Poisson Regression. Negative Binomial Regression. Multinomial Logistic Regression. Zero Inflated Poisson. Negative Binomial Regression.

Data 621: Homework 5 - Group 4

Sachid Deshmukh
Michael Yampol
Vishal Arora
Ann Liu-Ferrara

December 5th, 2019

Contents

---

0. Introduction

We are examining a dataset containing information about 12,000 commercially available wines, where the variables are mostly related to the chemical properties of the wine being sold.

The response variable `TARGET` is the number of sample cases of wine that were purchased by wine distribution companies after sampling a wine.

These cases would be used to provide tasting samples to restaurants and wine stores around the United States.

The more sample cases purchased, the more likely is a wine to be sold at a high-end restaurant.

We are asked to study the data in order to predict the number of wine cases ordered, based upon the wine characteristics, because as a wine manufacturer, if we can predict the number of cases ordered, then we can adjust our wine offering to maximize sales.

List of variables in the dataset:

VARIABLE NAME	DEFINITION
INDEX	Identification Variable (do not use)
TARGET	Number of Cases Purchased
AcidIndex	Proprietary method of testing total acidity of wine by using a weighted average
Alcohol	Alcohol Content
Chlorides	Chloride content of wine
CitricAcid	Citric Acid Content
Density	Density of Wine
FixedAcidity	Fixed Acidity of Wine
FreeSulfurDioxide	Sulfur Dioxide content of wine
LabelAppeal	Marketing Score indicating the appeal of label design for consumers. High numbers suggest customers like the label design. Negative numbers suggest customers don't like the design. (Many consumers purchase based on the visual appeal of the wine label design.) (Higher numbers suggest better sales.)
ResidualSugar	Residual Sugar of wine
STARS	Wine rating by a team of experts. 4 Stars = Excellent, 1 Star = Poor (A high number of stars suggests high sales)
Sulphates	Sulfate content of wine
TotalSulfurDioxide	Total Sulfur Dioxide of Wine
VolatileAcidity	Volatile Acid content of wine
pH	pH of wine

1. Data Exploration

Load training dataset

Examine training dataset (transposed, to fit on page)

	1	2	3	4	5	6
ï..INDEX	1.0000	2.00000	4.00000	5.0000	6.00000	7.0000
TARGET	3.0000	3.00000	5.00000	3.0000	4.00000	0.0000
FixedAcidity	3.2000	4.50000	7.10000	5.7000	8.00000	11.3000
VolatileAcidity	1.1600	0.16000	2.64000	0.3850	0.33000	0.3200
CitricAcid	-0.9800	-0.81000	-0.88000	0.0400	-1.26000	0.5900
ResidualSugar	54.2000	26.10000	14.80000	18.8000	9.40000	2.2000
Chlorides	-0.5670	-0.42500	0.03700	-0.4250	NA	0.5560
FreeSulfurDioxide	NA	15.00000	214.00000	22.0000	-167.00000	-37.0000
TotalSulfurDioxide	268.0000	-327.00000	142.00000	115.0000	108.00000	15.0000
Density	0.9928	1.02792	0.99518	0.9964	0.99457	0.9994
pH	3.3300	3.38000	3.12000	2.2400	3.12000	3.2000
Sulphates	-0.5900	0.70000	0.48000	1.8300	1.77000	1.2900
Alcohol	9.9000	NA	22.00000	6.2000	13.70000	15.4000
LabelAppeal	0.0000	-1.00000	-1.00000	-1.0000	0.00000	0.0000
AcidIndex	8.0000	7.00000	8.00000	6.0000	9.00000	11.0000
STARS	2.0000	3.00000	3.00000	1.0000	2.00000	NA

```
## [1] "Number of columns =  16"
```

```
## [1] "Number of rows =  12795"
```

As we can see in the preview, training data has 16 columns and 12795 rows .

Let's analyze datatypes of each column.

```
## 'data.frame':    12795 obs. of  16 variables:
##  $ ï..INDEX          : int  1 2 4 5 6 7 8 11 12 13 ...
##  $ TARGET            : int  3 3 5 3 4 0 0 4 3 6 ...
##  $ FixedAcidity      : num  3.2 4.5 7.1 5.7 8 11.3 7.7 6.5 14.8 5.5 ...
##  $ VolatileAcidity   : num  1.16 0.16 2.64 0.385 0.33 0.32 0.29 -1.22 0.27 -0.22 ...
##  $ CitricAcid        : num  -0.98 -0.81 -0.88 0.04 -1.26 0.59 -0.4 0.34 1.05 0.39 ...
##  $ ResidualSugar     : num  54.2 26.1 14.8 18.8 9.4 ...
##  $ Chlorides         : num  -0.567 -0.425 0.037 -0.425 NA 0.556 0.06 0.04 -0.007 -0.277 ...
##  $ FreeSulfurDioxide : num  NA 15 214 22 -167 -37 287 523 -213 62 ...
##  $ TotalSulfurDioxide: num  268 -327 142 115 108 15 156 551 NA 180 ...
##  $ Density           : num  0.993 1.028 0.995 0.996 0.995 ...
##  $ pH                : num  3.33 3.38 3.12 2.24 3.12 3.2 3.49 3.2 4.93 3.09 ...
##  $ Sulphates         : num  -0.59 0.7 0.48 1.83 1.77 1.29 1.21 NA 0.26 0.75 ...
##  $ Alcohol           : num  9.9 NA 22 6.2 13.7 15.4 10.3 11.6 15 12.6 ...
##  $ LabelAppeal       : int  0 -1 -1 -1 0 0 0 1 0 0 ...
##  $ AcidIndex         : int  8 7 8 6 9 11 8 7 6 8 ...
##  $ STARS             : int  2 3 3 1 2 NA NA 3 NA 4 ...
```

All the columns are either Integer or numeric types. Please note that Integer columns in the above dataset are excellent candidate for creating categorical variables, which can further enhance quality of our models.

Check missing values

Let's check whether any variables have missing values, i.e., values which are NULL or NA.

```
## [1] "Number of columns with missing values =  8"
```

```
## [1] "Names of columns with missing values =  ResidualSugar, Chlorides, FreeSulfurDioxide, TotalSulfu
```

We observe that the following 8 variables have missing values:

- ResidualSugar,
- Chlorides,
- FreeSulfurDioxide,
- TotalSulfurDioxide,
- pH,
- Sulphates,
- Alcohol, and
- STARS .

We will impute values for these variables for better model accuracy.

Distribution of TARGET variable

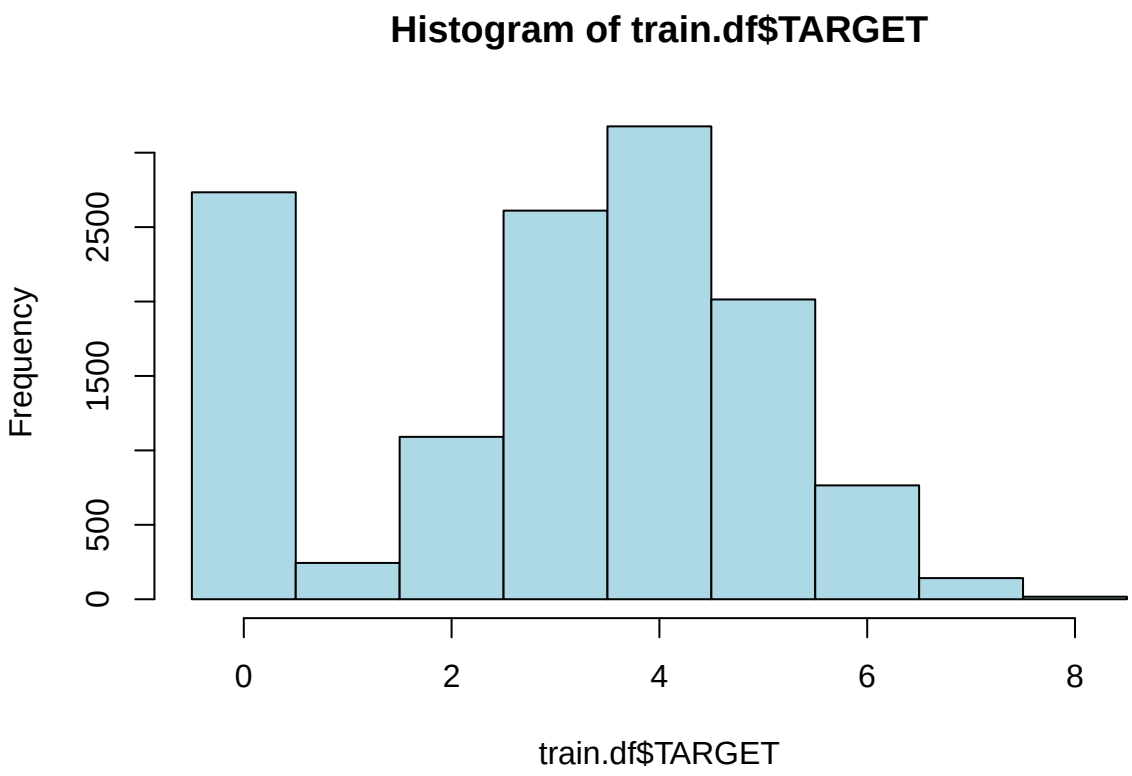
```
table(train.df$TARGET)
```

```
##
##    0    1    2    3    4    5    6    7    8
## 2734  244 1091 2611 3177 2014  765  142   17
```

```
summary(train.df$TARGET)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.00000 2.00000 3.00000 3.02907 4.00000 8.00000
```

```
hist(train.df$TARGET, breaks=seq(-0.5,8.5),col="lightblue")
```



From the above histogram we can see that the distribution of the target variable is not exactly Poisson distributed.

We can see a high number of observations with zeros (n=2734). Then there is another peak around 4 and then it decreases towards the right.

The distribution also shows a skew in the right hand side where there are very few observations as the sample count increases.

This dataset is a very good candidate for *Zero Inflated Poisson / Negative Binomial Regression* since we can see the large number of observations with zero count.

