read in our data

```
library(glmnet)
```

```
## Warning: package 'glmnet' was built under R version 4.0.2
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.0-2
```

```
library(tidyverse)
```

```
## Warning: package 'tidyverse' was built under R version 4.0.2
```

```
## -- Attaching packages ------------------------- tidyverse 1.3.0 --
```

```
## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.1     v dplyr   1.0.0
## v tidyr   1.1.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0
```

```
## Warning: package 'ggplot2' was built under R version 4.0.2
```

```
## Warning: package 'tibble' was built under R version 4.0.2
```

```
## Warning: package 'tidyr' was built under R version 4.0.2
```

```
## Warning: package 'readr' was built under R version 4.0.2
```

```
## Warning: package 'purrr' was built under R version 4.0.2
```

```
## Warning: package 'dplyr' was built under R version 4.0.2
```

```
## Warning: package 'stringr' was built under R version 4.0.2
```

```
## Warning: package 'forcats' was built under R version 4.0.2
```

```
## -- Conflicts ---------------------------- tidyverse_conflicts() --
## x tidyr::expand() masks Matrix::expand()
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## x tidyr::pack()   masks Matrix::pack()
## x tidyr::unpack() masks Matrix::unpack()
```

```
library(ggplot2)
train = read.csv("train_Madison.csv")
```

convert word counts to numeric percentage and conver star data to numeric

```
#convert stars to number
train[ , 2] = suppressWarnings(as.numeric(as.character(train[ , 2])))
##convert all numbers to percenatges
for (i in 7:length(train)) {
  train[ , i] = suppressWarnings(as.numeric(as.character(train[ , i])))
}
for (i in 9:length(train)) {
  train[ , i] = suppressWarnings(train[ , i] / train[ , 8])
}
```

remove any NA's in the training data set. Add an indicator for Madison.

```
train = train[(!is.na(train[ , i])), ]
train$Madison = c()
## add a column to contain our binomial madison or not variable.
for (i in 1:(nrow(train))) {
  if (train$city[i] == "Madison") {
    train$Madison[i]= 1
  } else {
    train$Madison[i] = 0
  }
}
```

Here we look at the the correlation for all words with star values and see how many are positive and negative.

```
positiveCorr.ID = c()
negativeCorr.ID = c()
##run a for loop to look at the postivie and negative correlations
for (i in 7:length(train)) {
  if (cor(train[ , 2],train[ , i]) > 0) {
    positiveCorr.ID = c(positiveCorr.ID, i)
  } else {
    negativeCorr.ID = c(negativeCorr.ID, i)
  }
}

##check number in each group
length(negativeCorr.ID)
```

```
## [1] 254
```

```
length(positiveCorr.ID)
```

```
## [1] 249
```

Here we can look at the absolute values of the correlation between each word and the star rating. There are very clearly a large number of words that have very little correlation: 105 words have less then .02 correlation. Slightly more words, 145, have comparitively high correlations above .05. The remaining 252 words comprimise the middle of the chart and land somewhere between .02 and .05. The bowed nature of our graph suggests that this data has been previously manipulated or sorted

```
corrValues=c()
for (i in 7:length(train)) {
 corrValues=c(corrValues,cor(train[ , 2],train[ , i]))
}

absCorrValues=abs(corrValues)
LowCorr<-which(absCorrValues<.02)
highCorr<-which(absCorrValues>=.05)

print("number of high correalted words")
```

```
## [1] "number of high correalted words"
```

```
length(highCorr)
```

```
## [1] 146
```

```
print("number of low correalted words")
```

```
## [1] "number of low correalted words"
```

```
length(LowCorr)
```

```
## [1] 105
```

