By Xiaodong Gu (xgu60)

title: ‘Project 2: Modeling and Evaluation’ subtitle: ’
CSE6242 - Data and Visual Analytics

Due: Friday, April 21, 2017 at 11:59 PM UTC-12:00 on T-Square

’ output: pdf_document: default html_document: default —

Data

We will use the same dataset as Project 1: `movies_merged` (https://s3.amazonaws.com/content.udacity-data.com/courses/gt-cs6242/project/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 (http://rmarkdown.rstudio.com) 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.

```
library(ggplot2)
```

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: None

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

```
# TODO: Remove all rows from df that do not correspond to movies
df = df[df$Type == "movie", ]
```

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.

```
# TODO: Remove rows with missing Gross value
df = df[!is.na(df$Gross), ]
df = df[df$Gross != 0,]
nrow(df)
```

```
## [1] 4273
```

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).

```
# TODO: Exclude movies released prior to 2000
df = df[df$Year >= 2000,]
```

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.

```
# TODO: Remove mismatched rows
#add column that match Year and Released
#nrow(df)
df$match = as.numeric(mapply(grepl, pattern=df$Year, x=df$Released))
df$match = df$match + as.numeric(is.na(df$Year) | is.na(df$Released))
df_match = df[df$match == 1, ]
#nrow(df_match)
df_dismatch = df[df$match == 0, ]
#nrow(df_dismatch)
df_add = df_dismatch[sample(nrow(df_dismatch), 164), ]
df = rbind(df_match, df_add)
#nrow(df)

df$match = NULL
df_match = NULL
df_dismatch = NULL
df_add = NULL
```

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.

```
# TODO: Exclude the `Domestic_Gross` column
df$Domestic_Gross = NULL
```

6. Process `Runtime` column

```
# TODO: Replace df$Runtime with a numeric column containing the runtime in minutes
calTime = function(str){
  if(str == "N/A"){
    return(NA)
  }

  strList = unlist(strsplit(str, " "))
  h.pos = grep("h", strList)
  if(!is.na(h.pos[1])){
    h.num = as.numeric(strList[h.pos[1] - 1])
  } else{
    h.num = 0
  }
  min.pos = grep("min", strList)
  if(!is.na(min.pos[1])){
    min.num = as.numeric(strList[min.pos[1] - 1])
  } else{
    min.num = 0
  }
  return (h.num * 60 + min.num)
}

df$Runtime = mapply(calTime, df$Runtime)
```

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).

```
# TODO(optional): Additional preprocessing
```

***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] 2700   38
```

```
names(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] "Gross"             "Date"
```

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.

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.

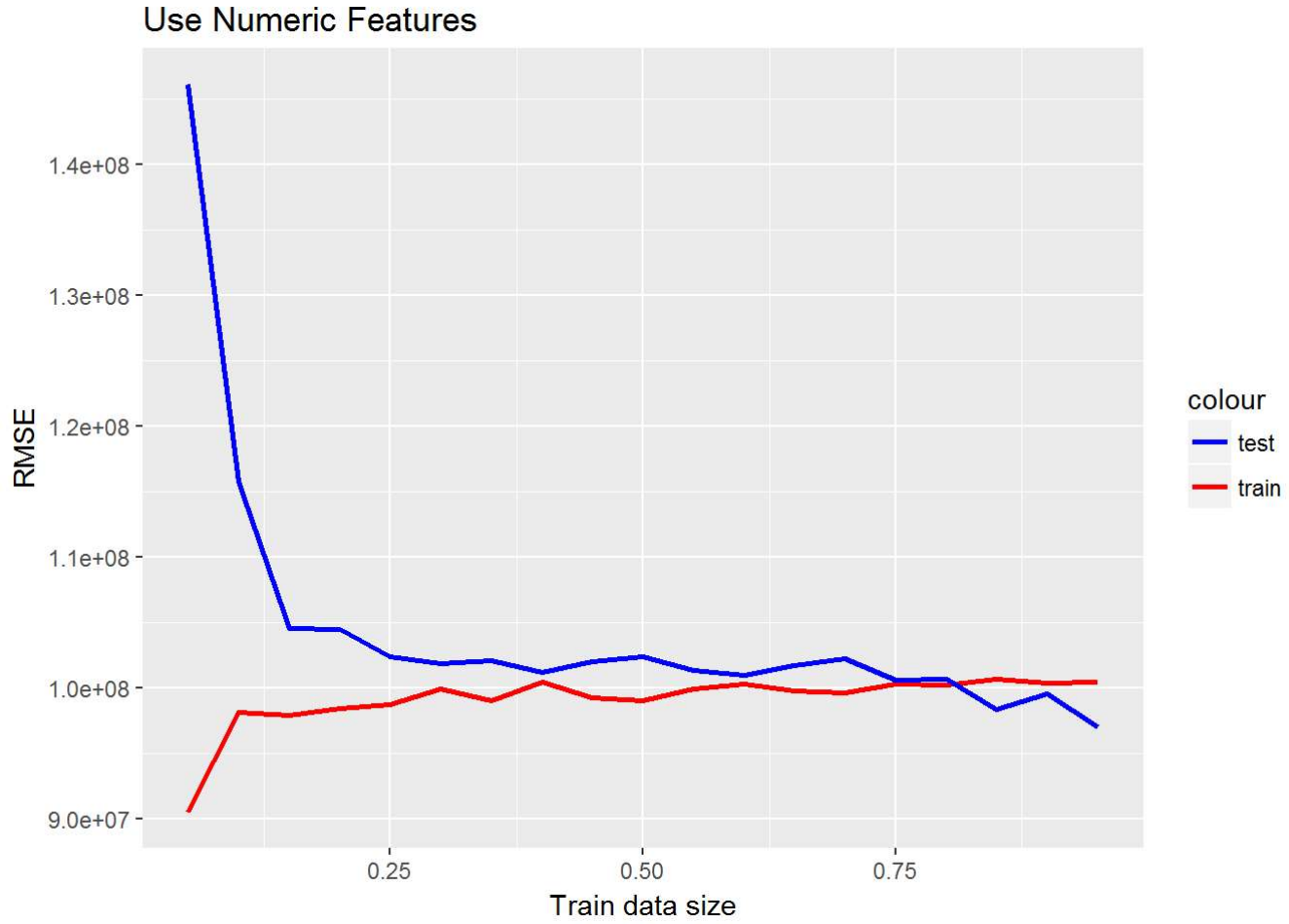
```
# TODO: Build & evaluate model 1 (numeric variables only)
#function to randimize samples and divide train and test sets
get_sample_sets = function(dataframe, train_perc){
  row = nrow(dataframe)
  index = sample(1:row, row)
  index_train = index[1 : (row * train_perc)]
  index_test = index[(row * train_perc + 1) : row]
  return (list(dataframe[index_train, ], dataframe[index_test, ]))
}

#calculate rmse
rmse = function(true, predict){
  len = length(predict[!is.na(predict)])
  res = sqrt(sum((true - predict) ^ 2, na.rm = TRUE) / len)
  return(res)
}
#1. Randomly divide the rows into two sets of sizes 5% and 95%.
sets = get_sample_sets(df, 0.05)
train_data = sets[[1]]
test_data = sets[[2]]

#2. Use the first set for training and the second for testing.
m1 = lm(Gross~Runtime+imdbRating+imdbVotes+
          tomatoUserMeter+tomatoUserRating+
          tomatoUserReviews+Budget, train_data)
pred_train = predict(m1, train_data)
pred_test = predict(m1, test_data)

#3. Print out the Root Mean Squared Error (RMSE) on the train and test sets.
print(c(rmse(train_data$Gross, pred_train), rmse(test_data$Gross, pred_test)))
```

```
## [1]  92779931 103619832
```

```
#4. Repeat the above data partition and model training and evaluation 50 times and average the R
MSE results so the results stabilize.
train_rmse = c()
test_rmse = c()
for(i in 1:100){
    sets = get_sample_sets(df, 0.05)
    m1 = lm(Gross~Runtime+imdbRating+imdbVotes+
              tomatoUserMeter+tomatoUserRating+
              tomatoUserReviews+Budget, sets[[1]])
    pred_train = predict(m1, sets[[1]])
    pred_test = predict(m1, sets[[2]])
    train_rmse = c(train_rmse, rmse(sets[[1]]$Gross, pred_train))
    test_rmse = c(test_rmse, rmse(sets[[2]]$Gross, pred_test))
  }
print(c(mean(train_rmse), mean(test_rmse)))
```

