Prediction of Installs based on Ratings and Reviews

Joseph Padilla and Ashish Tiwari

November 29, 2018

Contents

Abstract

We examined a data set from the Google Play App Store to determine if it was possible to accurately create predictions of the number of Installs an application would receive if it had a certain rating and number of comments. This proposed model worked out exceptionally well and very accurately predicts what is desired. An attempt was made to try to make a similar model to predict the rating an application would have, but this turned out to be a horrible response variable with the given parameters.

1 Introduction

The purpose of this paper is to describe the relationship between several facets of applications data from the Google Play Store. The main goal is to model the number of installs an application has using the rating and the number of comments.

A secondary goal is to determine if a free app would benefit from changing the app from free to paid. This is extremely beneficial to the developer since it would allow an extra source of income without having to rely on ads or methods that would detract from the user experience.

2 Methodology

We began by obtaining a data set of applications from the Google Play Store. This data came in the form of an excel file and needed to be imported into R so that it could be used. Due to the nature of the data file, every column consisted of string-class objects. In addition, Google has decided to categorize its data instead of displaying the actual number.

Installs were separated into categories which increased along powers of 10 and those values multiplied by 5. These values were all `string`-class objects since every number ended with `+`. This gives us an extremely logarithmic scale to deal with. Ratings came in the form of of values between 1.0 and 5.0. These values varied only in the tenths place. Prices used the string `free` and had costs containing dollar signs. All of these strings needed to have `char`-class characters removed. Once this was done, columns were converted to `double`-class objects.

Now that our data was capable of being handled by R we had to investigate it to gain

an understanding. The data consists of over 30 categories of application. This is an important thing to notice because specific types apps are likely to have different notions of what is worth paying money for. Another factor to consider is applications which have zero values for rating, reviews, or installs. Such data points are meaningless because apps cannot have zero installs or a zero rating by design of the Google Play Store. Thus, they were deleted. We also deleted applications which had less than 101 installs due to having almost no other data. We also discovered that our data had multiple versions of the same objects. This had to be removed as well.

After this data was sufficiently cleaned, it was separated into two groups: training and testing. We chose to break our data up as 80% training and 20% testing. We did this by first recording the number of apps in each category there were. We then found how many of each category went into training and testing. We then randomly chose applications divided them up. We now have a usable data set for the problem at hand.

We first begin with a model that takes Installs as the response with Rating and Reviews as the predictors. By considering the numbers in our data set, It is obvious that a transformation must be performed. The max value of our rating is five, with the max value of Installs being one billion. We begin by using a transformation on both the Reviews and the Installs, as these values are extremely large. Our first model is simple:

$$\log_{10} Installs \quad \sim \quad Rating \quad + \quad \log_{10} Reviews.$$

This simple model gives us an $R^2_{Adj} = .919$ with all significant predictors. Since everything looks good here, we begin an investigation of the model to make sure that we can confirm our constant variance assumption, as well as making sure that our predictors aren't multicollinear.

We discover that Variance Inflation Factor indicates that our data is almost perfectly uncorrelated with a value of 1.092282. We now move to confirm our constant variance assumption and reach a problem; our data is extremely discretized resulting in our residual vs. fitted graph to appear as a series of lines. Thought and consideration allows us to make a guess on the statement, but another measure was taken just to ensure that there is constant variance. By considering each line, it can be seen that there is a normal distribution along it. Each of these lines has the same pattern of a normal distribution, therefore, we claim that the this model has constant variance.

Our secondary measure was to introduce a bit of noise into the data. This was only done to check our constant variance and then the model was discarded. We added a random normal distribution so that we could fill in the gaps that were missing due to the discrete nature of the data. When this was done, we got a random looking residual vs fitted graphs which confirm our constant variance.

The next job we have is to check our Normality assumption. This is done with a QQ-plot. The QQ-plot looks good given that there are 7031 points, but there are some obvious outliers. It is also possible that there may be a heavy-tailed distribution. In our case, we claim that it's best to remove a several outliers as we have so many data points that shouldn't have too much of an effect.

Our next job is to examine the leverage points. It is worth nothing that given the size of our data set, no one point will have a large leverage value, but these must be examined as a way to determine quality of the model. We follow this with a test for outliers. We look at the studentized residual and determine that there are 52 values which have a studentized residual greater than 3. Finally, we search for influential points, and determine that a small bunch of points are good candidates.

Now that we've gone through and examined our data to ensure it is sufficiently linear, we must do something about the outliers, leverage points, and influential points. We now must determine what is a reasonable amount of data to discard. We came to the conclusion that 3σ rule was probably the best approach. This tells us that given our data set of 7031 entries, we may freely discard 20.

Doing this allowed us to raise our R^2_{Adj} slightly, leading to a better fit. Now, with this better model, we used our testing data to see how well the model works. We used the predict function in R and compared the predicted values of our test data to the actual values and got a very good fit overall.

3 Results and Discussion

In Section 2, we discussed how we got the results that we got. Here, we will show you those results and discuss what they mean.

We begin by looking at the summary of the model that we created

```
> FirstTrainModel_Installs <- lm(log10(Installs)~Rating+log10(Reviews+1),data=
trainingdata)
> summary(FirstTrainModel_Installs)

Call:
lm(formula = log10(Installs) ~ Rating + log10(Reviews + 1), data = trainingdata)

Residuals:
     Min       1Q   Median       3Q      Max
-1.58533 -0.26965  0.00162  0.26576  3.01747

Coefficients:
                    Estimate Std. Error t value Pr(>|t|)
(Intercept)         2.994087   0.043845   68.29   <2e-16 ***
Rating             -0.234904   0.010882  -21.59   <2e-16 ***
log10(Reviews + 1)  0.927209   0.003358  276.08   <2e-16 ***
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 0.4247 on 7028 degrees of freedom
  (4 observations deleted due to missingness)
Multiple R-squared:  0.9192,Adjusted R-squared:  0.9192
F-statistic: 3.999e+04 on 2 and 7028 DF,  p-value: < 2.2e-16
```

As you can see, our original model before we did any tests came out spectacular. Every parameter is significant and we have an overall good fit. The reason that we used log_{10} of the parameters is because they make more numerical sense to look at and get a feel of what values should be. Doing so forces Installs to be in a range of between 1 and 9, as opposed to 1 and 1,000,000,000. The same argument can be made for Reviews. You may notice that Reviews has a "$+1$" as part of the parameter for Reviews. This is because when comments equal zero, the log function is undefined, and without it, R pulls up error messages.

Next we examine the VIF of the model.