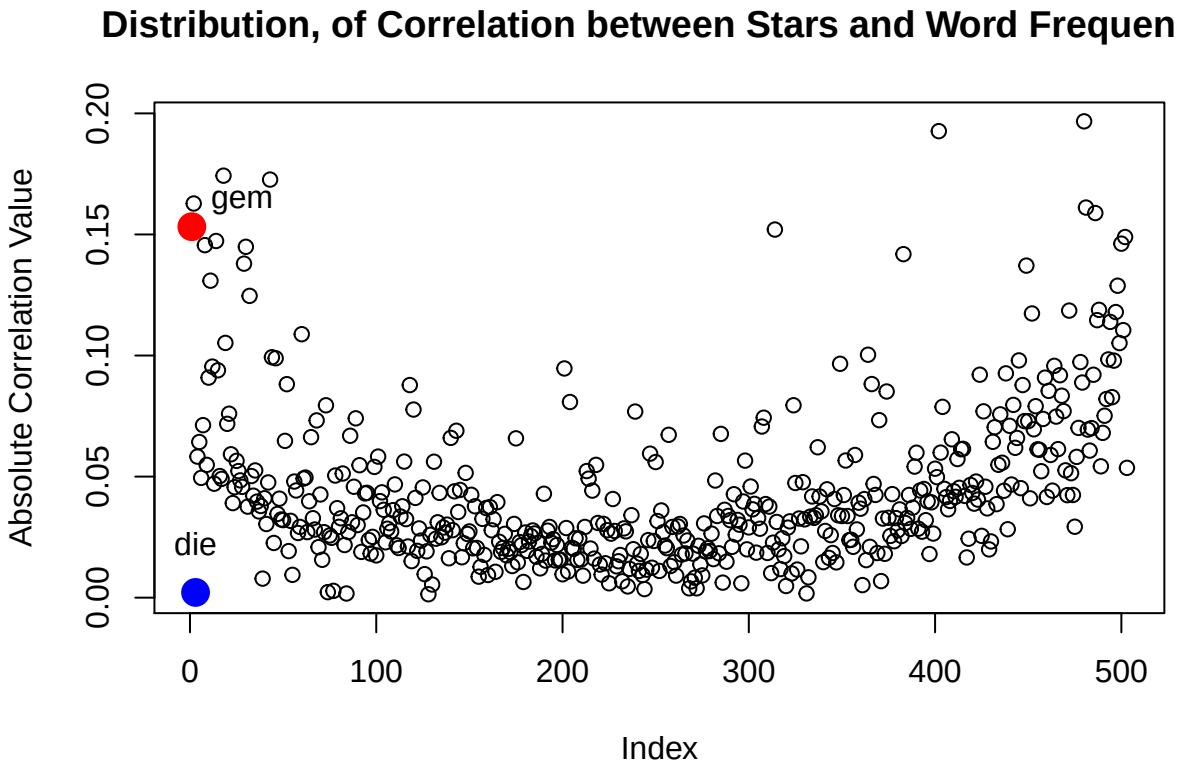
```
print("number of medium correalted words")
```

```
## [1] "number of medium correalted words"
```

```
length(absCorrValues)-(length(LowCorr)+length(highCorr))
```

```
## [1] 252
```

```
plot(absCorrValues,main="Distribution, of Correlation between Stars and Word Frequency",ylab=("Absolute
points(3,.00215,col="blue",cex=2,pch=16)
points(1,0.1532005,col="red",cex=2,pch=16)
text(28,0.1632005,"gem")
text(3,.02215,"die")
```



Here we will check our top 5 smallest and largest correlation to sanity check our results. We hope to see results that confirm our intutions. If Garbage is highly correlated with 5 stars, we will need to check our results but if gem is higly correlated with 5 stars, we will gain confidence.

```
#here we willcomment out the order command to keep the output clean and fresh. But if you were running
negativehigh.ID = c(486, 408, 487, 492, 320, 508)
negativehigh.name = colnames(train[ , negativehigh.ID])
negativehigh.cor = c()
for (i in 1:6) {
  cor.i = cor(train[ , 2], train[ , negativehigh.ID[i]], use = "complete.obs")
  negativehigh.cor[i] = cor.i
}
df.negative = data.frame(Variable = negativehigh.name, Correlation = negativehigh.cor)
##positively correlated Id numbers into a vector
positivehigh.ID = c(24, 49, 20, 14, 36, 35)
positivehigh.name = colnames(train[ , positivehigh.ID])
positivehigh.cor = c()
for (i in 1:6) {
  cor.i = cor(train[ , 2], train[ , positivehigh.ID[i]], use = "complete.obs")
  positivehigh.cor[i] = cor.i
}
df.positive = data.frame(Variable = positivehigh.name, Correlation = positivehigh.cor)

##order(corrValues)
print("here are our most negatively correlated words and their correlation")
```

```
## [1] "here are our most negatively correlated words and their correlation"
```

```
print(df.negative)
```

```
##   Variable Correlation
## 1  minutes  -0.1967165
## 2   minute  -0.1926718
## 3    asked  -0.1611139
## 4     told  -0.1588440
## 5    order  -0.1520287
## 6    worst  -0.1489433
```

```
#here we willcomment out the order command to keep the output clean and fresh. But if you were running
##order(corrValues,decreasing=TRUE)
print("here are our most postively correlated words and their correlation")
```

```
## [1] "here are our most postively correlated words and their correlation"
```

```
print(df.positive)
```

```
##    Variable Correlation
## 1 delicious   0.1742814
## 2     great   0.1726655
## 3  favorite   0.1473058
## 4   amazing   0.1455803
## 5      love   0.1448994
## 6      best   0.1379523
```