```
## [1]  92558285 152579955
```

```
#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%).
perc = seq(0.05, 0.95, 0.05)
train_rmse = c()
test_rmse = c()
for(i in 1:length(perc)){
  train_rmse_perc = c()
  test_rmse_perc = c()
  for(j in 1:100){
    sets = get_sample_sets(df, perc[i])
    m1 = lm(Gross~Runtime+imdbRating+imdbVotes+
              tomatoUserMeter+tomatoUserRating+
              tomatoUserReviews+Budget, sets[[1]])
    pred_train = predict(m1, sets[[1]])
    pred_test = predict(m1, sets[[2]])
    train_rmse_perc = c(train_rmse_perc, rmse(sets[[1]]$Gross, pred_train))
    test_rmse_perc = c(test_rmse_perc, rmse(sets[[2]]$Gross, pred_test))
  }
  train_rmse = c(train_rmse, mean(train_rmse_perc))
  test_rmse = c(test_rmse, mean(test_rmse_perc))
}

errors = data.frame(train_size = perc, train_rmse_01 = train_rmse, test_rmse_01 = test_rmse)

#6. Generate a graph of the averaged train and test RMSE as a function of the train set size
 (%).
plot = ggplot(errors, aes(train_size, train_rmse_01, color = "train")) +
  geom_line(size = 1) +
  geom_line(aes(train_size, test_rmse_01, color = "test"), size = 1)+
  scale_color_manual(values=c("train"="red",  "test"="blue"))+
  ggtitle("Use Numeric Features") +
  labs(x="Train data size", y="RMSE")
print(plot)
```



Q: List all the numeric variables you used.

A: At the beginning, all numerical features were applied. Some less valuable features were removed one by one (if removing one feature does not effect final rmse values, the feature was consider as less valuable feature). The final numeric variables used: Runtime, imdbRating, imdbVotes, tomatoUserMeter, tomatoUserRating, tomatoUserReviews, Budget.

To get smooth curve, all data points are averages of 100 runs.

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`).

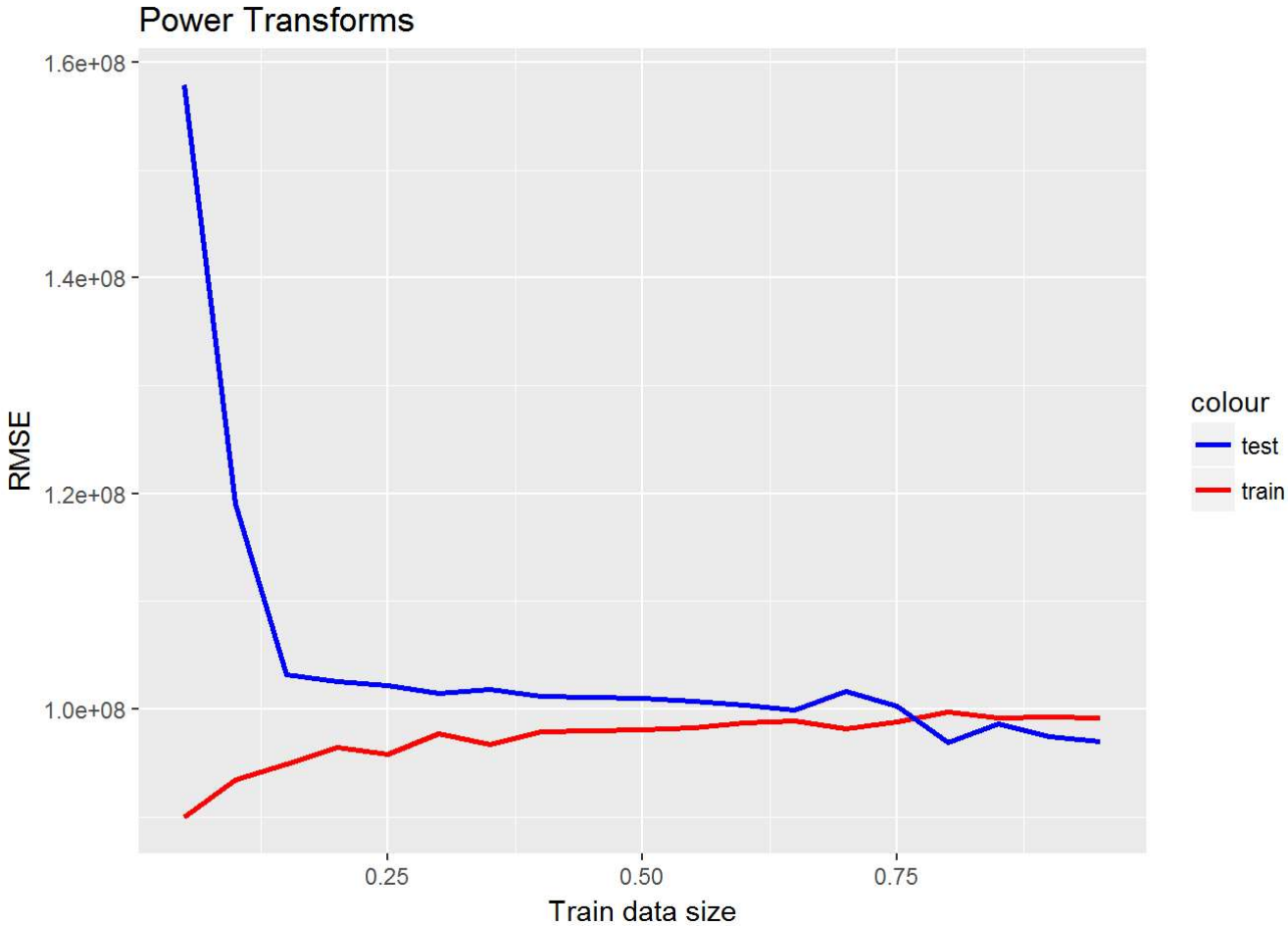
```
# TODO: Build & evaluate model 2 (transformed numeric variables only)
#power transforms
#1. Randomly divide the rows into two sets of sizes 5% and 95%.
sets = get_sample_sets(df, 0.05)
train_data = sets[[1]]
test_data = sets[[2]]

#2. Use the first set for training and the second for testing.
m1 = lm(Gross~Runtime+
          imdbRating+I(imdbRating^2)+I(imdbRating^3)+
          imdbVotes+
          tomatoUserMeter+I(tomatoUserMeter^2)+
          tomatoUserRating+I(tomatoUserRating^2)+I(tomatoUserRating^3)+
          tomatoUserReviews+
          Budget, train_data)
pred_train = predict(m1, train_data)
pred_test = predict(m1, test_data)

#3. Print out the Root Mean Squared Error (RMSE) on the train and test sets.
print(c(rmse(train_data$Gross, pred_train), rmse(test_data$Gross, pred_test)))
```

```
## [1] 124596587 123778973
```

```
#4. Repeat the above data partition and model training and evaluation 50 times and average the R
MSE results so the results stabilize.
train_rmse = c()
test_rmse = c()
for(i in 1:100){
    sets = get_sample_sets(df, 0.05)
    m1 = lm(Gross~Runtime+
            imdbRating+I(imdbRating^2)+I(imdbRating^3)+
            imdbVotes+
            tomatoUserMeter+I(tomatoUserMeter^2)+
            tomatoUserRating+I(tomatoUserRating^2)+I(tomatoUserRating^3)+
            tomatoUserReviews+
            Budget, sets[[1]])
    pred_train = predict(m1, sets[[1]])
    pred_test = predict(m1, sets[[2]])
    train_rmse = c(train_rmse, rmse(sets[[1]]$Gross, pred_train))
    test_rmse = c(test_rmse, rmse(sets[[2]]$Gross, pred_test))
  }
print(c(mean(train_rmse), mean(test_rmse)))
```

