Project 2: Modeling and Evaluation

GID: mmendiola3

Data

We will use the same dataset as Project 1: `movies_merged`.

Objective

Your goal in this project is to build a linear regression model that can predict the `Gross` revenue earned by a movie based on other variables. You may use R packages to fit and evaluate a regression model (no need to implement regression yourself). Please stick to linear regression, however.

Instructions

You should be familiar with using an RMarkdown Notebook by now. Remember that you have to open it in RStudio, and you can run code chunks by pressing *Cmd+Shift+Enter*.

Please complete the tasks below and submit this R Markdown file (as **pr2.Rmd**) containing all completed code chunks and written responses, as well as a PDF export of it (as **pr2.pdf**) which should include all of that plus output, plots and written responses for each task.

*Note that **Setup** and **Data Preprocessing** steps do not carry any points, however, they need to be completed as instructed in order to get meaningful results.*

Setup

Same as Project 1, load the dataset into memory:

```
load('movies_merged')
```

This creates an object of the same name (`movies_merged`). For convenience, you can copy it to `df` and start using it:

```
df = movies_merged
cat("Dataset has", dim(df)[1], "rows and", dim(df)[2], "columns", end="\n", file="")
```

```
## Dataset has 40789 rows and 39 columns
```

```
colnames(df)
```

```
##  [1] "Title"            "Year"             "Rated"
##  [4] "Released"         "Runtime"          "Genre"
##  [7] "Director"         "Writer"           "Actors"
## [10] "Plot"             "Language"         "Country"
## [13] "Awards"           "Poster"           "Metascore"
## [16] "imdbRating"       "imdbVotes"        "imdbID"
## [19] "Type"             "tomatoMeter"      "tomatoImage"
## [22] "tomatoRating"     "tomatoReviews"    "tomatoFresh"
## [25] "tomatoRotten"     "tomatoConsensus"  "tomatoUserMeter"
```

```
## [28] "tomatoUserRating"  "tomatoUserReviews" "tomatoURL"
## [31] "DVD"               "BoxOffice"         "Production"
## [34] "Website"           "Response"          "Budget"
## [37] "Domestic_Gross"    "Gross"             "Date"
```

Load R packages

Load any R packages that you will need to use. You can come back to this chunk, edit it and re-run to load any additional packages later.

```
suppressMessages(library(ggplot2))
suppressMessages(library(GGally))
suppressMessages(library(reshape2))
suppressMessages(library(caret))
suppressMessages(library(plyr))
suppressMessages(library(dplyr))
suppressMessages(library(splitstackshape))
suppressMessages(library(stringr))
suppressMessages(library(MASS))
suppressMessages(library(doParallel))
suppressMessages(library(rbenchmark))
```

```
# Setup parallel processing
cl <- makeCluster(detectCores()-2)
registerDoParallel(cl)
```

If you are using any non-standard packages (ones that have not been discussed in class or explicitly allowed for this project), please mention them below. Include any special instructions if they cannot be installed using the regular `install.packages('<pkg name>')` command.

Non-standard packages used:

```
reshape2
caret
MASS
plyr
dplyr
splitstackshape
stringr
doParallel
rbenchmark
```

Data Preprocessing

Before we start building models, we should clean up the dataset and perform any preprocessing steps that may be necessary. Some of these steps can be copied in from your Project 1 solution. It may be helpful to print the dimensions of the resulting dataframe at each step.

1. Remove non-movie rows

```r
# TODO: Remove all rows from df that do not correspond to movies
df <- subset(df, Type == 'movie')
nrow(df)
```

```
## [1] 40000
```

2. Drop rows with missing `Gross` value

Since our goal is to model `Gross` revenue against other variables, rows that have missing `Gross` values are not useful to us.

```r
# TODO: Remove rows with missing Gross value
df <- subset(df, !is.na(Gross))
nrow(df)
```

```
## [1] 4558
```

3. Exclude movies released prior to 2000

Inflation and other global financial factors may affect the revenue earned by movies during certain periods of time. Taking that into account is out of scope for this project, so let's exclude all movies that were released prior to the year 2000 (you may use `Released`, `Date` or `Year` for this purpose).

```r
# TODO: Exclude movies released prior to 2000
df <- subset(df, Year >= 2000)
nrow(df)
```

```
## [1] 3332
```

4. Eliminate mismatched rows

Note: You may compare the ***`Released`*** *column (string representation of release date) with either* ***`Year`*** *or* ***`Date`*** *(numeric representation of the year) to find mismatches. The goal is to avoid removing more than 10% of the rows.*

```r
# TODO: Remove mismatched rows
year_delta <- function(year_, date) {
  if (is.na(year_) || is.na(date)) {
    return(0)
  } else if (year_ == as.numeric(format(date, "%Y"))) {
    return(0)
  }
  date = as.Date(date)
  year_start = as.Date(ISOdate(year_, 1, 1))
  year_end = as.Date(ISOdate(year_, 12, 31))
  diff = min(abs(c(date - year_start, date - year_end)))
  return(diff)
}

df$DaysDiff = mapply(year_delta, df$Year, df$Released)

# Threshold of days between Release and Year
threshold = 90
```

```r
df_dedupped <- subset(df, DaysDiff < threshold)
sprintf("percent of rows removed: %f", 100 * (1 - (nrow(df_dedupped) / nrow(df))))
```

```
## [1] "percent of rows removed: 9.903962"
```

```r
# Commit changes to df
df_before_dedup <- cbind(df)
df <- df_dedupped
```

5. Drop `Domestic_Gross` column

`Domestic_Gross` is basically the amount of revenue a movie earned within the US. Understandably, it is very highly correlated with `Gross` and is in fact equal to it for movies that were not released globally. Hence, it should be removed for modeling purposes.

```r
# TODO: Exclude the `Domestic_Gross` column
df <- subset(df, select=-Domestic_Gross)
```

6. Process `Runtime` column

```r
# TODO: Replace df$Runtime with a numeric column containing the runtime in minutes
parseRuntime <- function(string) {
  parts = unlist(strsplit(string, ' '))
  if (length(parts) == 2) {
    return(as.numeric(parts[1]))
  } else if (length(parts) == 4) {
    return(as.numeric(parts[1]) * 60 + as.numeric(parts[3]))
  } else {
    return(NA)
  }
}

df$Runtime = sapply(df$Runtime, parseRuntime)
```

Perform any additional preprocessing steps that you find necessary, such as dealing with missing values or highly correlated columns (feel free to add more code chunks, markdown blocks and plots here as necessary).

```r
# TODO(optional): Additional preprocessing
# Generate month feature to replace Released
df$Month = as.numeric(format(df$Released, "%m"))

# Convert Metascore to num
df$Metascore <- as.numeric(df$Metascore)
```

```
## Warning: NAs introduced by coercion
```

```r
# Remove featues with clearly no correlation with Gross
df <- subset(df, select=-c(Date, Released, Poster, imdbID, Type, tomatoImage, tomatoURL, BoxOffice, Webs

# Filter major outliers
df <- subset(df, Gross < 2e9)
df <- subset(df, Gross > 1000)

# Clean column names
```

```
colnames(df) <- make.names(colnames(df), unique=T)

# Remove features with more than 10% NAs
pMiss <- function(x){round(sum(is.na(x))/length(x)*100)}
mc <- apply(df, 2, pMiss)
mc[mc>0]
```

```
##           Runtime         Metascore        imdbRating         imdbVotes
##                 1                 6                 1                 1
##       tomatoMeter      tomatoRating     tomatoReviews       tomatoFresh
##                 6                 6                 6                 6
##      tomatoRotten   tomatoUserMeter  tomatoUserRating tomatoUserReviews
##                 6                 4                 4                 2
##               DVD             Month
##                 5                 1
```

```
columns_to_keep <- names(mc[mc < 10])
df <- subset(df, select=columns_to_keep)

# Remove high correlation features
df_numeric <- df[, sapply(df, is.numeric)]
cor_matrix <- cor(subset(df_numeric, select=-Gross), use='pairwise.complete.obs')
colnames(cor_matrix[, findCorrelation(cor_matrix, cutoff=0.75)])
```

```
## [1] "tomatoFresh"     "tomatoRating"    "tomatoMeter"     "imdbRating"
## [5] "tomatoUserMeter"
```

```
df <- subset(df, select=-c(tomatoUserMeter, tomatoUserRating))

# Remove samples with NAs
df <- df[complete.cases(df), ]
```

***Note**: Do NOT convert categorical variables (like `Genre`) into binary columns yet. You will do that later as part of a model improvement task.*

Final preprocessed dataset

Report the dimensions of the preprocessed dataset you will be using for modeling and evaluation, and print all the final column names. (Again, `Domestic_Gross` should not be in this list!)

```
# TODO: Print the dimensions of the final preprocessed dataset and column names
dim(df)
```

```
## [1] 2472   27
```

```
colnames(df)
```

```
##  [1] "Title"             "Year"              "Rated"
##  [4] "Runtime"           "Genre"             "Director"
##  [7] "Writer"            "Actors"            "Plot"
## [10] "Language"          "Country"           "Awards"
## [13] "Metascore"         "imdbRating"        "imdbVotes"
## [16] "tomatoMeter"       "tomatoRating"      "tomatoReviews"
## [19] "tomatoFresh"       "tomatoRotten"      "tomatoConsensus"
## [22] "tomatoUserReviews" "DVD"               "Production"
## [25] "Budget"            "Gross"             "Month"
```

Evaluation Strategy

In each of the tasks described in the next section, you will build a regression model. In order to compare their performance, use the following evaluation procedure every time:

1. Randomly divide the rows into two sets of sizes 5% and 95%.
2. Use the first set for training and the second for testing.
3. Compute the Root Mean Squared Error (RMSE) on the train and test sets.
4. Repeat the above data partition and model training and evaluation 10 times and average the RMSE results so the results stabilize.
5. Repeat the above steps for different proportions of train and test sizes: 10%-90%, 15%-85%, . . . , 95%-5% (total 19 splits including the initial 5%-95%).
6. Generate a graph of the averaged train and test RMSE as a function of the train set size (%).

You can define a helper function that applies this procedure to a given model and reuse it.

```
# Plot RMSE over various training sizes
plot_sub_data <- function(df, title, data_type) {
  df <- melt(df, id.vars='train_size', measure.vars=c('train', 'test'))
  return(ggplot(df, aes(train_size, value, color=variable)) +
    geom_line() +
    geom_point() +
    ggtitle(title) +
    labs(x='Train size (ratio of original data)', y=data_type))
}

# Split data into train and test sets
split_data <- function(df, train_ratio=0.8) {
  train_indicies<- sample(1:nrow(df), floor(train_ratio * nrow(df)))
  return(list(train=df[train_indicies,], test=df[-train_indicies,]))
}

# Fit a model with a random train/test sample and return RMSE for both sets
test_model <-function (formula, df, train_ratio=0.8, y_label='Gross') {
  df_sample <- split_data(df, train_ratio)
  model <- lm(formula, df_sample$train)
  suppressWarnings(test_residuals <- df_sample$test[, y_label] - predict(model, df_sample$test))
  result <- list(train=sqrt(mean(model$residuals^2)),
                 test=sqrt(mean(test_residuals^2)))
  return(result)
}

# Caculate the average RMSE over a number of trials for a given formula, data, and train/test split
evaluate_model <- function(formula, df, train_ratio=0.95, y_label='Gross', trials=10) {
  result <- foreach(i=1:trials, .combine=rbind, .export=c('test_model', 'split_data')) %dopar% {
    as.data.frame(test_model(formula, df, train_ratio=train_ratio, y_label=y_label))
  }
  return(list(
            train=mean(result$train),
            test=mean(result$test)
        ))
}

# Evaluate performance of a model over a range of train/test set sizes
evaluate_model_range <- function(formula, df, y_label='Gross', trials=50, max_time=20) {
```

```
  sample_sizes <- seq(0.05, 0.95, 0.05)
  duration <- system.time(evaluate_model(formula, df, trials=detectCores()))[3]
  trials_till_max_time <- ceiling(max_time * 4/ (duration * length(sample_sizes)))
  trials <- min(trials, trials_till_max_time)
  print(paste(trials, 'trials'))
  rmse <- data.frame()
  for (i in sample_sizes) {
    rmse <- rbind(rmse, c(evaluate_model(formula, df, i, y_label, trials), train_size=i))
  }
  return(list(rmse=rmse, plot=plot_sub_data(rmse, 'RMSE vs. Training size', 'RMSE')))
}

# Run a t-test on the set of RMSEs observed over a number of trials for two different models
evaluate_model_change <- function(formula1, df1, formula2, df2, train_ratio=0.95, y_label='Gross', trial
  duration <- system.time(evaluate_model(formula1, df1, trials=detectCores()))[3]
  duration <- duration + system.time(evaluate_model(formula2, df2, trials=detectCores()))[3]
  trials_till_max_time <- ceiling(max_time * 4/ duration)
  trials <- min(trials, trials_till_max_time)
  print(paste(trials, 'trials'))
  m1 <- foreach(i=1:trials, .combine=rbind, .export=c('test_model', 'split_data')) %dopar% {
    as.data.frame(test_model(formula1, df1, train_ratio=train_ratio, y_label=y_label))
  }
  m2 <- foreach(i=1:trials, .combine=rbind, .export=c('test_model', 'split_data')) %dopar% {
    as.data.frame(test_model(formula2, df2, train_ratio=train_ratio, y_label=y_label))
  }
  return(list(train=t.test(m1$train, m2$train, alternative='greater'),
              test=t.test(m1$test, m2$test, alternative='greater')))
}
```

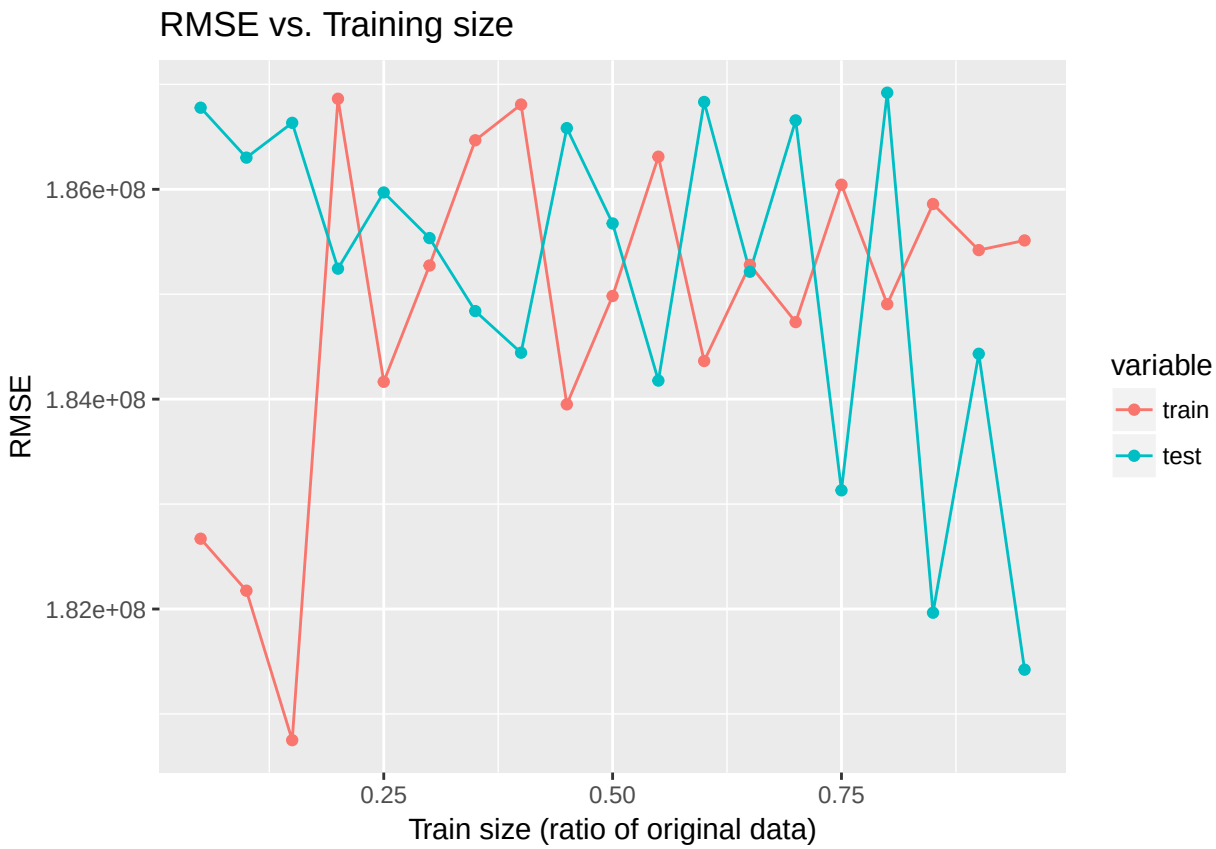
```
# Run benchmark on random data
df_random <- data.frame(x=runif(nrow(df)), Gross=df$Gross)
evaluate_model_range(Gross~., df_random)
```