Distribution of STARS feature

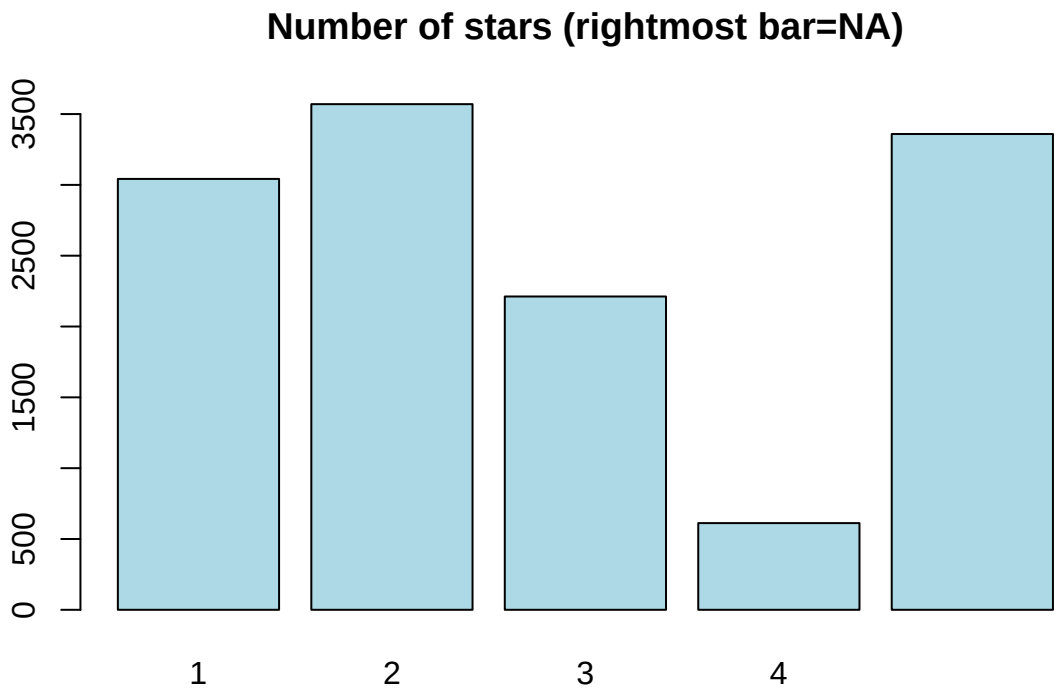
```
table(train.df$STARS, useNA="ifany")
```

```
##
##    1    2    3    4 <NA>
## 3042 3570 2212  612 3359
```

```
summary(train.df$STARS)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
## 1.00000 1.00000 2.00000 2.04175 3.00000 4.00000    3359
```

```
barplot(table(train.df$STARS, useNA="ifany"),col="lightblue",
        main="Number of stars (rightmost bar=NA)")
```



We note that there are many cases where the number of stars is NA.
Does this mean that the wine was simply unrated?
Does it mean that it was awful (meriting zero stars?)
Perhaps the manufacturer knew that the wine was not good, and opted for it to be unrated?

Effect of STARS present or missing on TARGET

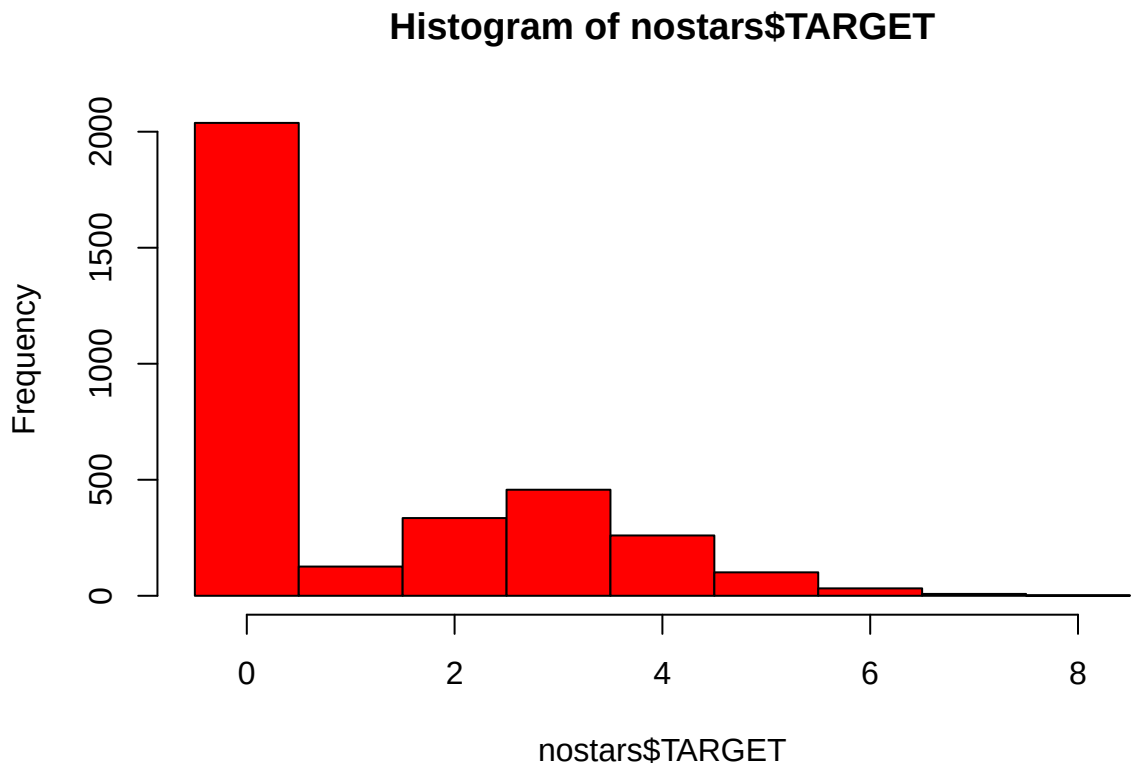
```
## Value of TARGET when STARS is missing
nostars <- train.df %>% filter(is.na(STARS))
# table of TARGET values when STARS is missing
table(nostars$TARGET)
```

```
##
##    0    1    2    3    4    5    6    7    8
## 2038  126  335  457  260  101   32    8    2
```

```r
summary(nostars$TARGET)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.00000 0.00000 0.00000 1.18369 3.00000 8.00000
```

```r
hist(nostars$TARGET, breaks=seq(-0.5,8.5),col="red")
```



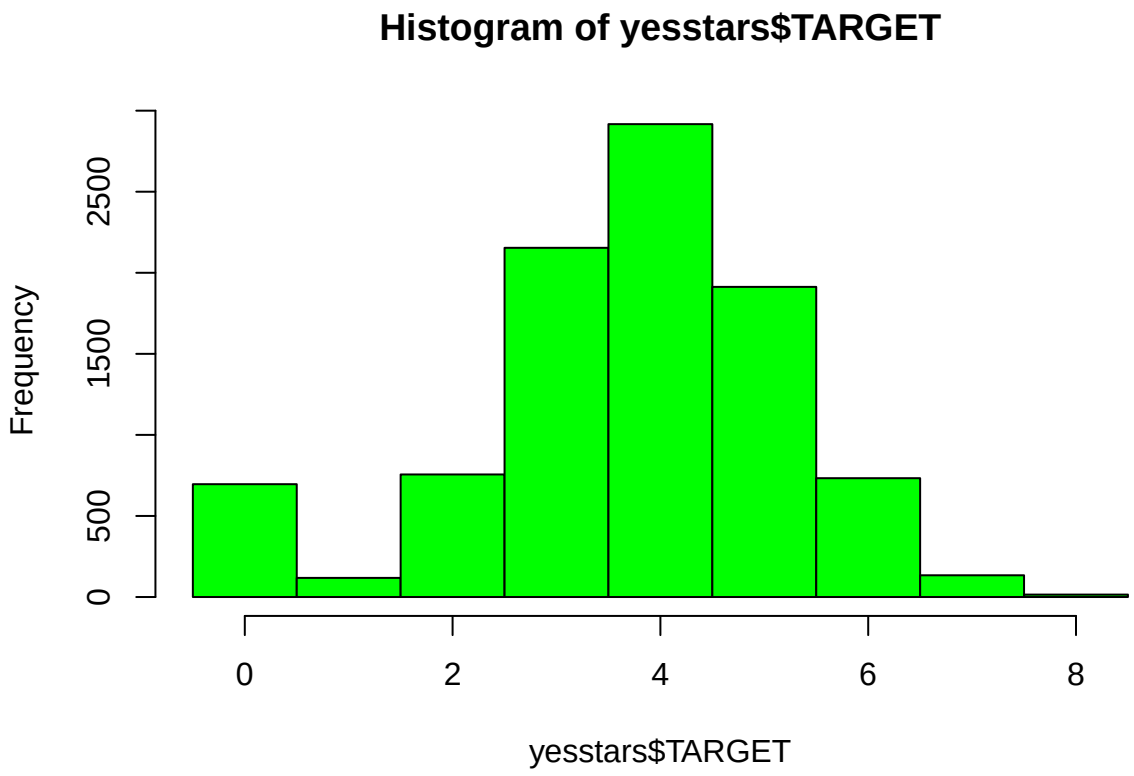

```r
## Value of TARGET when STARS is not missing
yesstars <- train.df %>% filter(!is.na(STARS))
# table of TARGET values when STARS is missing
table(yesstars$TARGET)
```

```
##
##    0    1    2    3    4    5    6    7    8
##  696  118  756 2154 2917 1913  733  134   15
```

```r
summary(yesstars$TARGET)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.00000 3.00000 4.00000 3.68599 5.00000 8.00000
```

```
hist(yesstars$TARGET, breaks=seq(-0.5,8.5),col="green")
```



The above summaries reveal that for wines which have a missing STARS rating,

- the average number of cases ordered is 1.18, and
- the median quantity ordered is zero.

For wines which DO have a STARS rating,

- the average number of cases ordered is 3.686, and
- the median quatity is 4.

In such a circumstance, perhaps we should replace each "NA" for STARS with a **zero**, rather than imputing a value using the `MICE` package, as we do for other missing values?

2. Data Preparation

Drop Index Column

Remove the `Index` column since it is just an identifier and won't contribute toward the predictive capability of the model

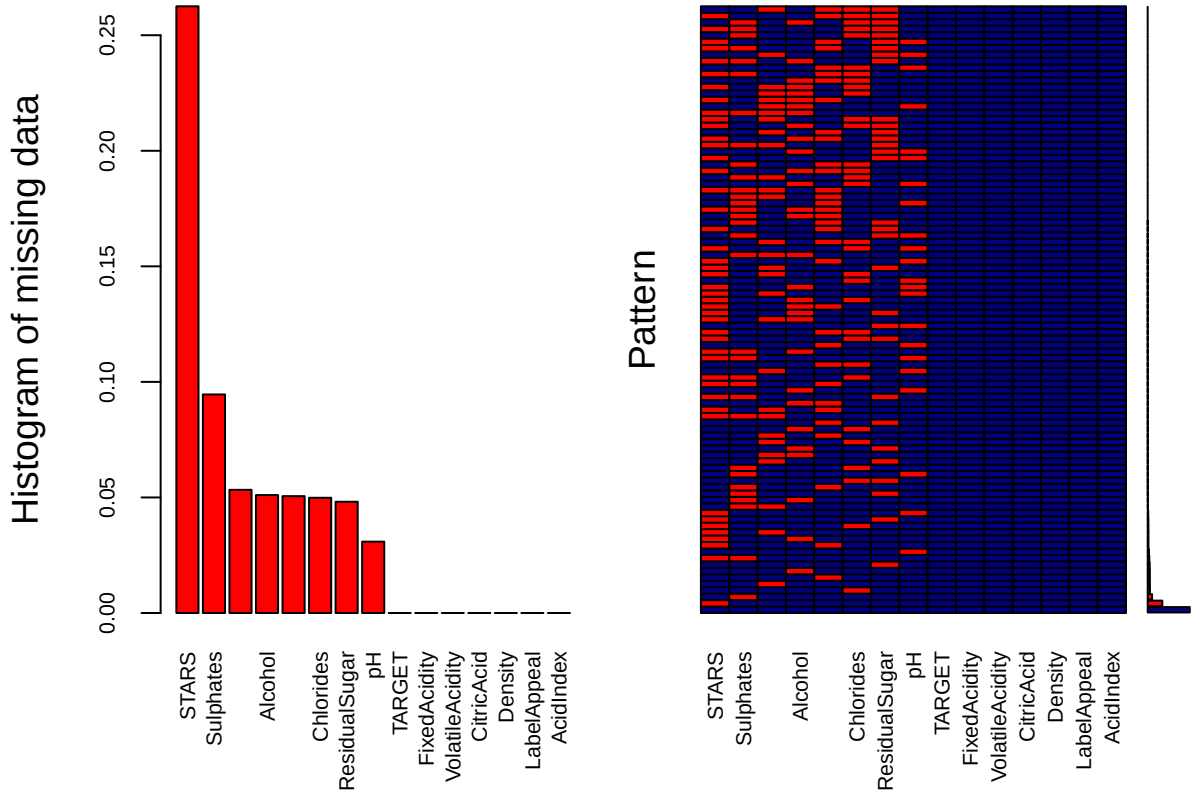
```
train.df = train.df[1:nrow(train.df), 2:ncol(train.df)]
```

Impute missing data

Which columns have missing data, and what is the pattern for the missing data?