```
## [1]  86383745 149908354
```

```
#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%).
train_rmse = c()
test_rmse = c()
for(i in 1:length(perc)){
  train_rmse_perc = c()
  test_rmse_perc = c()
  for(j in 1:100){
    sets = get_sample_sets(df, perc[i])
    m1 = lm(Gross~Runtime+
            imdbRating+I(imdbRating^2)+I(imdbRating^3)+
            imdbVotes+
            tomatoUserMeter+I(tomatoUserMeter^2)+
            tomatoUserRating+I(tomatoUserRating^2)+I(tomatoUserRating^3)+
            tomatoUserReviews+
            Budget, sets[[1]])
    pred_train = predict(m1, sets[[1]])
    pred_test = predict(m1, sets[[2]])
    train_rmse_perc = c(train_rmse_perc, rmse(sets[[1]]$Gross, pred_train))
    test_rmse_perc = c(test_rmse_perc, rmse(sets[[2]]$Gross, pred_test))
  }
  train_rmse = c(train_rmse, mean(train_rmse_perc))
  test_rmse = c(test_rmse, mean(test_rmse_perc))
}

errors$train_rmse_02 = train_rmse
errors$test_rmse_02 = test_rmse

plot = ggplot(errors, aes(train_size, train_rmse_02, color = "train")) +
  geom_line(size = 1) +
  geom_line(aes(train_size, test_rmse_02, color = "test"), size = 1)+
  scale_color_manual(values=c("train"="red",  "test"="blue")) +
  ggtitle("Power Transforms") +
  labs(x="Train data size", y="RMSE")
print(plot)
```



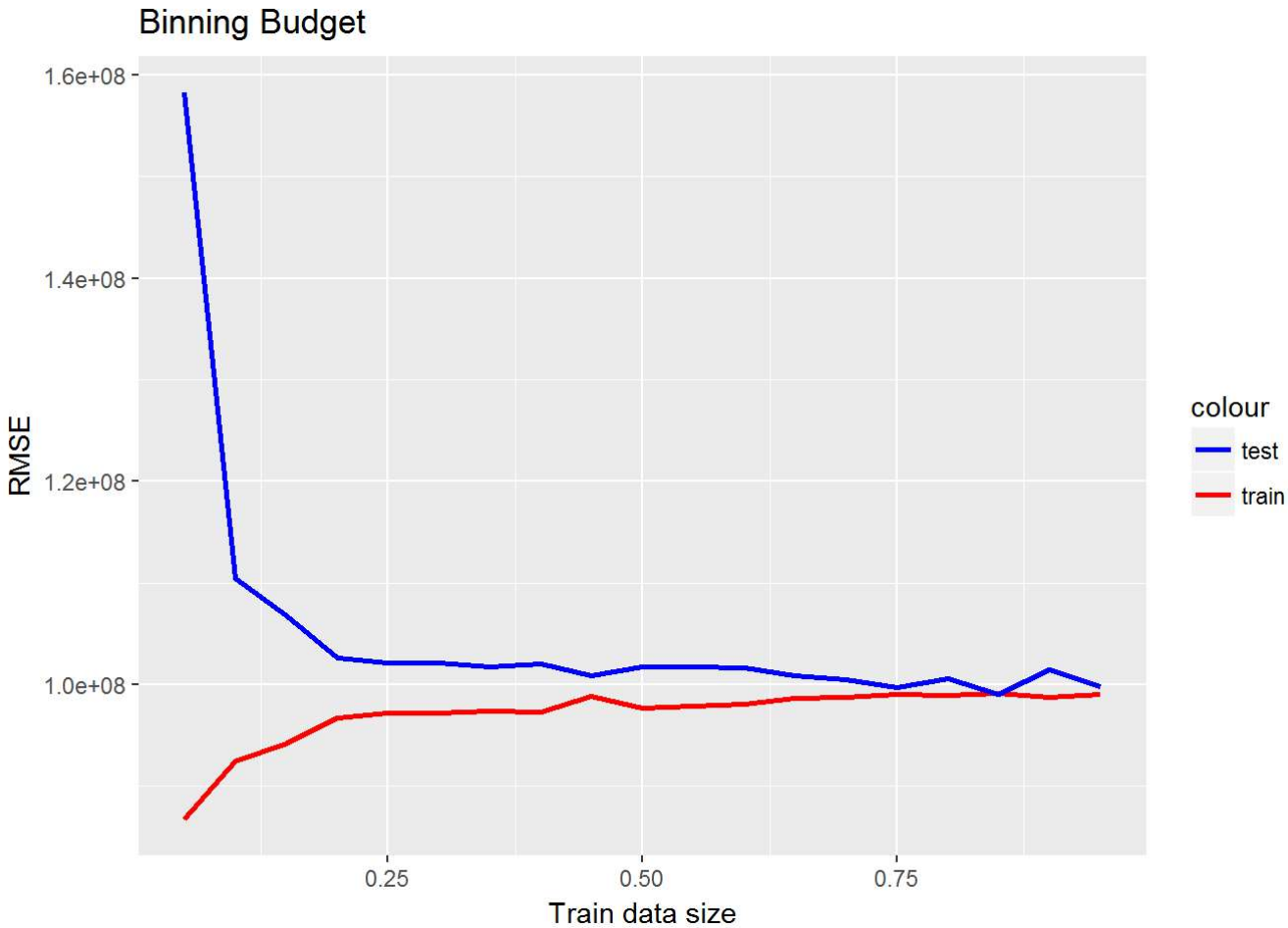

```
#binning Budget
df$is_budget_great_than_10M = as.numeric(df$Budget > 10000000)
df$is_budget_great_than_50M = as.numeric(df$Budget > 50000000)
df$is_budget_great_than_100M = as.numeric(df$Budget > 100000000)

#1. Randomly divide the rows into two sets of sizes 5% and 95%.
sets = get_sample_sets(df, 0.05)
train_data = sets[[1]]
test_data = sets[[2]]

#2. Use the first set for training and the second for testing.
m1 = lm(Gross~Runtime+imdbRating+imdbVotes+
            tomatoUserMeter+tomatoUserRating+tomatoUserReviews+
            Budget+
            is_budget_great_than_10M+
            is_budget_great_than_50M+
            is_budget_great_than_100M, train_data)
pred_train = predict(m1, train_data)
pred_test = predict(m1, test_data)

#3. Print out the Root Mean Squared Error (RMSE) on the train and test sets.
print(c(rmse(train_data$Gross, pred_train), rmse(test_data$Gross, pred_test)))
```

```
## [1]  97599500 112411831
```

```
#4. Repeat the above data partition and model training and evaluation 50 times and average the R
MSE results so the results stabilize.
train_rmse = c()
test_rmse = c()
for(i in 1:100){
    sets = get_sample_sets(df, 0.05)
    m1 = lm(Gross~Runtime+imdbRating+imdbVotes+
            tomatoUserMeter+tomatoUserRating+tomatoUserReviews+
            Budget+
            is_budget_great_than_10M+
            is_budget_great_than_50M+
            is_budget_great_than_100M, sets[[1]])
    pred_train = predict(m1, sets[[1]])
    pred_test = predict(m1, sets[[2]])
    train_rmse = c(train_rmse, rmse(sets[[1]]$Gross, pred_train))
    test_rmse = c(test_rmse, rmse(sets[[2]]$Gross, pred_test))
  }
print(c(mean(train_rmse), mean(test_rmse)))
```

```
## [1]  89229237 150795918
```

```
#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%).
train_rmse = c()
test_rmse = c()
for(i in 1:length(perc)){
  train_rmse_perc = c()
  test_rmse_perc = c()
  for(j in 1:100){
    sets = get_sample_sets(df, perc[i])
    m1 = lm(Gross~Runtime+imdbRating+imdbVotes+
            tomatoUserMeter+tomatoUserRating+tomatoUserReviews+
            Budget+
            is_budget_great_than_10M+
            is_budget_great_than_50M+
            is_budget_great_than_100M, sets[[1]])
    pred_train = predict(m1, sets[[1]])
    pred_test = predict(m1, sets[[2]])
    train_rmse_perc = c(train_rmse_perc, rmse(sets[[1]]$Gross, pred_train))
    test_rmse_perc = c(test_rmse_perc, rmse(sets[[2]]$Gross, pred_test))
  }
  train_rmse = c(train_rmse, mean(train_rmse_perc))
  test_rmse = c(test_rmse, mean(test_rmse_perc))
}