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

```
## [1] "50 trials"
```

```
## $rmse
##        train      test train_size
## 1  182669906 186777837       0.05
## 2  182174620 186301922       0.10
## 3  180750029 186633512       0.15
## 4  186863233 185243789       0.20
## 5  184165679 185970010       0.25
## 6  185273775 185535201       0.30
## 7  186467531 184839294       0.35
## 8  186808197 184442551       0.40
## 9  183950618 186583329       0.45
## 10 184983320 185674560       0.50
## 11 186310764 184177005       0.55
## 12 184363696 186832518       0.60
## 13 185279562 185214190       0.65
## 14 184734360 186657131       0.70
## 15 186043759 183130907       0.75
## 16 184905215 186920737       0.80
```

```
## 17 185859665 181964738       0.85
## 18 185420531 184432473       0.90
## 19 185512732 181421960       0.95
##
## $plot
```




```
evaluate_model_change(Gross~., df_random, Gross~.^2, df_random, trials=500)
```

```
## [1] "500 trials"
```

```
## $train
##
##  Welch Two Sample t-test
##
## data:  m1$train and m2$train
## t = 0.042016, df = 992.32, p-value = 0.4832
## alternative hypothesis: true difference in means is greater than 0
## 95 percent confidence interval:
##  -163115     Inf
## sample estimates:
## mean of x mean of y
## 185583496 185579225
##
##
## $test
##
##  Welch Two Sample t-test
##
```

```
## data:  m1$test and m2$test
## t = 0.12841, df = 994.49, p-value = 0.4489
## alternative hypothesis: true difference in means is greater than 0
## 95 percent confidence interval:
##  -2928911      Inf
## sample estimates:
## mean of x mean of y
## 180084878 179837107
```

Tasks

Each of the following tasks is worth 20 points. Remember to build each model as specified, evaluate it using the strategy outlined above, and plot the training and test errors by training set size (%).

1. Numeric variables

Use linear regression to predict `Gross` based on all available *numeric* variables.

```
# TODO: Build & evaluate model 1 (numeric variables only)
df_numeric <- df[, sapply(df, is.numeric)]
names(df_numeric)
```

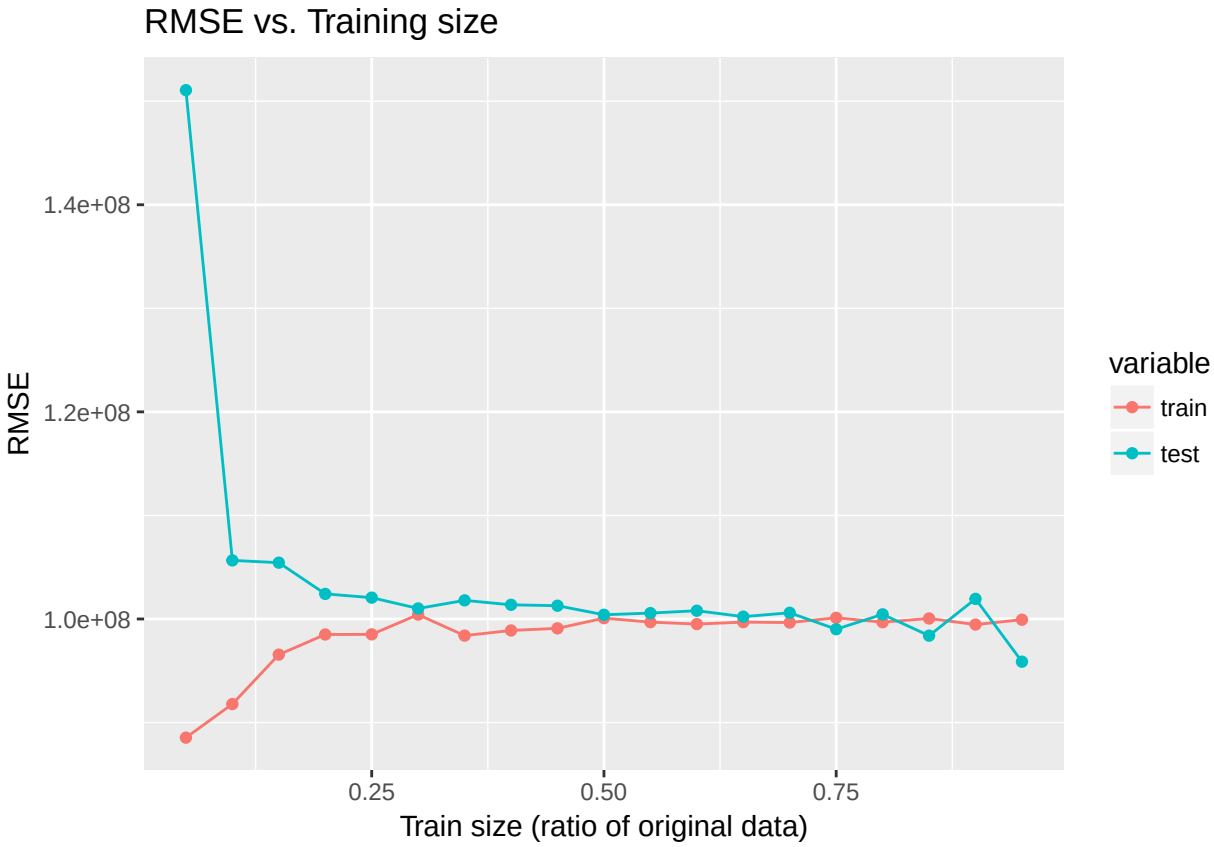
```
##  [1] "Year"              "Runtime"           "Metascore"
##  [4] "imdbRating"        "imdbVotes"         "tomatoMeter"
##  [7] "tomatoRating"      "tomatoReviews"     "tomatoFresh"
## [10] "tomatoRotten"      "tomatoUserReviews" "Budget"
## [13] "Gross"             "Month"
```

```
evaluate_model_range('Gross~.', df_numeric, 'Gross')
```

```
## [1] "50 trials"
```

```
## $rmse
##        train      test train_size
## 1   88541307 151065057       0.05
## 2   91785050 105657207       0.10
## 3   96557713 105437950       0.15
## 4   98502842 102422309       0.20
## 5   98513593 102060991       0.25
## 6  100420263 101015901       0.30
## 7   98393920 101797992       0.35
## 8   98892247 101369049       0.40
## 9   99097666 101281886       0.45
## 10 100078162 100408372       0.50
## 11  99696095 100568752       0.55
## 12  99510572 100798232       0.60
## 13  99696949 100226752       0.65
## 14  99656528 100597978       0.70
## 15 100115759  99014062       0.75
## 16  99689636 100449524       0.80
## 17 100048752  98389354       0.85
## 18  99462466 101944551       0.90
## 19  99930547  95876658       0.95
##
```

```
## $plot
```



Q: List all the numeric variables you used.

A:

[1] "Year" "Runtime" "Metascore" "imdbRating" "imdbVotes"
[6] "tomatoMeter" "tomatoRating" "tomatoReviews" "tomatoFresh" "tomatoRotten"
[11] "tomatoUserReviews" "Budget" "Month"

The resulting model produced an RMSE of ~1e8 for both training and test samples.

2. Feature transformations

Try to improve the prediction quality from **Task 1** as much as possible by adding feature transformations of the numeric variables. Explore both numeric transformations such as power transforms and non-numeric transformations of the numeric variables like binning (e.g. `is_budget_greater_than_3M`).

Setup functions

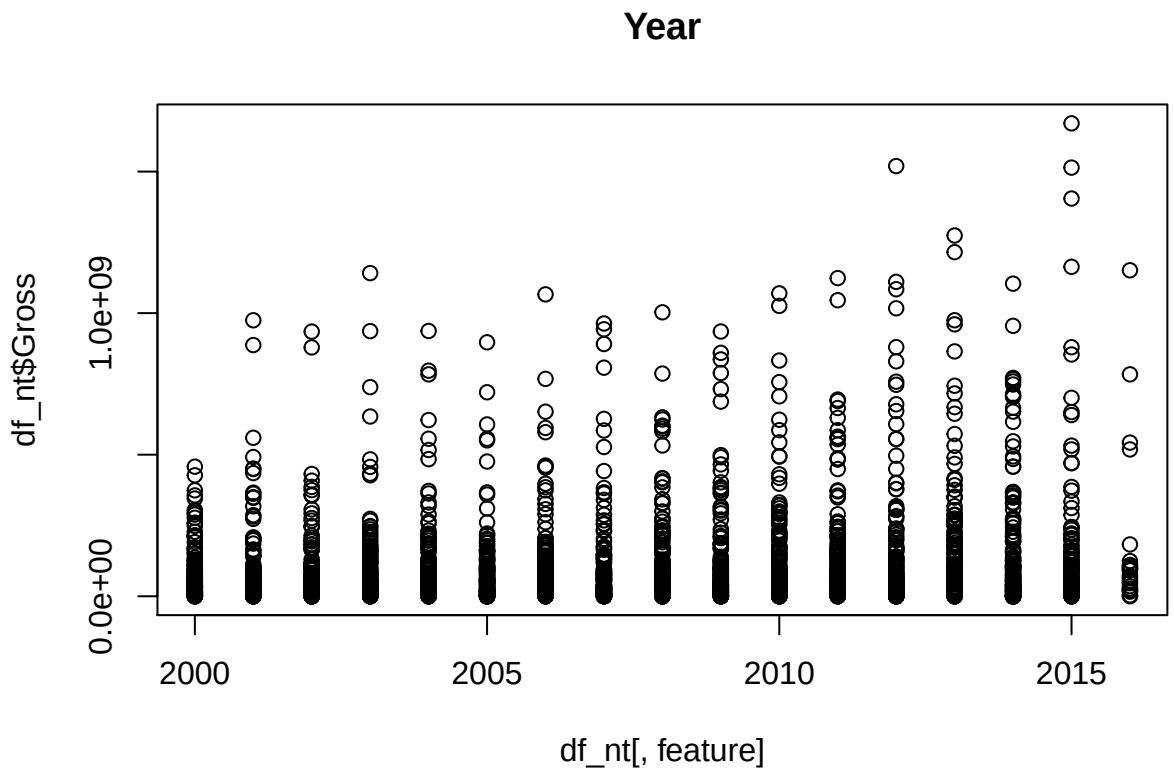
These help to evaluate numeric and non-numeric transformations

```
# TODO: Build & evaluate model 2 (transformed numeric variables only)

df_nt <- cbind(df_numeric)

# Generate a set of models based on a set of independent var transformations
reg_transforms <- function(x, y) {
```

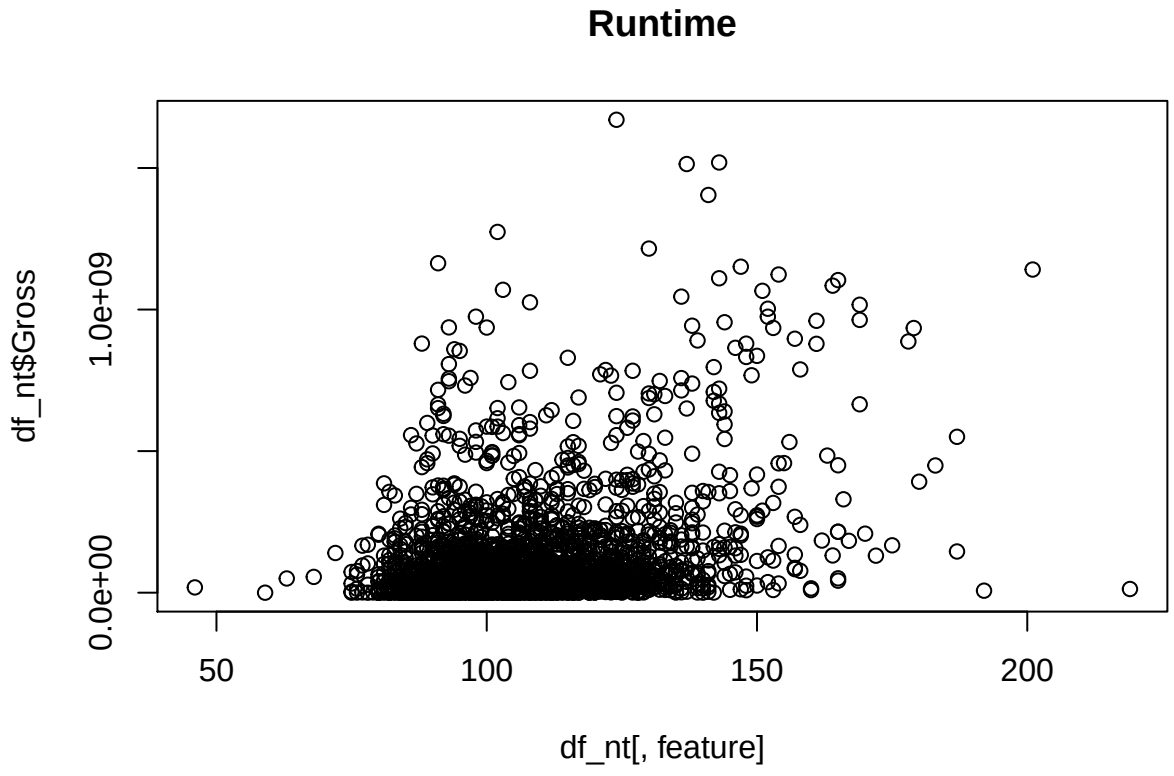
```
  return(list(
    linear = lm(y~x),
    log = lm(y~log(x+1)),
    quad = lm(y~I(x^2)),
    cubic = lm(y~I(x^3)),
    quartic = lm(y~I(x^4)),
    quintic = lm(y~I(x^5)),
    reciprocal = lm(y~1/(x+1))
  ))
}

get_r2 <- function(fit) {
  return(summary(fit)$r.squared)
}

get_rmse <- function(fit) {
  return(sqrt(mean(fit$residuals^2)))
}

get_cor <- function(fit) {
  return(as.vector(cor(fit$model[1], fit$model[2])))
}

# Calculate model fit metrics for a given model
eval_fit <- function(fit) {
  return(data.frame(r2=get_r2(fit), rmse=get_rmse(fit), cor=get_cor(fit)))
}

# Calculate the optimum number of ntile bins for a given feature
best_bins <- function(x, y) {
  fit <- lm(y ~ x)
  result <- list(r2=summary(fit)$r.squared, bins=0)
  for (i in 1:200) {
    b <- ntile(x, i)
    fit <- lm(y ~ x + b)
    r2 <- summary(fit)$r.squared
    if (r2 > result$r2) {
      result$r2 <- r2
      result$bins <- i
    }
  }
  return(result)
}
```