Let's leverage the VIM package to get this information.

```
## Warning in plot.aggr(res, ...): not enough vertical space to display frequencies (too many combinati
```




```
##
##  Variables sorted by number of missings:
##             Variable        Count
##                STARS 0.2625244236
##            Sulphates 0.0945681907
##  TotalSulfurDioxide 0.0533020711
```

```
##             Alcohol 0.0510355608
##   FreeSulfurDioxide 0.0505666276
##           Chlorides 0.0498632278
##       ResidualSugar 0.0481438062
##                  pH 0.0308714342
##              TARGET 0.0000000000
##        FixedAcidity 0.0000000000
##     VolatileAcidity 0.0000000000
##          CitricAcid 0.0000000000
##             Density 0.0000000000
##         LabelAppeal 0.0000000000
##           AcidIndex 0.0000000000
```

From the above missing values pattern we can see that variable `Stars` has the highest proportion of missing values, with about 25 percent of the observations missing any value for `Stars`.

Because no wine is rated "zero stars", the fact that the value is "NA" might indicate wine with 0 Star ratings.

Let's replace NA STARS values with zero, before imputing other missing variables:

```
train.df$STARS[is.na(train.df$STARS)]=0
summary(train.df$STARS)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.00000 0.00000 1.00000 1.50574 2.00000 4.00000
```

This change has reduced the average STARS value from 2.04 to 1.51 and has reduced the median value from 2 to 1.

All other variables have a very low proportion of missing observations.

This is good news because since this indicates good quality of input data.

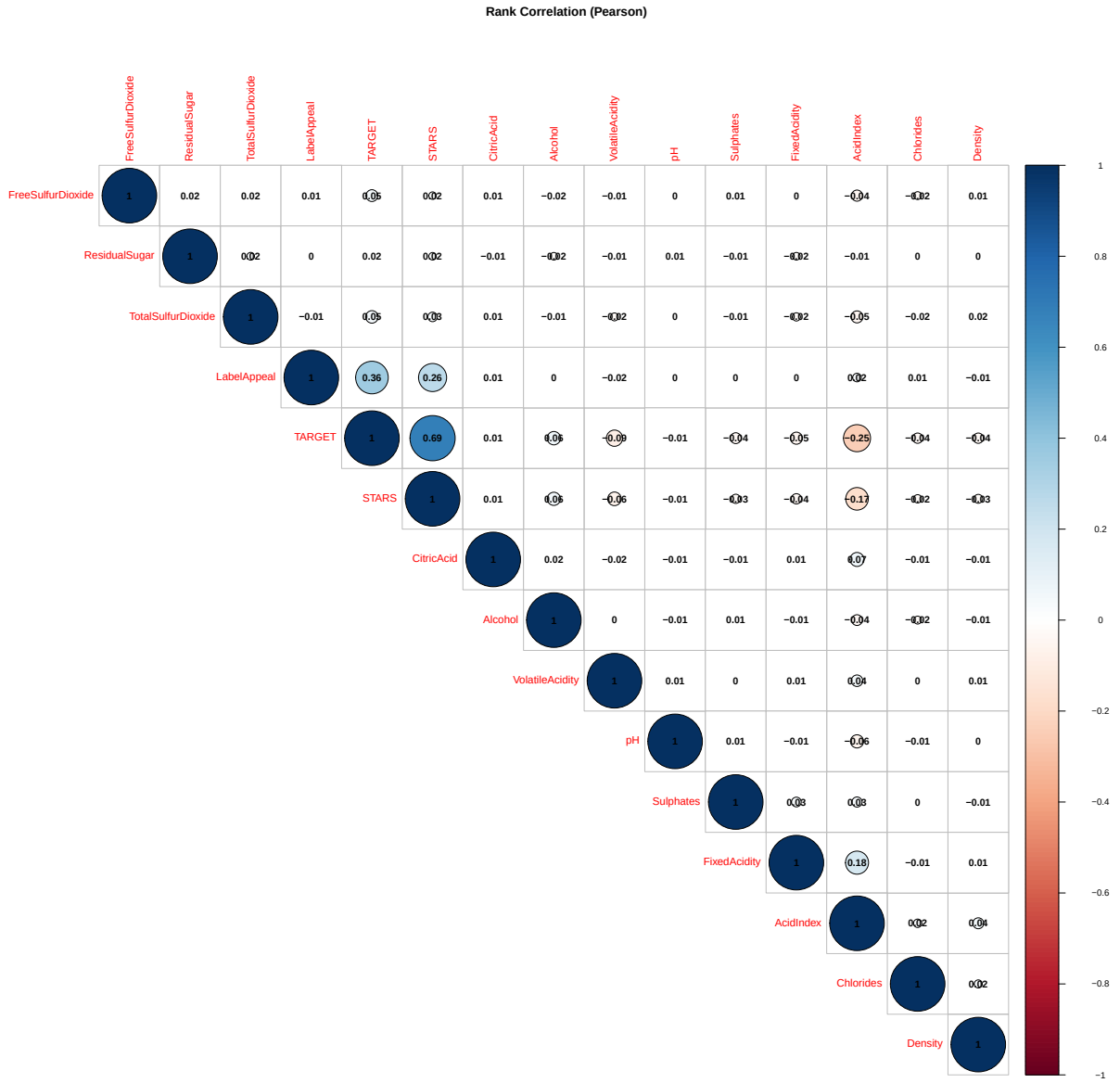
Let's use MICE package to impute missing values

```
##
##  iter imp variable
##   1   1  ResidualSugar  Chlorides  FreeSulfurDioxide  TotalSulfurDioxide  pH  Sulphates  Alcohol
##   1   2  ResidualSugar  Chlorides  FreeSulfurDioxide  TotalSulfurDioxide  pH  Sulphates  Alcohol
##   2   1  ResidualSugar  Chlorides  FreeSulfurDioxide  TotalSulfurDioxide  pH  Sulphates  Alcohol
##   2   2  ResidualSugar  Chlorides  FreeSulfurDioxide  TotalSulfurDioxide  pH  Sulphates  Alcohol
##   3   1  ResidualSugar  Chlorides  FreeSulfurDioxide  TotalSulfurDioxide  pH  Sulphates  Alcohol
##   3   2  ResidualSugar  Chlorides  FreeSulfurDioxide  TotalSulfurDioxide  pH  Sulphates  Alcohol
##   4   1  ResidualSugar  Chlorides  FreeSulfurDioxide  TotalSulfurDioxide  pH  Sulphates  Alcohol
##   4   2  ResidualSugar  Chlorides  FreeSulfurDioxide  TotalSulfurDioxide  pH  Sulphates  Alcohol
##   5   1  ResidualSugar  Chlorides  FreeSulfurDioxide  TotalSulfurDioxide  pH  Sulphates  Alcohol
##   5   2  ResidualSugar  Chlorides  FreeSulfurDioxide  TotalSulfurDioxide  pH  Sulphates  Alcohol
##   6   1  ResidualSugar  Chlorides  FreeSulfurDioxide  TotalSulfurDioxide  pH  Sulphates  Alcohol
##   6   2  ResidualSugar  Chlorides  FreeSulfurDioxide  TotalSulfurDioxide  pH  Sulphates  Alcohol
##   7   1  ResidualSugar  Chlorides  FreeSulfurDioxide  TotalSulfurDioxide  pH  Sulphates  Alcohol
```

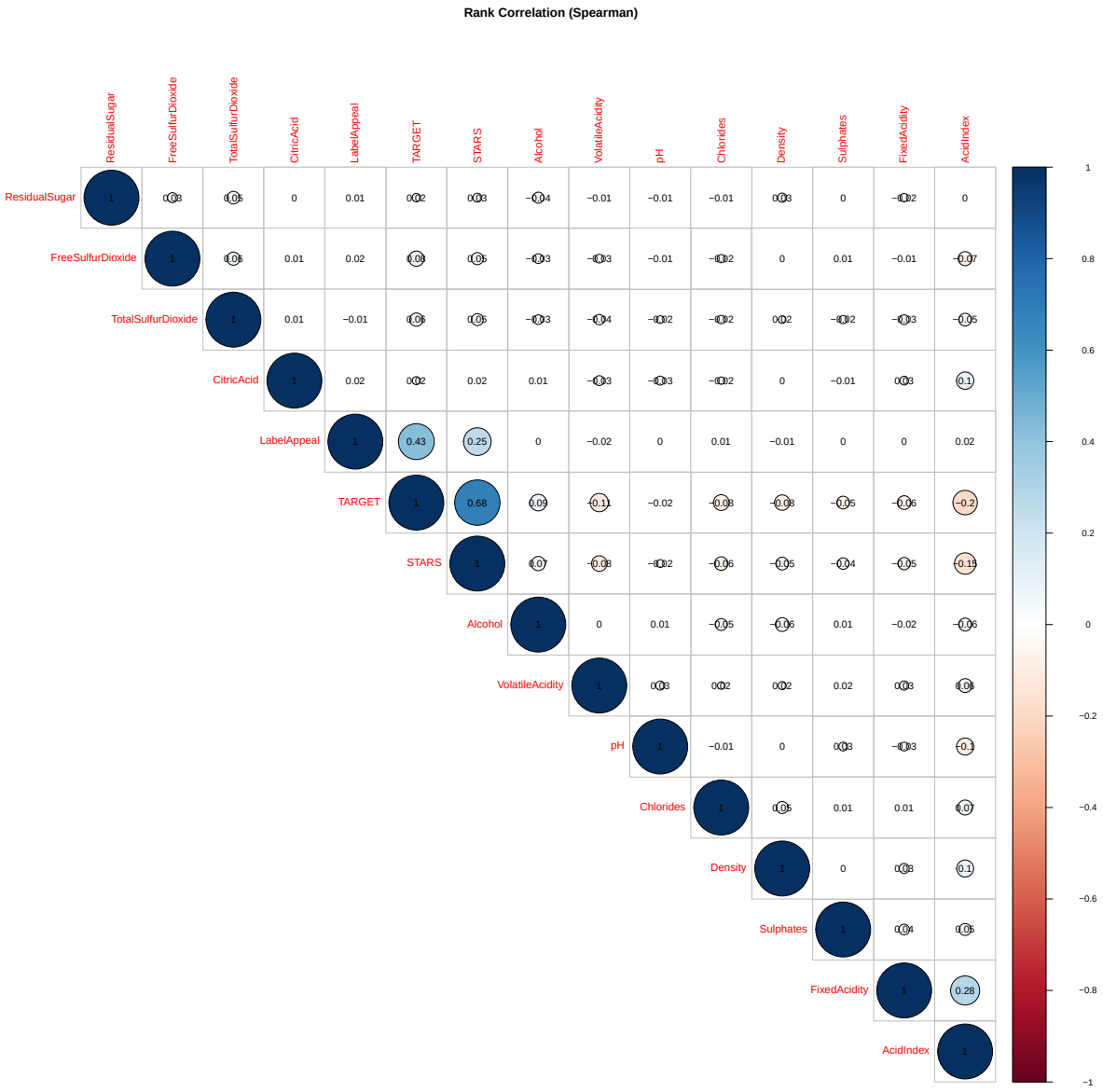
```
##    7   2  ResidualSugar  Chlorides  FreeSulfurDioxide  TotalSulfurDioxide  pH  Sulphates  Alcohol
##    8   1  ResidualSugar  Chlorides  FreeSulfurDioxide  TotalSulfurDioxide  pH  Sulphates  Alcohol
##    8   2  ResidualSugar  Chlorides  FreeSulfurDioxide  TotalSulfurDioxide  pH  Sulphates  Alcohol
##    9   1  ResidualSugar  Chlorides  FreeSulfurDioxide  TotalSulfurDioxide  pH  Sulphates  Alcohol
##    9   2  ResidualSugar  Chlorides  FreeSulfurDioxide  TotalSulfurDioxide  pH  Sulphates  Alcohol
##    10   1  ResidualSugar  Chlorides  FreeSulfurDioxide  TotalSulfurDioxide  pH  Sulphates  Alcohol
##    10   2  ResidualSugar  Chlorides  FreeSulfurDioxide  TotalSulfurDioxide  pH  Sulphates  Alcohol
```