errors$train_rmse_03 = train_rmse
errors$test_rmse_03 = test_rmse
plot = ggplot(errors, aes(train_size, train_rmse_03, color = "train")) +
  geom_line(size = 1) +
  geom_line(aes(train_size, test_rmse_03, color = "test"), size = 1)+
  scale_color_manual(values=c("train"="red",  "test"="blue")) +
  ggtitle("Binning Budget") +
  labs(x="Train data size", y="RMSE")
print(plot)
```



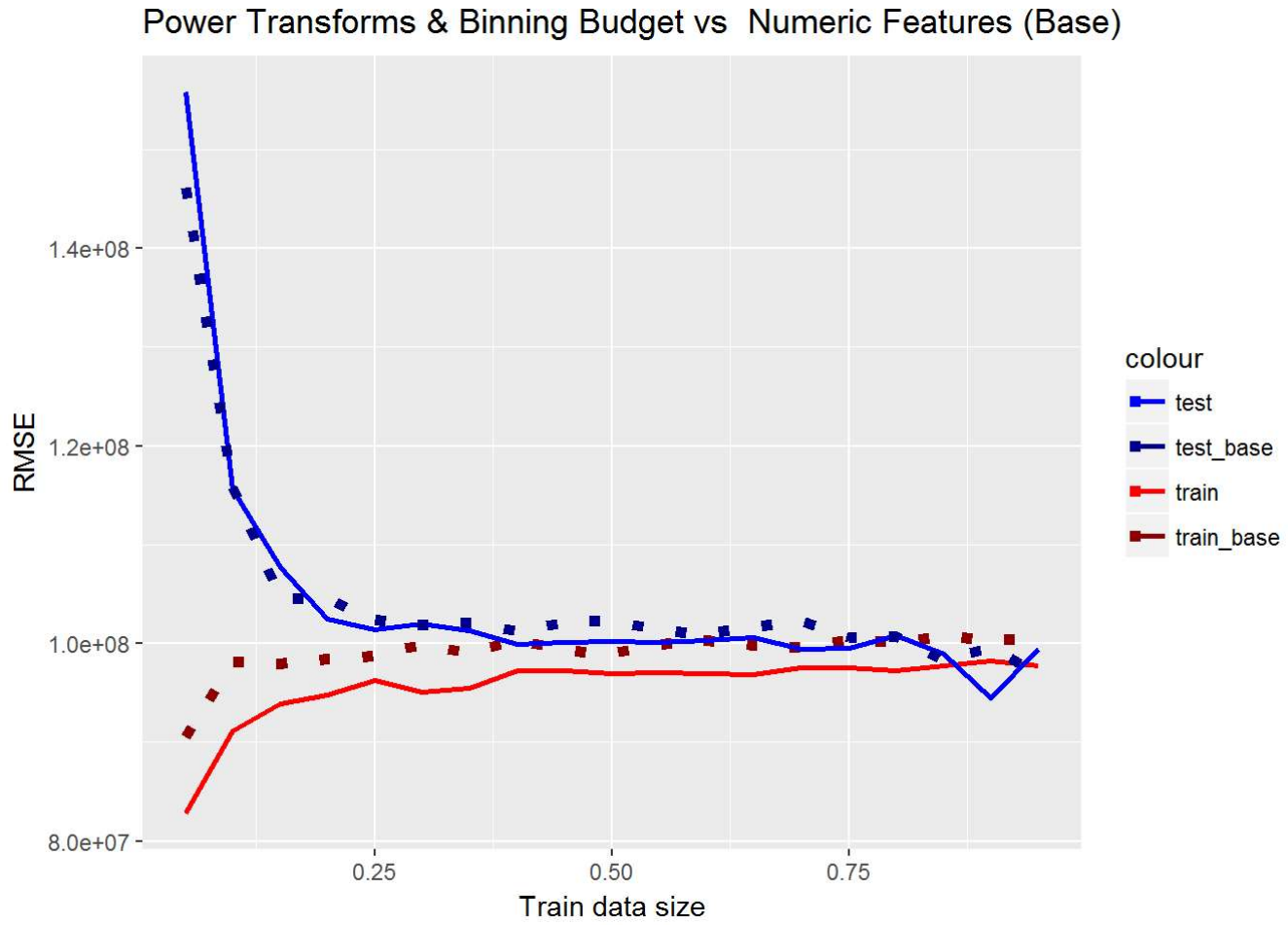

```
#combine power transform and binning
#1. Randomly divide the rows into two sets of sizes 5% and 95%.
sets = get_sample_sets(df, 0.05)
train_data = sets[[1]]
test_data = sets[[2]]

#2. Use the first set for training and the second for testing.
m1 = lm(Gross~Runtime+
            imdbRating+I(imdbRating^2)+I(imdbRating^3)+
            imdbVotes+
            tomatoUserMeter+I(tomatoUserMeter^2)+
            tomatoUserRating+I(tomatoUserRating^2)+I(tomatoUserRating^3)+
            tomatoUserReviews+
            Budget+
            is_budget_great_than_10M+
            is_budget_great_than_50M+
            is_budget_great_than_100M, train_data)
pred_train = predict(m1, train_data)
pred_test = predict(m1, test_data)

#3. Print out the Root Mean Squared Error (RMSE) on the train and test sets.
print(c(rmse(train_data$Gross, pred_train), rmse(test_data$Gross, pred_test)))
```

```
## [1]  79952000 249822370
```

```
#4. Repeat the above data partition and model training and evaluation 50 times and average the R
MSE results so the results stabilize.
train_rmse = c()
test_rmse = c()
for(i in 1:100){
    sets = get_sample_sets(df, 0.05)
    m1 = lm(Gross~Runtime+
            imdbRating+I(imdbRating^2)+I(imdbRating^3)+
            imdbVotes+
            tomatoUserMeter+I(tomatoUserMeter^2)+
            tomatoUserRating+I(tomatoUserRating^2)+I(tomatoUserRating^3)+
            tomatoUserReviews+
            Budget+
            is_budget_great_than_10M+
            is_budget_great_than_50M+
            is_budget_great_than_100M, sets[[1]])
    pred_train = predict(m1, sets[[1]])
    pred_test = predict(m1, sets[[2]])
    train_rmse = c(train_rmse, rmse(sets[[1]]$Gross, pred_train))
    test_rmse = c(test_rmse, rmse(sets[[2]]$Gross, pred_test))
  }
print(c(mean(train_rmse), mean(test_rmse)))
```

```
## [1]  83183987 140209094
```

```
#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%).
train_rmse = c()
test_rmse = c()
for(i in 1:length(perc)){
  train_rmse_perc = c()
  test_rmse_perc = c()
  for(j in 1:100){
    sets = get_sample_sets(df, perc[i])
    m1 = lm(Gross~Runtime+
            imdbRating+I(imdbRating^2)+I(imdbRating^3)+
            imdbVotes+
            tomatoUserMeter+I(tomatoUserMeter^2)+
            tomatoUserRating+I(tomatoUserRating^2)+I(tomatoUserRating^3)+
            tomatoUserReviews+
            Budget+
            is_budget_great_than_10M+
            is_budget_great_than_50M+
            is_budget_great_than_100M, sets[[1]])
    pred_train = predict(m1, sets[[1]])
    pred_test = predict(m1, sets[[2]])
    train_rmse_perc = c(train_rmse_perc, rmse(sets[[1]]$Gross, pred_train))
    test_rmse_perc = c(test_rmse_perc, rmse(sets[[2]]$Gross, pred_test))
  }
  train_rmse = c(train_rmse, mean(train_rmse_perc))
  test_rmse = c(test_rmse, mean(test_rmse_perc))
}

errors$train_rmse_04 = train_rmse
errors$test_rmse_04 = test_rmse
plot = ggplot(errors, aes(train_size, train_rmse_04, color = "train")) +
  geom_line(size = 1) +
  geom_line(aes(train_size, train_rmse_01, color = "train_base"), size = 2, linetype="dotted")+
  geom_line(aes(train_size, test_rmse_04, color = "test"), size = 1)+
  geom_line(aes(train_size, test_rmse_01, color = "test_base"), size = 2, linetype="dotted")+
  scale_color_manual(values=c("train"="red", "train_base"="red4",
                              "test"="blue", "test_base"="blue4")) +
  ggtitle("Power Transforms & Binning Budget vs  Numeric Features (Base)") +
  labs(x="Train data size", y="RMSE")