Visualize features

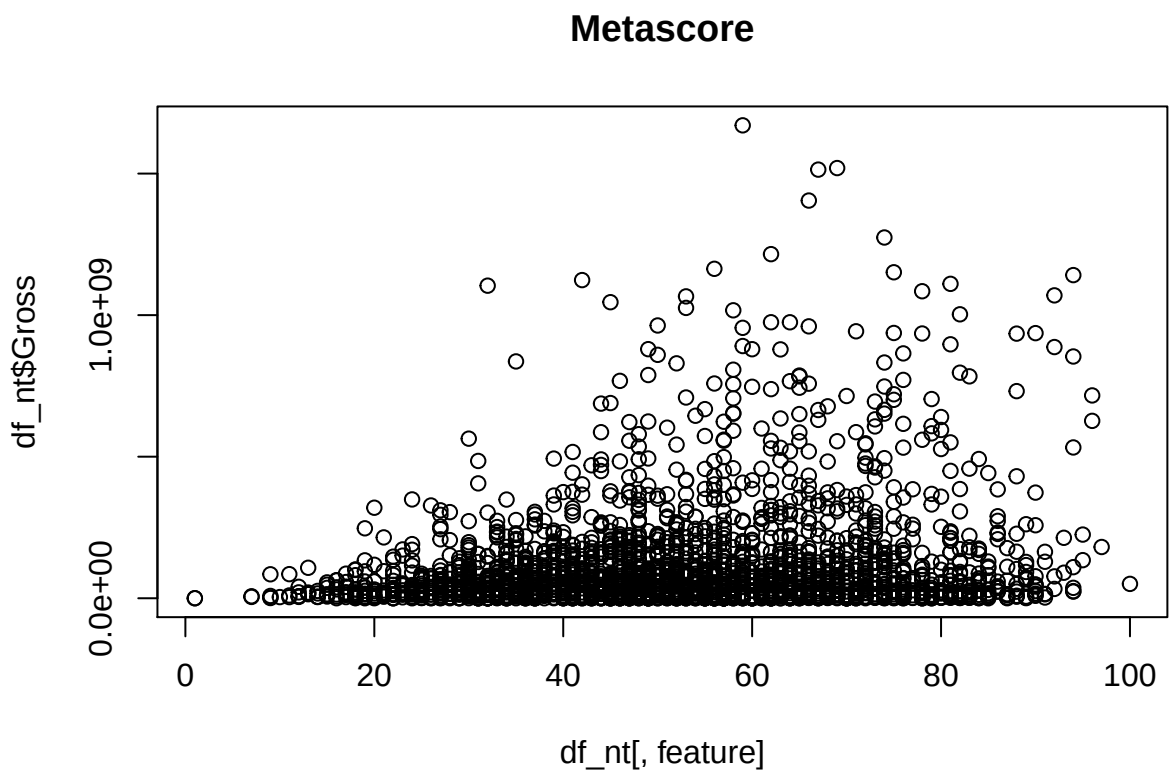
```
for (feature in setdiff(names(df_nt), 'Gross')) {
  print(plot(df_nt[,feature], df_nt$Gross, main=feature))
}
```


Year
df_nt$Gross
0.0e+00
1.0e+09
2000
2005
2010
2015
df_nt[, feature]

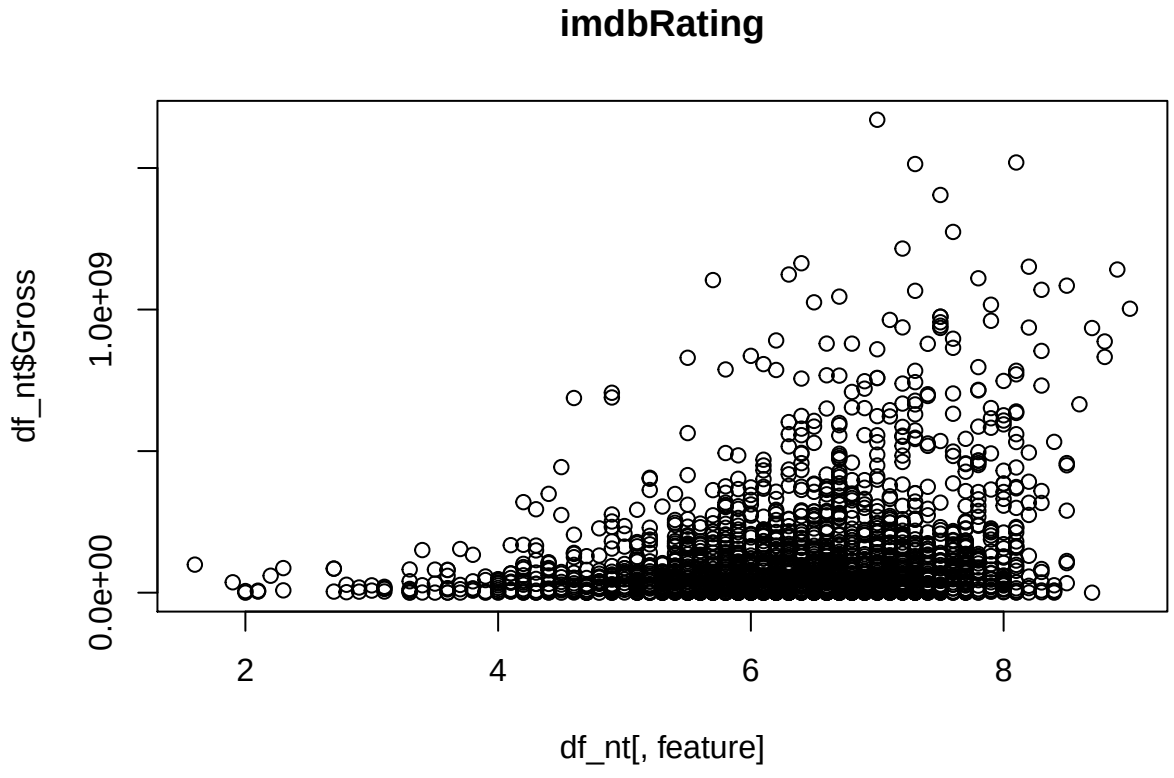
```
## NULL
```


Runtime
df_nt$Gross
0.0e+00
1.0e+09
50
100
150
200
df_nt[, feature]

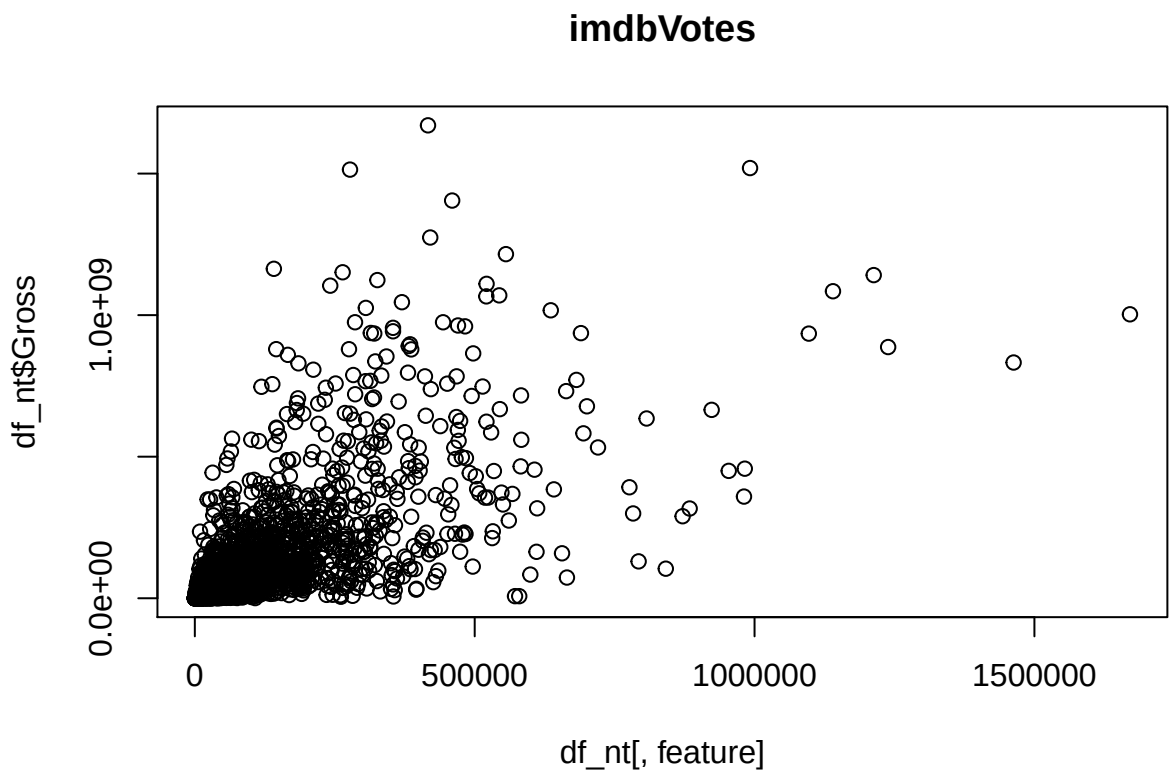
```
## NULL
```


Metascore
df_nt$Gross
0.0e+00
1.0e+09
0
20
40
60
80
100
df_nt[, feature]

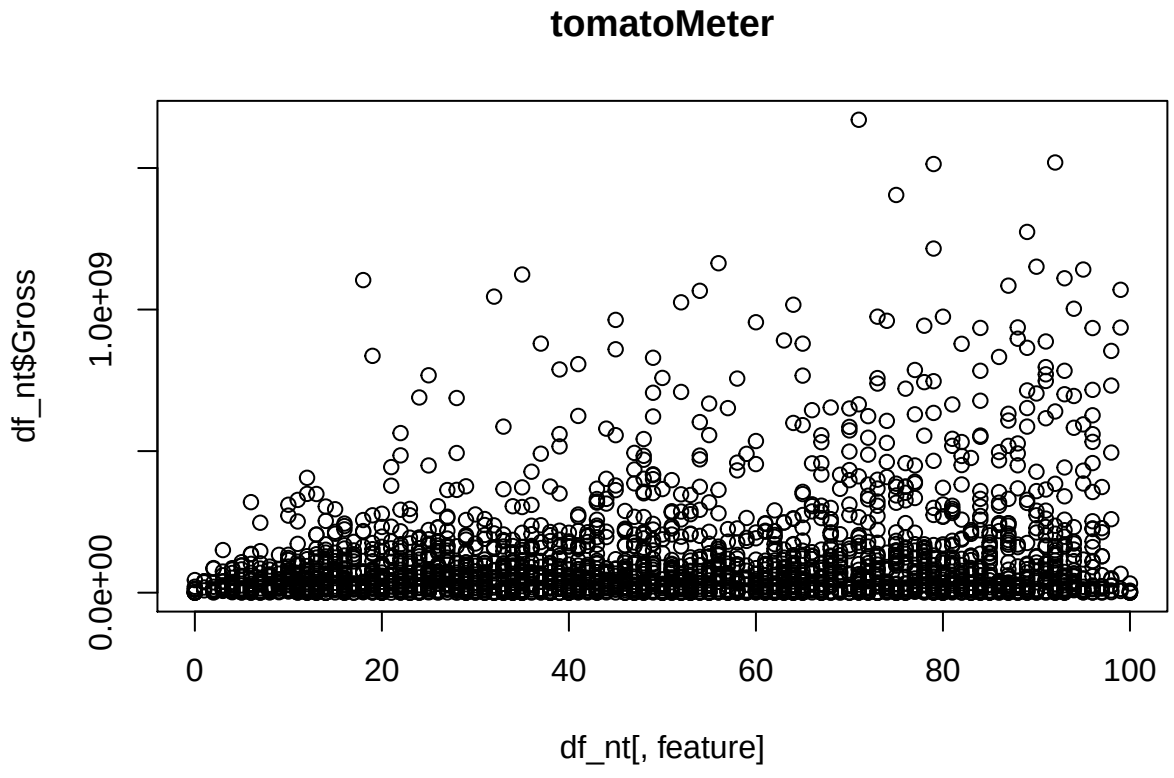
```
## NULL
```


imdbRating
df_nt$Gross
0.0e+00
1.0e+09
2
4
6
8
df_nt[, feature]

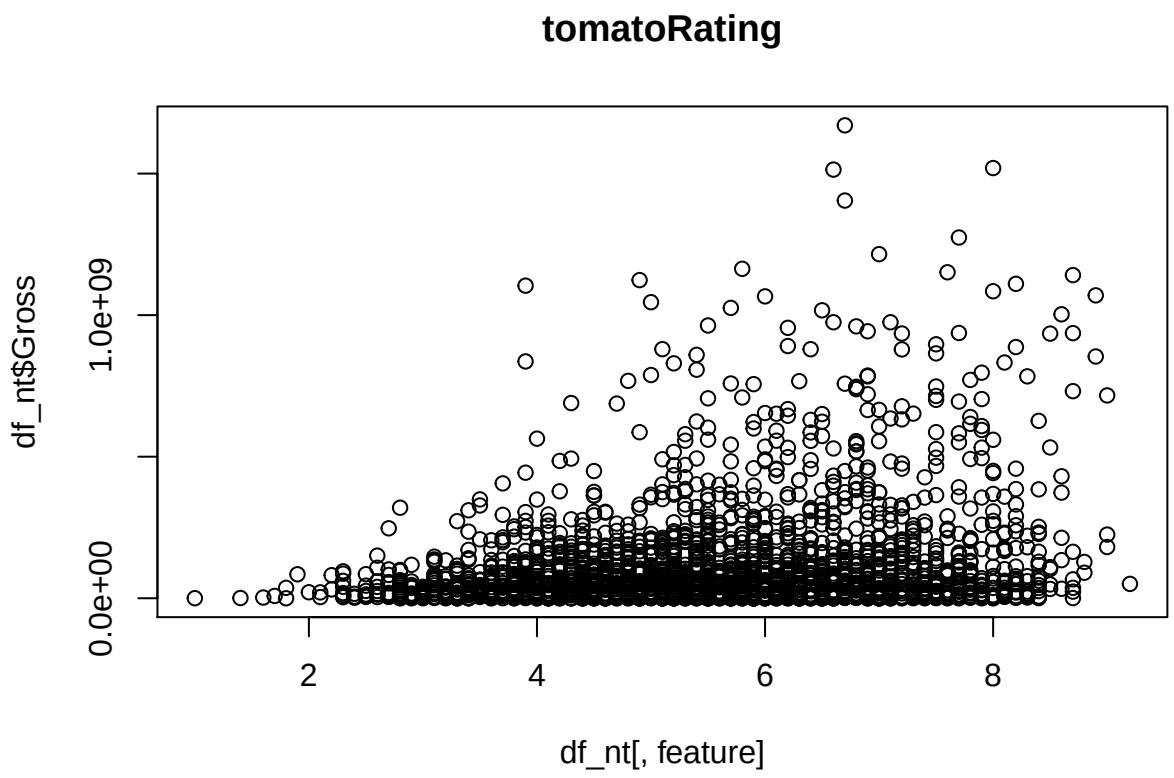
```
## NULL
```


imdbVotes
df_nt$Gross
0.0e+00
1.0e+09
0
500000
1000000
1500000
df_nt[, feature]