Check variable correlation

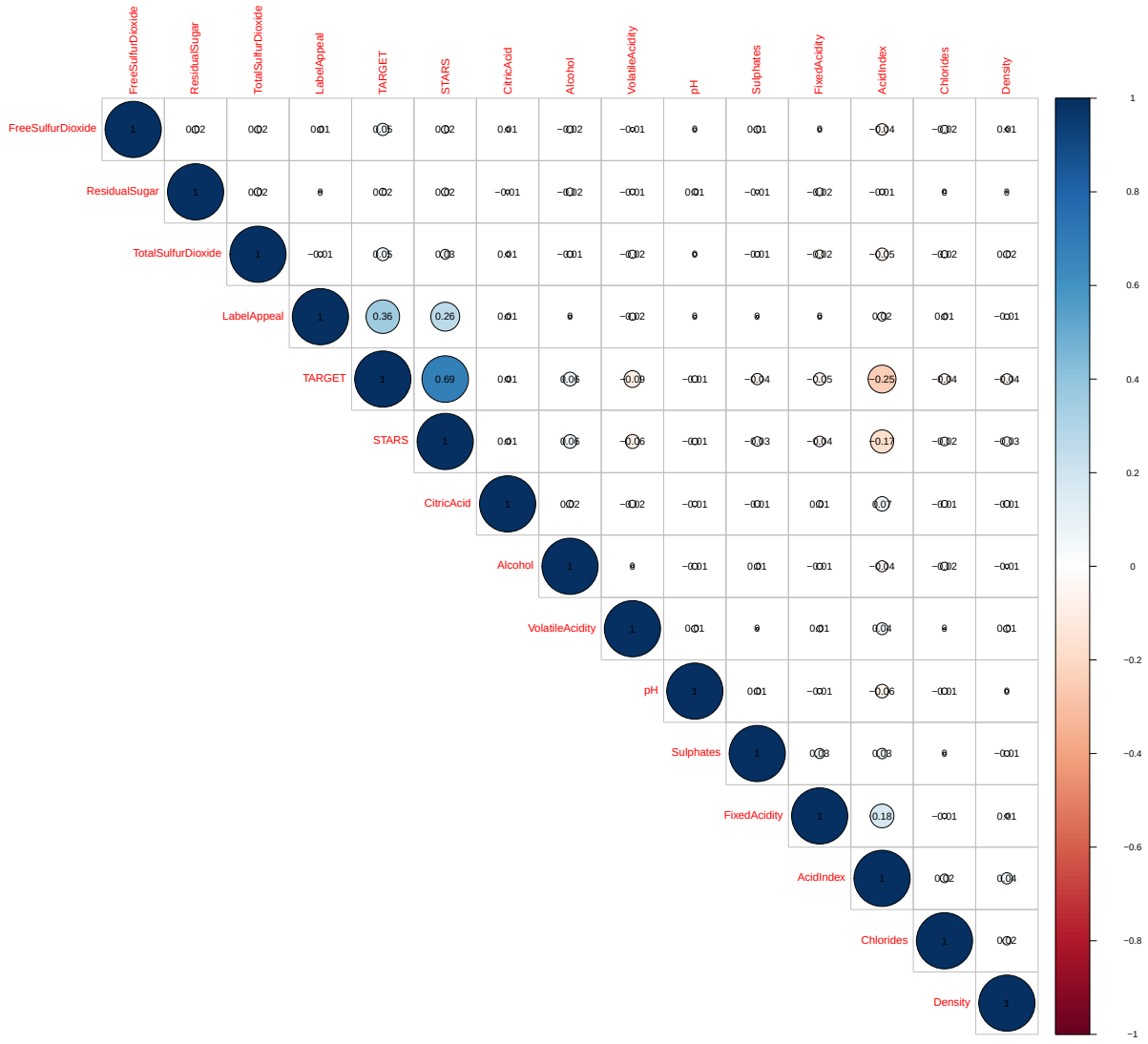
The variables which are highly correlated carry similar information and can affect model accuracy. Highly correlated variables also impact estimation of model coefficients. Let's determine which variables in the training datasets are highly correlated to each other.



Spearman rank correlation



Actual correlations


Actual correlations (not rank correlations)
FreeSulfurDioxide
ResidualSugar
TotalSulfurDioxide
LabelAppeal
TARGET
STARS
CitricAcid
Alcohol
VolatileAcidity
pH
Sulphates
FixedAcidity
AcidIndex
Chlorides
Density
0.36
0.26
0.69
−0.25
0.18
1
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1

From the above correlation graphs we can see that target variable TARGET is highly correlated with following variables:

- **STARS**

Additionally we can see a significant positive correlation with the following variable:

- **LabelAppeal**

Finally, there is a modest negative correlation with the following variable:

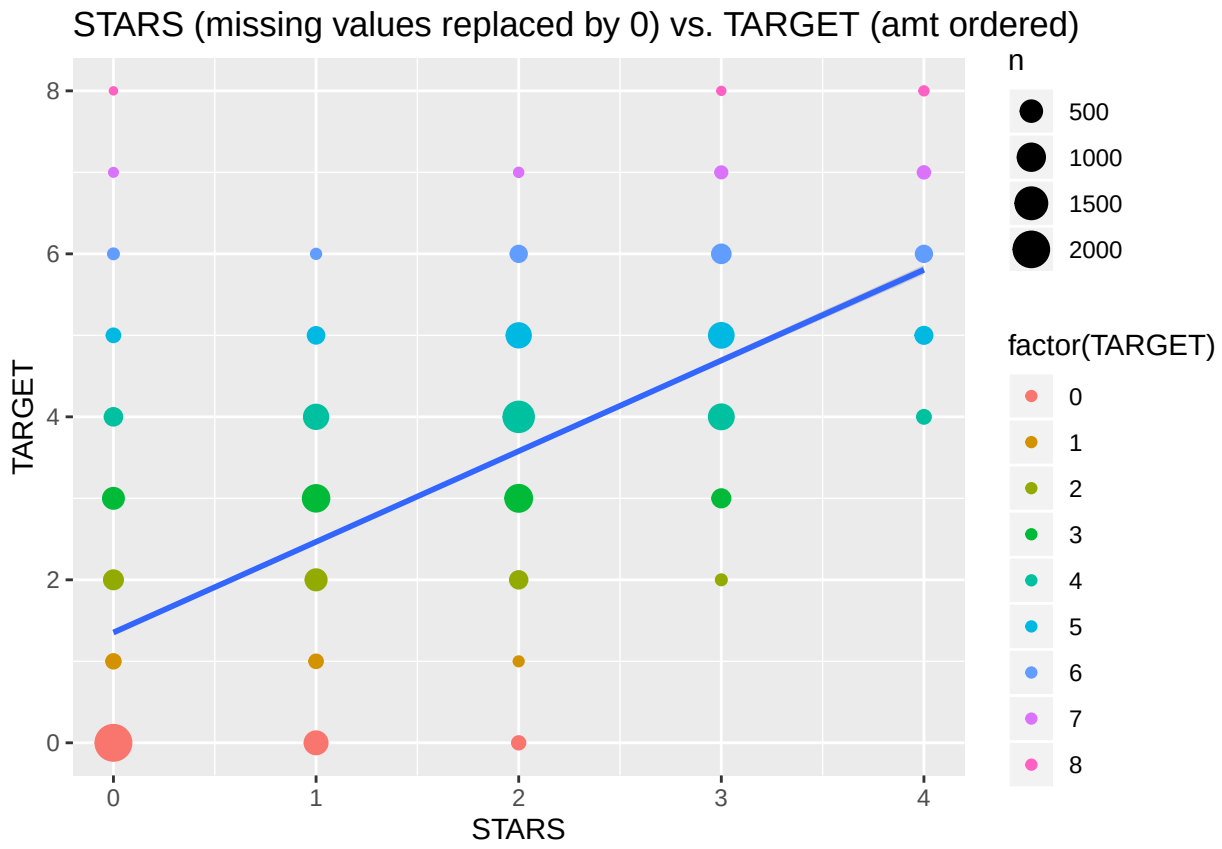
- **Acid Index**

Feature Transformations

Transform STARS Variable

Positive correlation between STARS and Target variable:

```
ggplot(train.df, aes(x= STARS, y = TARGET)) +
  geom_count(aes(colour = factor(TARGET))) +
  geom_smooth(method='lm') +
  ggtitle("STARS (missing values replaced by 0) vs. TARGET (amt ordered)")
```




```
train.df$StarTran = ifelse(train.df$STARS == 1, 2, train.df$STARS)
```

Table of raw STARS values:

```
table(train.df$STARS)
```

```
##
##    0    1    2    3    4
## 3359 3042 3570 2212  612
```

We will transform **STARS* by aggregating all of the “1” and “2” star ratings together, onto “2”

Table of transformed STARS values:

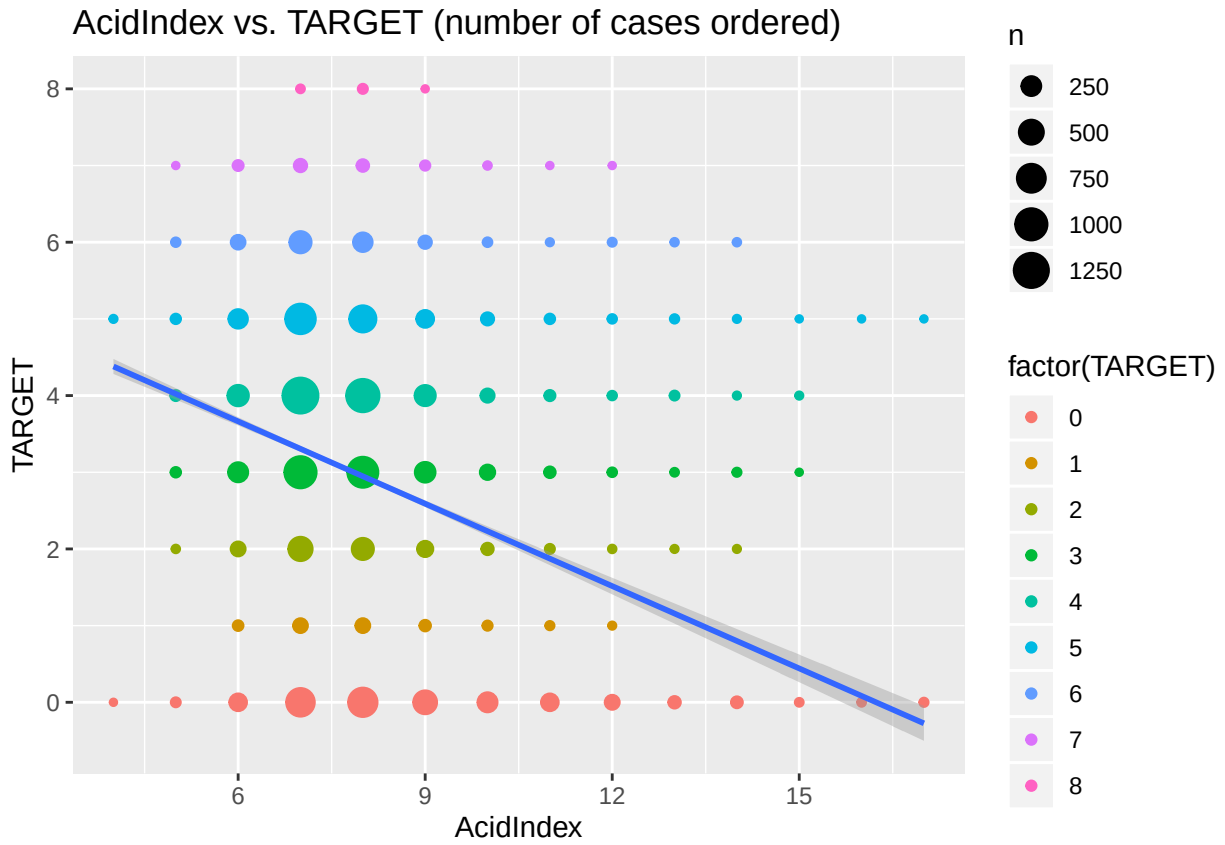
```
table(train.df$StarTran)
```

```
##
##    0    2    3    4
## 3359 6612 2212  612
```

Transform AcidIndex variable

Negative correlation between AcidIndex and Target variable:

```
ggplot(train.df, aes(x= AcidIndex, y = TARGET)) +
  geom_count(aes(colour = factor(TARGET))) +
  geom_smooth(method='lm') +
  ggtitle("AcidIndex vs. TARGET (number of cases ordered)")
```




```
train.df$AcidIndexTran = ifelse(train.df$AcidIndex <=6,  0, train.df$AcidIndex)
train.df$AcidIndexTran = ifelse((train.df$AcidIndexTran > 6 & train.df$AcidIndexTran < 12),
                                1, train.df$AcidIndexTran)
train.df$AcidIndexTran = ifelse(train.df$AcidIndexTran >= 12 ,  2, train.df$AcidIndexTran)
```

Table of raw AcidIndex values:

```
table(train.df$AcidIndex)
```

```
##
##    4    5    6    7    8    9   10   11   12   13   14   15   16   17
##    3   75 1197 4878 4142 1427  551  258  128   69   47    8    5    7
```

We transform **AcidIndex** into three groups by separating out the very low and very high values:

- 0: $4 \leq AcidIndex \leq 6$
- 1: $6 < AcidIndex < 12$
- 2: $12 \leq AcidIndex \leq 17$

Table of transformed AcidIndex values:

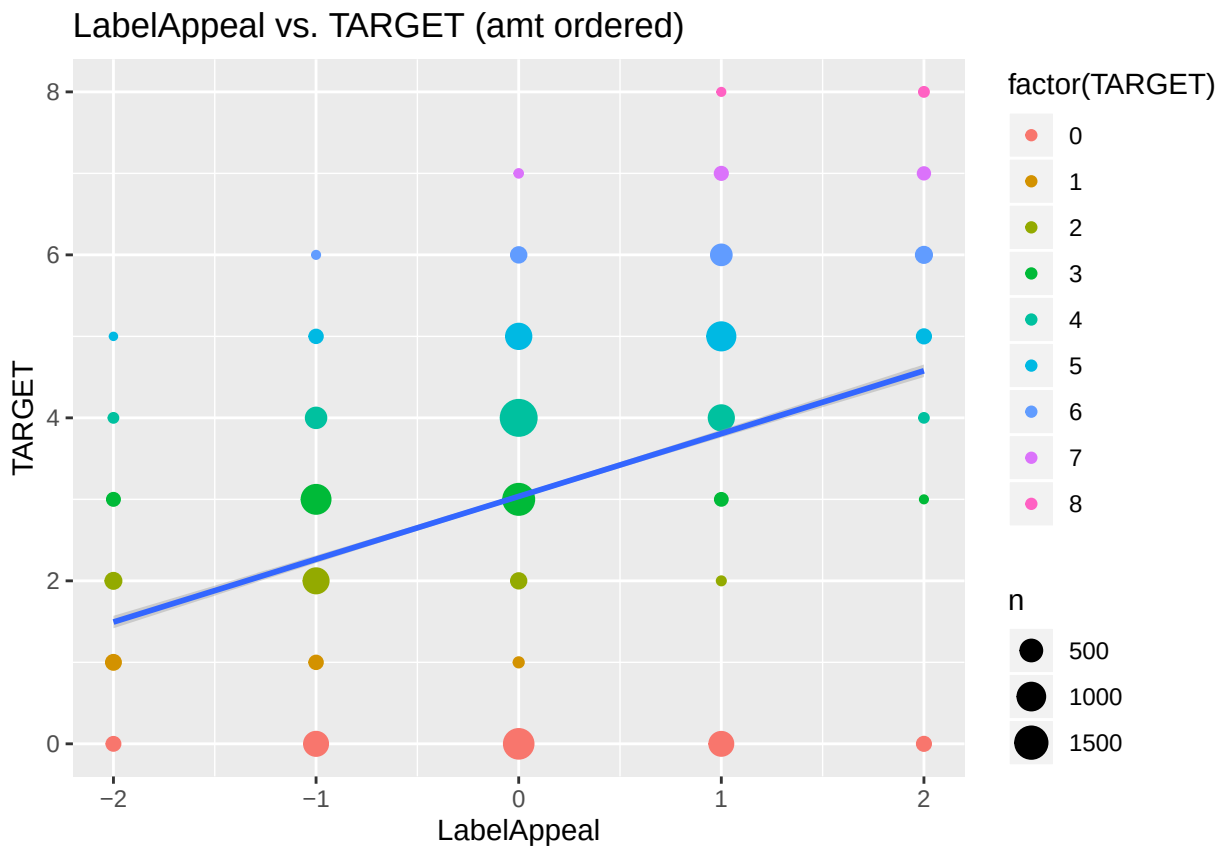
```
table(train.df$AcidIndexTran)
```

```
##
##     0     1     2
##  1275 11256   264
```

Transform LabelAppeal Variable

Positive correlation between LabelAppeal and target variable:

```
ggplot(train.df, aes(x= LabelAppeal, y = TARGET)) +
  geom_count(aes(colour = factor(TARGET))) +
  geom_smooth(method='lm') +
  ggtitle("LabelAppeal vs. TARGET (amt ordered)")
```




```
train.df$LabelAppealTran = ifelse(train.df$LabelAppeal <0,  0, train.df$LabelAppeal)
```

Table of raw LabelAppeal values:

```
table(train.df$LabelAppeal)
```

```
##
##   -2   -1    0    1    2
##  504 3136 5617 3048  490
```

We will transform **LabelAppeal** by aggregating all of the negative LabelAppeal values with zero (leaving "1" and "2" unchanged):

Table of transformed LabelAppeal values:

```
table(train.df$LabelAppealTran)
```

```
##
##    0    1    2
## 9257 3048  490
```

3. Build Models

Model fitting and evaluation

For model evaluation we will use Train/Test split technique.

We will use 70% of the data to train the model, and leverage the remaining 30% to test the model.

Model Evaluation Metrics

We will use following metrics for model evaluation and comparison:

- **RMSE** : We wlll use Root Mean Square error to evaluate the **Poisson** and **Negative Binomial** models.
- Root mean square indicates the quality of model fit.
- Lesser RMSE indicates better model fit and higher RMSE indicates poor model fit.
- RMSE can be calculated as $RMSE = \sqrt{\frac{\sum_{i=1}^{n}(y-\hat{y})^2}{n}}$.

where:

- y : Actual value
- $\hat{y}$: Predicted value
- **Model Accuracy** : For the **multinomial** model we will use **accuracy** as the model evaluation metric.
- Informally, accuracy is the fraction of predictions our model got right.
- Formally, accuracy has the following definition: $Accuracy = \frac{(TP+TN)}{(TP+FP+TN+FN)}$

where:

- **TP** : Stands for True Positives
- **TN** : Stands for True Negatives
- **FP** : Stands for False Positives
- **FN** : Stands for False Negatives

Now we are clear on our model fitting and evaluation method (Train/Test split) and also have model evaluation metrics (RMSE and Accuracy) which we will use to compare the model effectiveness.

We are all set to build different models and access their relative performance.

1. Split training data into train and test

```
train.df.class = dplyr::rename(train.df, target = TARGET)
dt = sort(sample(nrow(train.df.class), nrow(train.df.class)*.7))
train.data = train.df.class[dt,]
test.data = train.df.class[-dt,]
test.values = test.data$target
test.data = test.data[1:nrow(test.data), names(test.data)[names(test.data)!= 'target']]
```

There are 8956 cases in the training set and 3839 cases in the test set.
(The original dataset had 12795 cases .)