We tried to construct MLR models based on the variables with highest correlation and combination of mediocre variables.

```
# sort the correlation of the positive variables
positiveCorr.cor = c()
for (i in positiveCorr.ID) {
  cor.i = cor(train[ , 2], train[ , i], use = "complete.obs")
  positiveCorr.cor[i] = cor.i
}
positiveCorr.order = order(positiveCorr.cor, decreasing = TRUE)

# sort the correlation of the negative variables
negativeCorr.cor = c()
for (i in negativeCorr.ID) {
  cor.i = cor(train[ , 2], train[ , i], use = "complete.obs")
  negativeCorr.cor[i] = cor.i
}
negativeCorr.order = order(negativeCorr.cor, decreasing = TRUE)

# combined the 100 variables after 5 in each group

positivemedium.ID = positiveCorr.order[6:105]
train$p.medium.combined = rowSums(train[ , positivemedium.ID])
```

```r
negativemedium.ID = negativeCorr.order[6:105]
train$n.medium.combined = rowSums(train[ , negativemedium.ID])

# construct a new model with the combined numbers
lm.c.m = lm(train[ , 2] ~ train[ , 24] + train[ , 49] + train[ , 20] + train[ , 14] + train[ , 36] +
                          train[ , 486] + train[ , 408] + train[ , 487] + train[ , 492] + train[ , 320]
summary(lm.c.m)
```

```
##
## Call:
## lm(formula = train[, 2] ~ train[, 24] + train[, 49] + train[,
##     20] + train[, 14] + train[, 36] + train[, 486] + train[,
##     408] + train[, 487] + train[, 492] + train[, 320] + train[,
##     510] + train[, 511])
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.9089 -0.7169  0.2061  0.9483  5.7046
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)    3.66870    0.01509 243.076  < 2e-16 ***
## train[, 24]   31.30387    0.74200  42.188  < 2e-16 ***
## train[, 49]   13.82621    0.43516  31.773  < 2e-16 ***
## train[, 20]   24.99939    0.94095  26.568  < 2e-16 ***
## train[, 14]   23.79392    0.84507  28.156  < 2e-16 ***
## train[, 36]   17.79798    0.67951  26.192  < 2e-16 ***
## train[, 486] -35.69223    4.18969  -8.519  < 2e-16 ***
## train[, 408] -11.74981    3.86032  -3.044  0.00234 **
## train[, 487] -62.17819    2.25308 -27.597  < 2e-16 ***
## train[, 492] -58.60486    2.48230 -23.609  < 2e-16 ***
## train[, 320] -13.73699    0.63533 -21.622  < 2e-16 ***
## train[, 510]   0.25265    0.01242  20.341  < 2e-16 ***
## train[, 511]  -5.58113    0.18026 -30.962  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.162 on 51313 degrees of freedom
## Multiple R-squared:  0.1978, Adjusted R-squared:  0.1976
## F-statistic:  1054 on 12 and 51313 DF,  p-value: < 2.2e-16
```

The MLR model we constructed have a very low r squared and cannot beat the brenchmark in Kaggle. Thus we choose to use Lasso.

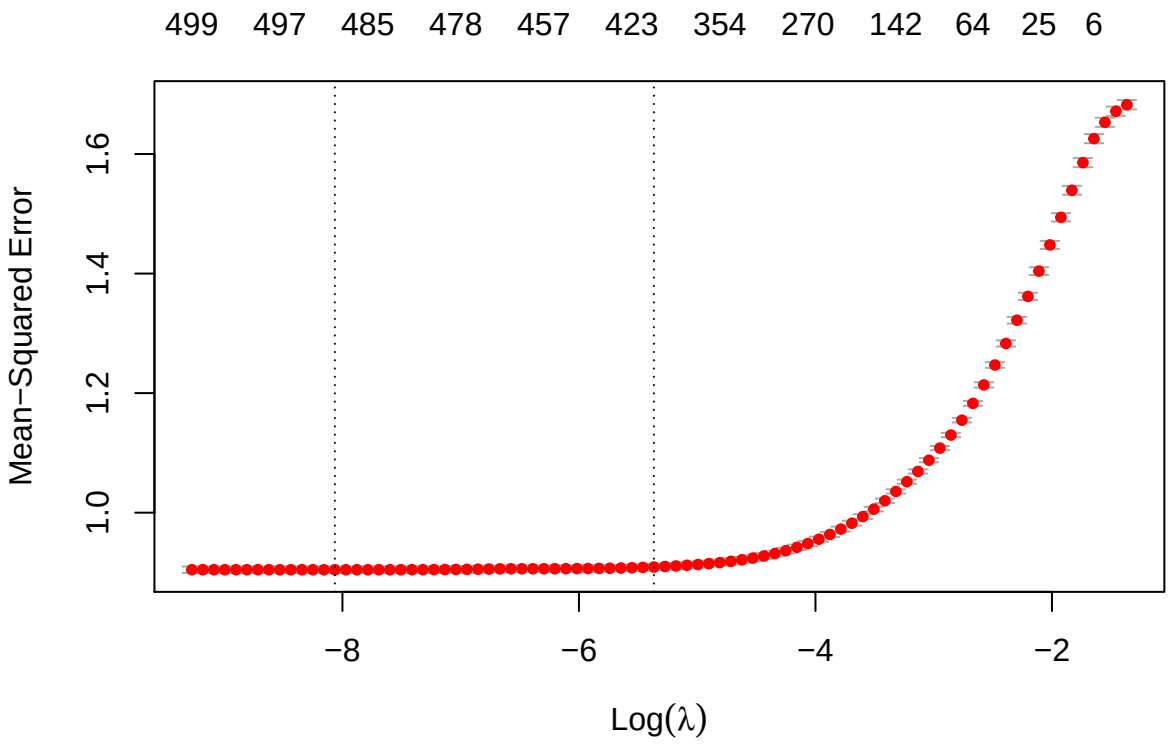
Please note. Here we comment out the code to load in our model and instead rerun the code to create the model. The results may not perfectly match the results described in the paper and rcode as that was written with a particular, saved lasso model. This code will create a new model.

```r
# Removes factors
x = as.matrix(train[,c(-2,-3,-4,-5,-6,-1, -510, -511)])
# single out star
y = as.integer(train[, 2])
# construct the Lasso model
 cv.lasso = cv.glmnet(x, y, family='gaussian', alpha=1, parallel=TRUE, standardize=TRUE, type.measure='
```

```
## Warning: executing %dopar% sequentially: no parallel backend registered
```

```
#load("cv.lasso.rda")

 #add two plots about cv.lasso
 plot(cv.lasso)
```




```
plot(cv.lasso$glmnet.fit,xvar="lambda",label=TRUE)
#find and add text names
colnames( x)[27]
```

```
## [1] "glad"
```