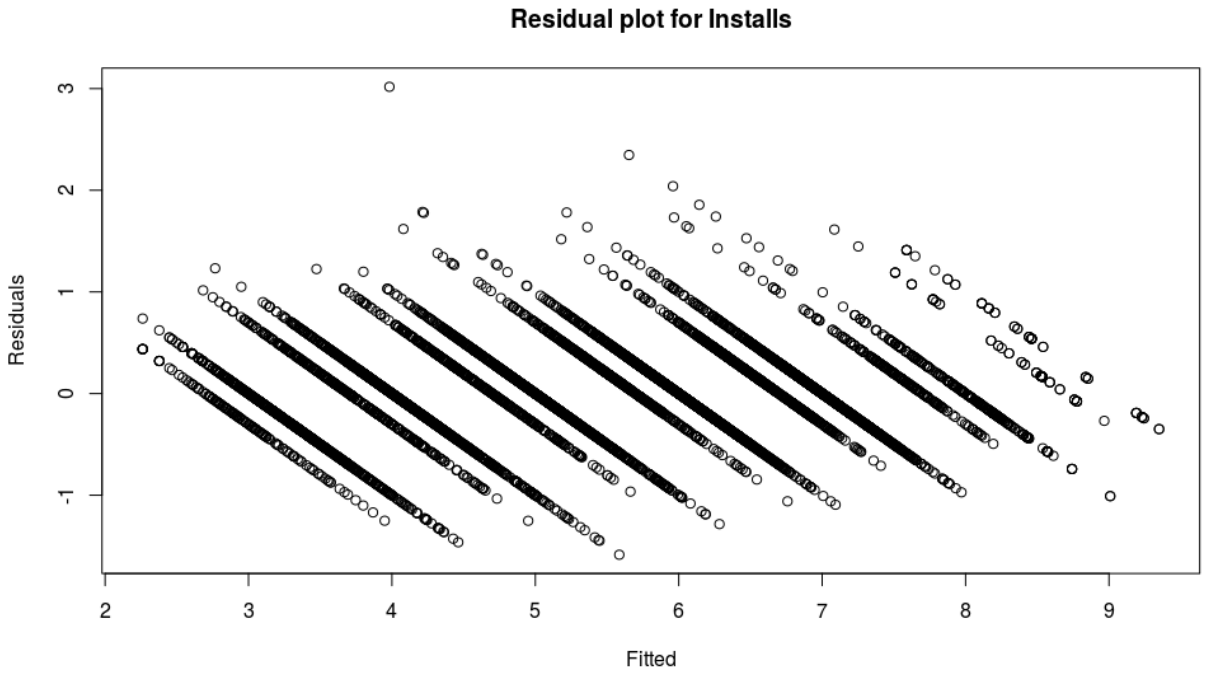
```
> vif(FirstTrainModel_Installs)
        Rating log10(Reviews + 1)
```

```
1.092264            1.092264
```

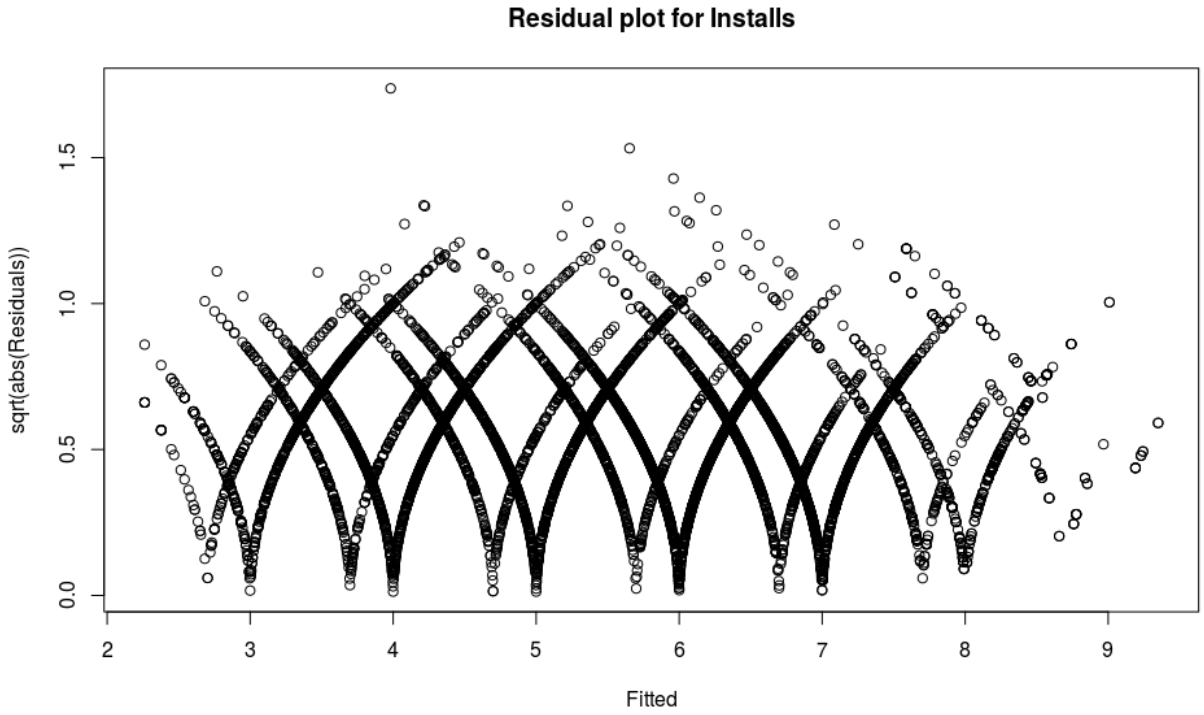
This clearly shows us that our data is almost perfectly non-collinear.

Now we have to confirm our constant variance assumption. This graph looks very strange because of the discrete nature of the data, but we claim that it has constant variance

```
res<-residuals(FirstTrainModel_Installs)
fit<-fitted(FirstTrainModel_Installs)
plot(fit,res,xlab="Fitted",ylab="Residuals",main="Residual plot for Installs")
abline(h=0)
```