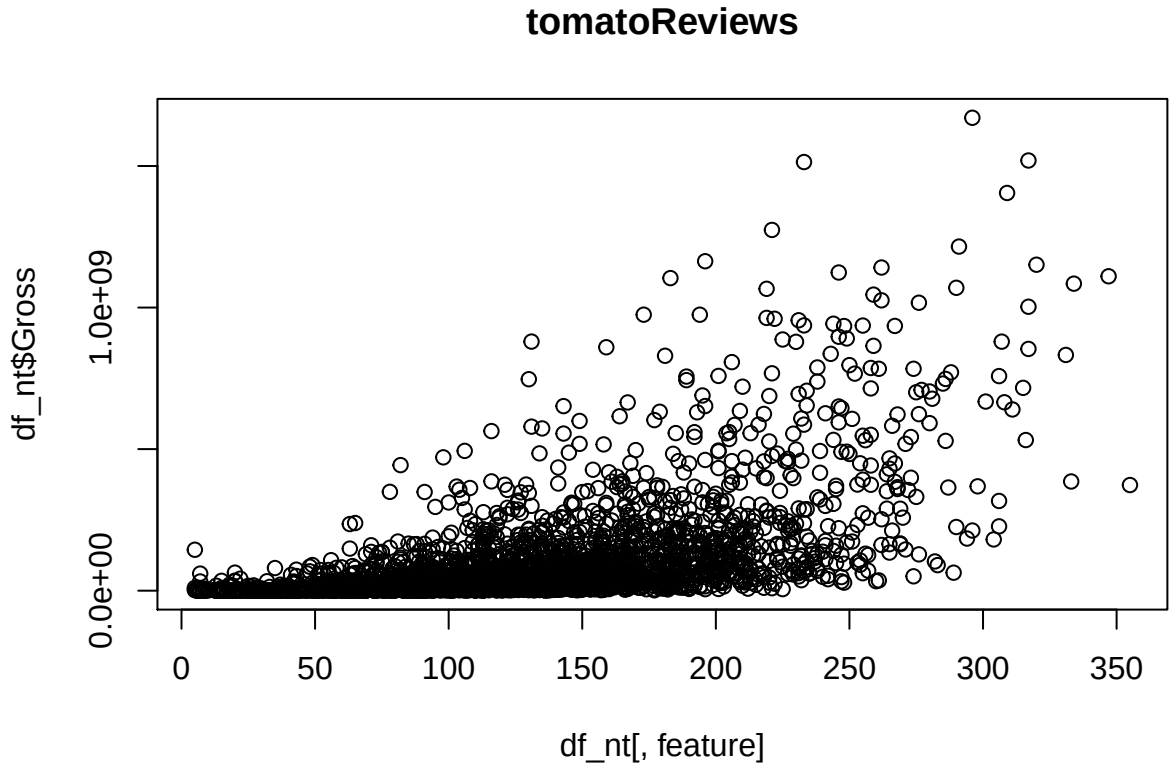
```
## NULL
```


tomatoMeter
df_nt$Gross
0.0e+00
1.0e+09
0
20
40
60
80
100
df_nt[, feature]

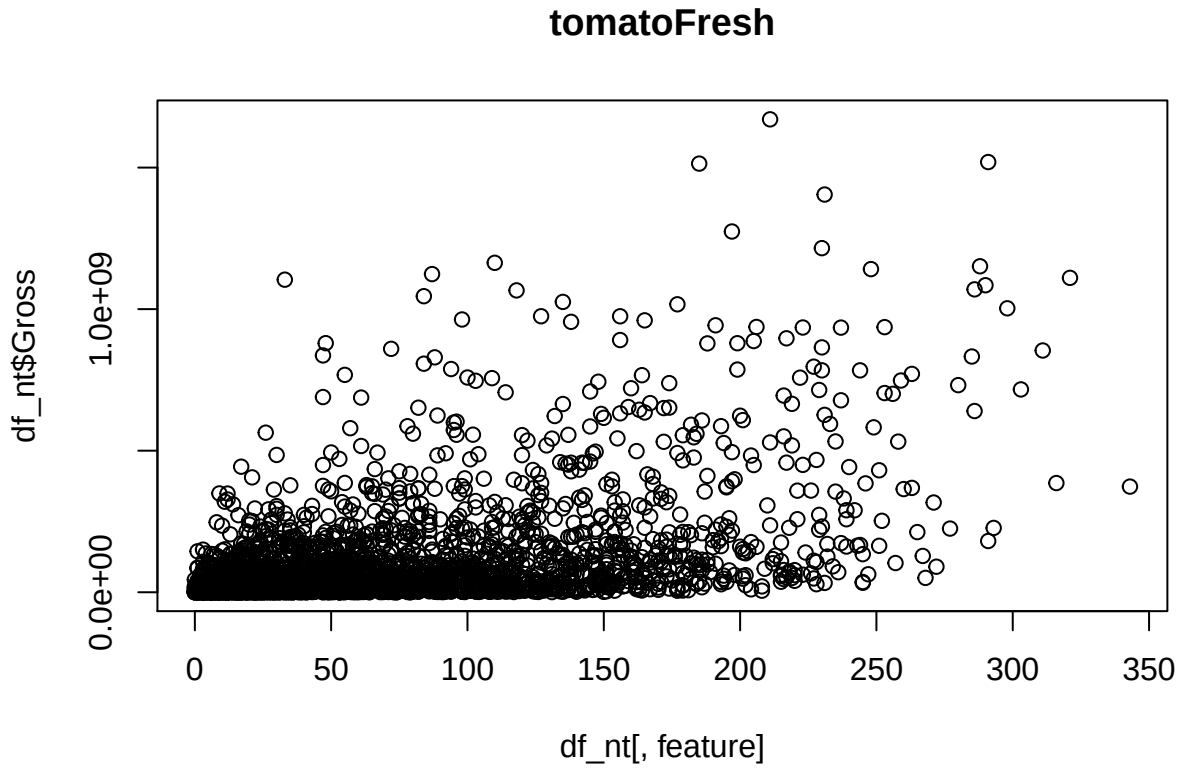
```
## NULL
```


tomatoRating
df_nt$Gross
0.0e+00
1.0e+09
2
4
6
8
df_nt[, feature]

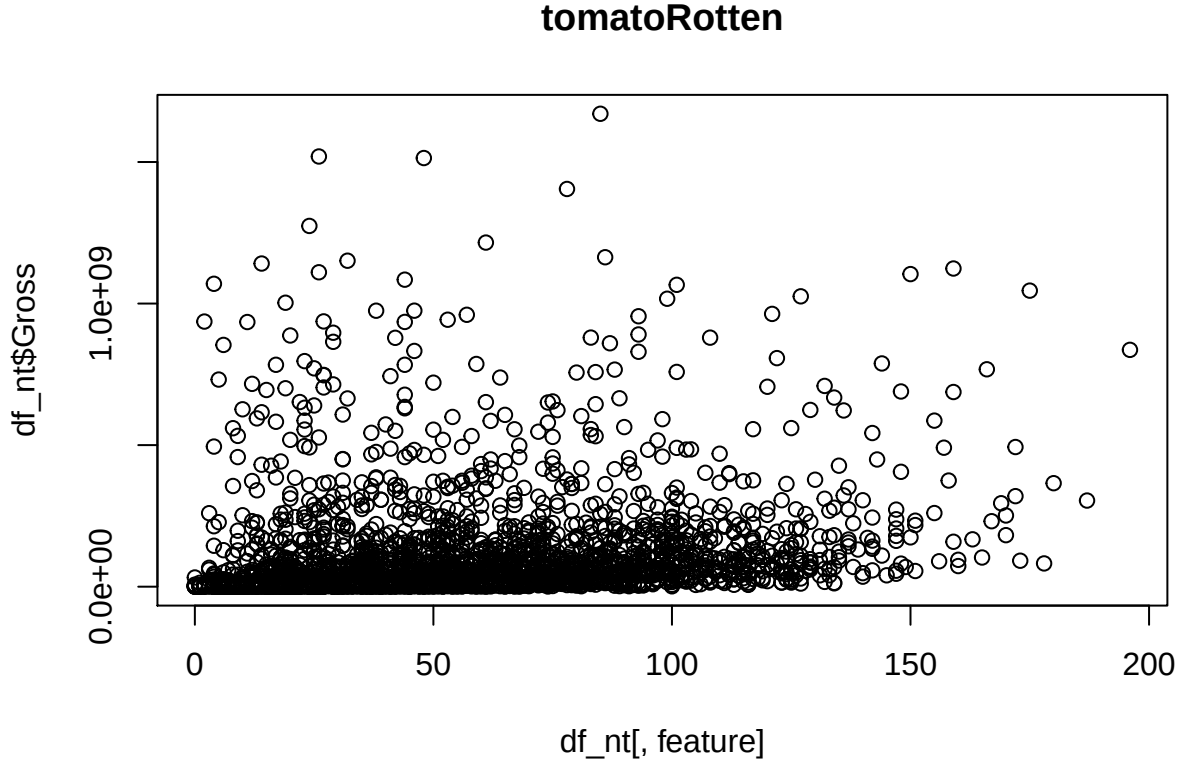
```
## NULL
```


tomatoReviews
df_nt$Gross
0.0e+00
1.0e+09
0
50
100
150
200
250
300
350
df_nt[, feature]

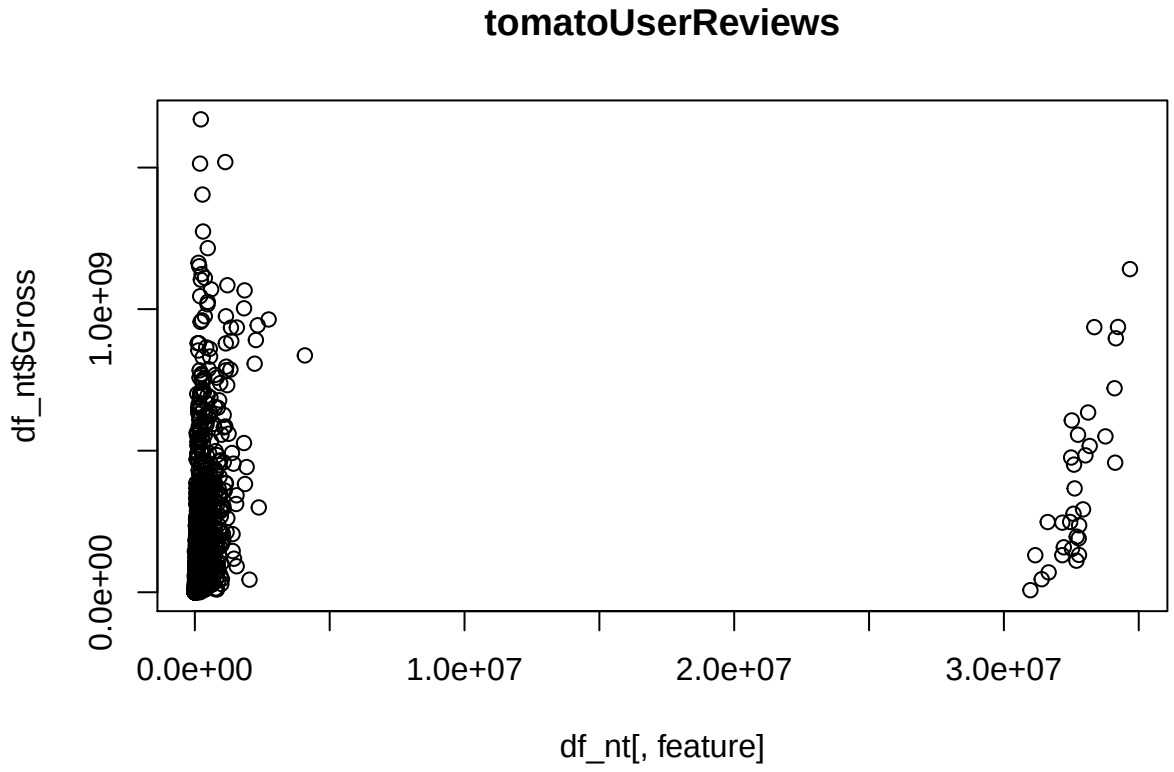
```
## NULL
```


tomatoFresh
df_nt$Gross
0.0e+00
1.0e+09
0
50
100
150
200
250
300
350
df_nt[, feature]

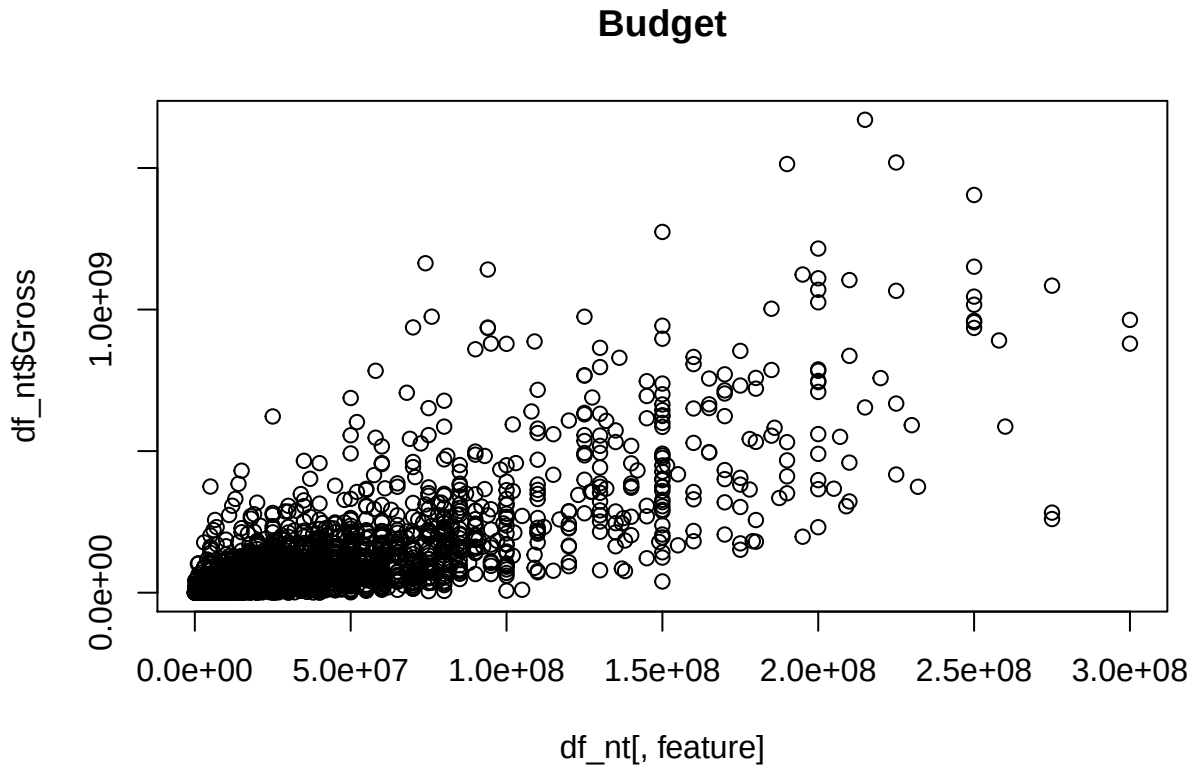
```
## NULL
```


tomatoRotten
df_nt$Gross
0.0e+00
1.0e+09
0
50
100
150
200
df_nt[, feature]

