Are Facebook likes, profits and movie reviews accurate predictors for IMDB score?

kvk229 & gsc326

11/22/2016

GAM model general additive model

log linear regression

```
# setting the working directory
setwd('/Users/kkiran/Desktop/fall_2016/fds/project/MovieScorePredictor/data')
# 'C:/Users/gogs/Documents/GitHub/MovieScorePredictor/Data'
# setwd('C:/Users/gogs/Documents/GitHub/MovieScorePredictor/Data')
movieData = read.xls("movie_data.xls")
#head(movieData)
```

Loaded the data into a data frame 'movieData'

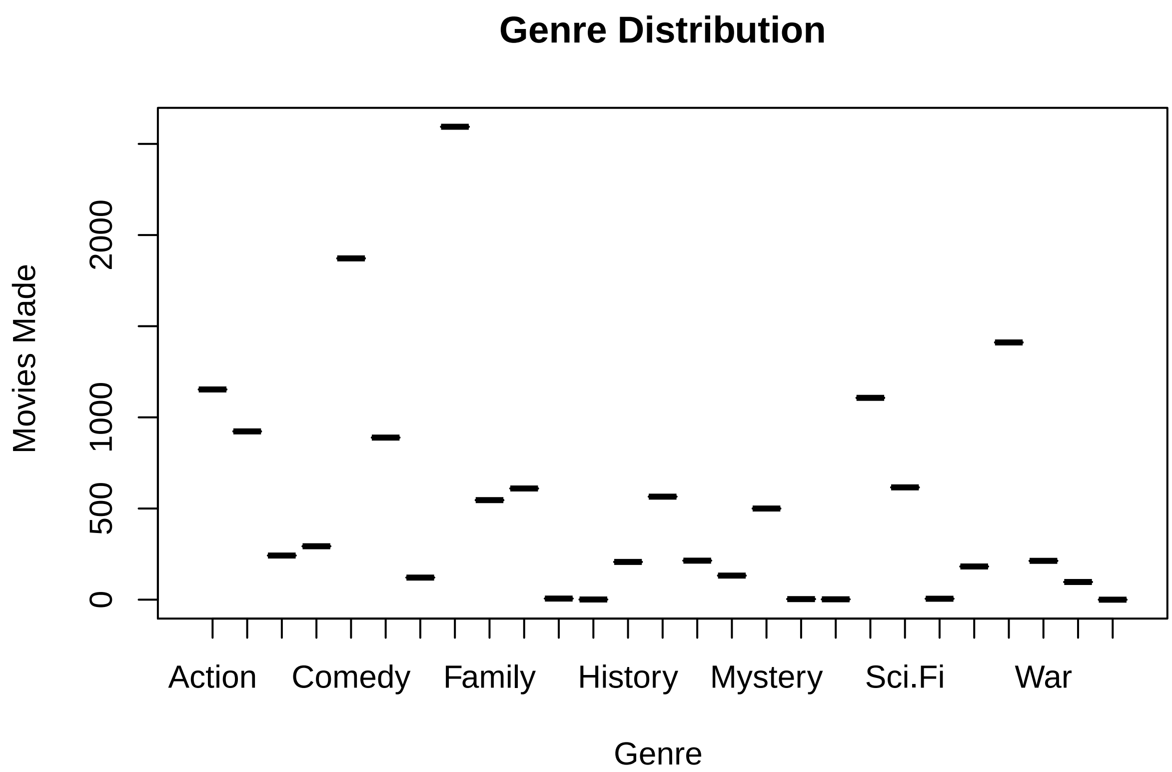
```
# identifying top 10 genres out of all the 26 genres to make the work more focussed

# gernes present in the data:
# 'Sci-Fi', 'Crime', Romance', Animation', Music', Comedy', War', genres', Horror', Film-Noir', Adventu

movieCount <- c()

for(i in 38:64)
{
  movieCount[i - 37] = sum(movieData[,i]);
}
movieCount
```

```
##  [1]  616  889 1107  242  214 1872  213    0  565    6  923    3    2 1411
## [15]   97  500    5 2594 1153  121  132  207  546  610    1  182  293
```

```
genreNames <- as.vector(colnames(movieData)[38:64])
genreNames
```

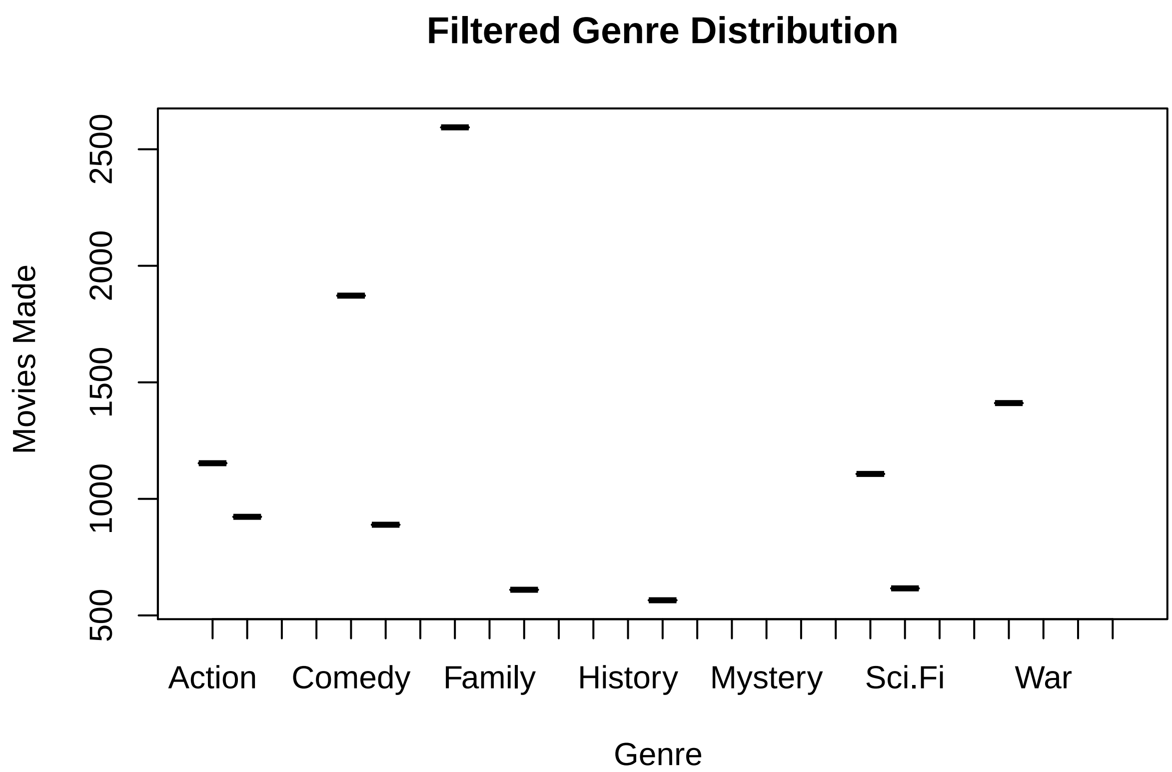
```
##  [1] "Sci.Fi"      "Crime"       "Romance"     "Animation"   "Music"
##  [6] "Comedy"      "War"         "genres.1"    "Horror"      "Film.Noir"
## [11] "Adventure"   "News"        "Reality.TV"  "Thriller"    "Western"
## [16] "Mystery"     "Short"       "Drama"       "Action"      "Documentary"
## [21] "Musical"     "History"     "Family"      "Fantasy"     "Game.Show"
## [26] "Sport"       "Biography"
```

```r
genreNames <- as.vector(genreNames)

genreMovieCount <- data.frame(genreNames, movieCount)
plot(genreMovieCount$genreNames, genreMovieCount$movieCount, main="Genre Distribution", xlab="Genre ", y
```



We can see that not all the genres have a considerable number of movies made in them, so we decided to extract the top 11 genres that have the most number of movies made in those particular genres.

```r
genreMovieCountSorted <- genreMovieCount[order(-movieCount),]
genreMovieCountSorted <- genreMovieCountSorted[c(1:10),]
plot(genreMovieCountSorted$genreNames, genreMovieCountSorted$movieCount, main="Filtered Genre Distributi
```



Now we have to remove the data for all the other genres from the data set, also we can delete the last column from the data set because it is repeated

```
movieData <- movieData[,-65]
columnNames <- colnames(movieData)
columnNames <- columnNames[1:37]

selectedNames <- genreMovieCountSorted$genreNames

columnNames <- as.vector(columnNames)
selectedNames <- as.vector(selectedNames)

names <- c(columnNames, selectedNames)
names
```

```
##  [1] "movie_title"               "actor_1_facebook_likes"
##  [3] "actor_2_facebook_likes"    "actor_3_facebook_likes"
##  [5] "director_facebook_likes"   "movie_facebook_likes"
##  [7] "cast_total_facebook_likes" "director_name"
##  [9] "actor_1_name"              "actor_2_name"
## [11] "actor_3_name"              "gross"
## [13] "budget"                    "imdb_score"
## [15] "num_critic_for_reviews"    "num_user_for_reviews"
## [17] "tomatoUserRating"          "tomatoRating"
## [19] "tomatoReviews"             "tomatoFresh"
## [21] "tomatoRotten"              "tomatoUserMeter"
## [23] "tomatoUserReviews"         "num_voted_users"
## [25] "imdbVotes"                 "Metascore"
## [27] "genres"                    "facenumber_in_poster"
```

```
## [29] "plot_keywords"              "movie_imdb_link"
## [31] "language"                   "country"
## [33] "content_rating"             "title_year"
## [35] "aspect_ratio"               "color"
## [37] "duration"                   "Drama"
## [39] "Comedy"                     "Thriller"
## [41] "Action"                     "Romance"
## [43] "Adventure"                  "Crime"
## [45] "Sci.Fi"                     "Fantasy"
## [47] "Horror"
```

```r
movieData1 <- subset(movieData, select = names)
# for(i in 38:47)
# {
#   print(c(colnames(movieData1[i]), sum(movieData1[,i])));
# }
```

Now movieData1 has the data about the genres that we are interested only. The next step is to clean the data by removing rows that dont have a considerable amount of data. If the facebook likes are missing, although we can get those details from other row, we are not proceeding so because the number of facebook likes are always changing and fetching data from other rows might not be a very good estimate for plugging in missing likes data.