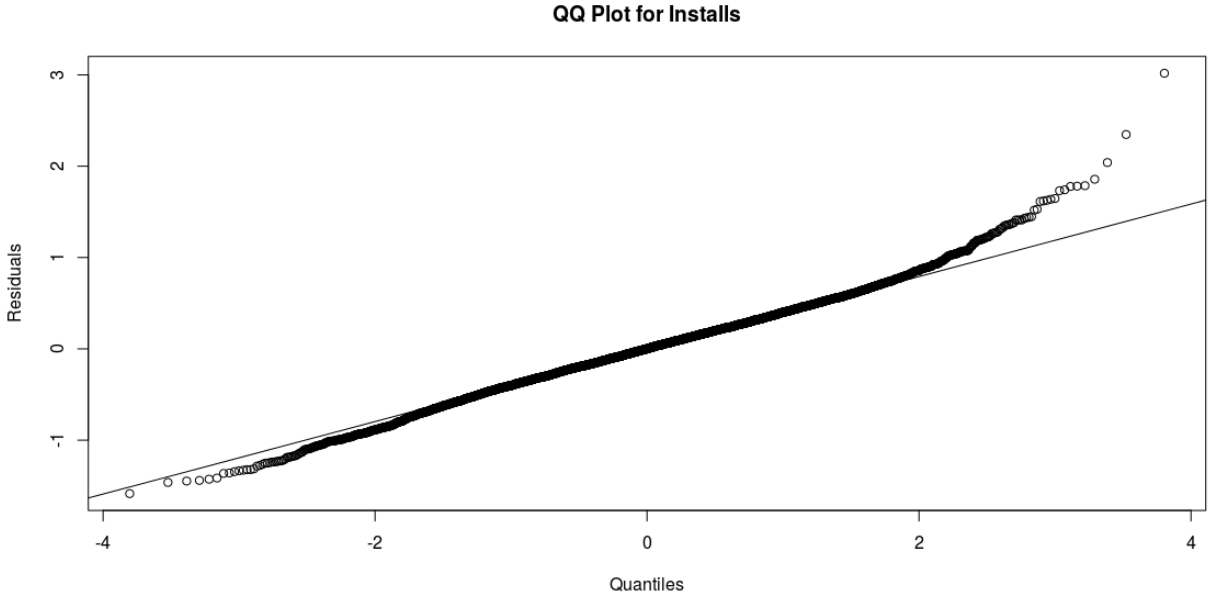


If you look at each line, you can notice that the values seem to have a normal distribution along them. This confirms our constant variance assumption. We also checked the $\sqrt{|\epsilon|}$ as well. This also appears to have a normal distribution along the curves.



We now have to confirm our normality assumption. We do this using a QQ graph

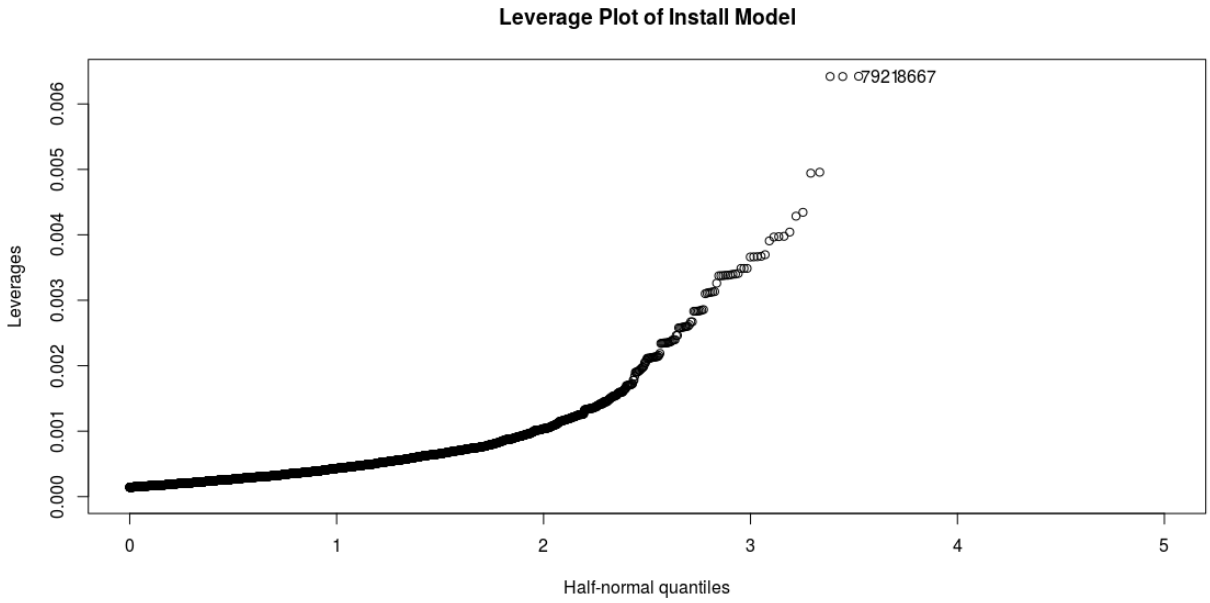
```
qqnorm(res,xlab="Quantiles",ylab="Residuals",main="QQ Plot for Installs")
qqline(res)
```



When we look at this plot can see that there is something going on here. Central values look perfect but once we pass the second standard deviation, we notice that the plot behaves strangely.It is clear that there is a heavy-tailed distribution going on here.

Our next analysis finds the leverage points of the model.

```
lev_install <- hatvalues(FirstTrainModel_Installs)
applications <- row.names(trainingdatatest)
halfnorm(lev_install,labs=applications,ylab="Leverages",xlim=c(0,5))
```



We can see here that there is a number of leverage points past 3 standard deviations. We claim that this is due to the heavy-tailed distribution.

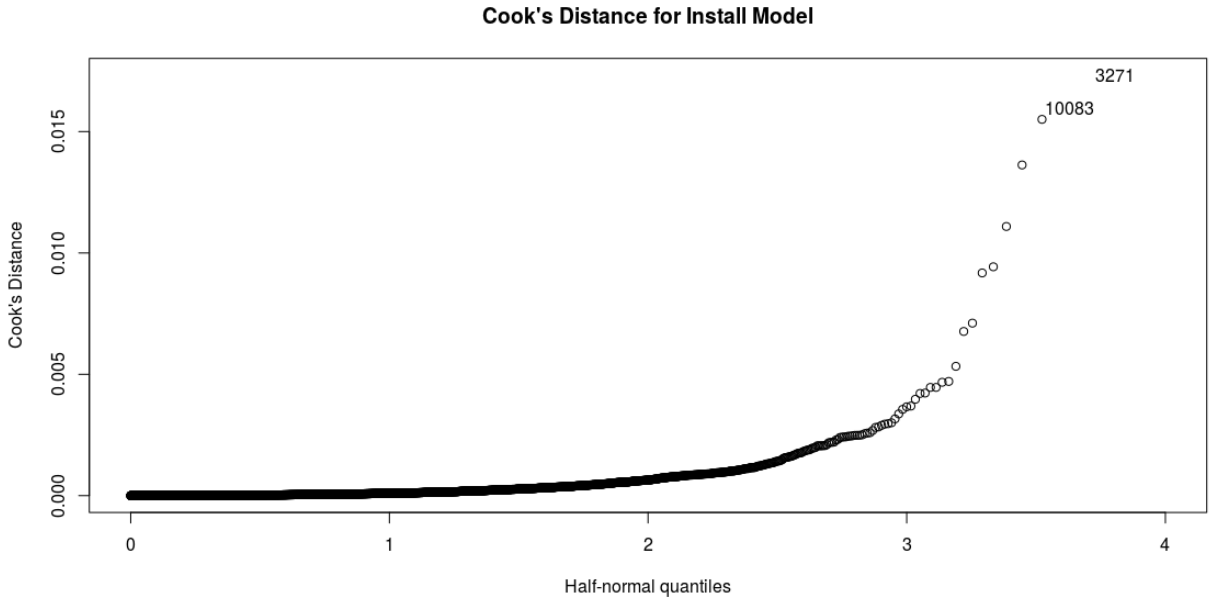
We now check for outliers using studentized residuals.

```
stdres_Installs<-rstandard(FirstTrainModel_Installs)
> studres[which.max(abs(studres))]
    3269
7.133426
> range(studres)
[1] -3.736782  7.133426
> sum((abs(studres)>3))
[1] 52
```

We see here that if we look at only values whose studentized residuals are greater than three standard deviations, our data set contains 52. This is a huge number, again thanks to our wide tailed distribution.