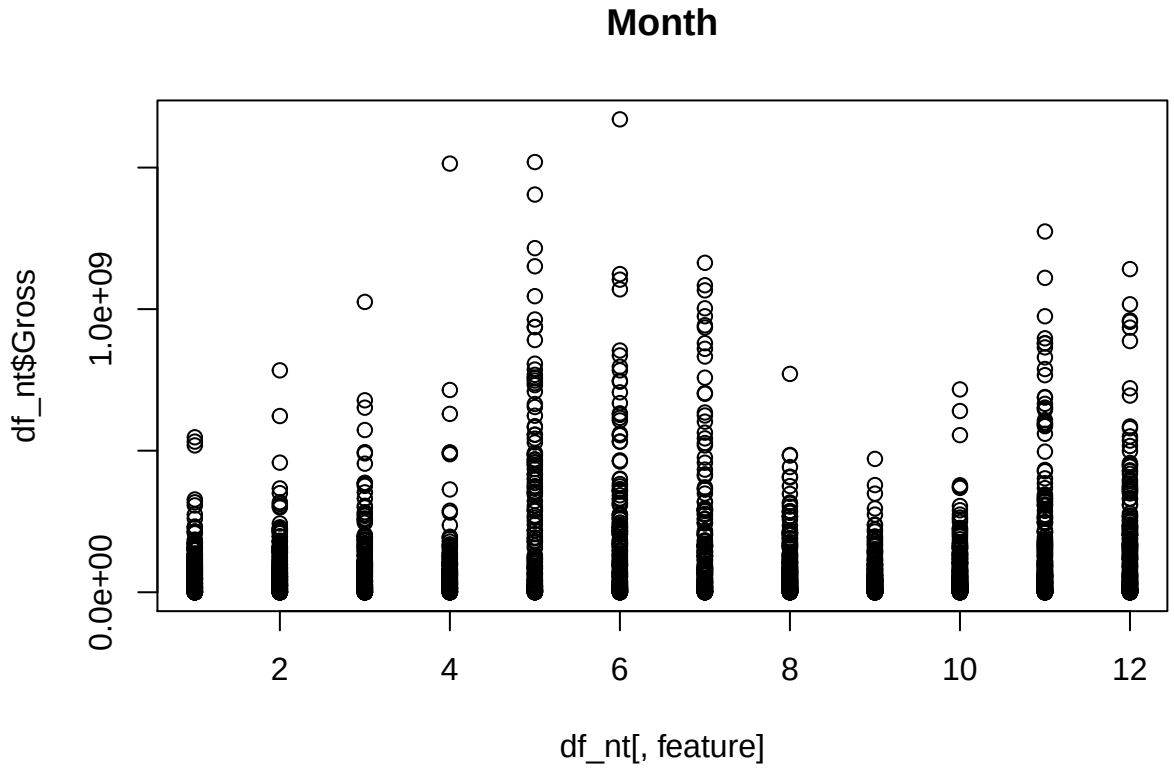
```
## NULL
```


tomatoUserReviews
df_nt$Gross
0.0e+00
1.0e+09
0.0e+00
1.0e+07
2.0e+07
3.0e+07
df_nt[, feature]

```
## NULL
```


Budget
df_nt$Gross
0.0e+00
1.0e+09
0.0e+00
5.0e+07
1.0e+08
1.5e+08
2.0e+08
2.5e+08
3.0e+08
df_nt[, feature]

```
## NULL
```




```
## NULL
```

Evaluate numeric transformations

```
for (feature in setdiff(names(df_numeric), 'Gross')) {
  print(feature)
  fits <- reg_transforms(df_numeric[, feature], df_numeric$Gross)
  perf <- sapply(fits, eval_fit)
  print(perf)
}
```

```
## [1] "Year"
##      linear     log        quad       cubic      quartic    quintic
## r2   0.02364445 0.02363339 0.02365547 0.02366644 0.02367737 0.02368825
## rmse 183319023  183320062  183317989  183316958  183315933  183314911
## cor  0.1537675  0.1537315  0.1538033  0.153839   0.1538745  0.1539099
##      reciprocal
## r2   0
## rmse 185525467
## cor  0.1537675
## [1] "Runtime"
##      linear    log        quad      cubic     quartic   quintic
## r2   0.109679  0.09741387 0.1190923 0.1231651 0.1196097 0.1076796
## rmse 175055941 176257604  174128055 173725052 174076902 175252387
## cor  0.3311781 0.3121119  0.3450975 0.3509488 0.3458464 0.3281457
##      reciprocal
## r2   0
## rmse 185525467
## cor  0.3311781
```

```
## [1] "Metascore"
##      linear     log        quad       cubic      quartic    quintic
## r2   0.03802642 0.03719076 0.03446113 0.03006267 0.02597657 0.02254549
## rmse 181963846  182042864  182300733  182715492  183099955  183422163
## cor  0.1950036  0.1928491  0.1856371  0.1733859  0.1611725  0.1501516
##      reciprocal
## r2   0
## rmse 185525467
## cor  0.1950036
## [1] "imdbRating"
##      linear     log        quad       cubic      quartic    quintic
## r2   0.05951562 0.05094285 0.06730634 0.07337383 0.07831007 0.08241604
## rmse 179919953  180738102  179173199  178589457  178113138  177715964
## cor  0.2439582  0.2257052  0.2594346  0.270876   0.2798394  0.2870819
##      reciprocal
## r2   0
## rmse 185525467
## cor  0.2439582
## [1] "imdbVotes"
##      linear    log       quad      cubic     quartic   quintic
## r2   0.456102  0.3102183 0.2461996 0.1132249 0.0627579 0.04146855
## rmse 136823990 154084645 161076329 174706991 179609552 181638003
## cor  0.6753533 0.5569725 0.496185  0.336489  0.2505153 0.2036383
##      reciprocal
## r2   0
## rmse 185525467
## cor  0.6753533
## [1] "tomatoMeter"
##      linear     log        quad       cubic      quartic    quintic
## r2   0.04011081 0.03668967 0.03754426 0.03430661 0.03122269 0.0284388
## rmse 181766601  182090229  182009442  182315320  182606198  182868379
## cor  0.2002768  0.1915455  0.1937634  0.1852204  0.1766994  0.1686381
##      reciprocal
## r2   0
## rmse 185525467
## cor  0.2002768
## [1] "tomatoRating"
##      linear     log        quad       cubic      quartic    quintic
## r2   0.04929701 0.04692088 0.05000303 0.04928745 0.04781734 0.04600436
## rmse 180894750  181120669  180827570  180895660  181035468  181207734
## cor  0.2220293  0.2166123  0.2236136  0.2220078  0.2186718  0.2144863
##      reciprocal
## r2   0
## rmse 185525467
## cor  0.2220293
## [1] "tomatoReviews"
##      linear    log       quad      cubic     quartic   quintic
## r2   0.3159372 0.1864631 0.3790154 0.395467  0.3814409 0.3496891
## rmse 153444573 167337073 146198845 144249244 145913052 149611185
## cor  0.5620829 0.4318137 0.6156423 0.6288617 0.617609  0.5913451
##      reciprocal
## r2   0
## rmse 185525467
## cor  0.5620829
```

```
## [1] "tomatoFresh"
##      linear    log       quad      cubic     quartic   quintic
## r2   0.2078324 0.1317985 0.227913  0.2183865 0.1950831 0.167198
## rmse 165124711 172867676 163018408 164021036 166448186 169306803
## cor  0.4558864 0.3630407 0.4774024 0.4673185 0.4416821 0.4088985
##      reciprocal
## r2   0
## rmse 185525467
## cor  0.4558864
## [1] "tomatoRotten"
##      linear     log        quad       cubic      quartic    quintic
## r2   0.02543419 0.01663468 0.03258367 0.03675347 0.03790522 0.03688179
## rmse 183150927  183975918  182477886  182084199  181975308  182072071
## cor  0.159481   0.1289755  0.1805095  0.1917119  0.1946926  0.1920463
##      reciprocal
## r2   0
## rmse 185525467
## cor  0.159481
## [1] "tomatoUserReviews"
##      linear     log       quad       cubic      quartic    quintic
## r2   0.04440259 0.237002  0.03227546 0.03310884 0.03437718 0.03565092
## rmse 181359795  162056042 182506952  182428349  182308658  182188378
## cor  0.2107192  0.4868285 0.1796537  0.1819584  0.1854108  0.1888145
##      reciprocal
## r2   0
## rmse 185525467
## cor  0.2107192
## [1] "Budget"
##      linear    log       quad      cubic     quartic   quintic
## r2   0.5872332 0.2860024 0.548111  0.4319684 0.3249191 0.2424858
## rmse 119194416 156766012 124715211 139826591 152433864 161472628
## cor  0.7663114 0.5347919 0.7403452 0.6572431 0.5700167 0.4924285
##      reciprocal
## r2   0
## rmse 185525467
## cor  0.7663114
## [1] "Month"
##      linear      log         quad        cubic       quartic
## r2   0.005137958 0.008499905 0.003036545 0.002664124 0.003121479
## rmse 185048243   184735310   185243575   185278171   185235684
## cor  0.07167955  0.09219493  0.05510485  0.05161515  0.0558702
##      quintic     reciprocal
## r2   0.003994366 0
## rmse 185154568   185525467
## cor  0.063201    0.07167955
```

```
# Numeric transformations
df_nt$Runtime_cubic <- df_nt$Runtime^3
df_nt$imdbRating_power <- df_nt$imdbRating^11
df_nt$tomatoReviews_cubic <- df_nt$tomatoReviews^3
df_nt$tomatoFresh_quad <- df_nt$tomatoFresh^2
df_nt$tomatoRotten_quant <- df_nt$tomatoRotten^4
df_nt$tomatoMonth_log <- log(df_nt$Month)
```

Evaluate binning of numeric features

```
# Binning transformations

for (feature in setdiff(names(df_numeric), 'Gross')) {
  b <- best_bins(df_numeric[, feature], df_numeric$Gross)
  if (b$bins > 0) {
    print(cbind(b, name=feature))
    df_nt[, paste(feature, '_bin', sep='')] <- ntile(df_numeric[, feature], b$bins)
  }
}
```

```
##      b          name
## r2   0.03068978 "Year"
## bins 15         "Year"
##      b         name
## r2   0.1352109 "Runtime"
## bins 8         "Runtime"
##      b          name
## r2   0.03877428 "Metascore"
## bins 17         "Metascore"
##      b          name
## r2   0.05987465 "imdbRating"
## bins 27         "imdbRating"
##      b         name
## r2   0.4715564 "imdbVotes"
## bins 8         "imdbVotes"
##      b         name
## r2   0.0425138 "tomatoMeter"
## bins 86        "tomatoMeter"
##      b          name
## r2   0.05206885 "tomatoRating"
## bins 23         "tomatoRating"
##      b         name
## r2   0.3598231 "tomatoReviews"
## bins 35        "tomatoReviews"
##      b         name
## r2   0.2235011 "tomatoFresh"
## bins 17        "tomatoFresh"
##      b          name
## r2   0.03909618 "tomatoRotten"
## bins 18         "tomatoRotten"
##      b         name
## r2   0.2625313 "tomatoUserReviews"
## bins 27        "tomatoUserReviews"
##      b         name
## r2   0.5962518 "Budget"
## bins 29        "Budget"
##      b          name
## r2   0.04112824 "Month"
## bins 2          "Month"
```

```
evaluate_model_change('Gross~.', df_numeric, 'Gross~.', df_nt)
```

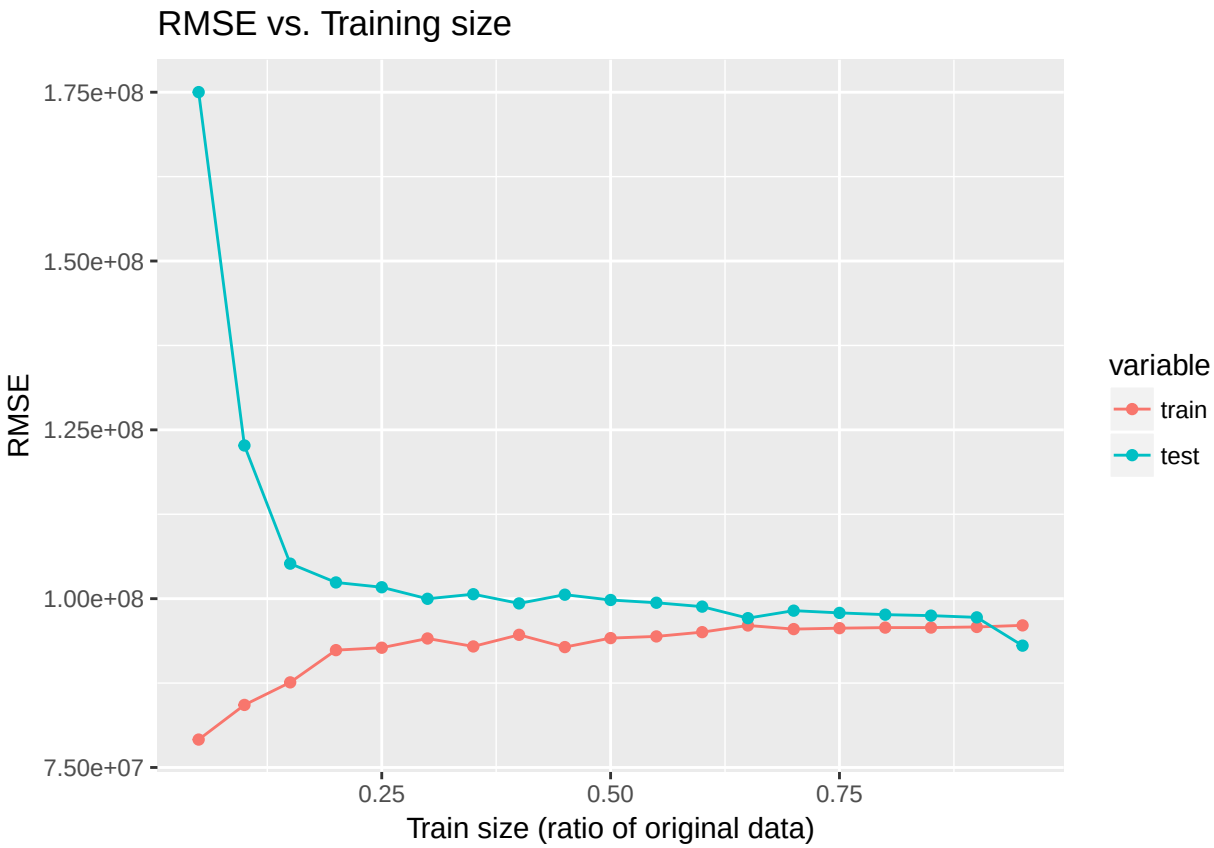
```
## [1] "100 trials"
```

```
## $train
##
##  Welch Two Sample t-test
##
## data:  m1$train and m2$train
## t = 25.326, df = 197.61, p-value < 2.2e-16
## alternative hypothesis: true difference in means is greater than 0
## 95 percent confidence interval:
##  3587245     Inf
## sample estimates:
## mean of x mean of y
##  99749413  95911746
##
##
## $test
##
##  Welch Two Sample t-test
##
## data:  m1$test and m2$test
## t = 1.5663, df = 197.75, p-value = 0.05944
## alternative hypothesis: true difference in means is greater than 0
## 95 percent confidence interval:
##  -235565.5       Inf
## sample estimates:
## mean of x mean of y
##  99460694  95184020
```

```
evaluate_model_range('Gross~.', df_nt)
```

```
## [1] "48 trials"
```

```
## $rmse
##       train      test train_size
## 1  79132056 175015308       0.05
## 2  84262617 122683559       0.10
## 3  87604894 105186530       0.15
## 4  92378724 102401456       0.20
## 5  92735995 101692539       0.25
## 6  94103816  99981186       0.30
## 7  92924850 100660971       0.35
## 8  94645301  99298310       0.40
## 9  92828286 100590199       0.45
## 10 94160277  99802941       0.50
## 11 94410263  99395486       0.55
## 12 95042625  98819261       0.60
## 13 96029117  97115890       0.65
## 14 95493364  98225131       0.70
## 15 95631519  97893450       0.75
## 16 95715917  97632763       0.80
## 17 95717860  97488862       0.85
## 18 95808854  97232065       0.90
## 19 96044002  93042406       0.95
##
## $plot
```



Q: Explain which transformations you used and why you chose them.