Note: The facebook data is present in the data itself, we didnt have to fetch the data manually.

```r
mean(is.na(movieData))
```

```
## [1] 0.006379511
```

```r
paste("only", mean(is.na(movieData)), " (mean) amount of data is null, so we can safely remove NAs")
```

```
## [1] "only 0.00637951120364862  (mean) amount of data is null, so we can safely remove NAs"
```

```r
row.has.na <- apply(movieData1, 1, function(x){any(is.na(x))})
numberOfNAs <- sum(row.has.na)

print (c("can remove ", numberOfNAs , "null rows from the table"))
```

```
## [1] "can remove "              "1242"
## [3] "null rows from the table"
```

```r
#removing the nulls
movieData1 <- na.omit(movieData1)
```

```r
NAcounter <- 0
indicesToRemove <- c()
index <- 1
# this is the working version
for (i in 1 : nrow(movieData1)) {
  if (any(movieData1[i,] == "N/A")) {
     # print (c(i, "yes" , movieData1[i,]))
```

```
    indicesToRemove[index] = i;
    index <- index + 1
    NAcounter <- NAcounter + 1
  }
    # print ("no")
}

print(length(indicesToRemove))
```

```
## [1] 567
```

```
# print(indicesToRemove)
print(c("total number of nulls", NAcounter))
```

```
## [1] "total number of nulls" "567"
```

```
#removing the '@NAcounter' number of rows that have NA in them
movieData2 <- movieData1[-indicesToRemove,]
# Now, movieData2 has no NA in any of the rows.

movieData2[movieData2==""] <- NA
row.has.na <- apply(movieData2, 1, function(x){any(is.na(x))})
numberOfNAs <- sum(row.has.na)
paste("There are ",numberOfNAs, " rows with empty cells, so we are removing them")
```

```
## [1] "There are  236  rows with empty cells, so we are removing them"
```

```
# removing the empty rows, (second round of filtering)
movieData2 <- na.omit(movieData2)
```

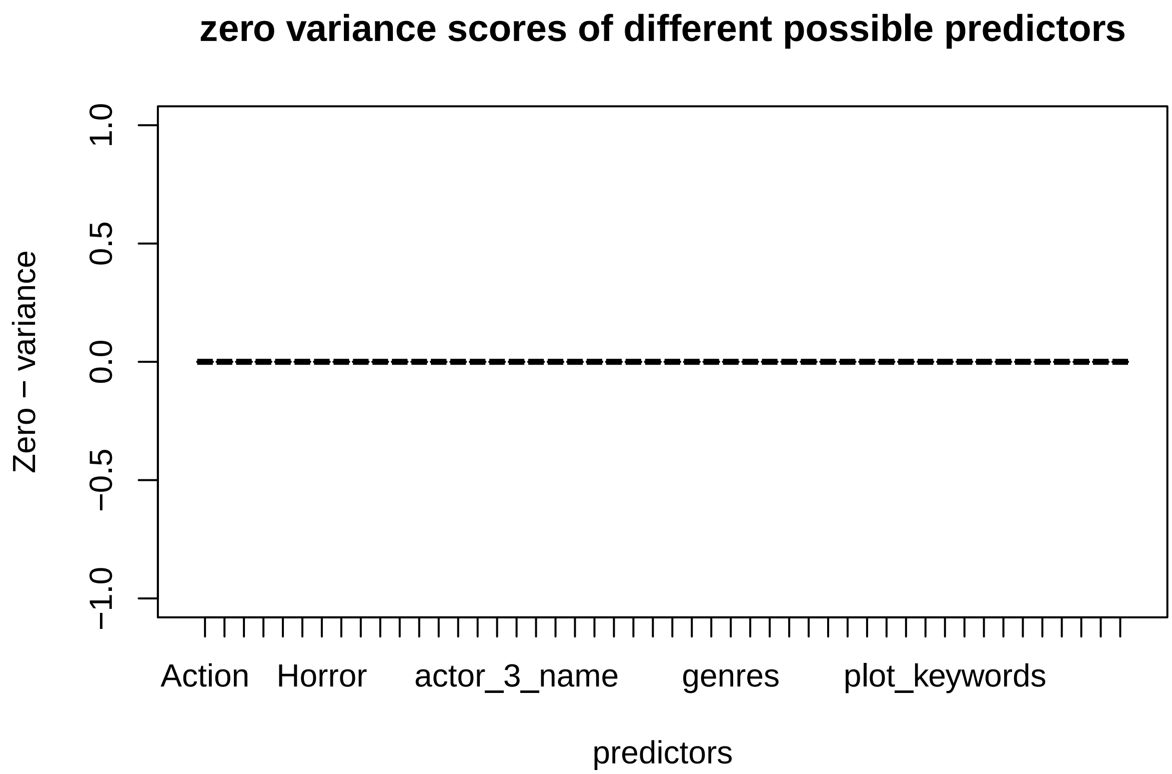
```
#couting the profits of a movie by subtracting the budget from the gross
movieData2$profits <- movieData2$gross - movieData2$budget
movieData3 <- movieData2[,c(c(1:13),48, c(14:47))]

# movieData3 is the final cleaned data that also has a column showing the profits made by the movie
#str(movieData3)
stat <- nearZeroVar(movieData3, saveMetrics = T)
class(stat$zeroVar)
```

```
## [1] "logical"
```

```
varDF <- cbind.data.frame(colnames(movieData3),stat$zeroVar)

#converting logical to binary
cols <- sapply(varDF, is.logical)
varDF[,cols] <- lapply(varDF[,cols], as.numeric)
```

```
## Warning in `[<-.data.frame`(`*tmp*`, , cols, value = list(0, 0, 0, 0, 0, :
## provided 48 variables to replace 1 variables
```

```
# for ( i in 1:nrow(varDF)) {
#  print(varDF[i,2])
# }

plot(varDF$`colnames(movieData3)`, varDF$`stat$zeroVar`, xlab = "predictors", ylab = "Zero - variance",
```




```
paste("we can see that all the columns have a zero score for the zero-variance score showing that all th
```

```
## [1] "we can see that all the columns have a zero score for the zero-variance score showing that all t
```

```
# We can see that none of the varibles have zero variance, so we can consider all the variables for stu
```

```
#coverting factor to int array
movieData3$tomatoUserRating <- as.numeric(as.character(movieData3$tomatoUserRating))
movieData3$tomatoRating <- as.numeric(as.character(movieData3$tomatoRating))
movieData3$tomatoReviews <- as.numeric(as.character(movieData3$tomatoReviews))
movieData3$tomatoFresh <- as.numeric(as.character(movieData3$tomatoFresh))
movieData3$tomatoRotten <- as.numeric(as.character(movieData3$tomatoRotten))
movieData3$tomatoUserMeter <- as.numeric(as.character(movieData3$tomatoUserMeter))
movieData3$tomatoUserReviews <- as.numeric(as.character(movieData3$tomatoUserReviews))
movieData3$imdbVotes <- as.numeric(as.character(movieData3$imdbVotes))
movieData3$Metascore <- as.numeric(as.character(movieData3$Metascore))
```

```
# checking the correlation
# str(movieData3)
cor(movieData3$imdb_score, movieData3[,c(7,12:14,16:26,38)], use = "pairwise.complete.obs")
```

```
##      cast_total_facebook_likes     gross     budget    profits
## [1,]                 0.0980045 0.2074095 0.02979009 0.02925642
##      num_critic_for_reviews num_user_for_reviews tomatoUserRating
## [1,]              0.3549605            0.3305453        0.8103758
##      tomatoRating tomatoReviews tomatoFresh tomatoRotten tomatoUserMeter
## [1,]     0.813982     0.2975506   0.5848666   -0.4436326       0.8440624
##      tomatoUserReviews num_voted_users imdbVotes  duration
## [1,]        0.08724109       0.4919135 0.4907566 0.3701765
```

```r
colnames(movieData3)[c(7,12:14,16:26,38)]
```

```
##  [1] "cast_total_facebook_likes" "gross"
##  [3] "budget"                    "profits"
##  [5] "num_critic_for_reviews"    "num_user_for_reviews"
##  [7] "tomatoUserRating"          "tomatoRating"
##  [9] "tomatoReviews"             "tomatoFresh"
## [11] "tomatoRotten"              "tomatoUserMeter"
## [13] "tomatoUserReviews"         "num_voted_users"
## [15] "imdbVotes"                 "duration"
```

```r
cNames <- paste("movieData3$",colnames(movieData3)[c(7,12:14,16:26,38)], sep = "")
# cNames
#formula contains all the columns that we want to include in the model
formula <- as.formula(paste("y ~ ", paste(cNames, collapse= "+")))
# formula
```

```r
#choice of linear regression vs logistic regression
# Linear regression: When the outcome(dependent variable) is continuous, i.e. infinite number of possib
# Logistic regression: When the outcome(dependent variable) has a limited set of values.

#because we are trying to predict the IMDB score of a movie and theoritically the score can have an inf
#linear regression



lmfit1.movieData <- lm(movieData3$imdb_score ~ movieData3$cast_total_facebook_likes + movieData3$gross +

# summary(lmfit1.movieData)

vif(lmfit1.movieData)
```

```
## movieData3$cast_total_facebook_likes                     movieData3$gross
##                             1.104381                             1.780129
##                    movieData3$budget    movieData3$num_critic_for_reviews
##                             1.022601                             4.204645
##      movieData3$num_user_for_reviews          movieData3$tomatoUserRating
##                             3.200089                             6.839647
##              movieData3$tomatoRating             movieData3$tomatoReviews
##                             5.104971                             7.447773
##               movieData3$tomatoFresh           movieData3$tomatoUserMeter
##                            11.679957                             7.997424
##         movieData3$tomatoUserReviews           movieData3$num_voted_users
##                             1.165426                           364.082025
##                 movieData3$imdbVotes                  movieData3$duration
##                           370.657616                             1.286969
```