We now look at influential points by examining Cook's Distance

```
cook_Installs <- cooks.distance(FirstTrainModel_Installs)
halfnorm(cook_Installs,labs=applications,ylab="Cook's Distance", xlim=c(0,4),
main="Cook's Distance for Install Model")
```

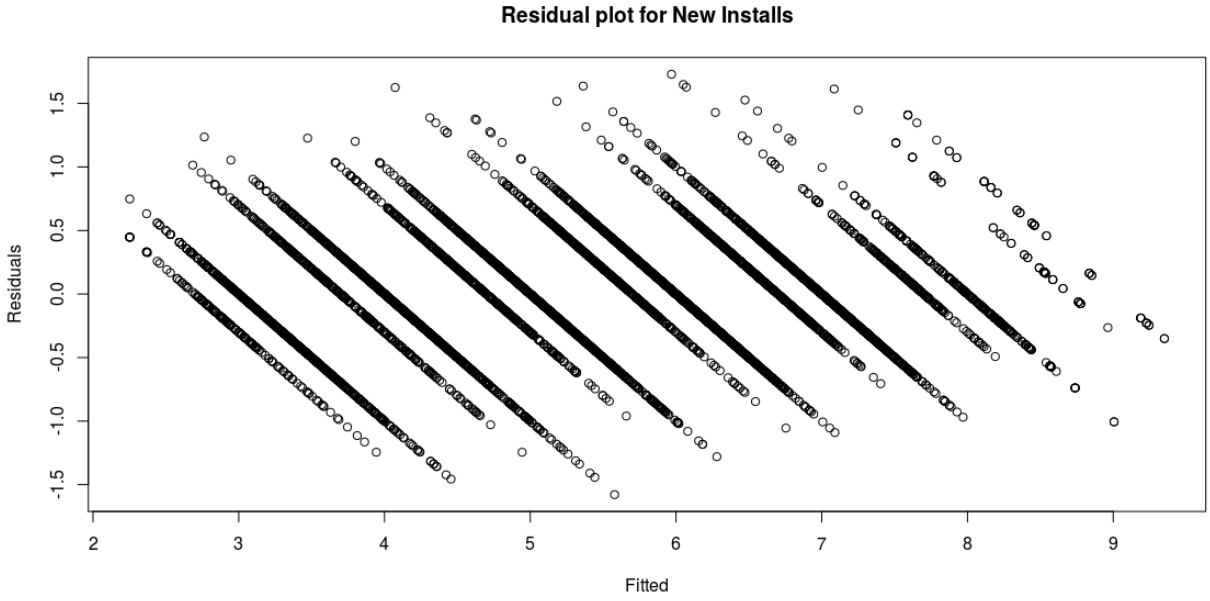


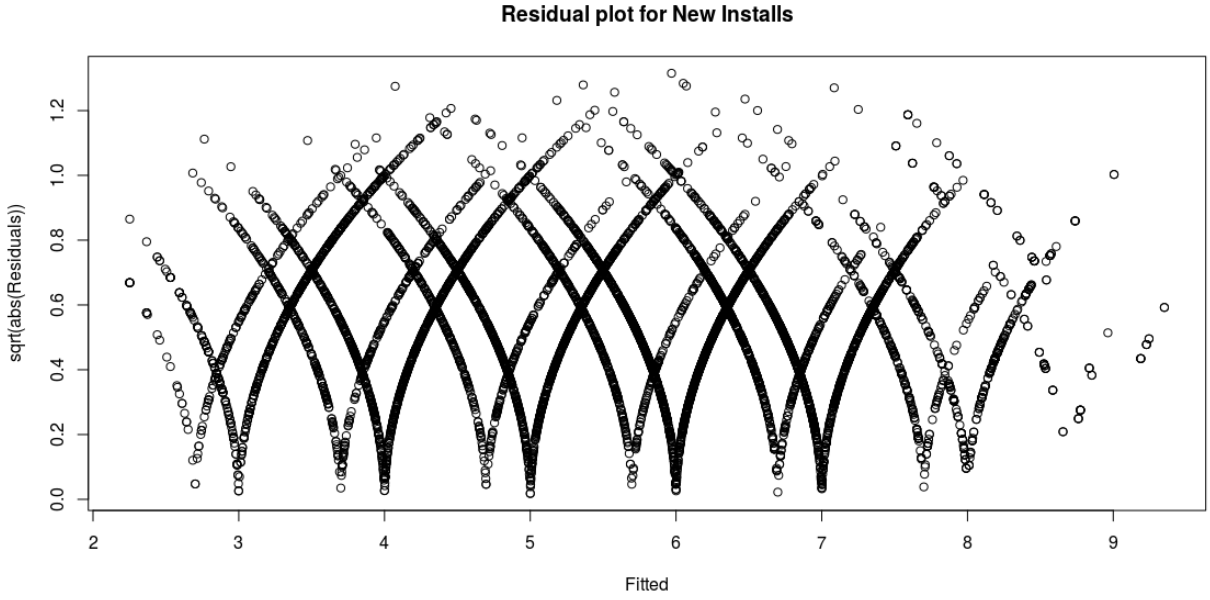
As with almost every other piece of data analyzing this data set, we see that past three standard deviations, the data becomes incredibly skewed. These are all potential candidates for being removed.