A:

Each feature was evaluted for a number of different numeric transforms:

```
linear = lm(y~x),
log = lm(y~log(x+1)),
quad = lm(y~I(x^2)),
cubic = lm(y~I(x^3)),
quartic = lm(y~I(x^4)),
quintic = lm(y~I(x^5)),
reciprocal = lm(y~1/(x+1))
```

A model was built with each transformation and the resulting R^2, RMSE, and correlation coefficient was inspected. For each feature, the model fit metrics were inspected before and after applying each of these transformations. Based on the results, the following numeric transformation features were added:

```
Runtime^3
imdbRating^11
tomatoReviews^3
tomatoFresh^2
tomatoRotten^4
log(Month)
```

Binning was also evaluated for each feature by splitting the data points into equal sized bins. Bin counts between 1 and 200 were tested and the bin count that resulted in the highest r^2 model was chosen. The following bin features where added:

```
bins 15         "Year"
```

```
bins 8          "Runtime"
bins 17          "Metascore"
bins 27          "imdbRating"
bins 8          "imdbVotes"
bins 86         "tomatoMeter"
bins 23          "tomatoRating"
bins 35         "tomatoReviews"
bins 17         "tomatoFresh"
bins 18          "tomatoRotten"
bins 27         "tomatoUserReviews"
bins 29         "Budget"
bins 2           "Month"
```

The final result of these transformations was an improvement of about 5% on the test RMSE on the training and test sets (~9.5e7). The t-test between the RMSE mean across trials for the numeric model verses the tranformed model shows a p-value of 0.01. This indicates the model improvement is statistically significant.

3. Non-numeric variables

Write code that converts genre, actors, directors, and other categorical variables to columns that can be used for regression (e.g. binary columns as you did in Project 1). Also process variables such as awards into more useful columns (again, like you did in Project 1). Now use these converted columns only to build your next model.

```
# TODO: Build & evaluate model 3 (converted non-numeric variables only)

# Create a non-numeric dataframe to work with
df_cat <- df[, sapply(df, function(x) !is.numeric(x))]
df_cat$Gross <- df$Gross

# Filter non-catagorical features
df_cat <- subset(df_cat, select=-c(DVD, Title, tomatoConsensus, Plot))

filter_columns <- function(df, column_names) {
  if (length(column_names) > 0) {
    filtered <- which(names(df) %in% column_names)
    if (length(filtered) > 0) {
      return(df[, -filtered])
    }
  }
  return(df)
}

# Converts the top N unique values to binary features
factor_to_features <- function(df, feature, top=10, append=TRUE) {
  df_factor <- subset(df, select=feature)
  df_features <- model.matrix(~., df_factor)[,-1]
  if (top > 0) {
    sums <- colSums(df_features)
    top_features = names(sort(sums, decreasing = TRUE)[1:min(top, length(sums))])
    df_features <- subset(df_features, select=top_features)
  }
  if (append) {
    df <- cbind(df, df_features)
```

```
  } else {
    df <- df_features
  }
  df <- filter_columns(df, c(feature, '(Intercept)'))
  colnames(df) <- make.names(colnames(df), unique=T)
  return(df)
}

# Converts the top N unique values to binary features. Assumes comma seperated values in original featu
string_to_features <- function(df, feature, top=10, append=TRUE) {
  df_sub <- subset(df, select=feature)
  df_sub <- cSplit_e(df_sub, feature, sep=',', mode='binary', type='character', fill=0, drop=TRUE)
  if (top > 0) {
    sums <- colSums(df_sub)
    top_features = names(sort(sums, decreasing = TRUE)[1:min(top, ncol(df_sub))])
    df_sub <- subset(df_sub, select=top_features)
  }
  if (append) {
    df <- cbind(df, df_sub)
  } else {
    df <- df_sub
  }
  df <- filter_columns(df, c(feature, 'na'))
  colnames(df) <- make.names(colnames(df), unique=T)
  return(df)
}

# Removes comments inside ()
clean_writer_strings <- function(string_) {
  return(str_replace_all(string_, ' *\\([^\\)]*\\)', ''))
}

# Clean 'Writer'
df_cat$Writer <- sapply(df_cat$Writer, clean_writer_strings)
```

```
# Create binary features for: 'Rated', 'Production'
for (feature in c('Rated', 'Production')) {
  df_cat <- factor_to_features(df_cat, feature)
}

# Create binary features for: 'Genre', 'Director', 'Writer', 'Actors', 'Language', 'Country'
for (feature in c('Genre', 'Director', 'Writer', 'Actors', 'Language', 'Country')) {
  df_cat = string_to_features(df_cat, feature)
}

# Clean columns
df_cat[c('Writer', 'na', 'Writer_N/A')] = NULL

# Parse Awards
parse_awards <- function(msg, patterns) {
  if (is.na(msg)) {
    return(NA)
  }
  msg = tolower(msg)
```

```
  value = 0
  for (pattern in patterns) {
    m = str_match(msg, pattern)[1, 2]
    if (!is.na(m)) {
      value = value + as.integer(m)
    }
  }
  return(value)
}

# Convert Awards feature to Wins and Nominations features
awards_patterns = c('won (\\d+)', "(\\d+) win")
nomination_patterns = c('nominated for (\\d+)', "(\\d+) nomination")
df_cat$Wins = sapply(df_cat$Awards, parse_awards, pattern=awards_patterns)
df_cat$Nominations = sapply(df_cat$Awards, parse_awards, pattern=nomination_patterns)
df_cat['Awards'] = NULL
```

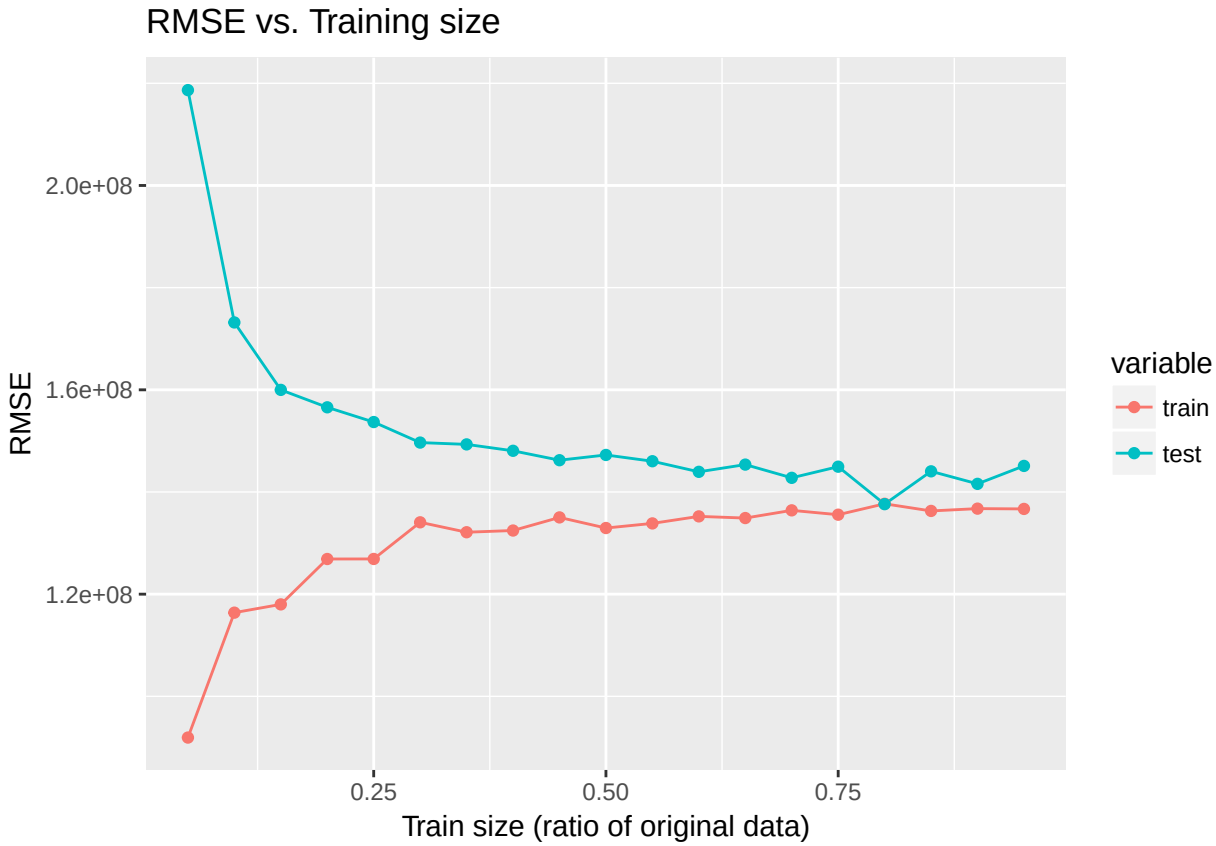
```
# Evaluate the non-numeric only model
evaluate_model('Gross~.', df_cat)
```

```
## $train
## [1] 136795910
##
## $test
## [1] 139267710
```

```
evaluate_model_range('Gross~.', df_cat)
```

```
## [1] "25 trials"
```

```
## $rmse
##        train      test train_size
## 1   91925435 218665359       0.05
## 2  116378004 173185633       0.10
## 3  117998349 159987397       0.15
## 4  126880720 156573788       0.20
## 5  126891770 153688360       0.25
## 6  134064284 149668078       0.30
## 7  132122320 149317401       0.35
## 8  132457351 148062226       0.40
## 9  135021519 146220663       0.45
## 10 132944563 147248094       0.50
## 11 133849012 146021322       0.55
## 12 135224398 143945366       0.60
## 13 134896695 145361543       0.65
## 14 136404849 142771760       0.70
## 15 135553399 144952249       0.75
## 16 137689824 137617464       0.80
## 17 136291649 144045547       0.85
## 18 136743136 141591602       0.90
## 19 136683600 145104388       0.95
##
## $plot
```



Q: Explain which categorical variables you used, and how you encoded them into features.

A:

I used the following original features:

'Rated', 'Production' 'Genre', 'Director', 'Writer', 'Actors', 'Language', 'Country' 'Awards'

Rated and Production were converted by taking the set of unique values and making each a seperate binary feature. These features were then reduced to the top ten most common.

The remaining features, besides Awards, were converted by parsing the string as comma seperated values and creating binary features from each of the unique extracted values. As before, only the top ten most common derived features were retained.

The Awards feature extraction uses regex parsing to pull out the number of wins and nominations. Wins and Nominations features replace Awards.

The resulting non-numeric only model resulted in an RMSE of ~1.4e8 for train and test samples.

4. Numeric and categorical variables

Try to improve the prediction quality as much as possible by using both numeric and non-numeric variables from **Tasks 2 & 3**.

```
# TODO: Build & evaluate model 4 (numeric & converted non-numeric variables)

# Combine numeric and non-numeric features
df_all <- merge(df_nt, df_cat)
```

```
evaluate_model_change('Gross~.', df_nt, 'Gross~.', df_all)
```

```
## [1] "100 trials"
```

```
## $train
##
##  Welch Two Sample t-test
##
## data:  m1$train and m2$train
## t = 31.72, df = 197.17, p-value < 2.2e-16
## alternative hypothesis: true difference in means is greater than 0
## 95 percent confidence interval:
##  4476538     Inf
## sample estimates:
## mean of x mean of y
##  95846880  91124294
##
##
## $test
##
##  Welch Two Sample t-test
##
## data:  m1$test and m2$test
## t = 1.7788, df = 197.35, p-value = 0.0384
## alternative hypothesis: true difference in means is greater than 0
## 95 percent confidence interval:
##  342340.4      Inf
## sample estimates:
## mean of x mean of y
##  96295388  91471064
```

```
evaluate_model_change('Gross~.', df_cat, 'Gross~.', df_all)
```

```
## [1] "100 trials"
```