Looking at the Variance Inflation Factor of the fitted model, we can see that imdbVotes and num_voted_users are two largest VIFs. so we try to eliminate them and make the model again.

```r
# coefficients of multi variate linear regression
paste("coefficients of fitted line by linear regression")
```

```
## [1] "coefficients of fitted line by linear regression"
```

```r
# summary(lmfit1.movieData)$coefficients
```

```r
 lmfit2.movieData <- lm(movieData3$imdb_score ~ movieData3$cast_total_facebook_likes + movieData3$gross

# summary(lmfit2.movieData)

lmfit2.movieData
```

```
##
## Call:
## lm(formula = movieData3$imdb_score ~ movieData3$cast_total_facebook_likes +
##     movieData3$gross + movieData3$budget + movieData3$num_critic_for_reviews +
##     movieData3$num_user_for_reviews + movieData3$tomatoUserRating +
##     +movieData3$tomatoRating + movieData3$tomatoReviews + movieData3$tomatoFresh +
##     movieData3$tomatoUserMeter + movieData3$tomatoUserReviews +
##     movieData3$duration, data = movieData3)
##
## Coefficients:
##                          (Intercept)
##                            1.103e+00
## movieData3$cast_total_facebook_likes
##                            1.633e-06
##                     movieData3$gross
##                           -1.043e-09
##                    movieData3$budget
##                            7.982e-11
##    movieData3$num_critic_for_reviews
##                            2.418e-04
##      movieData3$num_user_for_reviews
##                            1.256e-04
##          movieData3$tomatoUserRating
##                            5.397e-01
##              movieData3$tomatoRating
##                            3.519e-01
##             movieData3$tomatoReviews
##                            3.114e-03
##               movieData3$tomatoFresh
##                           -4.768e-03
##            movieData3$tomatoUserMeter
##                            1.857e-02
##         movieData3$tomatoUserReviews
##                            1.389e-08
##                 movieData3$duration
##                            2.786e-03
```

```r
vif(lmfit2.movieData)
```

```
## movieData3$cast_total_facebook_likes                            movieData3$gross
##                              1.087276                                    1.583824
##                     movieData3$budget     movieData3$num_critic_for_reviews
##                              1.019983                                    4.043964
##       movieData3$num_user_for_reviews           movieData3$tomatoUserRating
##                              2.072406                                    6.694771
##               movieData3$tomatoRating              movieData3$tomatoReviews
##                              5.104545                                    7.339284
##                movieData3$tomatoFresh            movieData3$tomatoUserMeter
##                             11.627723                                    7.993222
##          movieData3$tomatoUserReviews                   movieData3$duration
##                              1.161993                                    1.286622
```

```r
# we can see that the std error for the parameter estimates gets smaller.
```

```r
#calculating the MSFE and MAD for the predicted values
predictedScore <- predict(lmfit2.movieData)
# predictedScore
RSFE_v <- movieData3$imdb_score - predictedScore
# RSFE_v
RSFE <- sum(RSFE_v)
# RSFE
absRSFE <- abs(RSFE)
# absRSFE
length(RSFE_v)
```

```
## [1] 2998
```

```r
MSFE <- absRSFE / length(RSFE_v)
# MSFE
mean(lmfit2.movieData$residuals^2)
```

```
## [1] 0.1941867
```

```r
# calculating mad
Madoriginal <- mad(movieData3$imdb_score,center = median(movieData3$imdb_score),constant = 1)
MadRegression <- mad(predictedScore, center = median(predictedScore), constant = 1)
```

```r
# Analysing the linear regression model
# log.movieData <- log(movieData3[, c(7,12:26,38)])
trans = preProcess(x = movieData3[,c(7,12:14,16:26,38)], method=c("BoxCox", "center", "scale", "pca"))
trans
```

```
## Created from 2998 samples and 16 variables
##
## Pre-processing:
##   - Box-Cox transformation (13)
##   - centered (16)
##   - ignored (0)
```

```
##   - principal component signal extraction (16)
##   - scaled (16)
##
## Lambda estimates for Box-Cox transformation:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -0.8000  0.1000  0.2000  0.3308  0.5000  1.3000
##
## PCA needed 9 components to capture 95 percent of the variance
```

```r
trans$pcaComp
```

```
## NULL
```

```r
pca.movieData <- prcomp(x = movieData3[,c(7,12:14,16:26,38)], center = TRUE, scale. = TRUE)
print(pca.movieData)
```

```
## Standard deviations:
##  [1] 2.432503e+00 1.564950e+00 1.401758e+00 1.154068e+00 9.799094e-01
##  [6] 9.184594e-01 8.827092e-01 7.648408e-01 7.043480e-01 5.112467e-01
## [11] 4.578182e-01 3.645493e-01 2.603347e-01 3.693681e-02 1.698190e-15
## [16] 9.873104e-16
##
## Rotation:
##                                  PC1          PC2         PC3         PC4
## cast_total_facebook_likes  0.10874858 -0.159739097 -0.03150854 -0.07741693
## gross                      0.26316009 -0.208146502 -0.08207119  0.18149678
## budget                     0.02117848 -0.124725036  0.68823554  0.09564829
## profits                    0.05347272  0.063947013 -0.70036681 -0.04286223
## num_critic_for_reviews     0.31157844 -0.195203809  0.03556094 -0.34827959
## num_user_for_reviews       0.30429167 -0.199198393 -0.03953276  0.26783237
## tomatoUserRating           0.27568875  0.348381799  0.07634384 -0.09588632
## tomatoRating               0.28271713  0.375475248  0.05987576 -0.10215443
## tomatoReviews              0.28800338 -0.262819696  0.01164888 -0.44391120
## tomatoFresh                0.34435948  0.048346260  0.02976664 -0.34734537
## tomatoRotten              -0.06553253 -0.535196344 -0.02864047 -0.19999098
## tomatoUserMeter            0.26261601  0.413375677  0.06884329  0.02569731
## tomatoUserReviews          0.09772351 -0.126815013 -0.06026578  0.44463194
## num_voted_users            0.35162157 -0.114814179 -0.03940306  0.27747165
## imdbVotes                  0.35267478 -0.115679235 -0.03700495  0.27411764
## duration                   0.19402743 -0.002161922  0.05209069  0.17095599
##                                      PC5         PC6          PC7
## cast_total_facebook_likes  0.7442970309  0.57131652  0.199802020
## gross                      0.0371504409  0.07906111 -0.152176368
## budget                    -0.0007362571  0.05667193 -0.053212424
## profits                    0.0112150129 -0.03343718  0.009386848
## num_critic_for_reviews    -0.1138773738 -0.02993015 -0.049267286
## num_user_for_reviews      -0.0526316998 -0.15005822 -0.128820954
## tomatoUserRating           0.1329041063 -0.02261445  0.033480422
## tomatoRating              -0.1115986321  0.09782440  0.079849278
## tomatoReviews             -0.1770780212  0.01277372  0.140547048
## tomatoFresh               -0.2268091964  0.12742434  0.072353255
## tomatoRotten               0.0650392947 -0.18694074  0.125051821
## tomatoUserMeter            0.1129269225 -0.01149642  0.046752639
```

```
## tomatoUserReviews         -0.4651400109  0.45541260  0.538633766
## num_voted_users            0.0748453400 -0.04572894 -0.289635505
## imdbVotes                  0.0739755760 -0.04695573 -0.290051685
## duration                   0.2672984666 -0.60526603  0.637593608
##                                   PC8          PC9        PC10
## cast_total_facebook_likes  0.17506954  0.020426054 -0.05914578
## gross                     -0.49744154 -0.713144917 -0.12911647
## budget                    -0.07563258 -0.126002502 -0.02731393
## profits                   -0.06604902 -0.077397867 -0.00958438
## num_critic_for_reviews     0.13444239  0.001216694  0.31909171
## num_user_for_reviews       0.28194783  0.201509011 -0.72552433
## tomatoUserRating          -0.47326075  0.279209041  0.10658684
## tomatoRating               0.22778587 -0.159635157 -0.12269418
## tomatoReviews             -0.05719630  0.051030136 -0.09529204
## tomatoFresh                0.17192690 -0.149931431 -0.03681209
## tomatoRotten              -0.38133069  0.334541978 -0.10487794
## tomatoUserMeter           -0.33876910  0.253709205 -0.24588388
## tomatoUserReviews         -0.10465477  0.163945599  0.11853168
## num_voted_users            0.08615309  0.164713071  0.32513376
## imdbVotes                  0.08207903  0.162650775  0.32435965
## duration                   0.14236754 -0.207984553  0.13231874
##                                    PC11          PC12         PC13
## cast_total_facebook_likes  0.026671700  0.0054614142 -0.007909208
## gross                      0.057016165 -0.0444305061  0.009528170
## budget                    -0.005351173 -0.0004943584 -0.001489339
## profits                    0.021365572 -0.0120598955  0.004156011
## num_critic_for_reviews     0.714951697 -0.1901441546  0.242204070
## num_user_for_reviews       0.267208970  0.0549783993 -0.177867179
## tomatoUserRating           0.223888668  0.0837567248 -0.630769826
## tomatoRating              -0.232555094 -0.7595995477 -0.114768959
## tomatoReviews             -0.325220711  0.1710452076 -0.024027007
## tomatoFresh               -0.223312546  0.4218467584 -0.039739387
## tomatoRotten              -0.197653057 -0.3955499700  0.023535302
## tomatoUserMeter           -0.017069174  0.0747094918  0.703002764
## tomatoUserReviews          0.082555510 -0.0190121060  0.009940902
## num_voted_users           -0.238176326 -0.0055349746  0.024580202
## imdbVotes                 -0.223861943 -0.0111021480  0.028969046
## duration                  -0.011638249  0.0699895743  0.004428365
##                                    PC14          PC15          PC16
## cast_total_facebook_likes -0.0009811830  3.250883e-18  6.205474e-17
## gross                      0.0030325036 -2.761084e-02  1.953830e-01
## budget                     0.0013590672  9.620970e-02 -6.808101e-01
## profits                   -0.0004809376  9.777874e-02 -6.919131e-01
## num_critic_for_reviews     0.0095219764 -1.222113e-15 -3.261280e-16
## num_user_for_reviews       0.0010740188 -4.961309e-16  1.387779e-16
## tomatoUserRating           0.0011539765 -4.644722e-16  5.828671e-16
## tomatoRating               0.0002088994  1.474515e-16 -3.035766e-16
## tomatoReviews             -0.0023931164  6.634452e-01  9.375575e-02
## tomatoFresh               -0.0031922237 -6.276595e-01 -8.869865e-02
## tomatoRotten               0.0010873997 -3.824979e-01 -5.405326e-02
## tomatoUserMeter            0.0023808116  2.515349e-16 -7.979728e-17
## tomatoUserReviews         -0.0003350411  1.006140e-16  1.327063e-16
## num_voted_users            0.7038662332  1.040834e-17 -2.151057e-16
## imdbVotes                 -0.7102420018  4.770490e-16  3.330669e-16
```

```
## duration                    -0.0005571525  1.305379e-16 -5.551115e-17
```

```r
colnames(pca.movieData$rotation)
```