2. Poisson regression model with all variables

```
model.pois.all = glm(target~  ., family=poisson, data=train.data)
summary(model.pois.all)
```

```
##
## Call:
## glm(formula = target ~ ., family = poisson, data = train.data)
##
## Deviance Residuals:
##       Min         1Q     Median         3Q        Max
## -2.892845  -0.696200   0.074483   0.551342   3.335273
##
## Coefficients:
##                        Estimate    Std. Error   z value               Pr(>|z|)
## (Intercept)         1.3569707671  0.2315897835   5.85937        0.0000000046462 ***
## FixedAcidity       -0.0008125734  0.0009807230  -0.82855             0.40736175
## VolatileAcidity    -0.0256243820  0.0078042983  -3.28337             0.00102575 **
## CitricAcid          0.0098452991  0.0070979493   1.38706             0.16542272
## ResidualSugar      -0.0000334083  0.0001800611  -0.18554             0.85280635
## Chlorides          -0.0436112182  0.0190118229  -2.29390             0.02179625 *
## FreeSulfurDioxide   0.0001474326  0.0000415216   3.55075             0.00038414 ***
## TotalSulfurDioxide  0.0000731776  0.0000264503   2.76661             0.00566432 **
## Density            -0.0910793986  0.2267603912  -0.40165             0.68793810
## pH                 -0.0201789723  0.0089776809  -2.24768             0.02459647 *
## Sulphates          -0.0141462500  0.0065565105  -2.15759             0.03095986 *
## Alcohol             0.0025197960  0.0016330517   1.54300             0.12283118
## LabelAppeal         0.2045482795  0.0141889087  14.41607 < 0.00000000000000222 ***
## AcidIndex          -0.1046433140  0.0069941329 -14.96159 < 0.00000000000000222 ***
## STARS               0.1573295428  0.0157409257   9.99494 < 0.00000000000000222 ***
## StarTran            0.1700408628  0.0170174553   9.99214 < 0.00000000000000222 ***
## AcidIndexTran       0.0994955142  0.0232611942   4.27732        0.0000189158869 ***
## LabelAppealTran    -0.1020973534  0.0202069049  -5.05260        0.0000004358422 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 15819.65  on 8955  degrees of freedom
## Residual deviance: 10019.37  on 8938  degrees of freedom
## AIC: 32487.08
##
## Number of Fisher Scoring iterations: 5
```

```
rmse.pois.all = model.fit.evaluate.rmse(model.pois.all, test.data, test.values)
print(paste("RMSE: ",rmse.pois.all))
```

```
## [1] "RMSE:  2.63153576142571"
```

3. Reduced Poisson regression model with selected variables

We include the following variables:

- LabelAppeal
- STARS

- AcidIndex

```
model.pois.sel = glm(target~  LabelAppeal + STARS + AcidIndex , family=poisson, data=train.data)
summary(model.pois.sel)
```

```
##
## Call:
## glm(formula = target ~ LabelAppeal + STARS + AcidIndex, family = poisson,
##     data = train.data)
##
## Deviance Residuals:
##       Min         1Q     Median         3Q        Max
## -2.991058  -0.681008   0.055445   0.558621   3.266691
##
## Coefficients:
##                Estimate  Std. Error  z value               Pr(>|z|)
## (Intercept)  1.23215625  0.04364067  28.2341 < 0.000000000000000222 ***
## LabelAppeal  0.13491802  0.00722647  18.6700 < 0.000000000000000222 ***
## STARS        0.31066703  0.00541665  57.3541 < 0.000000000000000222 ***
## AcidIndex   -0.08931702  0.00532534 -16.7721 < 0.000000000000000222 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 15819.65  on 8955  degrees of freedom
## Residual deviance: 10217.58  on 8952  degrees of freedom
## AIC: 32657.29
##
## Number of Fisher Scoring iterations: 5
```

```
rmse.pois.sel = model.fit.evaluate.rmse(model.pois.sel, test.data, test.values)
print(paste("RMSE: ",rmse.pois.sel))
```

```
## [1] "RMSE:  2.6293571734572"
```

From the RMSE analysis of

- Poisson Regression model with all variables, and
- Reduced Poisson regression model with selected variables,

we can see that even though the RMSE of the Poisson regression model with all variables is smaller, there is not much difference.

Going forward we will prefer the reduced model, to maintain model simplicity and to make the model more interpretable by removing many variables of lesser significance.

4. Model Dispersion test

```
dispersiontest(model.pois.sel)
```

```
##
##  Overdispersion test
##
## data:  model.pois.sel
## z = -10.87096, p-value = 1
## alternative hypothesis: true dispersion is greater than 1
## sample estimates:
##  dispersion
## 0.858249504
```

The model dispersion test is 0.858, with a p-value equal to 1, which means that we *fail to reject* the null hypothesis that the dispersion is less than 1.

This indicates **under-dispersion**. For the Poisson assuption to hold true, we need mean and variance of the target variable to be the same.
If mean is greater than variance it results in over-dispersion.
If mean is less than variance it results in under-dispersion.

Because the Poisson model violates the assumption (Mean = Var), the result will be inaccurate calculation of standard errors for model coefficients.

The magnitude of model coefficients will be unaffected, but

inaccurate standard errors will yield inaccurate calculation of confidence intervals of the model coefficients and will in turn result in inaccurate inference.

Negative Binomial regression is well-suited for overdispersion.

For underdispersion, Negative Binomial regression is not reccomended.

However, let's try Negative Binomial Regression on the above dataset and compare the results:

5. Negative Binomial Regression

```
negbio = glm.nb(target~  LabelAppeal + STARS + AcidIndex , data=train.data)
```

```
## Warning in theta.ml(Y, mu, sum(w), w, limit = control$maxit, trace = control$trace > : iteration lim

## Warning in theta.ml(Y, mu, sum(w), w, limit = control$maxit, trace = control$trace > : iteration lim
```

```
summary(negbio)
```

```
##
## Call:
## glm.nb(formula = target ~ LabelAppeal + STARS + AcidIndex, data = train.data,
##     init.theta = 49864.60522, link = log)
##
## Deviance Residuals:
##       Min         1Q     Median         3Q        Max
## -2.990993  -0.680990   0.055434   0.558598   3.266593
##
## Coefficients:
##                Estimate  Std. Error  z value               Pr(>|z|)
## (Intercept)  1.23216445  0.04364217  28.2333 < 0.000000000000000222 ***
## LabelAppeal  0.13491742  0.00722674  18.6692 < 0.000000000000000222 ***
## STARS        0.31067100  0.00541686  57.3526 < 0.000000000000000222 ***
## AcidIndex   -0.08931901  0.00532552 -16.7719 < 0.000000000000000222 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for Negative Binomial(49864.6052) family taken to be 1)
##
##     Null deviance: 15818.99  on 8955  degrees of freedom
## Residual deviance: 10217.23  on 8952  degrees of freedom
## AIC: 32659.49
##
## Number of Fisher Scoring iterations: 1
##
##
##               Theta:  49865
##           Std. Err.:  61697
## Warning while fitting theta: iteration limit reached
##
##  2 x log-likelihood:  -32649.488
```

```
rmse.nb = model.fit.evaluate.rmse(negbio, test.data, test.values)
# Compute RMSE
print(paste("RMSE:", rmse.nb))
```

```
## [1] "RMSE: 2.6293571734572"
```

6. Robust Poisson regression model

In general it is a good idea to fit the Robust Poisson regression model to get accurate estimation for Std Errors.

Since the dataset indicates under-dispersion it is a good idea to fit Robust Poisson regression model and check whether we see any difference in the Std Error estimation for the model regression coefficients.

```
model.pois.sel.robust = glm(target~  LabelAppeal + STARS + AcidIndex ,
                            family=quasipoisson, data=train.data)
summary(model.pois.sel.robust)
```

```
##
## Call:
## glm(formula = target ~ LabelAppeal + STARS + AcidIndex, family = quasipoisson,
##     data = train.data)
##
## Deviance Residuals:
##       Min         1Q     Median         3Q        Max
## -2.991058  -0.681008   0.055445   0.558621   3.266691
##
## Coefficients:
##                Estimate  Std. Error  t value               Pr(>|t|)
## (Intercept)  1.23215625  0.04002509  30.7846 < 0.000000000000000222 ***
## LabelAppeal  0.13491802  0.00662776  20.3565 < 0.000000000000000222 ***
## STARS        0.31066703  0.00496789  62.5350 < 0.000000000000000222 ***
## AcidIndex   -0.08931702  0.00488414 -18.2872 < 0.000000000000000222 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for quasipoisson family taken to be 0.841165976)
##
##     Null deviance: 15819.65  on 8955  degrees of freedom
## Residual deviance: 10217.58  on 8952  degrees of freedom
## AIC: NA
##
## Number of Fisher Scoring iterations: 5
```

```
rmse.pois.sel.robust = model.fit.evaluate.rmse(model.pois.sel.robust, test.data, test.values)
print(rmse.pois.sel.robust)
```

```
## [1] 2.62935717
```

7. Compare Negative Binomial, Poisson Regression, and Robust Poisson Regression models: Coefficients and Std Errors

```
pois.coef = coef(model.pois.sel)
negbinom.coef = coef(negbio)
pois.stderr = se.coef(model.pois.sel)
negbinom.stderr = summary(negbio)$coefficients[, 2]
pois.robust.coef = coef(model.pois.sel.robust)
pois.robust.stderr = se.coef(model.pois.sel.robust)
df.analysis = cbind(pois.coef,   negbinom.coef,   pois.robust.coef,
                    pois.stderr, negbinom.stderr, pois.robust.stderr)
head(df.analysis,10) %>% kable() %>% kable_styling(c("striped", "bordered"))
```

	pois.coef	negbinom.coef	pois.robust.coef	pois.stderr	negbinom.stderr	pois.robust.stderr
(Intercept)	1.232156248	1.232164454	1.232156248	0.043640673	0.043642172	0.040025088
LabelAppeal	0.134918017	0.134917420	0.134918017	0.007226465	0.007226736	0.006627760
STARS	0.310667027	0.310670995	0.310667027	0.005416654	0.005416861	0.004967889
AcidIndex	-0.089317022	-0.089319009	-0.089317022	0.005325336	0.005325520	0.004884138

From the above table we can see that model coefficients and std errors for Poisson and Negative Binomial regression models are the same (up to 5 decimal places.)

This can be due to the fact that under-dispersion in the dataset is not severe enough to impact the accuracy of the Poisson regression model.

From the above table we can see that model coefficients for Poisson Regression and Robust Poisson Regression models are same, but the estimates for Std Errors are different.

This is expected since the dataset has under-dispersion.

Std Error estimations for regression coefficients of the Poisson regression model will not be accurate.

We need to rely on Std Error estimates from the Robust Poisson regression model, which is better suited for datasets exhibiting under-dispersion or over-dispersion.

If we need to use these coefficients for inference, it is better to rely on Std Error estimates from the Robust Poisson regression model to calculate the confidence intervals, rather than from the normal Poisson regression model, for better accuracy of inference.

8. Fit Zero-Inflated Poisson (ZIP) Regression model

```
## For technical reasons, we include the otherwise insignificant variable "FixedAcidity"
## because when omitting it, we get "NANs produced" on the Zero-Inflated Negative Binomial (below)
## and we want to keep the same formula to facilitate comparison between the two models

model.pois.zip= zeroinfl(target~  LabelAppeal + STARS + AcidIndex +FixedAcidity|STARS,
                         dist='poisson', data=train.data)
summary(model.pois.zip)
```

```
##
## Call:
## zeroinfl(formula = target ~ LabelAppeal + STARS + AcidIndex + FixedAcidity | STARS, data = train.data
##
## Pearson residuals:
##        Min         1Q     Median         3Q        Max
## -2.3031284 -0.5342597  0.0231726  0.4092066  2.9253167
##
## Count model coefficients (poisson with log link):
##                   Estimate   Std. Error  z value               Pr(>|z|)
## (Intercept)   1.358527526  0.047268000 28.74096 < 0.000000000000000222 ***
## LabelAppeal   0.224993670  0.007540099 29.83962 < 0.000000000000000222 ***
## STARS         0.101955422  0.006249841 16.31328 < 0.000000000000000222 ***
## AcidIndex    -0.033575977  0.006022439 -5.57515         0.000000024732 ***
## FixedAcidity  0.000220373  0.001001752  0.21999                0.82588
##
## Zero-inflation model coefficients (binomial with logit link):
##               Estimate Std. Error   z value               Pr(>|z|)
## (Intercept)  0.3581899  0.0434516   8.24342 < 0.000000000000000222 ***
## STARS       -2.2112170  0.0646638 -34.19560 < 0.000000000000000222 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Number of iterations in BFGS optimization: 13
## Log-likelihood: -14603.6 on 7 Df
```

```
rmse.pois.zip = model.fit.evaluate.rmse(model.pois.zip, test.data, test.values)
print(paste("RMSE: ",rmse.pois.zip))
```

```
## [1] "RMSE:  1.34704678284216"
```

We can see that the *Zero-Inflated Poisson* regression model has greatly improved the RMSE, indicating a much better fit for the model.