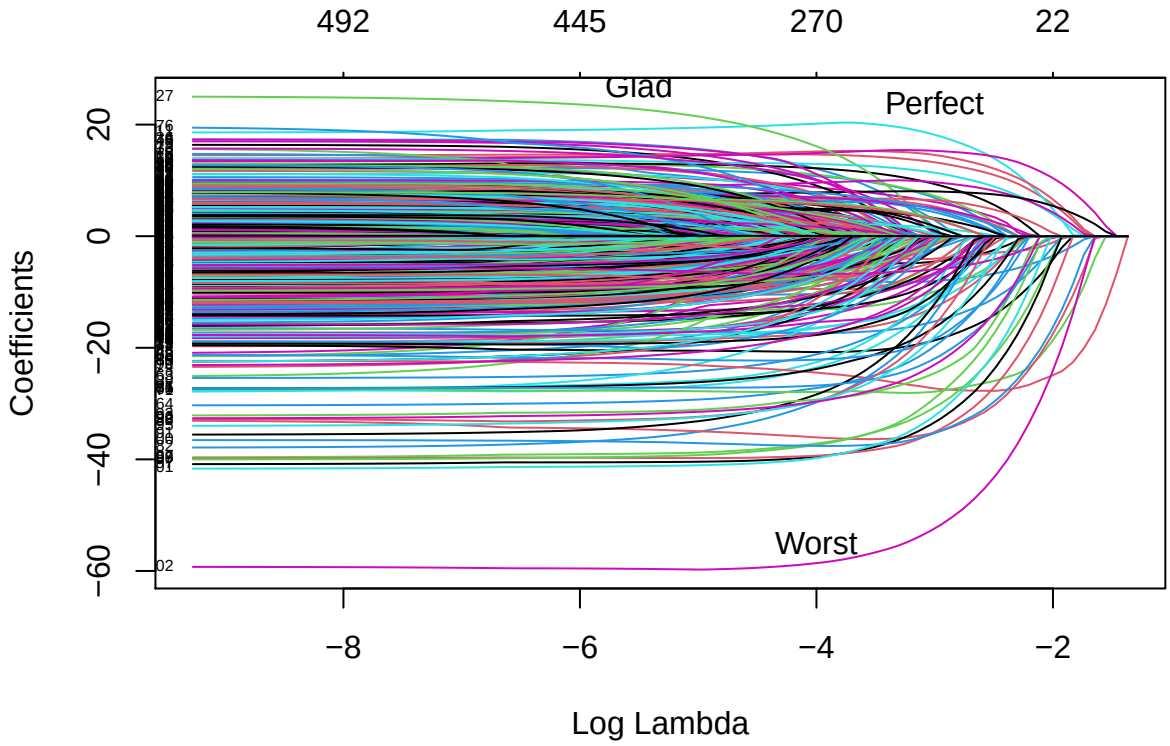
```
colnames( x)[502]
```

```
## [1] "worst"
```