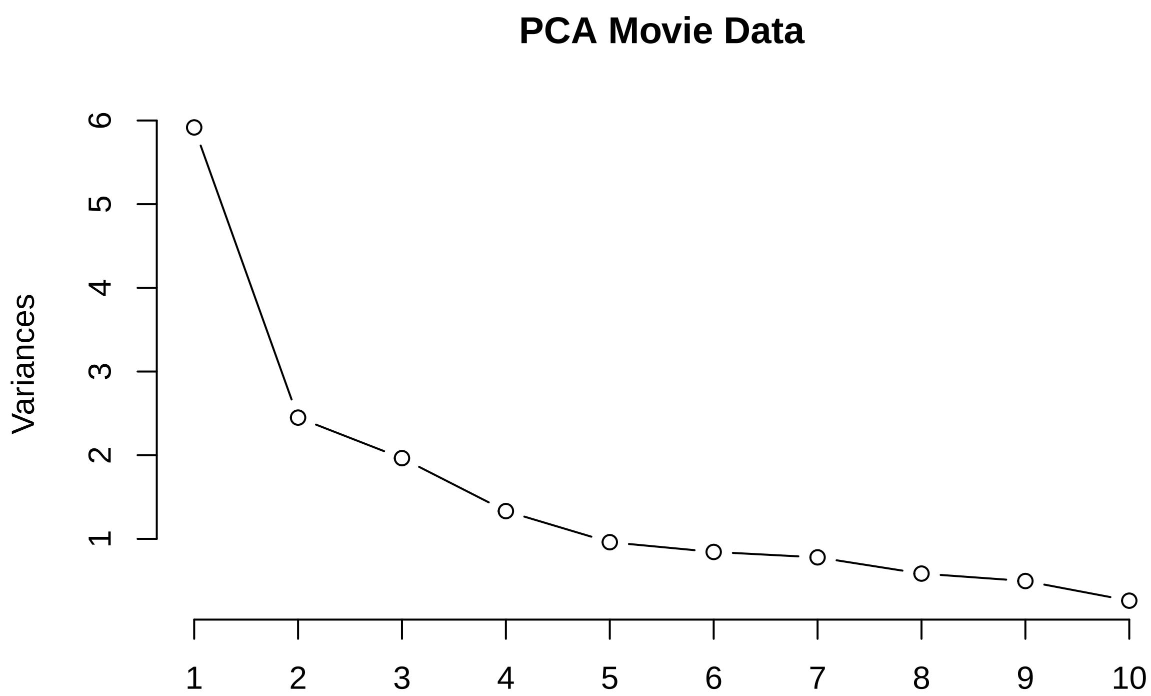
```
##  [1] "PC1"  "PC2"  "PC3"  "PC4"  "PC5"  "PC6"  "PC7"  "PC8"  "PC9"  "PC10"
## [11] "PC11" "PC12" "PC13" "PC14" "PC15" "PC16"
```

```r
# summary(pca.movieData)
pca.movieData
```

```
## Standard deviations:
##  [1] 2.432503e+00 1.564950e+00 1.401758e+00 1.154068e+00 9.799094e-01
##  [6] 9.184594e-01 8.827092e-01 7.648408e-01 7.043480e-01 5.112467e-01
## [11] 4.578182e-01 3.645493e-01 2.603347e-01 3.693681e-02 1.698190e-15
## [16] 9.873104e-16
##
## Rotation:
##                                PC1          PC2         PC3         PC4
## cast_total_facebook_likes  0.10874858 -0.159739097 -0.03150854 -0.07741693
## gross                      0.26316009 -0.208146502 -0.08207119  0.18149678
## budget                     0.02117848 -0.124725036  0.68823554  0.09564829
## profits                    0.05347272  0.063947013 -0.70036681 -0.04286223
## num_critic_for_reviews     0.31157844 -0.195203809  0.03556094 -0.34827959
## num_user_for_reviews       0.30429167 -0.199198393 -0.03953276  0.26783237
## tomatoUserRating           0.27568875  0.348381799  0.07634384 -0.09588632
## tomatoRating               0.28271713  0.375475248  0.05987576 -0.10215443
## tomatoReviews              0.28800338 -0.262819696  0.01164888 -0.44391120
## tomatoFresh                0.34435948  0.048346260  0.02976664 -0.34734537
## tomatoRotten              -0.06553253 -0.535196344 -0.02864047 -0.19999098
## tomatoUserMeter            0.26261601  0.413375677  0.06884329  0.02569731
## tomatoUserReviews          0.09772351 -0.126815013 -0.06026578  0.44463194
## num_voted_users            0.35162157 -0.114814179 -0.03940306  0.27747165
## imdbVotes                  0.35267478 -0.115679235 -0.03700495  0.27411764
## duration                   0.19402743 -0.002161922  0.05209069  0.17095599
##                                  PC5         PC6          PC7
## cast_total_facebook_likes  0.7442970309  0.57131652  0.199802020
## gross                      0.0371504409  0.07906111 -0.152176368
## budget                    -0.0007362571  0.05667193 -0.053212424
## profits                    0.0112150129 -0.03343718  0.009386848
## num_critic_for_reviews    -0.1138773738 -0.02993015 -0.049267286
## num_user_for_reviews      -0.0526316998 -0.15005822 -0.128820954
## tomatoUserRating           0.1329041063 -0.02261445  0.033480422
## tomatoRating              -0.1115986321  0.09782440  0.079849278
## tomatoReviews             -0.1770780212  0.01277372  0.140547048
## tomatoFresh               -0.2268091964  0.12742434  0.072353255
## tomatoRotten               0.0650392947 -0.18694074  0.125051821
## tomatoUserMeter            0.1129269225 -0.01149642  0.046752639
## tomatoUserReviews         -0.4651400109  0.45541260  0.538633766
## num_voted_users            0.0748453400 -0.04572894 -0.289635505
## imdbVotes                  0.0739755760 -0.04695573 -0.290051685
## duration                   0.2672984666 -0.60526603  0.637593608
##                                PC8          PC9        PC10
## cast_total_facebook_likes  0.17506954  0.020426054 -0.05914578
```

```
## gross                     -0.49744154 -0.713144917 -0.12911647
## budget                    -0.07563258 -0.126002502 -0.02731393
## profits                   -0.06604902 -0.077397867 -0.00958438
## num_critic_for_reviews     0.13444239  0.001216694  0.31909171
## num_user_for_reviews       0.28194783  0.201509011 -0.72552433
## tomatoUserRating          -0.47326075  0.279209041  0.10658684
## tomatoRating               0.22778587 -0.159635157 -0.12269418
## tomatoReviews             -0.05719630  0.051030136 -0.09529204
## tomatoFresh                0.17192690 -0.149931431 -0.03681209
## tomatoRotten              -0.38133069  0.334541978 -0.10487794
## tomatoUserMeter           -0.33876910  0.253709205 -0.24588388
## tomatoUserReviews         -0.10465477  0.163945599  0.11853168
## num_voted_users            0.08615309  0.164713071  0.32513376
## imdbVotes                  0.08207903  0.162650775  0.32435965
## duration                   0.14236754 -0.207984553  0.13231874
##                                   PC11          PC12         PC13
## cast_total_facebook_likes  0.026671700  0.0054614142 -0.007909208
## gross                      0.057016165 -0.0444305061  0.009528170
## budget                    -0.005351173 -0.0004943584 -0.001489339
## profits                    0.021365572 -0.0120598955  0.004156011
## num_critic_for_reviews     0.714951697 -0.1901441546  0.242204070
## num_user_for_reviews       0.267208970  0.0549783993 -0.177867179
## tomatoUserRating           0.223888668  0.0837567248 -0.630769826
## tomatoRating              -0.232555094 -0.7595995477 -0.114768959
## tomatoReviews             -0.325220711  0.1710452076 -0.024027007
## tomatoFresh               -0.223312546  0.4218467584 -0.039739387
## tomatoRotten              -0.197653057 -0.3955499700  0.023535302
## tomatoUserMeter           -0.017069174  0.0747094918  0.703002764
## tomatoUserReviews          0.082555510 -0.0190121060  0.009940902
## num_voted_users           -0.238176326 -0.0055349746  0.024580202
## imdbVotes                 -0.223861943 -0.0111021480  0.028969046
## duration                  -0.011638249  0.0699895743  0.004428365
##                                    PC14          PC15          PC16
## cast_total_facebook_likes -0.0009811830  3.250883e-18  6.205474e-17
## gross                      0.0030325036 -2.761084e-02  1.953830e-01
## budget                     0.0013590672  9.620970e-02 -6.808101e-01
## profits                   -0.0004809376  9.777874e-02 -6.919131e-01
## num_critic_for_reviews     0.0095219764 -1.222113e-15 -3.261280e-16
## num_user_for_reviews       0.0010740188 -4.961309e-16  1.387779e-16
## tomatoUserRating           0.0011539765 -4.644722e-16  5.828671e-16
## tomatoRating               0.0002088994  1.474515e-16 -3.035766e-16
## tomatoReviews             -0.0023931164  6.634452e-01  9.375575e-02
## tomatoFresh               -0.0031922237 -6.276595e-01 -8.869865e-02
## tomatoRotten               0.0010873997 -3.824979e-01 -5.405326e-02
## tomatoUserMeter            0.0023808116  2.515349e-16 -7.979728e-17
## tomatoUserReviews         -0.0003350411  1.006140e-16  1.327063e-16
## num_voted_users            0.7038662332  1.040834e-17 -2.151057e-16
## imdbVotes                 -0.7102420018  4.770490e-16  3.330669e-16
## duration                  -0.0005571525  1.305379e-16 -5.551115e-17
```

```r
plot(pca.movieData, type = "l", main = "PCA Movie Data")
```