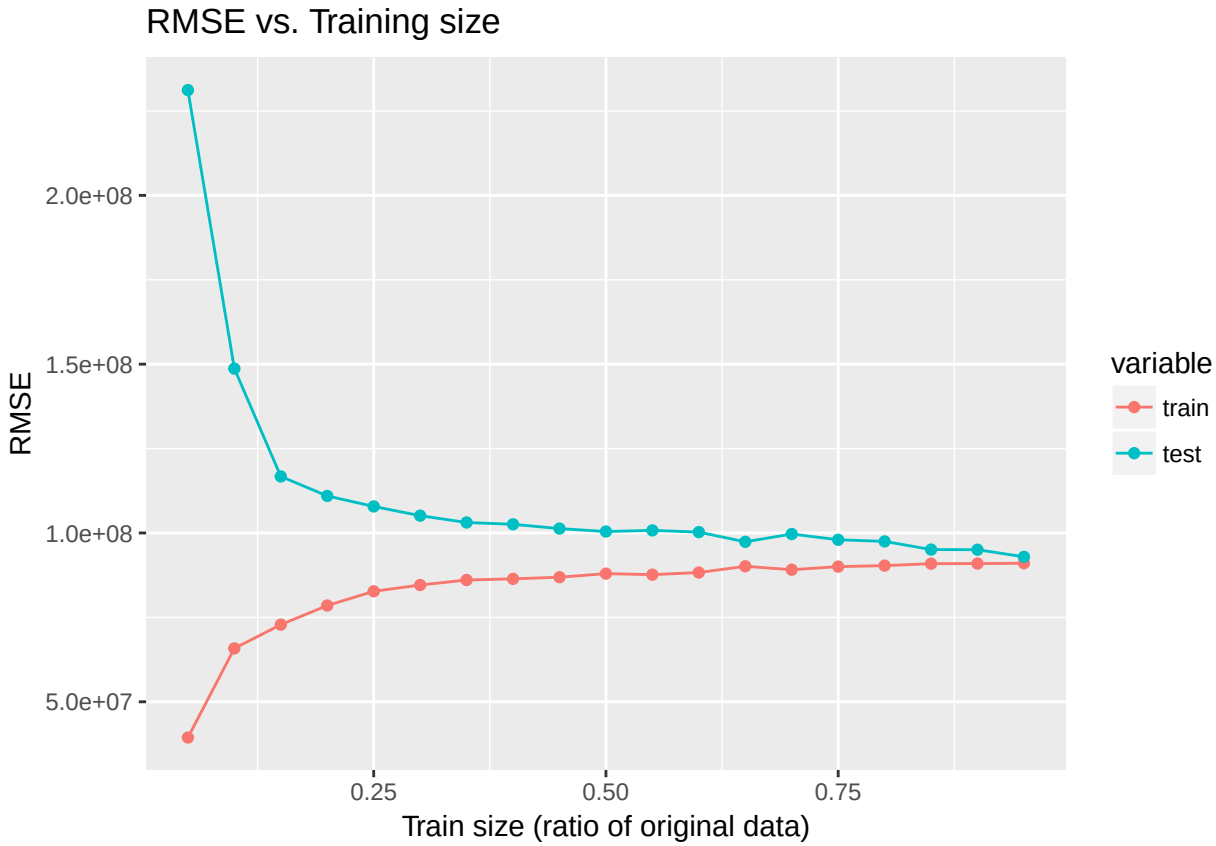
```
## $train
##
##  Welch Two Sample t-test
##
## data:  m1$train and m2$train
## t = 288.44, df = 197.95, p-value < 2.2e-16
## alternative hypothesis: true difference in means is greater than 0
## 95 percent confidence interval:
##  45556563      Inf
## sample estimates:
## mean of x mean of y
## 136799076  90979998
##
##
## $test
##
##  Welch Two Sample t-test
##
## data:  m1$test and m2$test
## t = 17.716, df = 197.71, p-value < 2.2e-16
```

```
## alternative hypothesis: true difference in means is greater than 0
## 95 percent confidence interval:
##  43816884      Inf
## sample estimates:
## mean of x mean of y
## 142594597  94269784
```

```
evaluate_model_range('Gross~.', df_all, max_time=120)
```

```
## [1] "50 trials"
```

```
## $rmse
##       train      test train_size
## 1  39399292 231172948       0.05
## 2  65816468 148685608       0.10
## 3  72852579 116738800       0.15
## 4  78513809 110970657       0.20
## 5  82720202 107861256       0.25
## 6  84585020 105123522       0.30
## 7  86076729 103108689       0.35
## 8  86408097 102573338       0.40
## 9  86901278 101301067       0.45
## 10 87959357 100428898       0.50
## 11 87656073 100766825       0.55
## 12 88282272 100252468       0.60
## 13 90136628  97382287       0.65
## 14 89114165  99693543       0.70
## 15 90039290  97985374       0.75
## 16 90331522  97504016       0.80
## 17 90909103  95089695       0.85
## 18 90936494  95043275       0.90
## 19 91037430  92939426       0.95
##
## $plot
```



The combined model, with numeric and non-numeric featues, performed significantly beter than the catagory only model and was about the same as the numeric only model. The best RMSE on the test set was 9.5e7. RMSE on the training set (95%) was better on the mixed model. Since the mixed model did not improve over the numeric model, the next step is to do some feature selection. We already know that the mixed model is performing similar to a model with less features, so there should be some features that are not contributing.

```
# Filter features with high correlation with other features
filter_high_cor <- function(df, cutoff=0.75) {
  if ('Gross' %in% colnames(df)) {
    x <- subset(df, select=-Gross)
  }
  cor_matrix <- cor(x, use='pairwise.complete.obs')
  high_cor <- findCorrelation(cor_matrix, cutoff=cutoff, names=TRUE)
  return(filter_columns(df, high_cor))
}

# Filter features with near zero variance
filter_near_zero_var <- function(df) {
  near_zero_var_columns <- names(df)[nearZeroVar(df)]
  return(filter_columns(df, near_zero_var_columns))
}

build_formula <- function(features) {
  return(paste('Gross ~', paste(features, collapse='+')))
}

# Perform univariate feature filtering
```

```
run_sbf <- function(formula, df) {
  ctrl <- sbfControl(functions = lmSBF,
                     method = "repeatedcv",
                     repeats = 5,
                     verbose = TRUE, saveDetails=T)
  return(sbf(form=formula,
             data=df,
             sbfControl = ctrl))
}

# Filter features
lmp_all <- run_sbf(Gross~., df_all)
df_filtered <- lmp_all$fit$model
colnames(df_filtered)[1] <- 'Gross'
```

```
paste('Keeping', ncol(df_filtered), 'out of', ncol(df_all), 'features')
```

```
## [1] "Keeping 78 out of 112 features"
```

```
evaluate_model_change('Gross~.', df_all, 'Gross~.', df_filtered)
```

```
## [1] "100 trials"
```

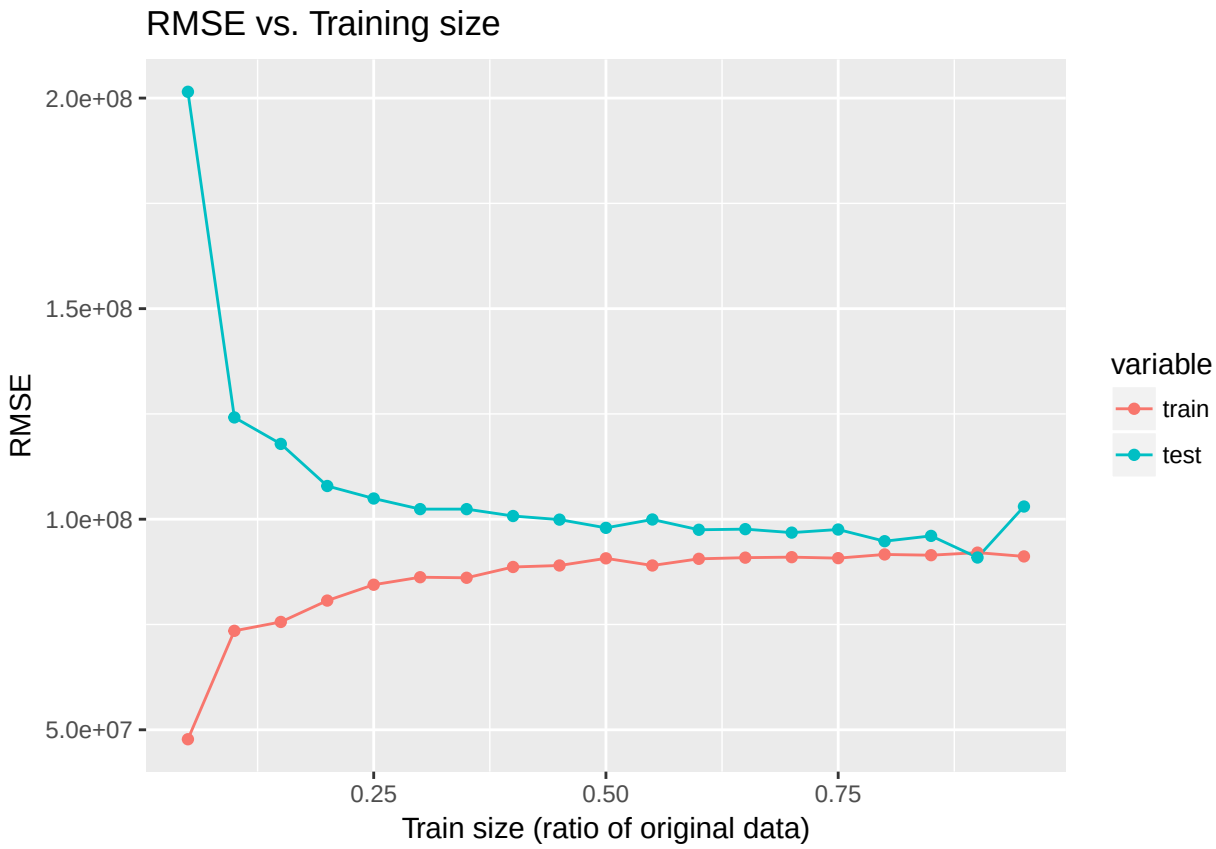
```
## $train
##
##  Welch Two Sample t-test
##
## data:  m1$train and m2$train
## t = -5.8739, df = 194.16, p-value = 1
## alternative hypothesis: true difference in means is greater than 0
## 95 percent confidence interval:
##  -992467     Inf
## sample estimates:
## mean of x mean of y
##  91056206  91830743
##
##
## $test
##
##  Welch Two Sample t-test
##
## data:  m1$test and m2$test
## t = 0.57273, df = 194.71, p-value = 0.2837
## alternative hypothesis: true difference in means is greater than 0
## 95 percent confidence interval:
##  -2617137      Inf
## sample estimates:
## mean of x mean of y
##  92941923  91554029
```

```
evaluate_model_range('Gross~.', df_filtered)
```

```
## [1] "27 trials"
```

```
## $rmse
##       train      test train_size
```

```
## 1  47749171 201497383       0.05
## 2  73496972 124172748       0.10
## 3  75605761 117881884       0.15
## 4  80677406 107891646       0.20
## 5  84442088 104921122       0.25
## 6  86225188 102397155       0.30
## 7  86092566 102393824       0.35
## 8  88656889 100760377       0.40
## 9  89012628  99904962       0.45
## 10 90708706  97952903       0.50
## 11 89014103  99936149       0.55
## 12 90595242  97492254       0.60
## 13 90872615  97642436       0.65
## 14 90991733  96817717       0.70
## 15 90754680  97556555       0.75
## 16 91634372  94779821       0.80
## 17 91454516  96038809       0.85
## 18 92066620  90908393       0.90
## 19 91168880 103016464       0.95
##
## $plot
```



Using univariate feature filtering on the full set of features (112 features) produced a simplified model with 78 features. RMSE on the simplified model improved slightly to 9.1e7 on the test set.

5. Additional features

Now try creating additional features such as interactions (e.g. `is_genre_comedy` x `is_budget_greater_than_3M`) or deeper analysis of complex variables (e.g. text analysis of full-text columns like `Plot`).

```
# TODO: Build & evaluate model 5 (numeric, non-numeric and additional features)

search_colnames <- function(term, df) {
  return(colnames(df)[which(str_detect(colnames(df), term))])
}

build_formula_by_term <- function(term, df) {
  return(paste(search_colnames(term, df_filtered), collapse='+'))
}
```

Interactions

Genre interactions

```
formula <- paste('Gross~(', build_formula_by_term('Genre', df_filtered), ')^2+.')
evaluate_model_change(Gross~., df_filtered, formula, df_filtered)
```

```
## [1] "100 trials"
```

```
## $train
##
##  Welch Two Sample t-test
##
## data:  m1$train and m2$train
## t = 8.0356, df = 197.78, p-value = 4.134e-14
## alternative hypothesis: true difference in means is greater than 0
## 95 percent confidence interval:
##  854216.8      Inf
## sample estimates:
## mean of x mean of y
##  91527102  90451724
##
##
## $test
##
##  Welch Two Sample t-test
##
## data:  m1$test and m2$test
## t = 1.9511, df = 197.88, p-value = 0.02623
## alternative hypothesis: true difference in means is greater than 0
## 95 percent confidence interval:
##  724773.2      Inf
## sample estimates:
## mean of x mean of y
##  97118639  92381669
```

Actor interactions

```
formula <- paste('Gross~(', build_formula_by_term('Actor', df_filtered), ')^2+.')
evaluate_model_change(Gross~., df_filtered, formula, df_filtered)
```

```
## [1] "100 trials"
```

```
## $train
##
##  Welch Two Sample t-test
##
## data:  m1$train and m2$train
## t = 0.97671, df = 195.14, p-value = 0.165
## alternative hypothesis: true difference in means is greater than 0
## 95 percent confidence interval:
##  -93225.24       Inf
## sample estimates:
## mean of x mean of y
##  91678730  91544034
##
##
## $test
##
##  Welch Two Sample t-test
##
## data:  m1$test and m2$test
## t = -0.67496, df = 196.55, p-value = 0.7498
## alternative hypothesis: true difference in means is greater than 0
## 95 percent confidence interval:
##  -5664732      Inf
## sample estimates:
## mean of x mean of y
##  94283850  95926512
```

Writer interactions

```
formula <- paste('Gross~(', build_formula_by_term('Writer', df_filtered), ')^2+.')
evaluate_model_change(Gross~., df_filtered, formula, df_filtered)
```

```
## [1] "100 trials"
```

```
## $train
##
##  Welch Two Sample t-test
##
## data:  m1$train and m2$train
## t = -0.49634, df = 197.99, p-value = 0.6899
## alternative hypothesis: true difference in means is greater than 0
## 95 percent confidence interval:
##  -323332.1       Inf
## sample estimates:
## mean of x mean of y
##  91535765  91610445
##
##
## $test
##
##  Welch Two Sample t-test
##
## data:  m1$test and m2$test
## t = 0.55074, df = 198, p-value = 0.2912
```

```
## alternative hypothesis: true difference in means is greater than 0
## 95 percent confidence interval:
##  -2896574      Inf
## sample estimates:
## mean of x mean of y
##  96626058  95178260
```

Budget, Year, Runtime interaction

```
evaluate_model_change(Gross~., df_filtered, Gross~(Budget_bin+Year_bin+Runtime_bin)^2+., df_filtered)
```

```
## [1] "100 trials"
```

```
## $train
##
##  Welch Two Sample t-test
##
## data:  m1$train and m2$train
## t = 6.882, df = 197.07, p-value = 3.838e-11
## alternative hypothesis: true difference in means is greater than 0
## 95 percent confidence interval:
##  679286.7      Inf
## sample estimates:
## mean of x mean of y
##  91716520  90822561
##
##
## $test
##
##  Welch Two Sample t-test
##
## data:  m1$test and m2$test
## t = -0.3308, df = 196.84, p-value = 0.6294
## alternative hypothesis: true difference in means is greater than 0
## 95 percent confidence interval:
##  -4752971      Inf
## sample estimates:
## mean of x mean of y
##  93553577  94346291
```