print(plot)
```



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

A: Power transform is applied on imbdRating and tomatoUserRating. They are both good indicators of the qualities of movies, and all have relately small values compared with other features. The budget 0f a movie was divided into three categories: >10M, >50M and >100M. After applying numeric and non-bumeric transforms, the rmse for train and test both improved.

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 collection of binary columns for Genres
#get a vector contains all genres
genre.all = unlist(strsplit(df$Genre, ", "))
genre.unique = unique(genre.all)
genre.unique = genre.unique[genre.unique != "N/A"]

#create new column, and insert 0 or 1
counts = c()
for(ge in genre.unique){
  colname = paste("G", ge, sep = ".")
  values = as.numeric(grepl(ge, df$Genre))
  df[, colname] = values
  counts = append(counts, sum(values))
}

df$Genre = NULL

#Convert Awards to 2 numeric columns: wins and nominations
#count wins from a string
countWin = function(str){
  str.list = unlist(strsplit(str, " "))
  m1 = grep("win", str.list)
  return (as.numeric(str.list[m1[1] - 1]))
}

#count nominations from a string
countNomination = function(str){
  str.list = unlist(strsplit(str, " "))
  m2 = grep("nomination", str.list)
  return (as.numeric(str.list[m2[1] - 1]))
}

#add two columns for wins and nominations
df$A.win = mapply(countWin, df$Awards)
df$A.win[is.na(df$A.win)] = 0
df$A.nomination = mapply(countNomination, df$Awards)
df$A.nomination[is.na(df$A.nomination)] = 0

#divide directors by clubs by the budget of movies they directed (100, 50, 10, 5, 1, 0.1 millio
n)
get_directors = function(df, budget){
  temp_df = df[df$Budget >= budget, ]
  directors = unlist(strsplit(temp_df$Director, ", "))
  directors = unique(directors)
  return (directors)
}

director_club = function(df){
  c1 = get_directors(df, 1E8)
  c2 = get_directors(df, 5E7)
  c3 = get_directors(df, 1E7)
```

```
  c4 = get_directors(df, 5E6)
  c5 = get_directors(df, 1E6)
  c6 = get_directors(df, 1E5)
  return (list(c1, c2, c3, c4, c5, c6))
}

director_search = function(str, club){
  directors_in_str = unlist(strsplit(str, ", "))
  for(director in directors_in_str){
    if(director %in% club){
      return (1)
    }
  }
  return (0)
}

#divide actors by clubs according the gross of movies they participated (100, 50, 10, 5, 1, 0.1
 million)
get_actors = function(df, budget){
  temp_df = df[df$Budget >= budget,]
  actors = unlist(strsplit(temp_df$Actors, ", "))
  actors = unique(actors)
  return (actors)
}

actor_club = function(df){
  c1 = get_actors(df, 1E8)
  c2 = get_actors(df, 5E7)
  c3 = get_actors(df, 1E7)
  c4 = get_actors(df, 5E6)
  c5 = get_actors(df, 1E6)
  c6 = get_actors(df, 1E5)
  return (list(c1, c2, c3, c4, c5, c6))
}

actor_search = function(str, club){
  actors_in_str = unlist(strsplit(str, ", "))
  for(actor in actors_in_str){
    if(actor %in% club){
      return (1)
    }
  }
  return (0)
}

add_director_actor_columns = function(df, d.clubs, a.clubs){
  df$d.c1 = sapply(df$Director, director_search, d.clubs[[1]])
  df$d.c2 = sapply(df$Director, director_search, d.clubs[[2]])
  df$d.c3 = sapply(df$Director, director_search, d.clubs[[3]])
  df$d.c4 = sapply(df$Director, director_search, d.clubs[[4]])
  df$d.c5 = sapply(df$Director, director_search, d.clubs[[5]])
  df$d.c6 = sapply(df$Director, director_search, d.clubs[[6]])

  df$a.c1 = sapply(df$Actors, actor_search, a.clubs[[1]])
```

```
  df$a.c2 = sapply(df$Actors, actor_search, a.clubs[[2]])
  df$a.c3 = sapply(df$Actors, actor_search, a.clubs[[3]])
  df$a.c4 = sapply(df$Actors, actor_search, a.clubs[[4]])
  df$a.c5 = sapply(df$Actors, actor_search, a.clubs[[5]])
  df$a.c6 = sapply(df$Actors, actor_search, a.clubs[[6]])
  return (df)
}


d_clubs = director_club(df)
a_clubs = actor_club(df)
df = add_director_actor_columns(df, d_clubs, a_clubs)

#create new data frame to store all new features
new_df = df[, c(36, 41:78)]
names(new_df)
```

```
##  [1] "Gross"         "G.Horror"      "G.Sci-Fi"      "G.Adventure"
##  [5] "G.Comedy"      "G.Family"      "G.Crime"       "G.Music"
##  [9] "G.Drama"       "G.Mystery"     "G.Thriller"    "G.Romance"
## [13] "G.Sport"       "G.Fantasy"     "G.Action"      "G.Documentary"
## [17] "G.Biography"   "G.History"     "G.Animation"   "G.Western"
## [21] "G.Musical"     "G.War"         "G.Short"       "G.News"
## [25] "G.Adult"       "A.win"         "A.nomination"  "d.c1"
## [29] "d.c2"          "d.c3"          "d.c4"          "d.c5"
## [33] "d.c6"          "a.c1"          "a.c2"          "a.c3"
## [37] "a.c4"          "a.c5"          "a.c6"
```