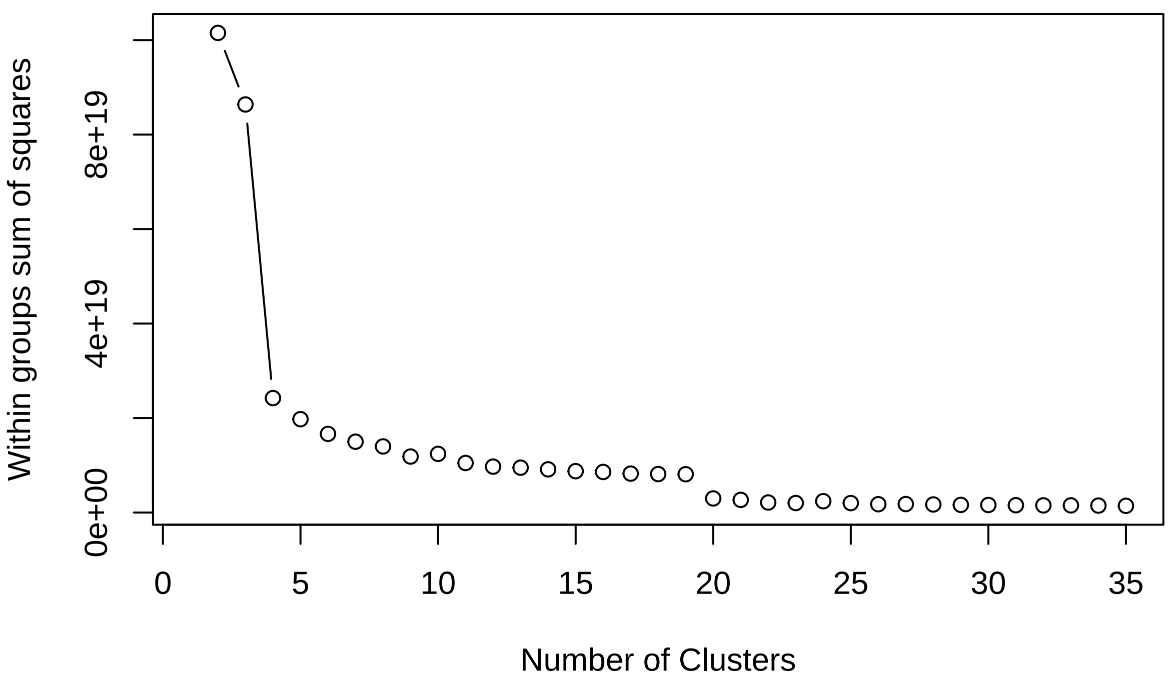


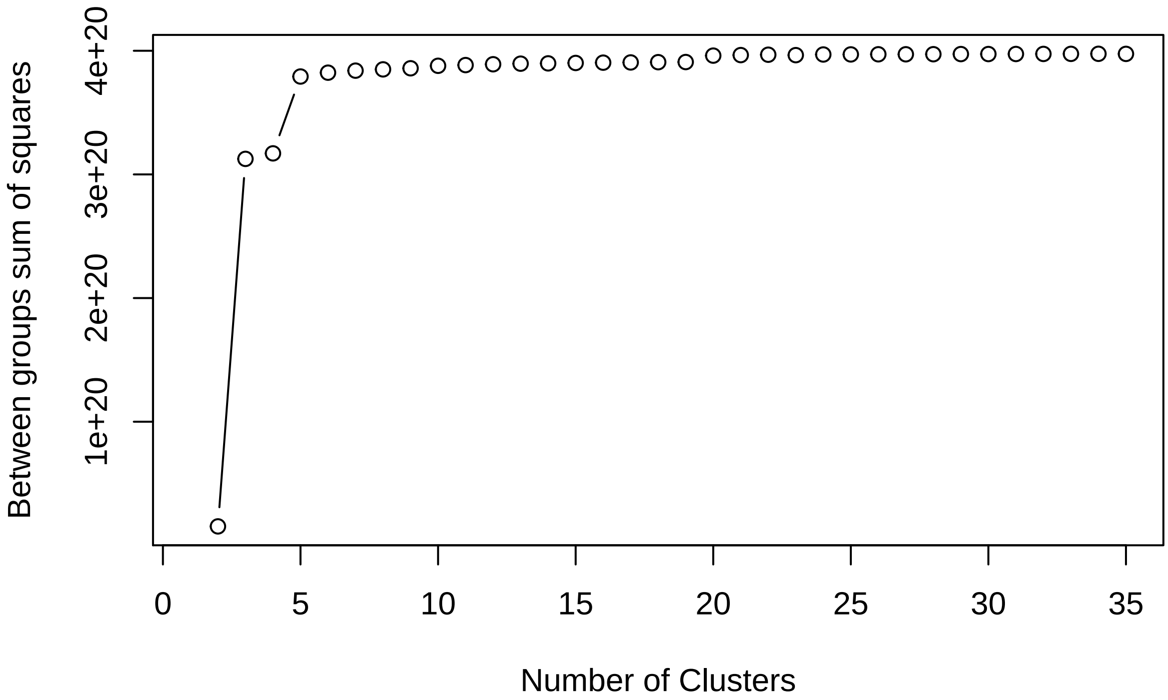

```
wssplot <- function(data, nc=15, seed=1234) {
  wss <- (nrow(data)-1)*sum(apply(data,2,var))
  bss <- (nrow(data)-1)*sum(apply(data,2,var))
  for (i in 2:nc) {
    set.seed(seed)
    wss[i] <- sum(kmeans(data[,c(7,12:26,38)], centers=i)$withinss)
    bss[i] <- sum(kmeans(data[,c(7,12:26,38)], centers=i)$betweenss)
    }

  plot(1:nc, wss, type="b", xlab="Number of Clusters",
       ylab="Within groups sum of squares")
  plot(1:nc, bss, type="b", xlab="Number of Clusters",
       ylab="Between groups sum of squares")
  }

wssplot(movieData3, nc=35)
```

```
## Warning in FUN(newX[, i], ...): NAs introduced by coercion

## Warning in FUN(newX[, i], ...): NAs introduced by coercion

## Warning in FUN(newX[, i], ...): NAs introduced by coercion

## Warning in FUN(newX[, i], ...): NAs introduced by coercion

## Warning in FUN(newX[, i], ...): NAs introduced by coercion

## Warning in FUN(newX[, i], ...): NAs introduced by coercion

## Warning in FUN(newX[, i], ...): NAs introduced by coercion
```

```
## Warning in FUN(newX[, i], ...): NAs introduced by coercion

## Warning in FUN(newX[, i], ...): NAs introduced by coercion

## Warning in FUN(newX[, i], ...): NAs introduced by coercion

## Warning in FUN(newX[, i], ...): NAs introduced by coercion

## Warning in FUN(newX[, i], ...): NAs introduced by coercion

## Warning in FUN(newX[, i], ...): NAs introduced by coercion

## Warning in FUN(newX[, i], ...): NAs introduced by coercion

## Warning in FUN(newX[, i], ...): NAs introduced by coercion

## Warning in FUN(newX[, i], ...): NAs introduced by coercion

## Warning in FUN(newX[, i], ...): NAs introduced by coercion

## Warning in FUN(newX[, i], ...): NAs introduced by coercion

## Warning in FUN(newX[, i], ...): NAs introduced by coercion

## Warning in FUN(newX[, i], ...): NAs introduced by coercion

## Warning in FUN(newX[, i], ...): NAs introduced by coercion

## Warning in FUN(newX[, i], ...): NAs introduced by coercion

## Warning in FUN(newX[, i], ...): NAs introduced by coercion

## Warning in FUN(newX[, i], ...): NAs introduced by coercion

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations
```

```
## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations
```






```
paste("from both of the graphs above, we can see that 20 is the optimum number of clusters because the l
```

```
## [1] "from both of the graphs above, we can see that 20 is the optimum number of clusters because the
```

```
paste("The within groups sum of squares was decreasing untill 20 clusters and the between groups sum of
```

```
## [1] "The within groups sum of squares was decreasing untill 20 clusters and the between groups sum of
```

```
set.seed(101)
sample = sample.split(movieData3, SplitRatio = .75)
train = subset(movieData3, sample == TRUE)
test = subset(movieData3, sample == FALSE)
```

```
#using Kmeans clustering

paste("The dimensions of the training data are")
```

```
## [1] "The dimensions of the training data are"
```

```
dim(train)
```

```
## [1] 2249   48
```

```
paste("The dimensions of the testing data are")
```

```
## [1] "The dimensions of the testing data are"
```

```r
dim(test)
```

```
## [1] 749  48
```

```r
k.means.fit <- kmeans(train[,c(7,12:14,16:26,38)], 20)
k.means.fit$centers
```

```
##    cast_total_facebook_likes       gross      budget       profits
## 1                  16549.536  67712868.0    24760156  4.295271e+07
## 2                   1173.000   2201412.0 12215500000 -1.221330e+10
## 3                  22365.793 263918931.7   183551724  8.036721e+07
## 4                     28.000    439162.0  1100000000 -1.099561e+09
## 5                  13438.308  27997612.7    74543684 -4.654607e+07
## 6                  13191.973  46733581.1    47512568 -7.789872e+05
## 7                  16859.471  89263989.2    83361345  5.902645e+06
## 8                  10226.813  12394205.4    32164579 -1.977037e+07
## 9                  23393.694  62875520.1   155547222 -9.267170e+07
## 10                 19563.950 251781661.9    63737500  1.880442e+08
## 11                 13626.344 114683644.3    42912295  7.177135e+07
## 12                 13164.065 142613315.6   159706522 -1.709321e+07
## 13                 15852.517 161229288.7    34210014  1.270193e+08
## 14                  6913.298   6061600.6     8344026 -2.282426e+06
## 15                 24615.283 181184689.7   115981132  6.520356e+07
## 16                  9870.771  33863374.1    16105434  1.775794e+07
## 17                 29787.556 364507148.3   133214815  2.312923e+08
## 18                    88.000   7292776.5   395000000 -3.877072e+08
## 19                 48618.500 641871762.5   202000000  4.398718e+08
## 20                  1209.333    901915.3  3033333333 -3.032431e+09
##    num_critic_for_reviews num_user_for_reviews tomatoUserRating
## 1                191.2552             392.9479         3.433854
## 2                363.0000             279.0000         3.200000
## 3                444.8276            1385.5517         3.637931
## 4                150.0000             430.0000         4.000000
## 5                165.2105             314.9850         3.063158
## 6                186.1421             374.8743         3.232240
## 7                217.5546             431.3109         3.279832
## 8                145.7729             208.2125         3.110256
## 9                294.1389             513.9167         3.172222
## 10               243.2500             771.1750         3.642500
## 11               205.3279             450.0984         3.454098
## 12               325.1087             853.5217         3.441304
## 13               227.4000             689.2500         3.670000
## 14               130.7601             187.0746         3.365154
## 15               308.6226             641.9057         3.588679
## 16               170.5229             345.1961         3.349673
## 17               409.7778            1472.2963         3.929630
## 18                71.0000             218.5000         3.250000
## 19               622.1667            2497.1667         4.083333
## 20               149.6667             248.6667         3.833333
##    tomatoRating tomatoReviews tomatoFresh tomatoRotten tomatoUserMeter
## 1      5.906250     123.80729    74.55208     49.25521        67.53646
## 2      3.600000     119.00000    10.00000    109.00000        50.00000
## 3      6.872414     252.44828   184.51724     67.93103        71.03448
```