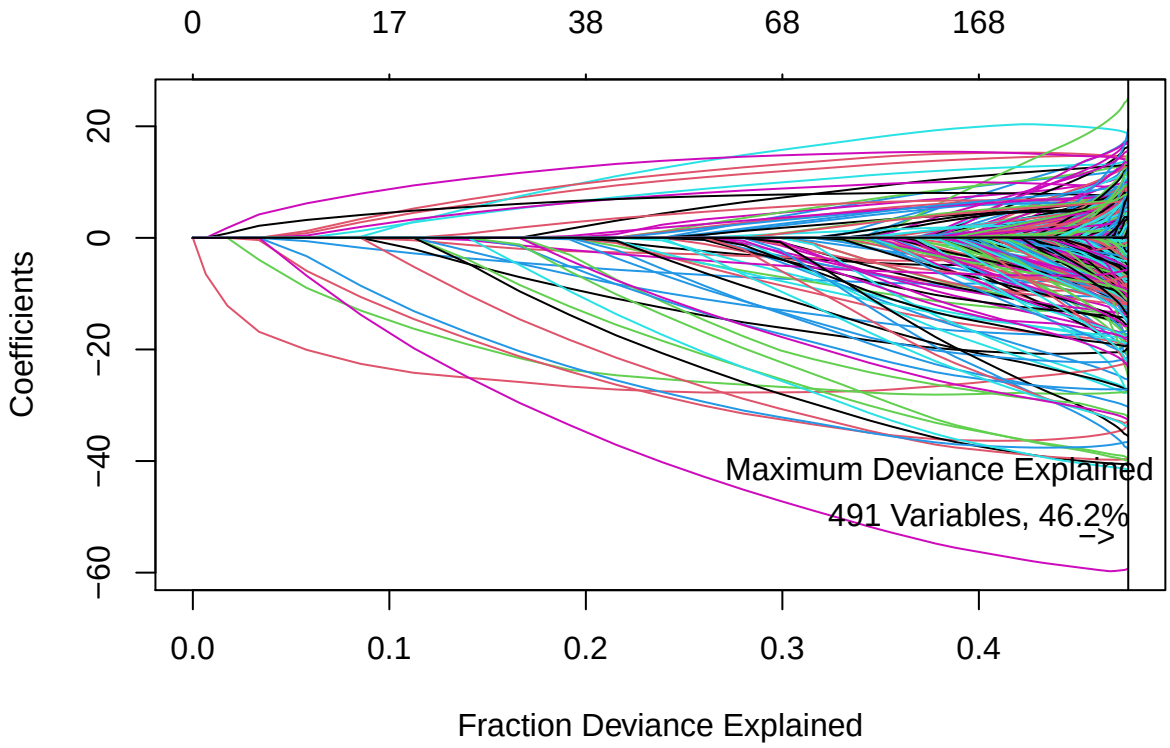
```
colnames( x)[11]
```

```
## [1] "perfect"
```

```
text(-4,-55,"Worst")
text(-5.5,27,"Glad")
text(-3,24,"Perfect")
```




```
## count number of coefficents set equal to 0 for both models.
plot(cv.lasso$glmnet.fit,xvar="dev",label=FALSE)
abline(abline(v = .476))
text(.38,-42,"Maximum Deviance Explained")
text(.40,-50,"491 Variables, 46.2%")
text(.46,-54,"->")
```



First we print out the mse for the absolute min mse. Then we print out the lambda for the 1 SE adjustment the authors suggest. So moving 1 SE increase our mse by about .044. Next we square root them to derive RMSE

```
##find lamda with min mse and pull its mse out
min=which(cv.lasso$lambda==cv.lasso$lambda.min)
mse.min<-cv.lasso$cvm[min]
##MSE and RMSE and adj R^2
print("MSE AND RMSE")
```

```
## [1] "MSE AND RMSE"
```

```
mse.min
```

```
## [1] 0.9044904
```

```
sqrt(mse.min)
```

```
## [1] 0.951047
```

```
#calculate r2
Min.rsq = 1 - cv.lasso$cvm[min]/var(y)
print("R^2")
```

```
## [1] "R^2"
```

```
Min.rsq
```

```
## [1] 0.4625313
```

```
##find lamda within 1 SE of the min MSE and pull its MSE out
onese=which(cv.lasso$lambda==cv.lasso$lambda.1se)
mse.onese<-cv.lasso$cvm[onese]
##MSE and RMSE and adj R^2
print("MSE AND RMSE")
```

```
## [1] "MSE AND RMSE"
```

```
mse.onese
```

```
## [1] 0.9090916
```

```
sqrt(mse.onese)
```

```
## [1] 0.9534629
```

```
#calculate r2
One.Se.rsq = 1 - cv.lasso$cvm[onese]/var(y)
print("R^2")
```

```
## [1] "R^2"
```

```
One.Se.rsq
```

```
## [1] 0.4597972
```

Next we print out the coefficents for each of the two lambdas. This shows only the head but more can be shown if head is removed.

```
##create a data type to hold coefficents for each lamda type
CoefForMIN=as.matrix(coef(cv.lasso, cv.lasso$lambda.min))
CoefFor1SE=as.matrix(coef(cv.lasso, cv.lasso$lambda.1se))
##print out coefficent tables
print("Coefficients of the MIN model")
```

```
## [1] "Coefficients of the MIN model"
```

```
head(CoefForMIN)
```

```
##                        1
## (Intercept)  3.757453939
## nchar        0.001050279
## nword       -0.005969495
## gem         -0.796631799
## incredible  17.298365388
## die          9.696949210
```

```
print("Coefficients of the 1SE model")
```

```
## [1] "Coefficients of the 1SE model"
```

```
head(CoefFor1SE)
```

```
##                          1
## (Intercept)  3.7539554778
## nchar        0.0000000000
## nword       -0.0003265998
## gem          0.0000000000
## incredible  16.2524029553
## die          7.0706126006
```