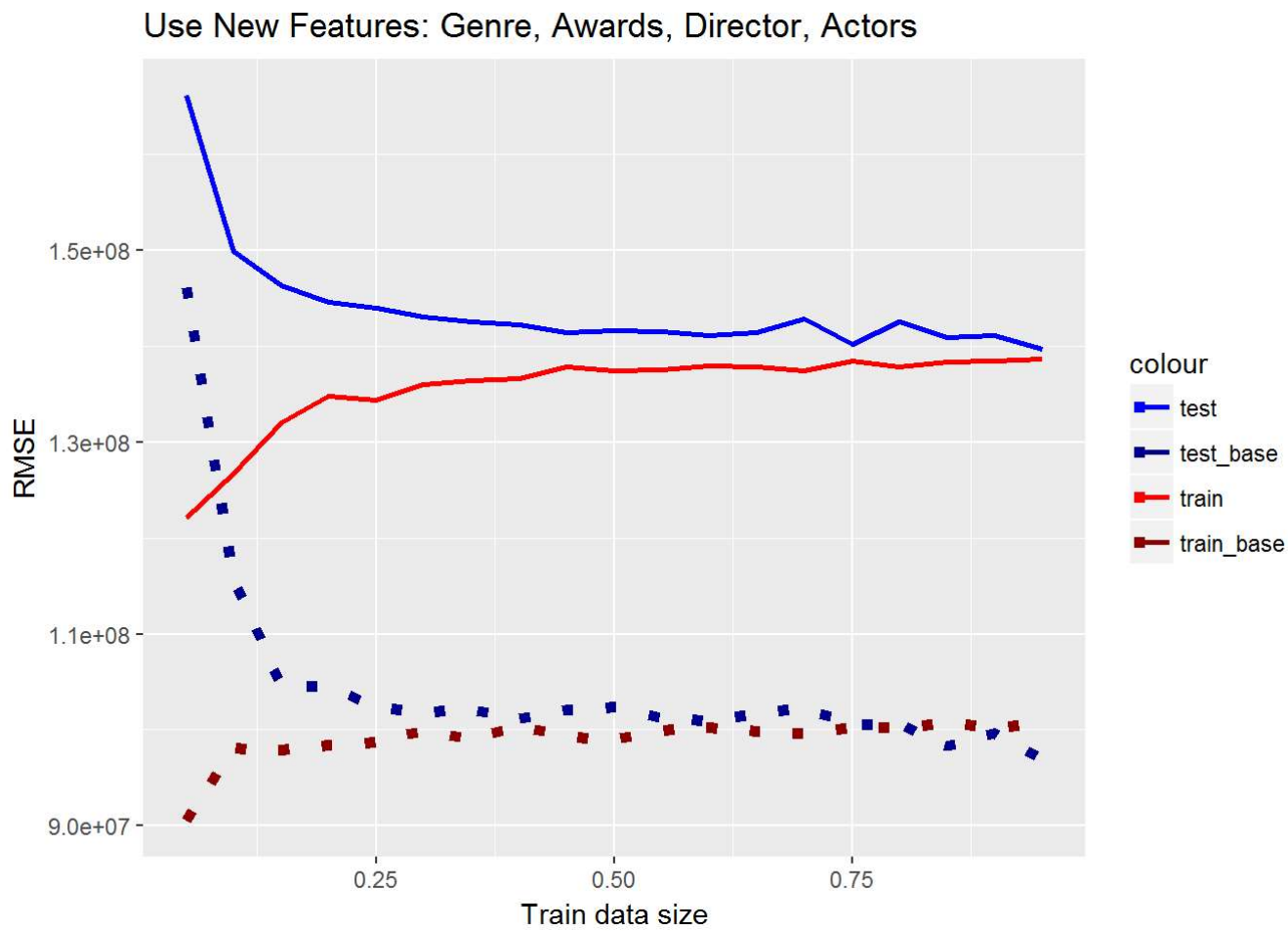
```
#linear regression use converted columns
#1. Randomly divide the rows into two sets of sizes 5% and 95%.
sets = get_sample_sets(new_df, 0.05)
train_data = sets[[1]]
test_data = sets[[2]]

#2. Use the first set for training and the second for testing.
m1 = lm(Gross~., train_data)
pred_train = predict(m1, train_data)
pred_test = predict(m1, test_data)

#3. Print out the Root Mean Squared Error (RMSE) on the train and test sets.
print(c(rmse(train_data$Gross, pred_train), rmse(test_data$Gross, pred_test)))
```

```
## [1] 118900818 158048694
```

```
#4. Repeat the above data partition and model training and evaluation 50 times and average the R
MSE results so the results stabilize.
train_rmse = c()
test_rmse = c()
for(i in 1:100){
    sets = get_sample_sets(new_df, 0.05)
    train_data = sets[[1]]
    test_data = sets[[2]]

    m1 = lm(Gross~., train_data)
    pred_train = predict(m1, train_data)
    pred_test = predict(m1, test_data)
    train_rmse = c(train_rmse, rmse(train_data$Gross, pred_train))
    test_rmse = c(test_rmse, rmse(test_data$Gross, pred_test))
  }
print(c(mean(train_rmse), mean(test_rmse)))
```

```
## [1] 118316222 163394651
```

```
#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%).
train_rmse = c()
test_rmse = c()
for(i in 1:length(perc)){
  train_rmse_perc = c()
  test_rmse_perc = c()
  for(j in 1:100){
    sets = get_sample_sets(new_df, perc[i])
    train_data = sets[[1]]
    test_data = sets[[2]]

    m1 = lm(Gross~., train_data)
    pred_train = predict(m1, train_data)
    pred_test = predict(m1, test_data)
    train_rmse_perc = c(train_rmse_perc, rmse(train_data$Gross, pred_train))
    test_rmse_perc = c(test_rmse_perc, rmse(test_data$Gross, pred_test))
  }
  train_rmse = c(train_rmse, mean(train_rmse_perc))
  test_rmse = c(test_rmse, mean(test_rmse_perc))
}


errors$train_rmse_05 = train_rmse
errors$test_rmse_05 = test_rmse
plot = ggplot(errors, aes(train_size, train_rmse_05, color = "train")) +
  geom_line(size = 1) +
  geom_line(aes(train_size, train_rmse_01, color = "train_base"), size = 2, linetype="dotted")+
  geom_line(aes(train_size, test_rmse_05, color = "test"), size = 1)+
  geom_line(aes(train_size, test_rmse_01, color = "test_base"), size = 2, linetype="dotted")+
  scale_color_manual(values=c("train"="red", "train_base"="red4",
                             "test"="blue", "test_base"="blue4")) +
  ggtitle("Use New Features: Genre, Awards, Director, Actors") +
  labs(x="Train data size", y="RMSE")
print(plot)
```



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

A: Categorical variables used are: Genre, Awards, Director, Actors. The Genre is handled same as described in project 1. New binary columns, e.g. G.Sci-Fi, G.Action are added. The Awards column is processed to generate two numeric columns:A.win, A.nomination. Directors and actors are clssified into six different clubs, based on the budget of movies (100, 50, 10, 5, 1, 0.1 million) that they participated. new binary columns: d.c1 - d.c6, a.c1 - a.c6 are added. For example, if a movie's director is in the 100 million director club, then d.c1 = 1. The assumpsion is that for two movies with similar budgets, the one has directors or actors with experience in high budget movies will likely beat the other one.

If only use new features for prediction, the results are not as good as use previous numerical features (base line).

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)
new_df = df[, c(5, 15, 16, 26:28, 35, 36, 38:78)]
names(new_df)
```

```
##  [1] "Runtime"                   "imdbRating"
##  [3] "imdbVotes"                 "tomatoUserMeter"
##  [5] "tomatoUserRating"          "tomatoUserReviews"
##  [7] "Budget"                    "Gross"
##  [9] "is_budget_great_than_10M"  "is_budget_great_than_50M"
## [11] "is_budget_great_than_100M" "G.Horror"
## [13] "G.Sci-Fi"                  "G.Adventure"
## [15] "G.Comedy"                  "G.Family"
## [17] "G.Crime"                   "G.Music"
## [19] "G.Drama"                   "G.Mystery"
## [21] "G.Thriller"                "G.Romance"
## [23] "G.Sport"                   "G.Fantasy"
## [25] "G.Action"                  "G.Documentary"
## [27] "G.Biography"               "G.History"
## [29] "G.Animation"               "G.Western"
## [31] "G.Musical"                 "G.War"
## [33] "G.Short"                   "G.News"
## [35] "G.Adult"                   "A.win"
## [37] "A.nomination"              "d.c1"
## [39] "d.c2"                      "d.c3"
## [41] "d.c4"                      "d.c5"
## [43] "d.c6"                      "a.c1"
## [45] "a.c2"                      "a.c3"
## [47] "a.c4"                      "a.c5"
## [49] "a.c6"
```

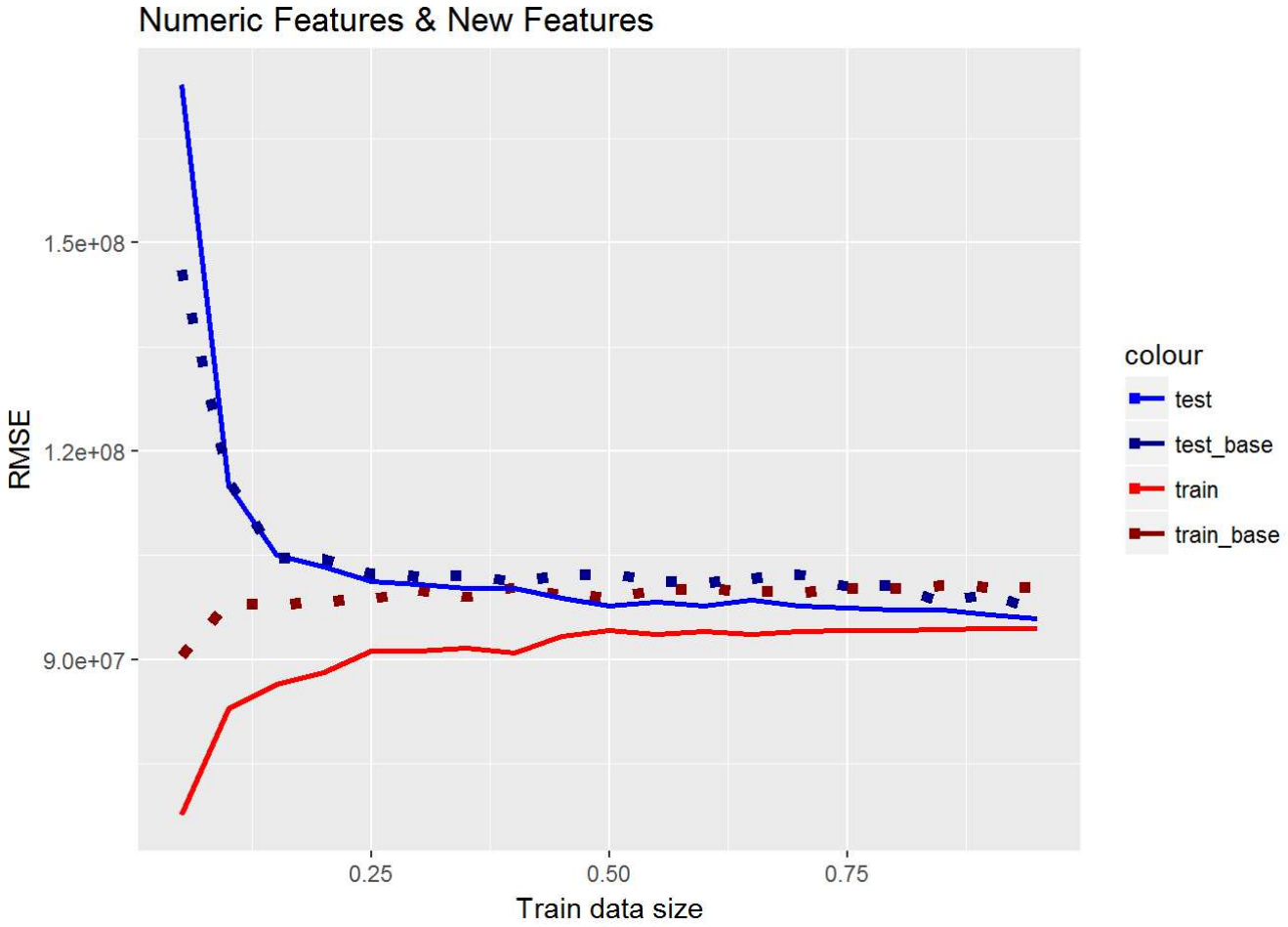
```
#1. Randomly divide the rows into two sets of sizes 5% and 95%.
sets = get_sample_sets(new_df, 0.05)
train_data = sets[[1]]
test_data = sets[[2]]