```
## 4     7.500000     46.00000    40.00000     6.00000        90.00000
## 5     5.097744    126.51128    54.12782    72.38346        50.94737
## 6     5.331148    130.74863    61.02186    69.72678        57.95082
## 7     5.589076    153.18487    80.52941    72.65546        58.42857
## 8     5.067033    109.99634    49.61905    60.37729        51.87179
## 9     5.261111    183.72222    78.58333   105.13889        50.66667
## 10    6.710000    147.92500   102.40000    45.52500        75.85000
## 11    6.291803    140.33607    95.61475    44.72131        69.39344
## 12    6.060870    197.56522   118.26087    79.30435        62.10870
## 13    6.988333    143.20000   109.71667    33.48333        79.63333
## 14    5.926580     92.53160    59.08266    33.44895        63.62075
## 15    6.098113    187.00000   118.39623    68.60377        66.92453
## 16    5.829739    117.78431    72.48693    45.29739        64.96078
## 17    7.192593    243.25926   192.14815    51.11111        80.18519
## 18    5.050000     66.50000    26.50000    40.00000        59.50000
## 19    7.800000    286.66667   249.00000    37.66667        84.16667
## 20    7.633333     84.33333    72.66667    11.66667        87.66667
##    tomatoUserReviews num_voted_users imdbVotes duration
## 1          921602.95       131468.71 134443.70 108.4896
## 2           53348.00        68883.00  93664.00 110.0000
## 3         7407187.72       403357.83 412243.00 136.6552
## 4          147140.00       106160.00 108399.00 124.0000
## 5          149492.98        80849.70  83410.17 119.2030
## 6          353784.74       102738.87 104758.60 113.4481
## 7          538440.55       142402.88 145209.62 115.9496
## 8           76536.48        44776.53  45603.40 109.2234
## 9          200362.17       124343.94 136598.19 120.8611
## 10        3178748.10       367515.78 373808.55 114.4500
## 11        1730981.29       199206.19 203400.23 114.7049
## 12         404829.09       224382.48 240450.39 122.4130
## 13        2142255.23       322003.85 327945.15 120.0167
## 14          57296.52        46001.20  46892.35 102.5057
## 15        2218239.11       276171.17 282376.21 117.6038
## 16         282438.60       101484.71 103488.17 105.6438
## 17        5720295.93       574576.15 589836.19 131.7778
## 18          24083.50        28777.50  28952.50 229.0000
## 19        6913243.67       960746.00 980335.50 165.6667
## 20          98405.33        93554.33  96158.33 126.6667
```

```
# k.means.fit$cluster
k.means.fit$size
```

```
##  [1] 192   1  29   1 133 183 119 273  36  40 122  46  60 617  53 306  27
## [18]   2   6   3
```

```
kmeansPrediction <- as.data.frame(k.means.fit$centers)
kmeansPrediction <- kmeansPrediction[,5]
kmeansPrediction
```

```
##  [1] 191.2552 363.0000 444.8276 150.0000 165.2105 186.1421 217.5546
##  [8] 145.7729 294.1389 243.2500 205.3279 325.1087 227.4000 130.7601
## [15] 308.6226 170.5229 409.7778  71.0000 622.1667 149.6667
```

```r
closest.cluster <- function(x) {
  cluster.dist <- apply(k.means.fit$centers, 1, function(y) sqrt(sum((x-y)^2)))
  # print(c( "cluster: " ,(which.min(cluster.dist)[1]), kmeansPrediction[which.min(cluster.dist)[1]]))
  # print( kmeansPrediction[which.min(cluster.dist)[1]])
  return (kmeansPrediction[which.min(cluster.dist)[1]])
}

clusters2 <- apply(test[,c(7,12:14,16:26,38)], 1, closest.cluster)
```

```r
# analysis of k means clustering approach
RSFE_v1 <- test$imdb_score - clusters2
# RSFE_v1
RSFE1 <- sum(RSFE_v1)
# RSFE1
absRSFE1 <- abs(RSFE1)
absRSFE1
```

```
## [1] 129724.3
```

```r
length(RSFE_v1)
```

```
## [1] 749
```

```r
MSFE1 <- absRSFE1 / length(RSFE_v1)
MSFE1
```

```
## [1] 173.1967
```

```r
# calculating the mad
madtest <- mad(test$imdb_score, center = median(test$imdb_score), constant = 1)
madKmeans <- mad(clusters2, center = median(clusters2), constant = 1)
```

```r
#Using random forests for predicting the IMDB score

set.seed(7)
rfdf <- movieData3[sample(nrow(movieData3)), ]
rf.train <- rfdf[1:2200,]
rf.test <- rfdf[2201:nrow(rfdf), ]
paste("The dimensions of the training data are")
```

```
## [1] "The dimensions of the training data are"
```

```r
dim(rf.train)
```

```
## [1] 2200   48
```

```r
paste("The dimensions of the testing data are")
```

```
## [1] "The dimensions of the testing data are"
```

```r
dim(rf.test)
```

```
## [1] 798  48
```

```r
set.seed(5)

rf.rfModel <- randomForest(rfdf$imdb_score ~ rfdf$cast_total_facebook_likes + rfdf$gross + rfdf$budget +

rf.rfModel
```

```
##
## Call:
##  randomForest(formula = rfdf$imdb_score ~ rfdf$cast_total_facebook_likes +      rfdf$gross + rfdf$bud
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 5
##
##           Mean of squared residuals: 0.1498318
##                     % Var explained: 86.29
```

```r
#validating the random forest model
#RMSE
rf.predictedValues <- predict(rf.rfModel, rfdf)
# rf.predictedValues


RSFE_v2 <- rfdf$imdb_score - rf.predictedValues
# RSFE_v2
RSFE2 <- sum(RSFE_v2)
# RSFE2
absRSFE2 <- abs(RSFE2)
absRSFE2
```

```
## [1] 13.66584
```

```r
length(RSFE_v1)
```

```
## [1] 749
```

```r
MSFE2 <- absRSFE2 / length(RSFE_v2)

mean(rf.rfModel$mse)
```

```
## [1] 0.1554552
```

```r
# calculating median absolute deviation

madtestrf <- mad(rfdf$imdb_score,center = median(rfdf$imdb_score),constant = 1)
madpredictedrf <- mad(rf.predictedValues, center = median(rf.predictedValues), constant = 1)

print(c("MSFE for prediction using random forests" , MSFE2))
```

```
## [1] "MSFE for prediction using random forests"
## [2] "0.00455831776740199"
```

```r
print(c("MSFE for prediction using K means clustering" , MSFE1))
```

```
## [1] "MSFE for prediction using K means clustering"
## [2] "173.196683616234"
```

```r
print(c("MSFE for prediction using Linear Regression" , MSFE))
```

```
## [1] "MSFE for prediction using Linear Regression"
## [2] "1.09274386292992e-14"
```

```r
print(c("MAD for Original data:" , madtestrf))
```

```
## [1] "MAD for Original data:" "0.699999999999999"
```

```r
print(c("MAD for prediction using random forests" , madpredictedrf))
```

```
## [1] "MAD for prediction using random forests"
## [2] "0.643714999999992"
```

```r
print(c("MAD for Original data:" , madtestrf))
```

```
## [1] "MAD for Original data:" "0.699999999999999"
```

```r
print(c("MAD for prediction using K means clustering" , madKmeans))
```

```
## [1] "MAD for prediction using K means clustering"
## [2] "24.7499820440997"
```

```r
print(c("MAD for Original data:" , Madoriginal))
```