This is due to the fact that dataset has an unusually high number of cases with zeros for the STARS (indicating either that the wine was not rated, or that was so bad that it merited zero stars.)

This makes this dataset a better fit for ***Zero Inflated Poisson*** regression model.

Note that we need to specify the parameter for Zero Inflated Poisson regression model, specifying which feature contributes to more occurences of cases with zero target variable. The obvious choice here is the STARS variable. It is intuitive to think that if the number of STARS are more it will contribute to increased count of wine sample purchase.

We fit our model designating `STARS` as the parameter which impacts the number of records with zero `TARGET` value and we can see that it this results in smaller RMSE, indicating better model fit compared to the regular Poisson regression model.

9. Fit Zero-Inflated Negative Binomial Regression model

```
## we include the otherwise insignificant variable "FixedAcidity" because
## when omitting it, we get "NANs produced"
model.nb.zip= hurdle(target~  LabelAppeal + STARS + AcidIndex + FixedAcidity|STARS,
                     dist='negbin', data=train.data)
summary(model.nb.zip)
```

```
##
## Call:
## hurdle(formula = target ~ LabelAppeal + STARS + AcidIndex + FixedAcidity | STARS, data = train.data,
##
## Pearson residuals:
##        Min         1Q     Median         3Q        Max
## -2.2494020 -0.5662055  0.0189724  0.4134355  2.9043652
##
## Count model coefficients (truncated negbin with log link):
##                  Estimate   Std. Error  z value               Pr(>|z|)
## (Intercept)   1.249504008  0.047435112 26.34133 < 0.000000000000000222 ***
## LabelAppeal   0.244684157  0.007800967 31.36588 < 0.000000000000000222 ***
## STARS         0.095570765  0.006318988 15.12438 < 0.000000000000000222 ***
## AcidIndex    -0.019342169  0.005955934 -3.24755              0.00116405 **
## FixedAcidity  0.000176682  0.001031383  0.17131              0.86398289
## Log(theta)   16.037285502  4.206721643  3.81230              0.00013768 ***
## Zero hurdle model coefficients (binomial with logit link):
##               Estimate Std. Error  z value               Pr(>|z|)
## (Intercept) -0.4597019  0.0408054 -11.2657 < 0.000000000000000222 ***
## STARS        1.9854252  0.0488461  40.6465 < 0.000000000000000222 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Theta: count = 9223687.872263
## Number of iterations in BFGS optimization: 41
## Log-likelihood: -14534.2 on 8 Df
```

```
rmse.nb.zip = model.fit.evaluate.rmse(model.nb.zip, test.data, test.values)
print(paste("RMSE: ", rmse.nb.zip))
```

```
## [1] "RMSE:  1.34926875702818"
```

We can see that Zero Inflated Poisson regression model has lower RMSE (1.3470) compared to Zero Inflated Negative Binomial model (1.3493).

This is due to the fact that dataset has under-dispersion, and the Negative Binomial model is not recommended when there is under-dispersion in the dataset.

10. Fit Multinomial Logistic Regression model with selected variables

Since this is a Multinomial Regression model, we will use *Accuracy* as a model evaluation metric, as stated in the model evaluation criteria above.

```
train.data.class = train.data
train.data.class$target = factor(train.data.class$target)
model.multi= multinom(target~  LabelAppeal + STARS + AcidIndex ,  data=train.data.class)
```

```
## # weights:  45 (32 variable)
## initial  value 19678.343315
## iter  10 value 14640.922446
## iter  20 value 13955.585567
## iter  30 value 12007.936275
## iter  40 value 11254.145703
## iter  50 value 11216.354868
## iter  60 value 11210.260302
## iter  70 value 11210.165348
## iter  80 value 11210.156725
## iter  90 value 11210.150300
## final  value 11210.142345
## converged
```

```
summary(model.multi)
```

```
## Call:
## multinom(formula = target ~ LabelAppeal + STARS + AcidIndex,
##     data = train.data.class)
##
## Coefficients:
##       (Intercept)  LabelAppeal       STARS    AcidIndex
## 1  -3.56215068481 -3.075346920 0.767408757 -0.202056200
## 2   0.00413791183 -1.933889917 1.371445951 -0.348831255
## 3   1.19124300209 -0.751245793 1.834662275 -0.349697233
## 4   1.57793463304  0.465208442 2.368736604 -0.465323689
## 5   0.30163604977  1.569799368 2.915334855 -0.567381184
## 6  -2.39959679092  2.738225390 3.431219242 -0.645421320
## 7  -6.18361824882  3.819690505 3.903232712 -0.731349595
## 8 -11.56847507202  5.151666255 3.593086018 -0.493250738
##
## Std. Errors:
##   (Intercept)  LabelAppeal        STARS    AcidIndex
## 1 0.584012232 0.1506103666 0.1256249084 0.0674085850
## 2 0.323299793 0.0775804841 0.0669096956 0.0397153532
## 3 0.240314965 0.0546472498 0.0566578261 0.0297279296
## 4 0.254890633 0.0561804511 0.0590491576 0.0320740383
## 5 0.322937599 0.0713114242 0.0665768849 0.0405439577
## 6 0.476068527 0.1043785779 0.0830429304 0.0579611220
## 7 0.955536601 0.2019495604 0.1460022545 0.1106959039
## 8 2.479318677 0.7808219443 0.3550280670 0.2329913466
##
## Residual Deviance: 22420.2847
## AIC: 22484.2847
```

```
accuracy.multi = model.fit.evaluate.mcr(model.multi, test.data, test.values)
print(paste("ACCURACY: ",accuracy.multi))
```

```
## [1] "ACCURACY:  0.489971346704871"
```

From the above Accuracy metric we can say that with the Accuracy score of 0.49, we can predict wine sample purchase count with 49% accuracy.

This is not a great accuracy score.

This is expected since the Multinomial regression model is not relevant for this problem statement.

The Multinomial regression model can be used to predict un-ordered target classes with multiple values.

However in this problem statement we are dealing with a numerical target variable which is wine sample purchase count.

The target variable is ordered here (indeed, it is numeric) and that makes the Poisson regression model more applicable here compared to the Multinomial Logistic Regression model.

11. Fit multiple linear regression model with all the variables

```
model.multilr= lm(target~  .,  data=train.data)
summary(model.multilr)
```

```
##
## Call:
## lm(formula = target ~ ., data = train.data)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -4.568013 -0.892066  0.065170  0.874513  5.982340
##
## Coefficients:
##                         Estimate    Std. Error   t value               Pr(>|t|)
## (Intercept)         3.5612819334  0.5276390186   6.74947      0.000000000015762 ***
## FixedAcidity       -0.0009288688  0.0022334254  -0.41589             0.67749737
## VolatileAcidity    -0.0785318767  0.0178742166  -4.39358      0.000011278281028 ***
## CitricAcid          0.0261836984  0.0162930011   1.60705             0.10807831
## ResidualSugar      -0.0000777723  0.0004129248  -0.18835             0.85061047
## Chlorides          -0.1325216594  0.0435272122  -3.04457             0.00233691 **
## FreeSulfurDioxide   0.0003977059  0.0000943850   4.21366      0.000025373879422 ***
## TotalSulfurDioxide  0.0002162364  0.0000602429   3.58941             0.00033320 ***
## Density            -0.3206330738  0.5188115043  -0.61801             0.53658147
```

```
## pH                -0.0485387239  0.0204426577  -2.37438             0.01759924 *
## Sulphates         -0.0422617871  0.0149595047  -2.82508             0.00473737 **
## Alcohol            0.0110918280  0.0037156366   2.98518             0.00284183 **
## LabelAppeal        0.4433765108  0.0275717139  16.08085 < 0.000000000000000222 ***
## AcidIndex         -0.2401506896  0.0140740009 -17.06343 < 0.000000000000000222 ***
## STARS              0.7863156094  0.0339053982  23.19146 < 0.000000000000000222 ***
## StarTran           0.1913928324  0.0341283070   5.60804      0.000000021072532 ***
## AcidIndexTran      0.2019967355  0.0540462702   3.73748             0.00018703 ***
## LabelAppealTran    0.0199143383  0.0446144809   0.44636             0.65534450
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.31294 on 8938 degrees of freedom
## Multiple R-squared:  0.532097,   Adjusted R-squared:  0.531207
## F-statistic: 597.896 on 17 and 8938 DF,  p-value: < 0.000000000000000222
```

```
rmse.multilr = model.fit.evaluate.rmse(model.multilr, test.data, test.values)
print(paste("RMSE: ", rmse.multilr))
```

```
## [1] "RMSE:  1.37271442094835"
```

12. Fit multiple linear regression model with selected variables and cubic transformation of Stars variable

```
model.multilr.sel= lm(target~  LabelAppeal + poly(STARS,3) + AcidIndex ,  data=train.data)
summary(model.multilr.sel)
```

```
##
## Call:
## lm(formula = target ~ LabelAppeal + poly(STARS, 3) + AcidIndex,
##     data = train.data)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -4.804154 -0.856520  0.048103  0.833110  6.221630
##
## Coefficients:
##                     Estimate  Std. Error   t value               Pr(>|t|)
## (Intercept)        4.6439260   0.0833586  55.71020 < 0.000000000000000222 ***
## LabelAppeal        0.4738166   0.0162521  29.15420 < 0.000000000000000222 ***
## poly(STARS, 3)1 108.7478720   1.3822072  78.67697 < 0.000000000000000222 ***
## poly(STARS, 3)2 -20.0755748   1.3193231 -15.21657 < 0.000000000000000222 ***
## poly(STARS, 3)3   3.6933778   1.3092642   2.82096              0.0047986 **
## AcidIndex        -0.2064560   0.0105757 -19.52179 < 0.000000000000000222 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.30488 on 8950 degrees of freedom
## Multiple R-squared:  0.537206,   Adjusted R-squared:  0.536948
## F-statistic: 2077.81 on 5 and 8950 DF,  p-value: < 0.000000000000000222
```

```
rmse.multilr.sel = model.fit.evaluate.rmse(model.multilr.sel, test.data, test.values)
print(rmse.multilr.sel)
```

```
## [1] 1.36643804
```

Interestingly we are getting a very close RMSE between the Zero Inflated Poisson Regression model and Multiple Linear regression model with selected variables (plus the cubic transformed Stars variable).

This is possible here because the distribution of the target variable was not strictly Poisson.

The distribution looked more like a bimodal with skew on the right side.

We can say that both Zero-Inflated Poisson regression models perform as well as the multiple linear regression model with cubic term of Stars variable.

4. Select Models

From the above model fits we can conclude the following:

- For ***Inference*** we select the **Robust Poisson Regression** model. We are selecting this model for inference even though it doesn't have great RMSE. We are doing this because:
- Robust Poisson regression model is interpretable compared to Zero-Inflated Poisson regression model, and
- Std Error estimation is better for Robust Poisson regression model. This is more important here because the dataset has under-dispersion, and we need to stick to the Robust Poisson regression model for inference rather than the simple Poisson regression model.
- For ***Prediction*** we select the **Zero Inflated Poisson** Regression model.
- We are selecting this model because it has **lower RMSE**.
- This not only indicates better model fit, but it also makes this model a perfect fit for the given dataset because this dataset has an unusually high number of wine sample sale observations with zero count, as well as an unusually high number of wines which were unrated (STARS=NA), which we decided to replace with zeros.