Look at how many variables are set to 0 for each lambda value.

```
## count number of coefficents set equal to 0 for both models.
removedMIN=which(CoefForMIN==0)
removed1se=which(CoefFor1SE==0)
print("number of removed variables in Min MSE model")
```

```
## [1] "number of removed variables in Min MSE model"
```

```
length(removedMIN)
```

```
## [1] 11
```

```
print("number of removed variables in 1 SE model")
```

```
## [1] "number of removed variables in 1 SE model"
```

```
length(removed1se)
```

```
## [1] 94
```

next we will save our model to preserve it.

```
#save model
 save(cv.lasso, file = "cv.lasso.rda")
```

read in our test data, convert word counts to numeric percentage and conver star data to numeric. remove any NA's in the training data set. create a new binary variable called Madison. This will replace the current Madison column that denotes the frequency of madison in text. This new varaible will be a 1 if a city is Madison and 0 otherwise.

```
library(tidyverse)
test <- read_csv("test_Madison.csv")
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   name = col_character(),
##   city = col_character(),
##   text = col_character()
## )

## See spec(...) for full column specifications.
```

```
#convert our numbers to percentages
for (i in 8:507) {
  test[ , i] = suppressWarnings(test[ , i] / test[ , 7])
}
```

create a new binary variable called Madison. This will replace the current Madison column that denotes the frequency of madison in text. This new varaible will be a 1 if a city is Madison and 0 otherwise.

```
#create a madison binomail varialbe in our test data
test$Madison = c()
for (i in 1:(nrow(test))) {
  if (test$city[i] == "Madison") {
    test$Madison[i]= 1
  } else {
    test$Madison[i] = 0
  }
}
```

```
## Warning: Unknown or uninitialised column: 'Madison'.
```

Add an empty column to replace the deleted column, create a test matrix and then use our fitted model to predict y.

```
#add colmn to repalce star column
test = test[ , c(1, 508, 2:507)]

#create our x matrix to plug into model
testMatrix = as.matrix(test[,c(-3,-4,-5,-6,-1)])
prdictedY = predict(cv.lasso, testMatrix)
q = prdictedY
#turncate stars
for (i in 1:(length(q))) {
  if (q[i] > 5) {
    q[i] = 5
  } else if (q[i] < 1) {
    q[i] = 1
  }
}
YOUR_PREDICTIONS <- q
##create kaggle submissions
out_df <- tibble(
  Id = test$Id,
  Expected = YOUR_PREDICTIONS
)
```

Draw the histogram, residual plot and QQ-plot of the model based on train dataset. As this is a task of prediction and we have a very large data set, we will not consider Cook??s distance, Pii and Leverage-Influence plots.R also currently lacks the capabilities to create these models.

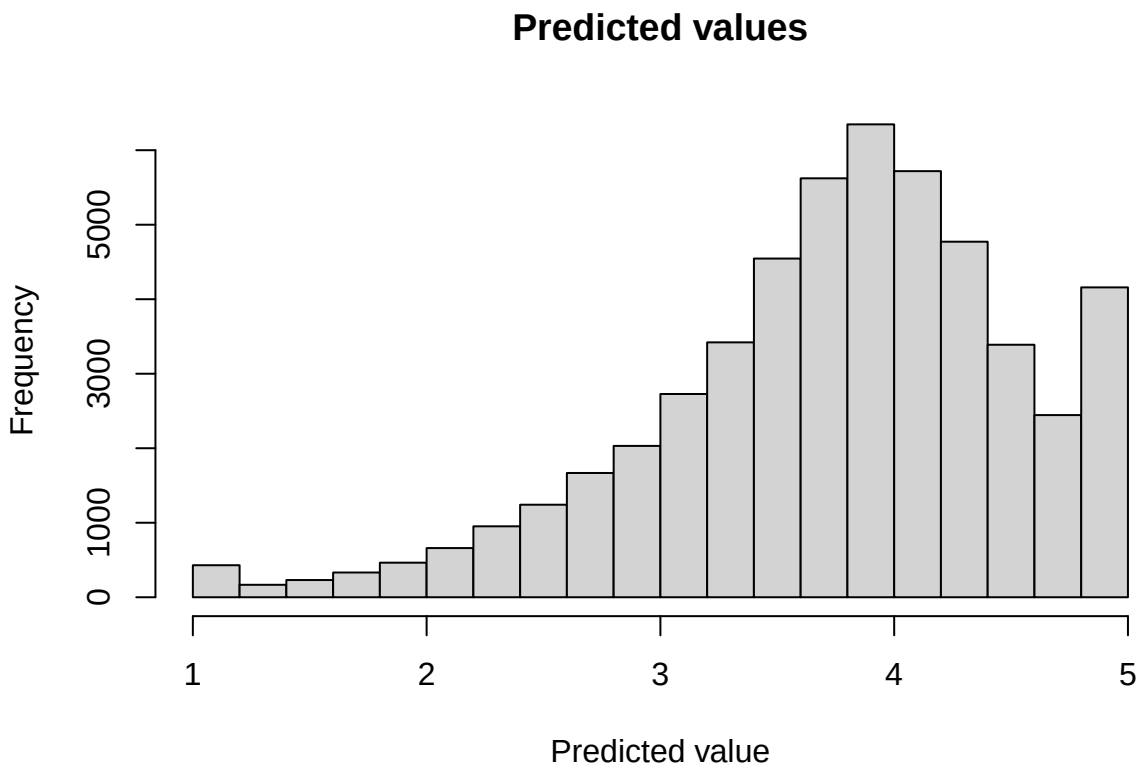
```
p = predict(cv.lasso, x)
for (i in 1:(length(p))) {
  if (p[i] > 5) {
    p[i] = 5
  } else if (p[i] < 1) {
    p[i] = 1
  }
}
##create grrpahs and look at predictions.
star = train$star
residual = p - star
summary(p)
```

```
##        1
##  Min.   :1.000
##  1st Qu.:3.318
##  Median :3.836
##  Mean   :3.743
##  3rd Qu.:4.276
##  Max.   :5.000
```