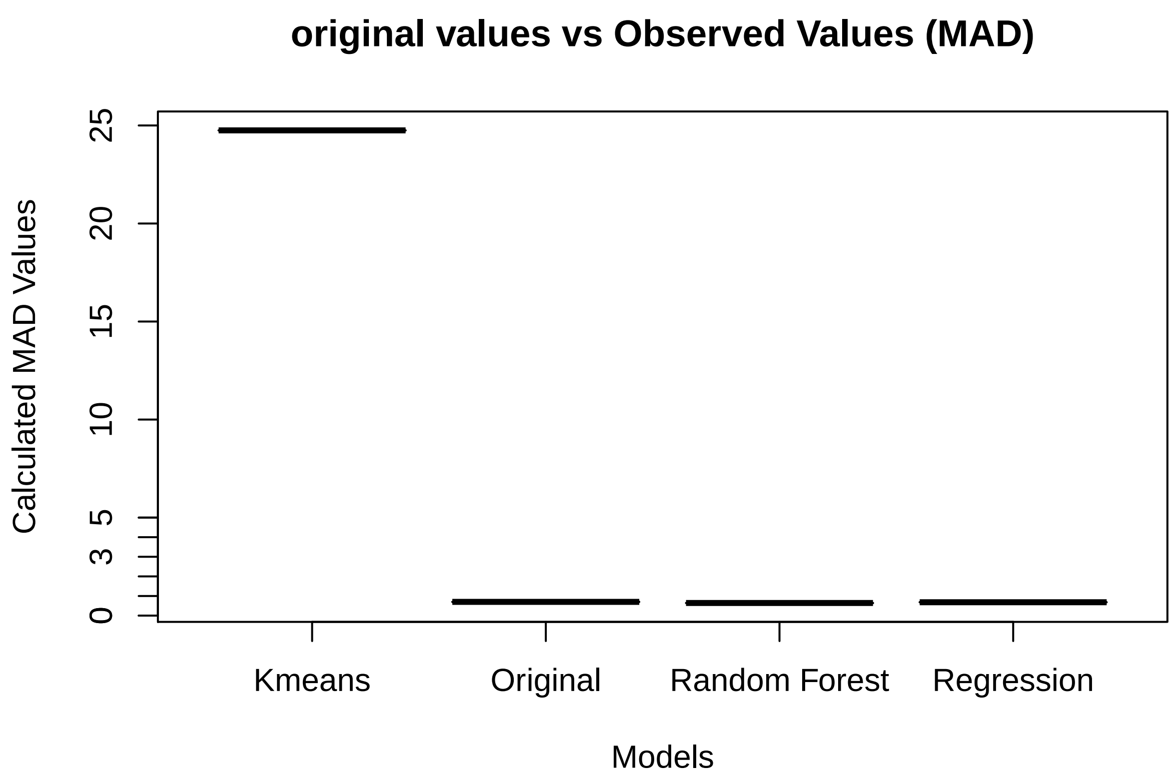
```
## [1] "MAD for Original data:" "0.699999999999999"
```

```r
print(c("MAD for prediction using Linear Regression" , MadRegression))
```