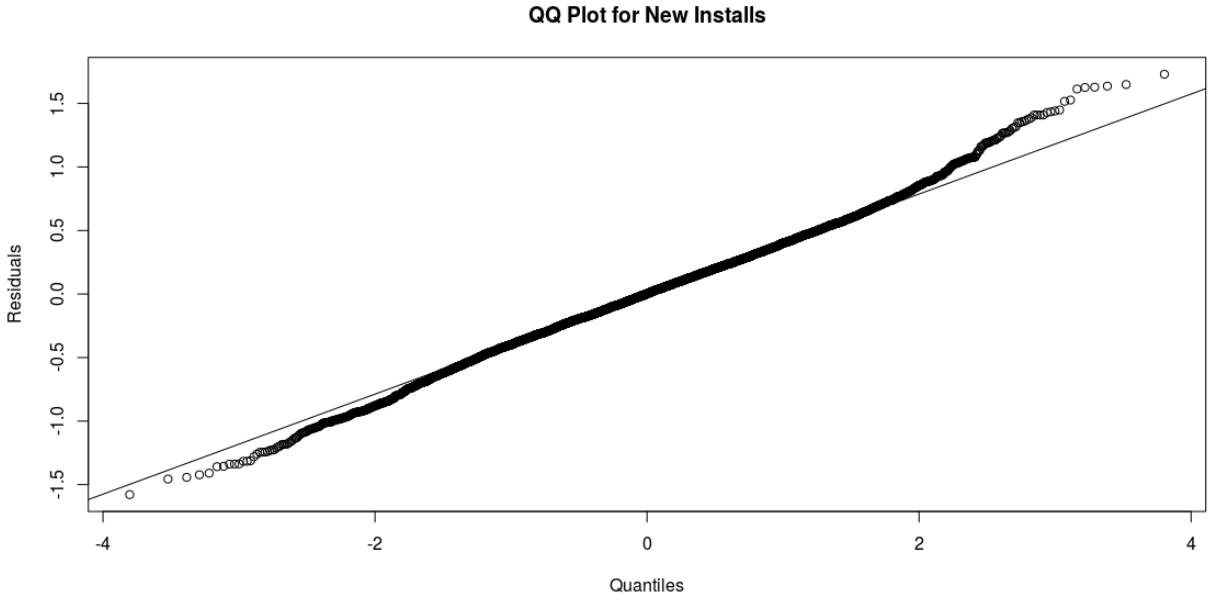
Now that the the data has been shown to be effectively linear, we must decide how to remove points from the data to solve some of these glaring issues. We considered dropped down to 2.5σ rule as to delete more points to fix the problems with the heavy-tail, but we decided against it because we didn't want to falsify the data by deleting more points than we should. We systematically went through and deleted points that showed up repeatedly in all of the graphs to capture the most glaring values which skewed the most data. With that said, our new model is here with all of the associated graphs of the previous tests to show the improvement

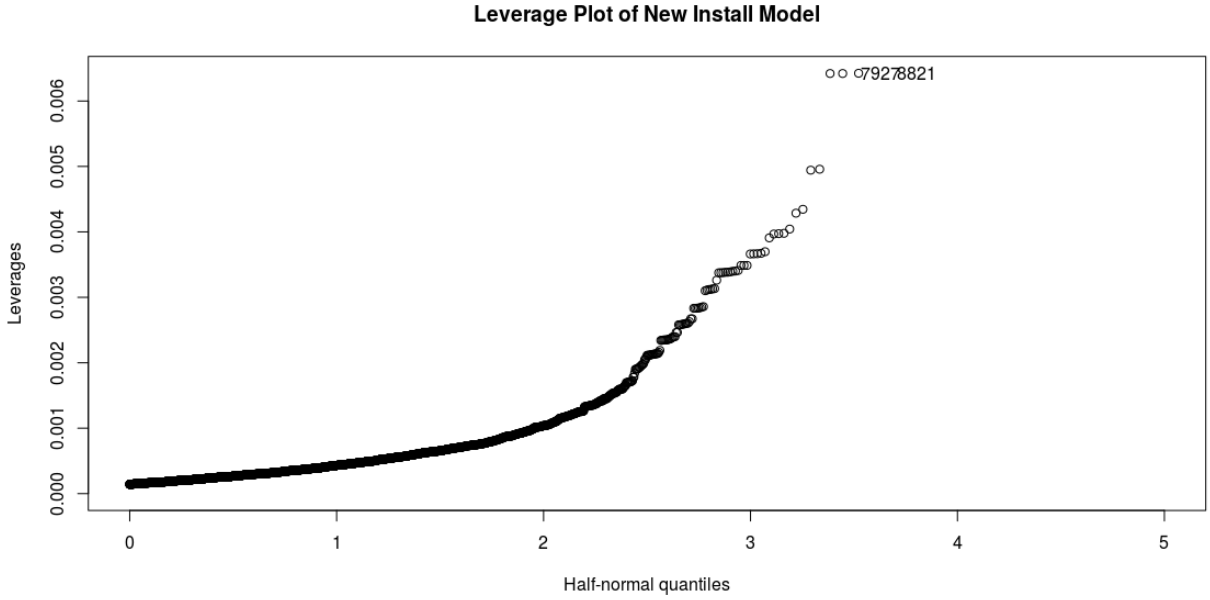
```
> SecondTrainModel_Installs <- lm(log10(Installs)~Rating+log10(Reviews+1),data=trainingd
> summary(SecondTrainModel_Installs)

Call:
lm(formula = log10(Installs) ~ Rating + log10(Reviews + 1), data = trainingdatatest)

Residuals:
     Min       1Q   Median       3Q      Max
-1.57870 -0.26558  0.00238  0.26542  1.72964

Coefficients:
                    Estimate Std. Error t value Pr(>|t|)
(Intercept)         3.027509   0.043780   69.15   <2e-16 ***
Rating             -0.243684   0.010853  -22.45   <2e-16 ***
log10(Reviews + 1)  0.927731   0.003306  280.65   <2e-16 ***
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 0.4178 on 7009 degrees of freedom
  (4 observations deleted due to missingness)
Multiple R-squared:  0.9217,Adjusted R-squared:  0.9217
F-statistic: 4.125e+04 on 2 and 7009 DF,  p-value: < 2.2e-16
```

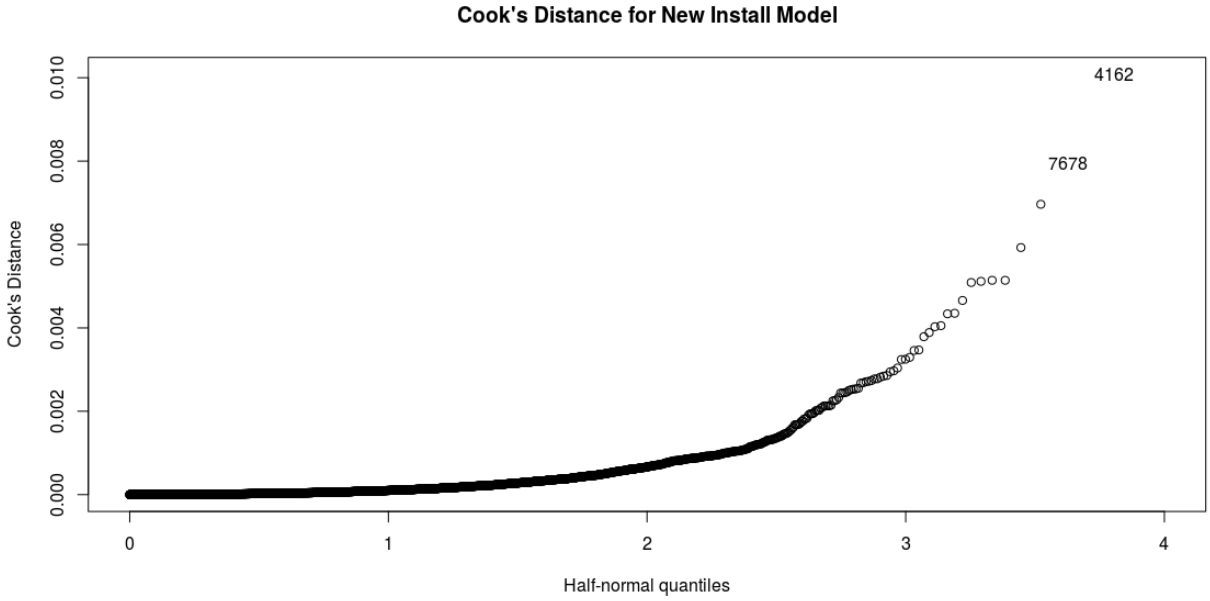
As before we see that the new model is not much better, but graphs will show that many of the huge leverage, influential, and outliers are gone.