#2. Use the first set for training and the second for testing.
m1 = lm(Gross~imdbRating+I(imdbRating^2)+I(imdbRating^3)+
            tomatoUserMeter+I(tomatoUserMeter^2)+
            tomatoUserRating+I(tomatoUserRating^2)+I(tomatoUserRating^3)+
            ., train_data)
pred_train = predict(m1, train_data)
pred_test = predict(m1, test_data)

#3. Print out the Root Mean Squared Error (RMSE) on the train and test sets.
print(c(rmse(train_data$Gross, pred_train), rmse(test_data$Gross, pred_test)))
```

```
## [1] 101228877 170660278
```

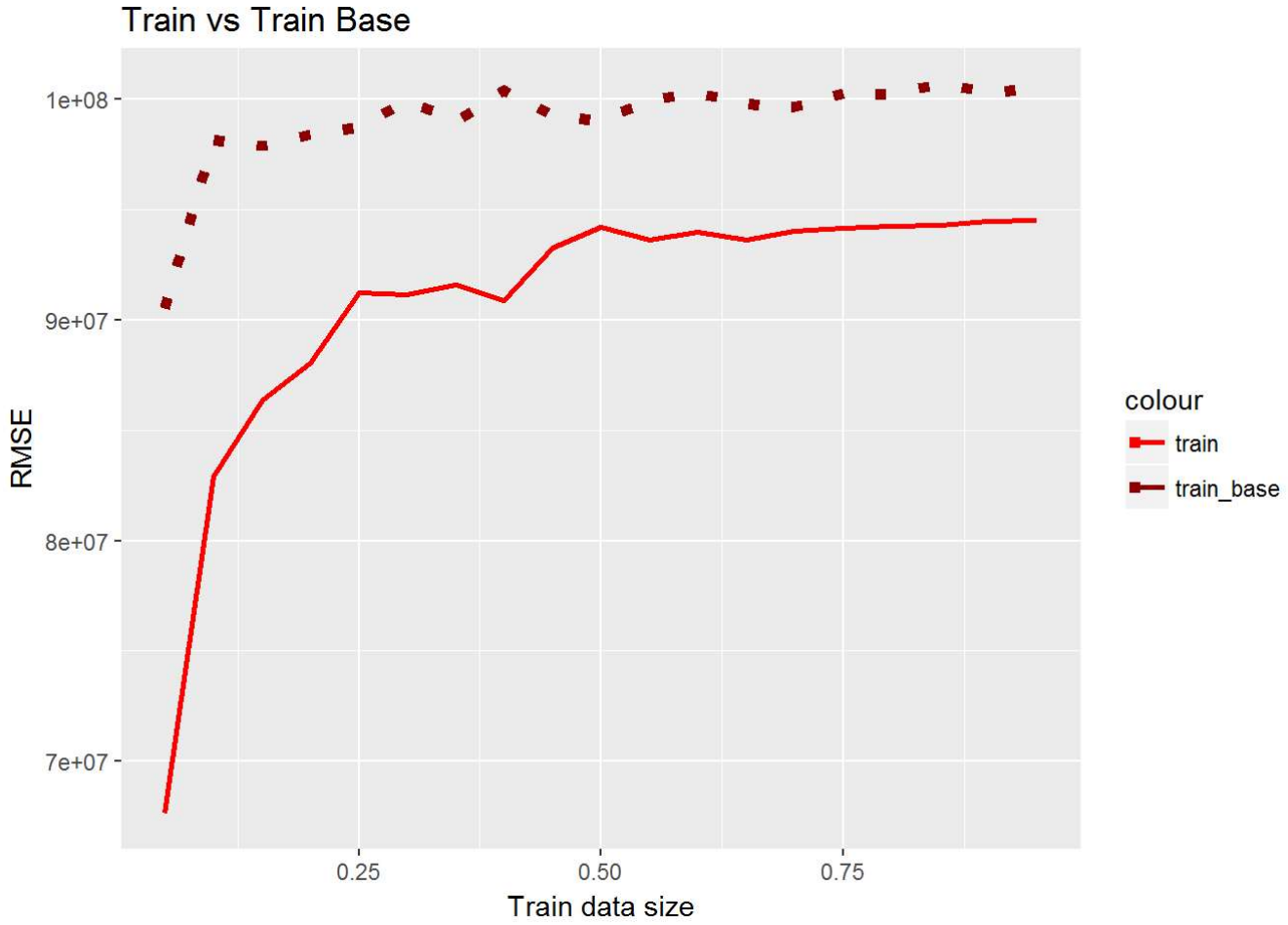
```
#4. Repeat the above data partition and model training and evaluation 50 times and average the R
MSE results so the results stabilize.
train_rmse = c()
test_rmse = c()
for(i in 1:100){
    sets = get_sample_sets(new_df, 0.05)
    train_data = sets[[1]]
    test_data = sets[[2]]

    m1 = lm(Gross~imdbRating+I(imdbRating^2)+I(imdbRating^3)+
            tomatoUserMeter+I(tomatoUserMeter^2)+
            tomatoUserRating+I(tomatoUserRating^2)+I(tomatoUserRating^3)+
            ., train_data)
    pred_train = predict(m1, train_data)
    pred_test = predict(m1, test_data)
    train_rmse = c(train_rmse, rmse(train_data$Gross, pred_train))
    test_rmse = c(test_rmse, rmse(test_data$Gross, pred_test))
  }
print(c(mean(train_rmse), mean(test_rmse)))
```