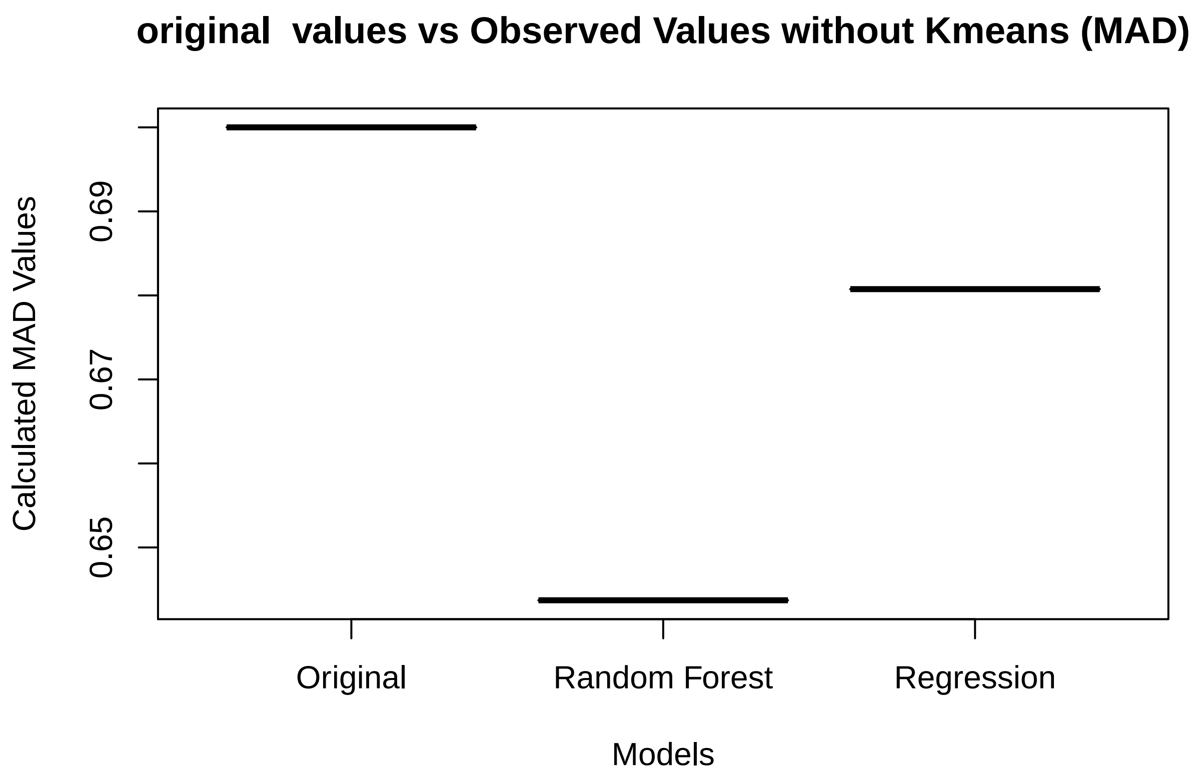
```
## [1] "MAD for prediction using Linear Regression"
## [2] "0.680744307507511"
```

```r
# Visualization of results
visualization <- as.data.frame(c(1:6))
visualization$new <- c(1:6)
colnames(visualization) <- c("MSFE","MAD")
row.names(visualization) <- c("Original","Regression","Kmeans","Random Forest","x","y")
visualization[1,1] <- 0
visualization[2,1] <- MSFE
```

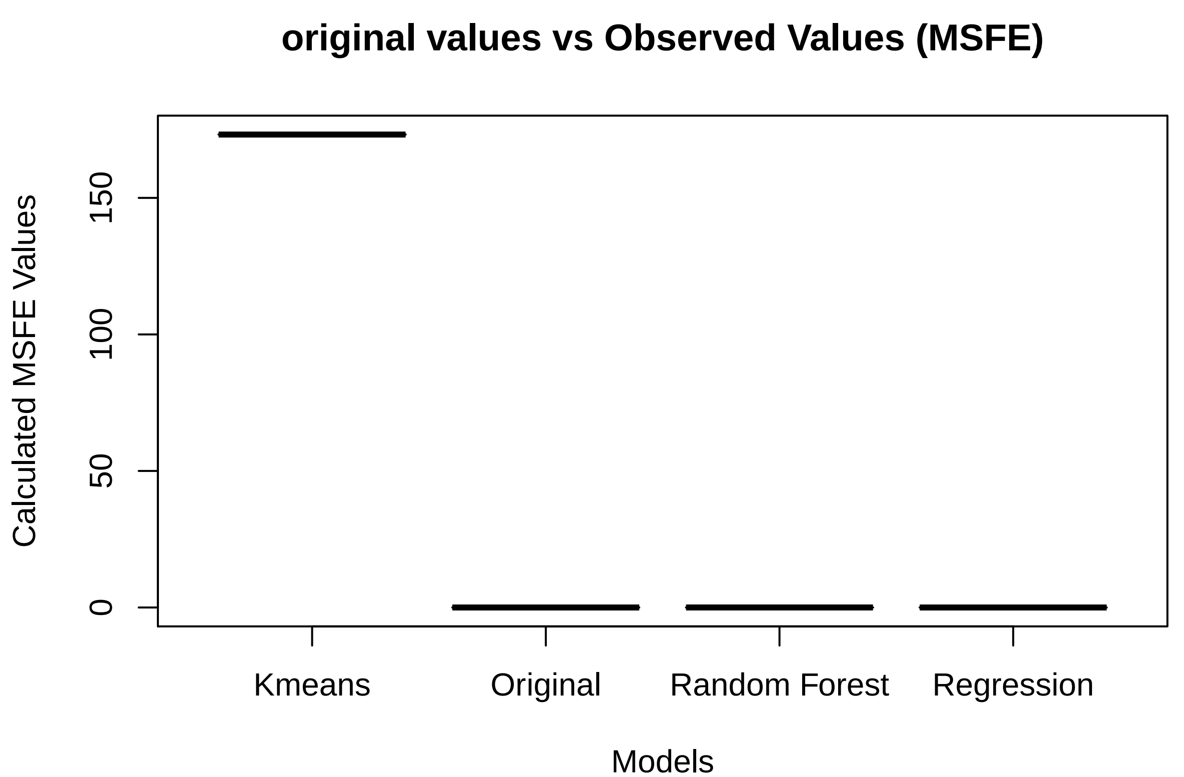
```r
visualization[3,1] <- MSFE1
visualization[4,1] <- MSFE2
visualization[1,2] <- Madoriginal
visualization[2,2] <- MadRegression
visualization[3,2] <- madKmeans
visualization[4,2] <- madpredictedrf
visualization <- visualization[c(1:4),]
plot(as.factor(rownames(visualization)),visualization$MAD, xlab="Models", ylab="Calculated MAD Values",
axis(2,at = c(0:5))
```



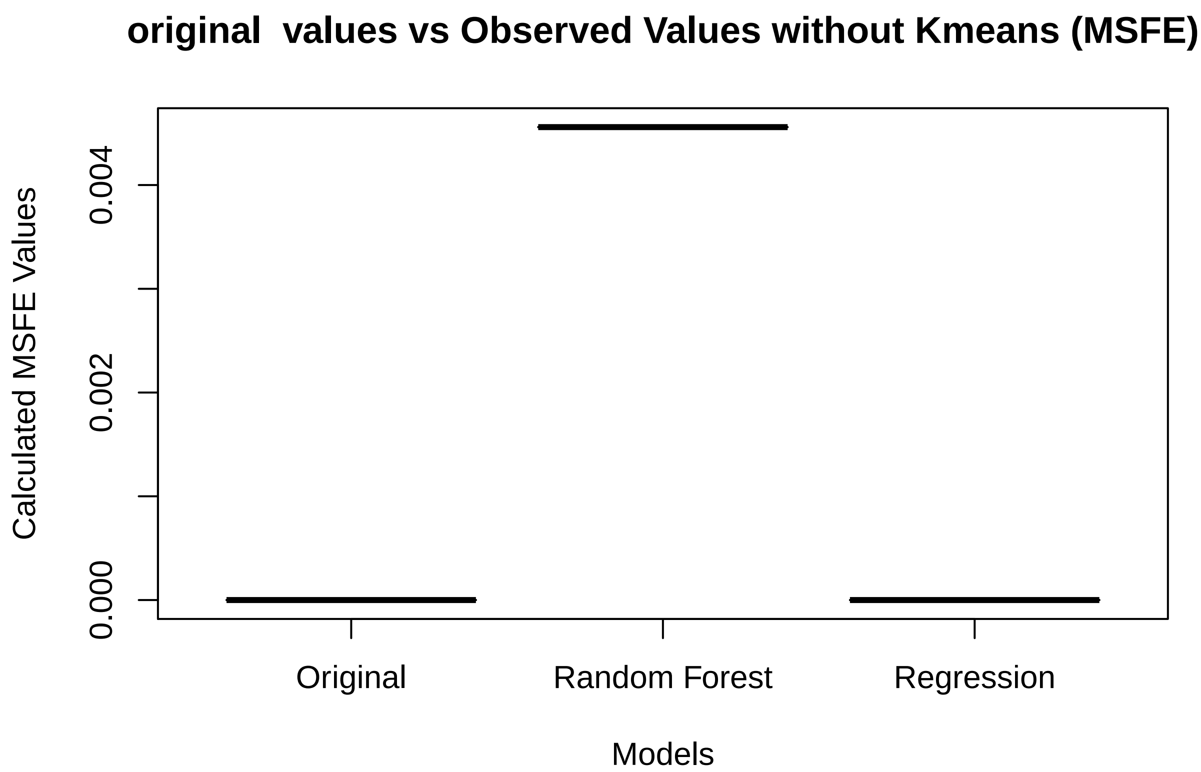

```r
plot(as.factor(rownames(visualization[c(1,2,4),])),visualization$MAD[c(1,2,4)], xlab="Models", ylab="Ca
```




```
# Visualizing MSFE values
plot(as.factor(rownames(visualization)),visualization$MSFE, xlab="Models", ylab="Calculated MSFE Values
```



```r
plot(as.factor(rownames(visualization[c(1,2,4),])),visualization$MSFE[c(1,2,4)], xlab="Models", ylab="Ca
```




```r
paste("Looking at the MSFE values for all the three models, we can clearly see that the k-means cluster
```

```
## [1] "Looking at the MSFE values for all the three models, we can clearly see that the k-means cluste
```

```r
paste("order of performances: Linear Regression BETTER THAN Random Forests BETTER THAN K Means Clusteri
```

```
## [1] "order of performances: Linear Regression BETTER THAN Random Forests BETTER THAN K Means Cluster
```