
## 21   loyal                    Fuel       2  Alabama  0.4527136
## 24   loyal         Smokeless (951)       1  Wyoming  0.4131277
## 26   loyal            Juice/tonics       4  Wyoming  0.5338463
## 34   loyal               Candy/Gum       1 Oklahoma  0.3362973
## 36   loyal            Juice/tonics       1  Alabama  0.4554110
## 37   loyal                  Energy       4 Oklahoma  0.4765806
```

```
table2 <- table(predict_test>.5, data_test$loyalty)
my_confusion_matrix(table2)
```

```
##
##         not loyal loyal
##   FALSE     80501 55670
##   TRUE      49499 74266
```

```
## [[1]]
## [1] "74266 = True Positive (TP), Hit"
##
## [[2]]
## [1] "80501 = True Negative (TN), Rejection"
##
## [[3]]
## [1] "49499 = False Positive (FP), Type 1 Error"
##
## [[4]]
## [1] "55670 = False Negative (FN), Type 2 Error"
##
## [[5]]
## [1] "0.5954 = Accuracy (TP+TN/(TP+TN+FP+FN))"
##
## [[6]]
## [1] "0.5716 = Sensitivity, Recall, Hit Rate, True Positive Rate (How many
positives did the model get right? TP/(TP+FN))"
##
## [[7]]
## [1] "0.6192 = Specificity, Selectivity, True Negative Rate (How many negat
ives did the model get right? TN/(TN+FP))"
##
## [[8]]
## [1] "0.6001 = Precision, Positive Predictive Value (How good are the model
's positive predictions? TP/(TP+FP))"
##
## [[9]]
## [1] "0.5912 = Negative Predictive Value (How good are the model's negative
predictions? TN/(TN+FN)"
```