Residual plot for New Installs
Residuals
Fitted
-1.5
-1.0
-0.5
0.0
0.5
1.0
1.5
2
3
4
5
6
7
8
9


Residual plot for New Installs
sqrt(abs(Residuals))
Fitted
0.0
0.2
0.4
0.6
0.8
1.0
1.2
2
3
4
5
6
7
8
9


QQ Plot for New Installs
Residuals
Quantiles
-1.5
-1.0
-0.5
0.0
0.5
1.0
1.5
-4
-2
0
2
4

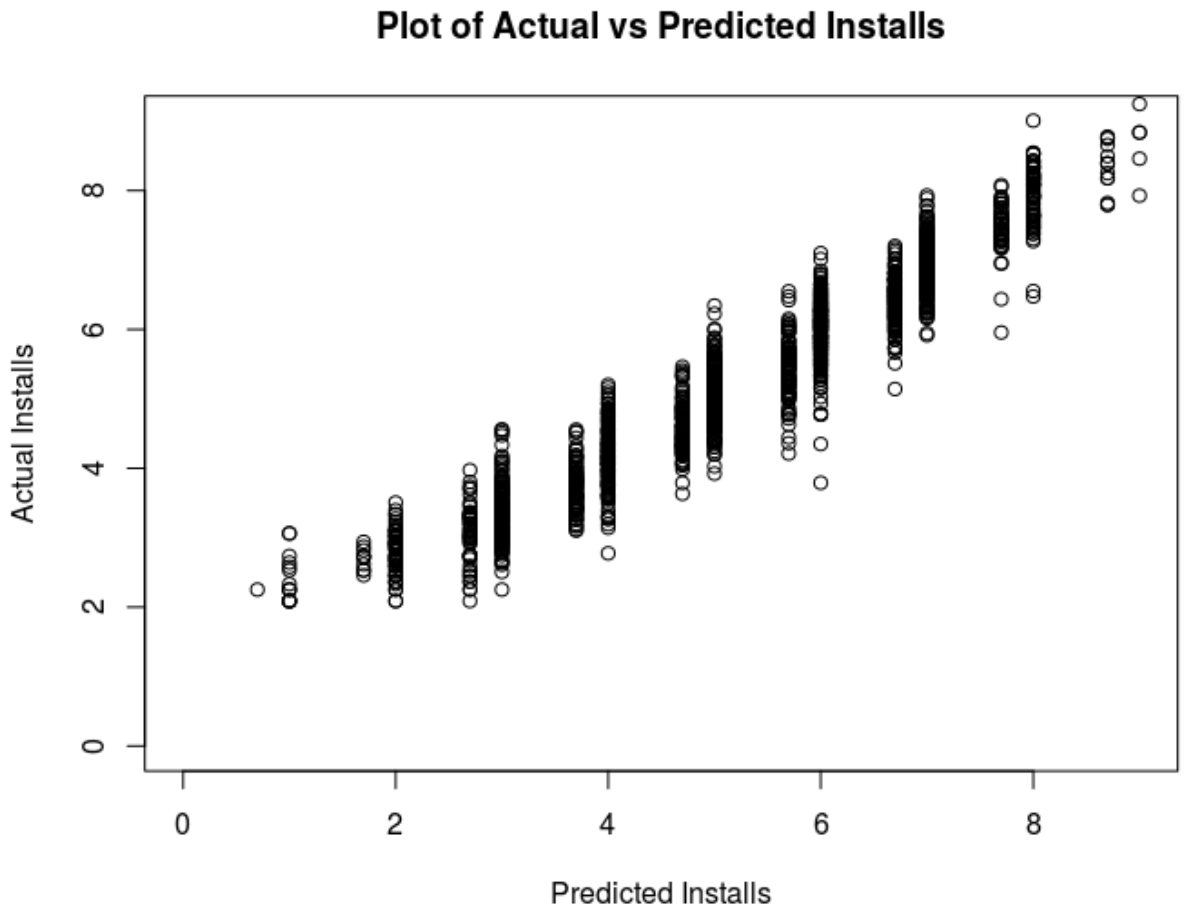

Leverage Plot of New Install Model
79278821
Leverages
Half-normal quantiles
0.000
0.001
0.002
0.003
0.004
0.005
0.006
0
1
2
3
4
5



When looking at these graphs, one may notice that the change between them isn't great, but it does make a difference. If we had decided to use a 2.5σ rule, we would have had a huge amount of points to delete, which would fix many of the problems that you can see that are caused by the heavy-tailed distribution.

The most important part of all of this is to test our data set which it has never seen

```
predictfitinstalls <- predict(SecondTrainModel_Installs,dfTest)
plot(log10(dfTest$Installs),predictfitinstalls,xlab="Predicted Installs",
ylab="Actual Installs",main="Plot of Actual vs Predicted Installs",xlim=
c(0,9),ylim=c(0,9))
```



This graph shows how well our model predicts the response. There is a clear linear trend in the data points and this graph demonstrates that it predicts the value within two orders of magnitude.

4 Conclusion

We examined a data set from the Google Play App Store to determine if it was possible to accurately create predictions of the number of Installs an application would receive if it had a certain rating and number of comments. This proposed model worked out exceptionally well and very accurately predicts what is desired.

An attempt was made to try to make a similar model to predict the rating an application would have, but this turned out to be a horrible response variable with the given parameters. The parameters were horribly multicollinear, and the highest possible R^2_{Adj} achieved was .12. This was a failed endeavour and was not included because it was essentially a meaningless

model that gave no further information.