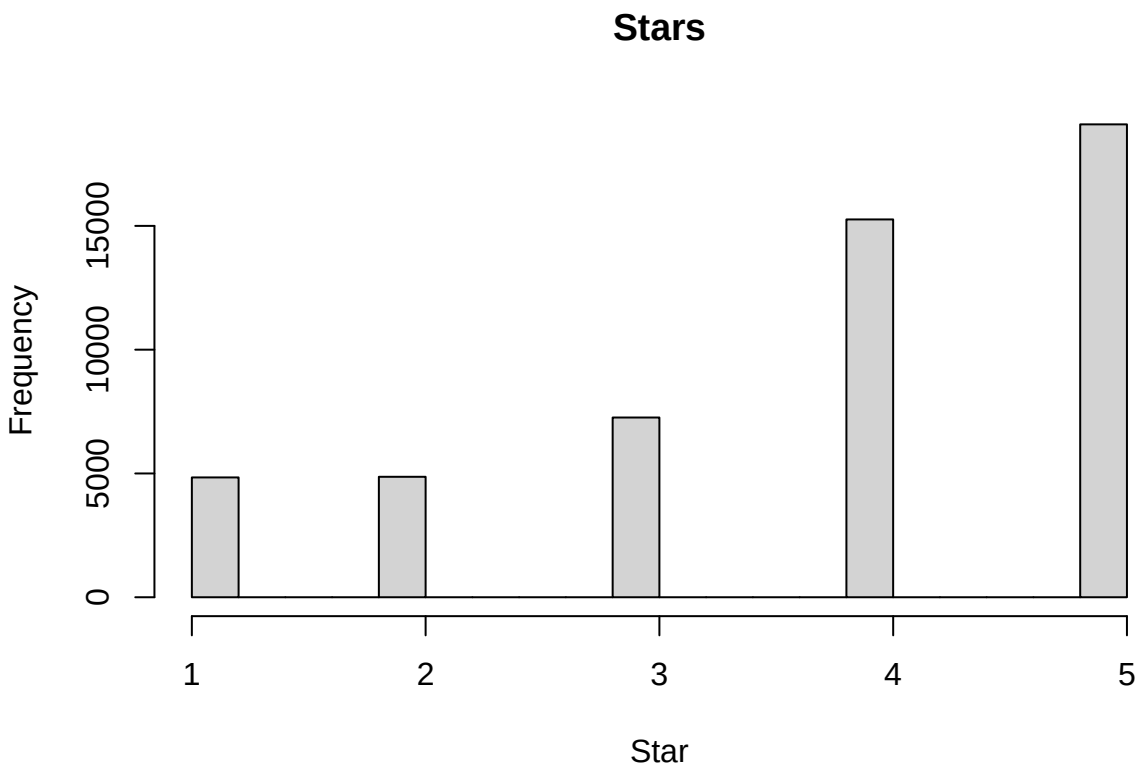
```
summary(star)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   3.000   4.000   3.758   5.000   5.000
```

```
hist(p, xlab = "Predicted value", main = "Predicted values")
```



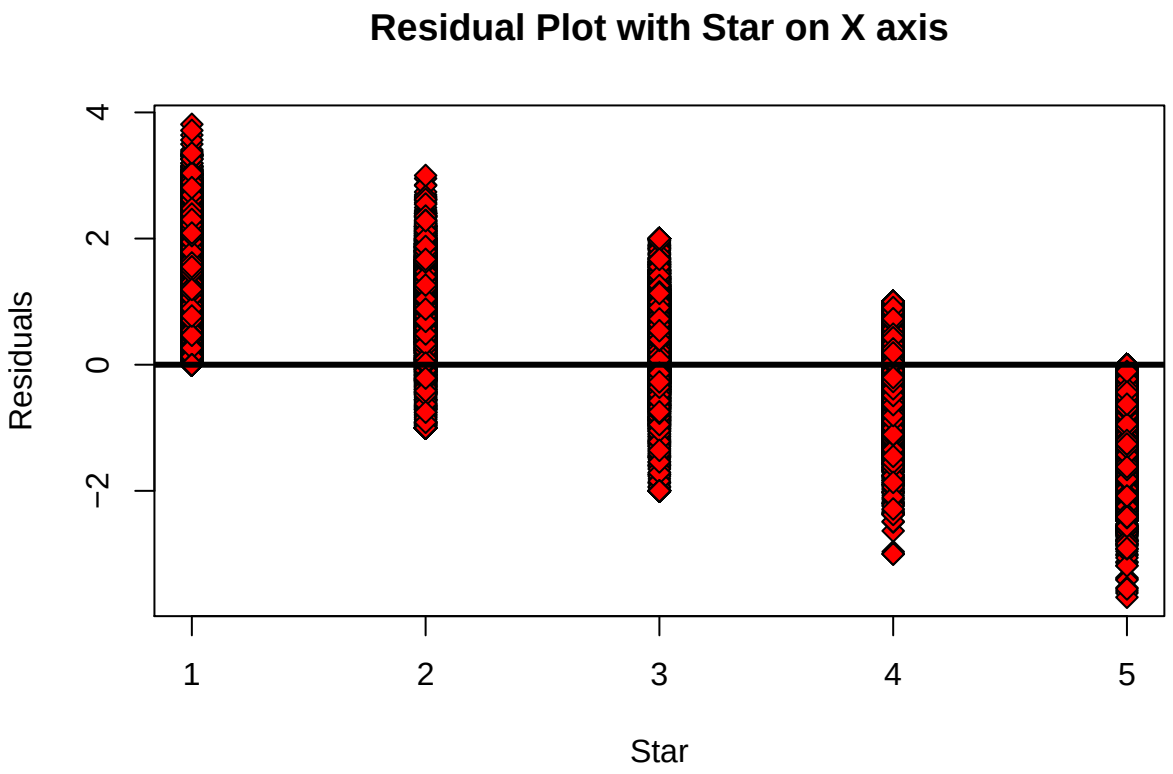

```
hist(star, xlab = "Star", main = "Stars")
```