The Zero-Inflated Poisson regression model can handle such datasets well and can greatly improve prediction accuracy by taking into account features which can influence the occurence of zero records while making predictions.

1. Model Inference

Robust Poisson Regression

```
summary(model.pois.sel.robust)
```

```
##
## Call:
## glm(formula = target ~ LabelAppeal + STARS + AcidIndex, family = quasipoisson,
##     data = train.data)
##
## Deviance Residuals:
##       Min         1Q     Median         3Q        Max
## -2.991058  -0.681008   0.055445   0.558621   3.266691
##
## Coefficients:
##                Estimate  Std. Error  t value               Pr(>|t|)
## (Intercept)  1.23215625  0.04002509  30.7846 < 0.000000000000000222 ***
## LabelAppeal  0.13491802  0.00662776  20.3565 < 0.000000000000000222 ***
## STARS        0.31066703  0.00496789  62.5350 < 0.000000000000000222 ***
## AcidIndex   -0.08931702  0.00488414 -18.2872 < 0.000000000000000222 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for quasipoisson family taken to be 0.841165976)
##
##     Null deviance: 15819.65  on 8955  degrees of freedom
## Residual deviance: 10217.58  on 8952  degrees of freedom
## AIC: NA
##
## Number of Fisher Scoring iterations: 5
```

From the above model summary we can say that following predictors are statistically significant while predicting target sample wine sale count:

- **LabelAppeal**: Positive coefficient indicates that a one unit increase in Label Appeal **increases** the wine sample count
- **STARS**: Positive coefficient indicates that a one unit increase in STARS will **increase** the wine sample sale count
- **AcidIndex**: Negative coefficient indicates that a one unit increase in Acid index will **reduce** the wine sample sale count

This goes with our intuition as well. The above coefficients indicate that better LabelAppeal and STARS ratings increase the wine sample sale count.

We can say that people don't like wine with high acidity. As AcidIndex increases, it negatively impacts wine sample sale count.

The coefficient for Stars is 0.31. This indicates that with all other coefficients held constant, one unit increase in Stars increases log of wine sale by 0.31 .

Calculate impact of STARS count on actual wine sale: $e^{0.31}$

```
exp(0.31)
```

```
## [1] 1.36342511
```

This analysis indicates that with all other coefficient held constant, one unit increase in `STARS` increases the wine sample sale count by 36.3%. This is a significant impact and that's why the STARS rating is very important for wine sales – restaurants are more interested in stocking higher-rated wines because they don't want to disappoint their diners with unpalatable wines.

The coefficient for LabelAppeal is 0.1349. This indicates that with all other coefficients held constant, one unit increase in LabelAppeal increases log of wine sale by 0.1349 .

Calculate impact of LabelAppeal on wine sale: $e^{0.1349}$

```
exp(0.1349)
```

```
## [1] 1.14442234
```

This analysis indicates that with all other variables held constant, a one unit increase in Label Appeal increases wine sample sale count by 14.4%.

The coefficient for AcidIndex is -0.0893. This indicates that with all other coefficients held constant, one unit increase in AcidIndex increases log of wine sale by -0.0893 .

Calculate impact of AcidIndex on wine sale: $e^{-0.0893}$

```
exp(-0.0893)
```

```
## [1] 0.914571161
```

This analysis indicates that with all other variables held constant, a one unit increase in AcidIndex DECREASES wine sample sale count by 8.54%.

2. Model Prediction

For *prediction* we will use the Zero Inflated Poisson Regression model due to its lower RMSE and better model fit.

```
summary(model.pois.zip)
```

```
##
## Call:
## zeroinfl(formula = target ~ LabelAppeal + STARS + AcidIndex + FixedAcidity | STARS, data = train.data
##
## Pearson residuals:
##        Min         1Q     Median         3Q        Max
## -2.3031284 -0.5342597  0.0231726  0.4092066  2.9253167
##
## Count model coefficients (poisson with log link):
##                   Estimate   Std. Error  z value               Pr(>|z|)
## (Intercept)   1.358527526  0.047268000 28.74096 < 0.000000000000000222 ***
## LabelAppeal   0.224993670  0.007540099 29.83962 < 0.000000000000000222 ***
## STARS         0.101955422  0.006249841 16.31328 < 0.000000000000000222 ***
## AcidIndex    -0.033575977  0.006022439 -5.57515         0.000000024732 ***
## FixedAcidity  0.000220373  0.001001752  0.21999                0.82588
##
## Zero-inflation model coefficients (binomial with logit link):
##               Estimate Std. Error   z value               Pr(>|z|)
## (Intercept)  0.3581899  0.0434516   8.24342 < 0.000000000000000222 ***
## STARS       -2.2112170  0.0646638 -34.19560 < 0.000000000000000222 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Number of iterations in BFGS optimization: 13
## Log-likelihood: -14603.6 on 7 Df
```

These coefficients differ from the those in the regular Poisson regression model because the Zero Inflation component has separated out those cases for which the target values are "Certain Zeros", as driven by the STARS value.

From the Zero-Inflation model, the intercept 0.3581899 represents log(OddsRatio) that TARGET=0 if STARS=0.
Therefore, the OddsRatio of this occuring is $e^{0.3581899} = 1.4307$, which indicates an increased likelihood of occurance since OddsRatio > 1.
The coefficient -2.2112170 represents the change to log(Odds Ratio) with each added STAR rating for the wine. So when a wine has a 1 star rating, the log(OddsRatio) becomes 0.3581899 - 2.2112170 = -1.8520271, which means the Odds Ratio becomes $e^{-1.8520271} = 0.1569$ which indicates an unlikely occurence because this OddsRatio < 1.

Once such cases are removed, the above Poisson model shows different coefficients, where a one-unit increase in LabelAppeal indicates a 0.225 increase in log(wine sales), which translates to a 25.23 percent increase in wine sales.

After removal of the Zero-Inflated cases, the Poisson model coefficient of 0.101955 on STARS indicates that an increase in rating of 1 star indicates a 0.101955 increase in log(wine sales), which translates to a 10.73 percent increase in wine sales.

3. Read evaluation data and impute missing columns.

```
evaluation = read.csv("./Data/wine-evaluation-data.csv",
                      stringsAsFactors = FALSE,
                      na.strings=c("NA","NaN", " ", ""))

#### How many cases in the evaluation data with missing STARS ?
summary(evaluation$STARS)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##  1.0000  1.0000  2.0000  2.0401  3.0000  4.0000     841
```

```
table(evaluation$STARS,useNA = "ifany")
```

```
##
##    1    2    3    4 <NA>
##  828  902  600  164  841
```

```
#### Let's replace NA STARS values with **zero**, before imputing other missing variables:
evaluation$STARS[is.na(evaluation$STARS)]=0

#### What does STARS data look like now ?
summary(evaluation$STARS)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.00000 0.00000 1.00000 1.52564 2.00000 4.00000
```

```
table(evaluation$STARS,useNA = "ifany")
```

```
##
##   0   1   2   3   4
## 841 828 902 600 164
```

```
#### Making this change has reduced the mean STARS value from 2.04 to 1.53,
#### and the median from 2 to 1.

#### Impute the other missing data points
comp.data.evaluation <- mice(evaluation,m=2,maxit=10,meth='pmm',seed=500)
```

```
##
##  iter imp variable
##    1   1  ResidualSugar  Chlorides  FreeSulfurDioxide  TotalSulfurDioxide  pH  Sulphates  Alcohol
##    1   2  ResidualSugar  Chlorides  FreeSulfurDioxide  TotalSulfurDioxide  pH  Sulphates  Alcohol
##    2   1  ResidualSugar  Chlorides  FreeSulfurDioxide  TotalSulfurDioxide  pH  Sulphates  Alcohol
##    2   2  ResidualSugar  Chlorides  FreeSulfurDioxide  TotalSulfurDioxide  pH  Sulphates  Alcohol
##    3   1  ResidualSugar  Chlorides  FreeSulfurDioxide  TotalSulfurDioxide  pH  Sulphates  Alcohol
##    3   2  ResidualSugar  Chlorides  FreeSulfurDioxide  TotalSulfurDioxide  pH  Sulphates  Alcohol
##    4   1  ResidualSugar  Chlorides  FreeSulfurDioxide  TotalSulfurDioxide  pH  Sulphates  Alcohol
##    4   2  ResidualSugar  Chlorides  FreeSulfurDioxide  TotalSulfurDioxide  pH  Sulphates  Alcohol
```

```
##   5   1  ResidualSugar  Chlorides  FreeSulfurDioxide  TotalSulfurDioxide  pH  Sulphates  Alcohol
##   5   2  ResidualSugar  Chlorides  FreeSulfurDioxide  TotalSulfurDioxide  pH  Sulphates  Alcohol
##   6   1  ResidualSugar  Chlorides  FreeSulfurDioxide  TotalSulfurDioxide  pH  Sulphates  Alcohol
##   6   2  ResidualSugar  Chlorides  FreeSulfurDioxide  TotalSulfurDioxide  pH  Sulphates  Alcohol
##   7   1  ResidualSugar  Chlorides  FreeSulfurDioxide  TotalSulfurDioxide  pH  Sulphates  Alcohol
##   7   2  ResidualSugar  Chlorides  FreeSulfurDioxide  TotalSulfurDioxide  pH  Sulphates  Alcohol
##   8   1  ResidualSugar  Chlorides  FreeSulfurDioxide  TotalSulfurDioxide  pH  Sulphates  Alcohol
##   8   2  ResidualSugar  Chlorides  FreeSulfurDioxide  TotalSulfurDioxide  pH  Sulphates  Alcohol
##   9   1  ResidualSugar  Chlorides  FreeSulfurDioxide  TotalSulfurDioxide  pH  Sulphates  Alcohol
##   9   2  ResidualSugar  Chlorides  FreeSulfurDioxide  TotalSulfurDioxide  pH  Sulphates  Alcohol
##   10   1  ResidualSugar  Chlorides  FreeSulfurDioxide  TotalSulfurDioxide  pH  Sulphates  Alcohol
##   10   2  ResidualSugar  Chlorides  FreeSulfurDioxide  TotalSulfurDioxide  pH  Sulphates  Alcohol

## Warning: Number of logged events: 1
```

```
evaluation = complete(comp.data.evaluation)
```

4. Perform prediction and write out the results

```
#### Perform the prediction
out = predict(model.pois.zip, evaluation)
evaluation$TARGET = round(out)

#### write the results
write.table(evaluation, "./Data/PredictedOutcome_HW5.csv", row.names = FALSE, sep=",")
```

5. Appendix

```r
model.fit.evaluate.mcr <- function(model, test.data, test.values) {
  output = predict(model, test.data)
  mat = table(output, test.values)
  mcr = sum(diag(mat))/sum(mat)
  return(mcr)
}

model.fit.evaluate.rmse <- function(model, test.data, test.values) {
  output = predict(model, test.data)
  rmse = sqrt(sum((output - test.values)^2)/length(test.values))
}
```