References

A R code

```
googleplaystore<-read_csv("googleplaystore.csv")
googleplaystore<-unique(googleplaystore) # Removing copies of row
googleplaystore<-googleplaystore[!duplicated(googleplaystore[c('App')]),]
googleplaystore$X14<-NULL # Removing the column X14 from df
testgoogleDF <- googleplaystore
#deleting string entires to convert into double entries
testgoogleDF$Installs <- gsub("100[+]","100",testgoogleDF$Installs)
testgoogleDF$Installs <- gsub("10,000[+]","10000",testgoogleDF$Installs)
testgoogleDF$Installs <- gsub("100,000[+]","100000",testgoogleDF$Installs)
testgoogleDF$Installs <- gsub("1,000,000,[+]","1000000",testgoogleDF$Installs)
testgoogleDF$Installs <- gsub("1,000,000,[+]","1000000",testgoogleDF$Installs)
testgoogleDF$Installs <- gsub("5,000,000,[+]","5000000",testgoogleDF$Installs)
testgoogleDF$Installs <- gsub("5,000,000[+]","5000000",testgoogleDF$Installs)
testgoogleDF$Installs <- gsub("50,000,000[+]","50000000",testgoogleDF$Installs)
testgoogleDF$Installs <- gsub("500,000[+]","500000",testgoogleDF$Installs)
testgoogleDF$Installs <- gsub("50,000[+]","50000",testgoogleDF$Installs)
testgoogleDF$Installs <- gsub("1,000,000[+]","1000000",testgoogleDF$Installs)
testgoogleDF$Installs <- gsub("10,000,000[+]","10000000",testgoogleDF$Installs)
testgoogleDF$Installs <- gsub("5,000[+]","5000",testgoogleDF$Installs)
testgoogleDF$Installs <- gsub("100,000,000[+]","100000000",testgoogleDF$Installs)
testgoogleDF$Installs <- gsub("1,000,000,000[+]","1000000000",testgoogleDF$Installs)
testgoogleDF$Installs <- gsub("1,000[+]","1000",testgoogleDF$Installs)
testgoogleDF$Installs <- gsub("5,000[+]","5000",testgoogleDF$Installs)
testgoogleDF$Installs <- gsub("500,000,000[+]","500000000",testgoogleDF$Installs)
testgoogleDF$Installs <- gsub("50[+]","50",testgoogleDF$Installs)
testgoogleDF$Installs <- gsub("500[+]","500",testgoogleDF$Installs)
testgoogleDF$Installs <- gsub("5[+]","5",testgoogleDF$Installs)
testgoogleDF$Installs <- gsub("10[+]","10",testgoogleDF$Installs)
testgoogleDF$Installs <- gsub("1[+]","1",testgoogleDF$Installs)
testgoogleDF$Installs <- gsub("100[+]","100",testgoogleDF$Installs)
```

```
testgoogleDF$Price <- gsub("[$]","0",testgoogleDF$Price)
#Converting entire columns to double so they can be maniuplated
testgoogleDF$Installs <- as.numeric(as.character(testgoogleDF$Installs))
testgoogleDF$Price <- as.numeric(as.character(testgoogleDF$Price))
testgoogleDF<-testgoogleDF[!(testgoogleDF$Installs<101),]
testgoogleDF<-testgoogleDF[!(testgoogleDF$Rating=="NaN"),]
#Creating Index vector to manipulate data based on category
index <- c(1:10841)
testgoogleDF <- cbind(testgoogleDF,index)
#it is of note that specific categories our our dataframe can be
#pulled based on the code below.  It will create a subdataframe
#consisting of only the category and also contain the index.
#Art_and_Design <- subset(testgoogleDF, Category == "ART_AND_DESIGN")
#This above code creates a subdataframe which contains only apps from
#which have the category equal to ART_AND_DESIGN.
#Below we have created a dataframe for each category
Art_and_Design <- subset(testgoogleDF, Category == "ART_AND_DESIGN")
Auto_and_Vehicles <- subset(testgoogleDF, Category == "AUTO_AND_VEHICLES")
Beauty <- subset(testgoogleDF, Category == "BEAUTY")
Books_and_Reference <- subset(testgoogleDF, Category == "BOOKS_AND_REFERENCE")
Business <- subset(testgoogleDF, Category == "BUSINESS")
Comics <- subset(testgoogleDF, Category == "COMICS")
Communication <- subset(testgoogleDF, Category == "COMMUNICATION")
Dating <- subset(testgoogleDF, Category == "DATING")
Education <- subset(testgoogleDF, Category == "EDUCATION")
Entertainment <- subset(testgoogleDF, Category == "ENTERTAINMENT")
Events <- subset(testgoogleDF, Category == "EVENTS")
Finance <- subset(testgoogleDF, Category == "FINANCE")
Food_and_Drink <- subset(testgoogleDF, Category == "FOOD_AND_DRINK")
Health_and_Fitness <- subset(testgoogleDF, Category == "HEALTH_AND_FITNESS")
House_and_Home <- subset(testgoogleDF, Category == "HOUSE_AND_HOME")
Libraries_and_Demo <- subset(testgoogleDF, Category == "LIBRARIES_AND_DEMO")
Lifestyle <- subset(testgoogleDF, Category == "LIFESTYLE")
Game <- subset(testgoogleDF, Category == "GAME")
Family_Cat <- subset(testgoogleDF, Category == "FAMILY")
Medical <- subset(testgoogleDF, Category == "MEDICAL")
```

```
Social <- subset(testgoogleDF, Category == "SOCIAL")
Shopping <- subset(testgoogleDF, Category == "SHOPPING")
Photography <- subset(testgoogleDF, Category == "PHOTOGRAPHY")
Sports <- subset(testgoogleDF, Category == "SPORTS")
Travel_and_Local <- subset(testgoogleDF, Category == "TRAVEL_AND_LOCAL")
Tools_Cat <- subset(testgoogleDF, Category == "TOOLS")
Personalization <- subset(testgoogleDF, Category == "PERSONALIZATION")
Productivity <- subset(testgoogleDF, Category == "PRODUCTIVITY")
Parenting <- subset(testgoogleDF, Category == "PARENTING")
Weather <- subset(testgoogleDF, Category == "WEATHER")
Video_Players <- subset(testgoogleDF, Category == "VIDEO_PLAYERS")
News_and_Magazines <- subset(testgoogleDF, Category == "NEWS_AND_MAGAZINES")
Maps_and_Navigation <- subset(testgoogleDF, Category == "MAPS_AND_NAVIGATION")

# Set some input variables to define the splitting.
# Input 1. The data frame that you want to split into training, validation, and test.
df <- Art_and_Design

# Input 2. Set the fractions of the dataframe you want to split into training,
# validation, and test.
fractionTraining   <- 0.80
fractionValidation <- 0.00
fractionTest       <- 0.20

# Compute sample sizes.
sampleSizeTraining   <- floor(fractionTraining   * nrow(df))
sampleSizeValidation <- floor(fractionValidation * nrow(df))
sampleSizeTest       <- floor(fractionTest       * nrow(df))