Production staff counts and description lengths

```
# Try engineering new features

count_entries <- function(column) {
  return(str_count(column, ',') + 1)
}

# Try entity counts
count_features <- c('Genre', 'Director', 'Writer', 'Actors', 'Language', 'Country')
df_count <- as.data.frame(sapply(df[, count_features], count_entries))
colnames(df_count) <- sapply(colnames(df_count), function(n) return(paste('Count', n)))
evaluate_model_change(Gross~., df_filtered, Gross~., cbind(df_filtered, df_count))
```

```
## [1] "100 trials"
```

```
## $train
##
##  Welch Two Sample t-test
##
## data:  m1$train and m2$train
## t = 1.8908, df = 194.39, p-value = 0.03007
## alternative hypothesis: true difference in means is greater than 0
## 95 percent confidence interval:
##  30799.61      Inf
## sample estimates:
## mean of x mean of y
##  91668167  91423510
##
##
## $test
##
##  Welch Two Sample t-test
##
## data:  m1$test and m2$test
## t = -0.73827, df = 196.11, p-value = 0.7694
## alternative hypothesis: true difference in means is greater than 0
## 95 percent confidence interval:
##  -5696176      Inf
## sample estimates:
## mean of x mean of y
##  94568666  96327525
```

```
# Try string lengths
length_features <- c('Title', 'Plot', 'tomatoConsensus')
df_length <- as.data.frame(sapply(df[, length_features], str_length))
colnames(df_length) <- sapply(colnames(df_length), function(n) return(paste('Length', n)))
evaluate_model_change(Gross~., df_filtered, Gross~., cbind(df_filtered, df_length))
```

```
## [1] "100 trials"
```

```
## $train
##
##  Welch Two Sample t-test
##
## data:  m1$train and m2$train
## t = 2.0147, df = 197.4, p-value = 0.02265
## alternative hypothesis: true difference in means is greater than 0
## 95 percent confidence interval:
##  56104.72      Inf
## sample estimates:
## mean of x mean of y
##  91628883  91316688
##
##
## $test
##
##  Welch Two Sample t-test
##
## data:  m1$test and m2$test
## t = -1.5618, df = 196.22, p-value = 0.94
```

```
## alternative hypothesis: true difference in means is greater than 0
## 95 percent confidence interval:
##  -8746731      Inf
## sample estimates:
## mean of x mean of y
##  94366703  98616517
```

Ordered Rating scale

```
# Try Rated scale
rate_scale <- function(rating) {
  ratings <- c('G', 'PG', 'PG-13', 'R', 'NC-17')
  if (rating %in% ratings) {
    return(which(ratings == rating))
  } else {
    return(6)
  }
}

df_rating <- cbind(df_filtered)
df_rating$rate_scale <- sapply(df$Rated, rate_scale)
evaluate_model_change(Gross~., df_filtered, Gross~., df_rating)
```

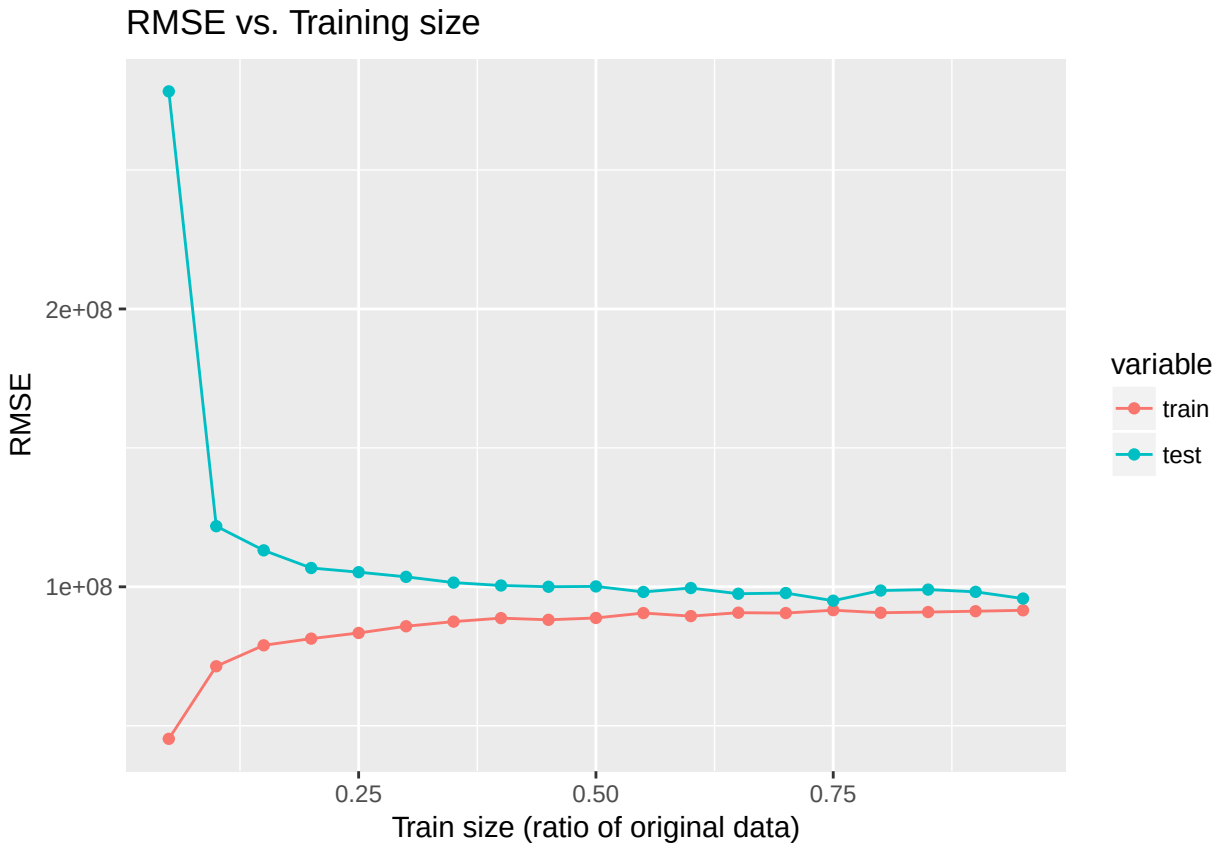
```
## [1] "100 trials"
```

```
## $train
##
##  Welch Two Sample t-test
##
## data:  m1$train and m2$train
## t = -1.5448, df = 192.75, p-value = 0.938
## alternative hypothesis: true difference in means is greater than 0
## 95 percent confidence interval:
##  -454844.1       Inf
## sample estimates:
## mean of x mean of y
##  91504129  91723873
##
##
## $test
##
##  Welch Two Sample t-test
##
## data:  m1$test and m2$test
## t = 1.5029, df = 195.15, p-value = 0.06725
## alternative hypothesis: true difference in means is greater than 0
## 95 percent confidence interval:
##  -381790     Inf
## sample estimates:
## mean of x mean of y
##  97236366  93407355
```

```
evaluate_model_range(Gross~., df_rating)
```

```
## [1] "26 trials"
```

```
## $rmse
##       train      test train_size
## 1  45292183 278294101       0.05
## 2  71410423 121819671       0.10
## 3  78952222 113122269       0.15
## 4  81356237 106773037       0.20
## 5  83390638 105268117       0.25
## 6  85803522 103586744       0.30
## 7  87480460 101515680       0.35
## 8  88729722 100472710       0.40
## 9  88126055 100025652       0.45
## 10 88792576 100141589       0.50
## 11 90553449  98157211       0.55
## 12 89459121  99559355       0.60
## 13 90684405  97512638       0.65
## 14 90551540  97757396       0.70
## 15 91587564  95001276       0.75
## 16 90668537  98663888       0.80
## 17 90916312  99001033       0.85
## 18 91222596  98194725       0.90
## 19 91544437  95772605       0.95
##
## $plot
```



Count of top Actors, Writers, Languages, Countries, Directors, and Production studios.

```
# Try top count
df_top <- cbind(df_filtered)
df_top$top_actor_count <- rowSums(string_to_features(df, 'Actors', top=10, append=F))
df_top$top_writer_count <- rowSums(string_to_features(df, 'Writer', top=100, append=F))
df_top$top_language_count <- rowSums(string_to_features(df, 'Language', top=10, append=F))
df_top$top_country_count <- rowSums(string_to_features(df, 'Country', top=5, append=F))
df_top$top_director <- rowSums(factor_to_features(df, 'Director', top=100, append=F))
df_top$top_production <- rowSums(factor_to_features(df, 'Production', top=10, append=F))

evaluate_model_change(Gross~., df_filtered, Gross~., df_top)
```

```
## [1] "100 trials"
```

```
## $train
##
##  Welch Two Sample t-test
##
## data:  m1$train and m2$train
## t = 1.8639, df = 194.01, p-value = 0.03192
## alternative hypothesis: true difference in means is greater than 0
## 95 percent confidence interval:
##  33048.9     Inf
## sample estimates:
## mean of x mean of y
##  91544399  91252645
##
##
## $test
##
##  Welch Two Sample t-test
##
## data:  m1$test and m2$test
## t = -0.89938, df = 196.78, p-value = 0.8152
## alternative hypothesis: true difference in means is greater than 0
## 95 percent confidence interval:
##  -6952229      Inf
## sample estimates:
## mean of x mean of y
##  96336921  98787029
```

```
# evaluate_model(Gross~.^2, df_top)
```

Genre featues reintroduced

```
df_genre <- string_to_features(df, 'Genre', top=100, append=F)
df_genre_eval <- merge(df_filtered, df_genre)
evaluate_model_change(Gross~., df_filtered, Gross~., df_genre_eval, trials=10, max_time=120)
```

```
## [1] "10 trials"
```

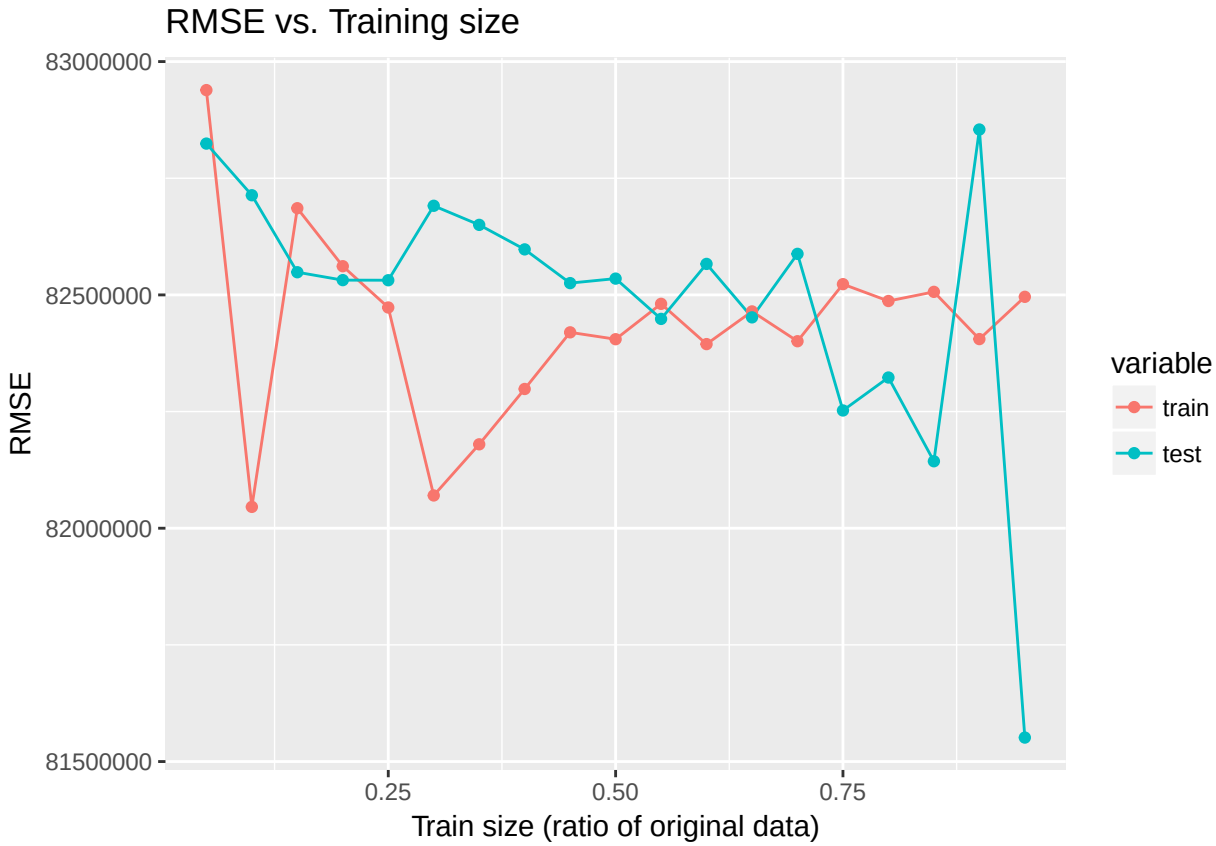
```
## $train
##
```

```
##  Welch Two Sample t-test
##
## data:  m1$train and m2$train
## t = 26.812, df = 9.106, p-value = 2.806e-10
## alternative hypothesis: true difference in means is greater than 0
## 95 percent confidence interval:
##  8670048     Inf
## sample estimates:
## mean of x mean of y
##  91761668  82456272
##
##
## $test
##
##  Welch Two Sample t-test
##
## data:  m1$test and m2$test
## t = 1.4498, df = 9.0992, p-value = 0.09034
## alternative hypothesis: true difference in means is greater than 0
## 95 percent confidence interval:
##  -2605010      Inf
## sample estimates:
## mean of x mean of y
##  92220965  82308926
```

```
evaluate_model_range(Gross~., df_genre_eval, trials=10, max_time=3000)
```

```
## [1] "10 trials"
```

```
## $rmse
##       train     test train_size
## 1  82938873 82824259       0.05
## 2  82045735 82713592       0.10
## 3  82685789 82548587       0.15
## 4  82561512 82531660       0.20
## 5  82473039 82531456       0.25
## 6  82070209 82690974       0.30
## 7  82179836 82650093       0.35
## 8  82298407 82597531       0.40
## 9  82419749 82525276       0.45
## 10 82405059 82534934       0.50
## 11 82480860 82448563       0.55
## 12 82394558 82566401       0.60
## 13 82464608 82451873       0.65
## 14 82400761 82587977       0.70
## 15 82523029 82252331       0.75
## 16 82486905 82322994       0.80
## 17 82506499 82143549       0.85
## 18 82405304 82854430       0.90
## 19 82495918 81551349       0.95
##
## $plot
```



Q: Explain what new features you designed and why you chose them.

A:

A number of interaction models were evaluated, mixes of Genres, Actors, and Writers. The intuition was that movies with a mix of specific sets of Actors, Writers, or Genres might do beter than others. A custum pairwise interaction with Budget_bin, Year_bin, and Runtime_bin was also attempted to explore the possible relationships of Budget and Runtime over the years. While some of these models improved RMSE on the training set, none of them made a significant improvement on the test set.

New features were created on character data indicating the number of elements (comma seperated values) and field lengths. This indicates the number of Writers, Directors, etc. that were invovled in a movie. The length transformation also indicates how much was written in the Plot and tomatoConsensus fields. Unfortunately, none of these new features contributed significantly to improving the model.

A feature was created to capture ordering in the Rated feature as a numeric value, with G being the lowest and increasing from there. This new feature did not significantly improve the model either.

Yet another feature was added that indicated the quantity of top N Actors, etc. that were associated with each movie. As an example, a movie could have 3 of the top 10 most prolific writers. This was calculated for Actors, Writers, Languages, Countries, Directors, and Production studios. Again, these new features did not improve the model performance.

The final attempt was to reintroduce features that where previously filtered to reduce dementionality. In this final model, all Genres were captured in binary featues instead of limiting it to the top 10. The resulting model was able to reduce the test set RMSE to around 8.2e7. This leads to the belief that there are likely other predictive features in the dataset that would be discovered with an exhaustive, and potentially computationally costly, search. An effort was put into exploring recursive feature selection, but was abandoned in light of the computational time required given the number of features available.