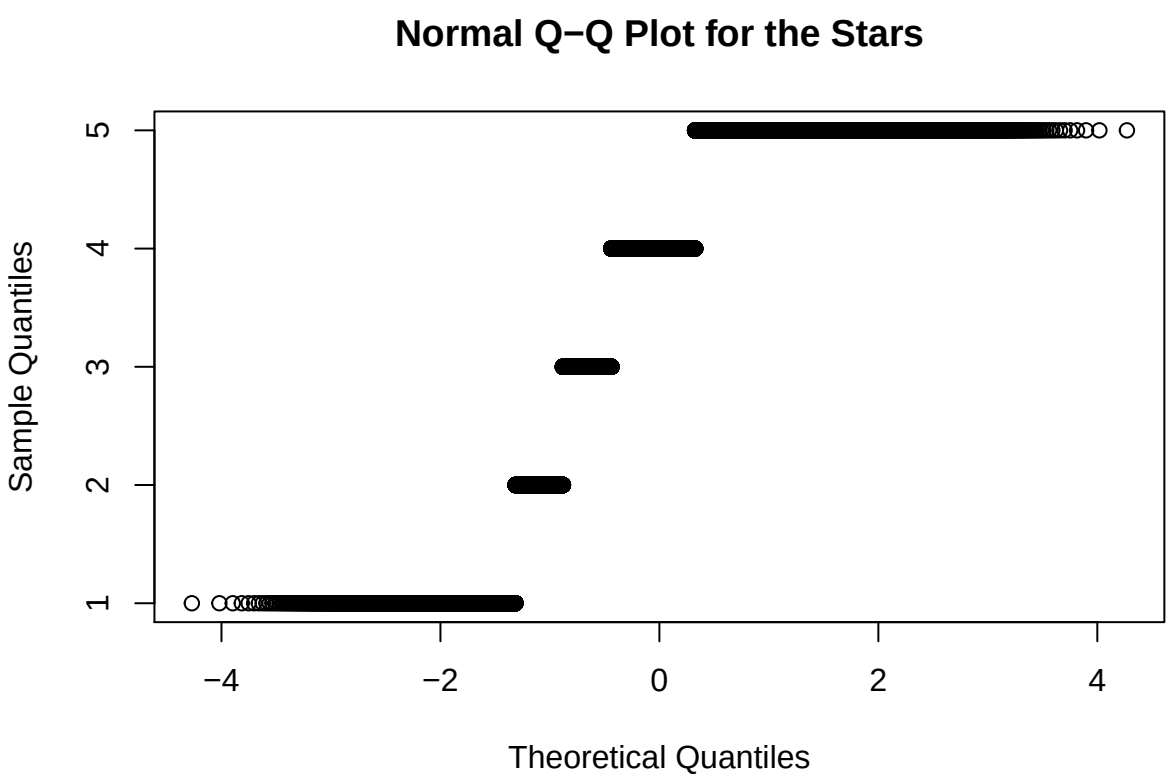

```
print("The star histogram shows that our projected stars is centered at 4.3 but our actual data set is
      4.3. This is likely due to the model making predictions to minimize MSE.")
```

```
## [1] "The star histogram shows that our projected stars is centered at 4.3 but our actual data set is
```

```
df.resid = tibble(residual = residual, star = star)
plot(df.resid$star,df.resid$residual,
     xlab="Star",
     ylab="Residuals",
     main="Residual Plot with Star on X axis",pch=23,bg="red",cex=1.2)
abline(a=0,b=0,col="black",lwd=3)
```




```
qqnorm(star, main = "Normal Q-Q Plot for the Stars")
```


Normal Q−Q Plot for the Stars
Sample Quantiles
1
2
3
4
5
−4
−2
0
2
4
Theoretical Quantiles