# Create the randomly-sampled indices for the dataframe. Use setdiff() to
# avoid overlapping subsets of indices.
indicesTraining    <- sort(sample(seq_len(nrow(df)), size=sampleSizeTraining))
indicesNotTraining <- setdiff(seq_len(nrow(df)), indicesTraining)
indicesValidation  <- sort(sample(indicesNotTraining, size=sampleSizeValidation))
indicesTest        <- setdiff(indicesNotTraining, indicesValidation)
```

```
# Finally, output the three dataframes for training, validation and test.
dfTraining   <- df[indicesTraining, ]
dfValidation <- df[indicesValidation, ]
dfTest       <- df[indicesTest, ]

# Set some input variables to define the splitting.
# Input 1. The data frame that you want to split into training, validation, and test.
df <- Auto_and_Vehicles

# Input 2. Set the fractions of the dataframe you want to split into training,
# validation, and test.
fractionTraining   <- 0.80
fractionValidation <- 0.00
fractionTest       <- 0.20

# Compute sample sizes.
sampleSizeTraining   <- floor(fractionTraining   * nrow(df))
sampleSizeValidation <- floor(fractionValidation * nrow(df))
sampleSizeTest       <- floor(fractionTest       * nrow(df))

# Create the randomly-sampled indices for the dataframe. Use setdiff() to
# avoid overlapping subsets of indices.
indicesTraining    <- sort(sample(seq_len(nrow(df)), size=sampleSizeTraining))
indicesNotTraining <- setdiff(seq_len(nrow(df)), indicesTraining)
indicesValidation  <- sort(sample(indicesNotTraining, size=sampleSizeValidation))
indicesTest        <- setdiff(indicesNotTraining, indicesValidation)

# Finally, output the three dataframes for training, validation and test.
dfTraining_temp   <- df[indicesTraining, ]
dfValidation_temp <- df[indicesValidation, ]
dfTest_temp       <- df[indicesTest, ]

dfTraining <- rbind(dfTraining,dfTraining_temp)
dfTesting <- rbind(dfTesting,dfTesting_temp)

##Continue the above function on every category dataframe that was created above.
```

```
trainingdatatest <- dfTraining
trainingdatatest<-trainingdatatest[!(trainingdatatest$Installs<101),]
trainingdatatest<-trainingdatatest[!(trainingdatatest$Rating=="NaN"),]
FirstTrainModel_Installs <- lm(log10(Installs)~Rating+log10(Reviews+1),data=trainingdata
summary(FirstTrainModel_Installs)
vif(FirstTrainModel_Installs)
summary(FirstTrainModel_Rating)
plot(FirstTrainModel_Rating)
vif(FirstTrainModel_Installs)
res<-residuals(FirstTrainModel_Installs)
fit<-fitted(FirstTrainModel_Installs)
plot(fit,res,xlab="Fitted",ylab="Residuals",main="Residual plot for Installs")
abline(h=0)
plot(fit,sqrt(abs(res)),xlab="Fitted",ylab="sqrt(abs(Residuals))",main="Residual plot fo
qqnorm(res,xlab="Quantiles",ylab="Residuals",main="QQ Plot for Installs")
qqline(res)
lev_install <- hatvalues(FirstTrainModel_Installs)
applications <- row.names(trainingdatatest)
halfnorm(lev_install,labs=applications,ylab="Leverages",xlim=c(0,5),main="Leverage Plot
stdres_Installs<-rstandard(FirstTrainModel_Installs)
qqnorm(stdres_Installs)
qqline(stdres_Installs)
studres<-rstudent(FirstTrainModel_Installs)
studres[which.max(abs(studres))]
range(studres)
sum((abs(studres)>3))
cook_Installs <- cooks.distance(FirstTrainModel_Installs)
halfnorm(cook_Installs,labs=applications,ylab="Cook's Distance", xlim=c(0,4),main="Cook'
installdataset <- trainingdatatest
trainingdatatest<-trainingdatatest[!(trainingdatatest$index==3490),]
trainingdatatest<-trainingdatatest[!(trainingdatatest$index==6320),]
trainingdatatest<-trainingdatatest[!(trainingdatatest$index==8842),]
trainingdatatest<-trainingdatatest[!(trainingdatatest$index==9177),]
trainingdatatest<-trainingdatatest[!(trainingdatatest$index==10083),]
trainingdatatest<-trainingdatatest[!(trainingdatatest$index==3269),]
```

```
trainingdatatest<-trainingdatatest[!(trainingdatatest$index==3262),]
trainingdatatest<-trainingdatatest[!(trainingdatatest$index==3497),]
trainingdatatest<-trainingdatatest[!(trainingdatatest$index==3244),]
trainingdatatest<-trainingdatatest[!(trainingdatatest$index==4007),]
trainingdatatest<-trainingdatatest[!(trainingdatatest$index==8858),]
trainingdatatest<-trainingdatatest[!(trainingdatatest$index==7127),]
trainingdatatest<-trainingdatatest[!(trainingdatatest$index==7924),]
trainingdatatest<-trainingdatatest[!(trainingdatatest$index==8759),]
trainingdatatest<-trainingdatatest[!(trainingdatatest$index==10724),]
trainingdatatest<-trainingdatatest[!(trainingdatatest$index==3241),]
trainingdatatest<-trainingdatatest[!(trainingdatatest$index==7921),]
trainingdatatest<-trainingdatatest[!(trainingdatatest$index==8667),]
trainingdatatest<-trainingdatatest[!(trainingdatatest$index==5049),]
trainingdatatest<-trainingdatatest[!(trainingdatatest$index==4601),]
trainingdatatest<-trainingdatatest[!(trainingdatatest$index==3243),]
res<-residuals(SecondTrainModel_Installs)
fit<-fitted(SecondTrainModel_Installs)
plot(fit,res,xlab="Fitted",ylab="Residuals",main="Residual plot for New Installs")
abline(h=0)
plot(fit,sqrt(abs(res)),xlab="Fitted",ylab="sqrt(abs(Residuals))",main="Residual plot fo
qqnorm(res,xlab="Quantiles",ylab="Residuals",main="QQ Plot for New Installs")
qqline(res)
lev_install <- hatvalues(SecondTrainModel_Installs)
applications <- row.names(trainingdatatest)
halfnorm(lev_install,labs=applications,ylab="Leverages",xlim=c(0,5),main="Leverage Plot
stdres_Installs<-rstandard(SecondTrainModel_Installs)
qqnorm(stdres_Installs)
qqline(stdres_Installs)
studres<-rstudent(SecondTrainModel_Installs)
studres[which.max(abs(studres))]
range(studres)
sum((abs(studres)>3))
cook_Installs <- cooks.distance(SecondTrainModel_Installs)
halfnorm(cook_Installs,labs=applications,ylab="Cook's Distance", xlim=c(0,4),main="Cook'
predictfitinstalls <- predict(SecondTrainModel_Installs,dfTest)
plot(log10(dfTest$Installs),predictfitinstalls,xlab="Predicted Installs",ylab="Actual In
```