```
## [1]  68086316 180046877
```

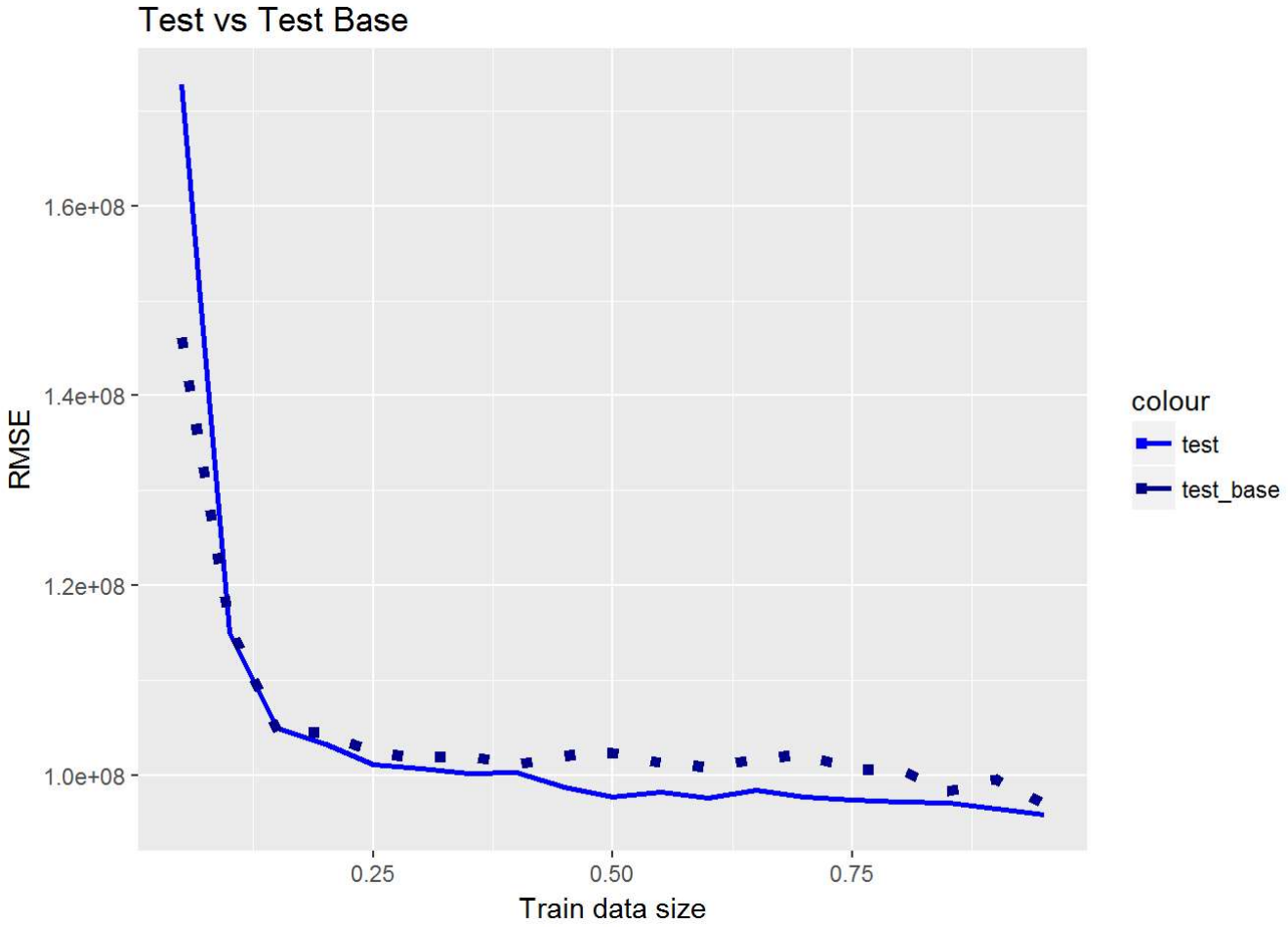
```
#5. Repeat the above steps for different proportions of train and test sizes: 10%-90%, 15%-85%,
 ..., 95%-5% (total 19 splits including the initial 5%-95%).
train_rmse = c()
test_rmse = c()
for(i in 1:length(perc)){
  train_rmse_perc = c()
  test_rmse_perc = c()
  for(j in 1:100){
    sets = get_sample_sets(new_df, perc[i])
    train_data = sets[[1]]
    test_data = sets[[2]]


    m1 = lm(Gross~imdbRating+I(imdbRating^2)+I(imdbRating^3)+
            tomatoUserMeter+I(tomatoUserMeter^2)+
            tomatoUserRating+I(tomatoUserRating^2)+I(tomatoUserRating^3)+
            ., train_data)
    pred_train = predict(m1, train_data)
    pred_test = predict(m1, test_data)
    train_rmse_perc = c(train_rmse_perc, rmse(train_data$Gross, pred_train))
    test_rmse_perc = c(test_rmse_perc, rmse(test_data$Gross, pred_test))
  }
  train_rmse = c(train_rmse, mean(train_rmse_perc))
  test_rmse = c(test_rmse, mean(test_rmse_perc))
}

errors$train_rmse_06 = train_rmse
errors$test_rmse_06 = test_rmse
plot1 = ggplot(errors, aes(train_size, train_rmse_06, color = "train")) +
  geom_line(size = 1) +
  geom_line(aes(train_size, train_rmse_01, color = "train_base"), size = 2, linetype="dotted")+
  geom_line(aes(train_size, test_rmse_06, color = "test"), size = 1)+
  geom_line(aes(train_size, test_rmse_01, color = "test_base"), size = 2, linetype="dotted")+
  scale_color_manual(values=c("train"="red", "train_base"="red4",
                              "test"="blue", "test_base"="blue4")) +
  ggtitle("Numeric Features & New Features") +
  labs(x="Train data size", y="RMSE")

plot2 = ggplot(errors, aes(train_size, train_rmse_06, color = "train")) +
  geom_line(size = 1) +
  geom_line(aes(train_size, train_rmse_01, color = "train_base"), size = 2, linetype="dotted")+
  scale_color_manual(values=c("train"="red", "train_base"="red4"))+
  ggtitle("Train vs Train Base") +
  labs(x="Train data size", y="RMSE")
plot3 = ggplot(errors, aes(train_size, test_rmse_06, color = "test")) +
  geom_line(size = 1) +
  geom_line(aes(train_size, test_rmse_01, color = "test_base"), size = 2, linetype="dotted")+
  scale_color_manual(values=c("test"="blue", "test_base"="blue4")) +
  ggtitle("Test vs Test Base") +
  labs(x="Train data size", y="RMSE")

print(plot1)
```


Numeric Features & New Features
RMSE
1.5e+08
1.2e+08
9.0e+07
0.25
0.50
0.75
Train data size
colour
test
test_base
train
train_base

```
print(plot2)
```


Train vs Train Base
1e+08
9e+07
8e+07
7e+07
RMSE
0.25
0.50
0.75
Train data size
colour
train
train_base

```
print(plot3)
```



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)
new_df = df[, c(5, 15, 16, 26:28, 35, 36, 38:78)]
names(new_df)
```

```
##  [1] "Runtime"                   "imdbRating"
##  [3] "imdbVotes"                 "tomatoUserMeter"
##  [5] "tomatoUserRating"          "tomatoUserReviews"
##  [7] "Budget"                    "Gross"
##  [9] "is_budget_great_than_10M"  "is_budget_great_than_50M"
## [11] "is_budget_great_than_100M" "G.Horror"
## [13] "G.Sci-Fi"                  "G.Adventure"
## [15] "G.Comedy"                  "G.Family"
## [17] "G.Crime"                   "G.Music"
## [19] "G.Drama"                   "G.Mystery"
## [21] "G.Thriller"                "G.Romance"
## [23] "G.Sport"                   "G.Fantasy"
## [25] "G.Action"                  "G.Documentary"
## [27] "G.Biography"               "G.History"
## [29] "G.Animation"               "G.Western"
## [31] "G.Musical"                 "G.War"
## [33] "G.Short"                   "G.News"
## [35] "G.Adult"                   "A.win"
## [37] "A.nomination"              "d.c1"
## [39] "d.c2"                      "d.c3"
## [41] "d.c4"                      "d.c5"
## [43] "d.c6"                      "a.c1"
## [45] "a.c2"                      "a.c3"
## [47] "a.c4"                      "a.c5"
## [49] "a.c6"
```

```
#1. Randomly divide the rows into two sets of sizes 5% and 95%.
sets = get_sample_sets(new_df, 0.05)
train_data = sets[[1]]
test_data = sets[[2]]

#2. Use the first set for training and the second for testing.
m1 = lm(Gross~imdbRating+I(imdbRating^2)+I(imdbRating^3)+
            tomatoUserMeter+I(tomatoUserMeter^2)+
            tomatoUserRating+I(tomatoUserRating^2)+I(tomatoUserRating^3)+
            I(imdbRating * tomatoUserRating) +
            I(G.Adventure * is_budget_great_than_50M) +
            I(G.Action * is_budget_great_than_50M) +
            I(G.Animation * is_budget_great_than_50M) +
            I(G.Fantasy * is_budget_great_than_50M) +
            I(d.c1 * a.c1) +
            I(d.c1 * a.c1 * is_budget_great_than_50M) +
            ., train_data)
pred_train = predict(m1, train_data)
pred_test = predict(m1, test_data)

#3. Print out the Root Mean Squared Error (RMSE) on the train and test sets.
print(c(rmse(train_data$Gross, pred_train), rmse(test_data$Gross, pred_test)))
```

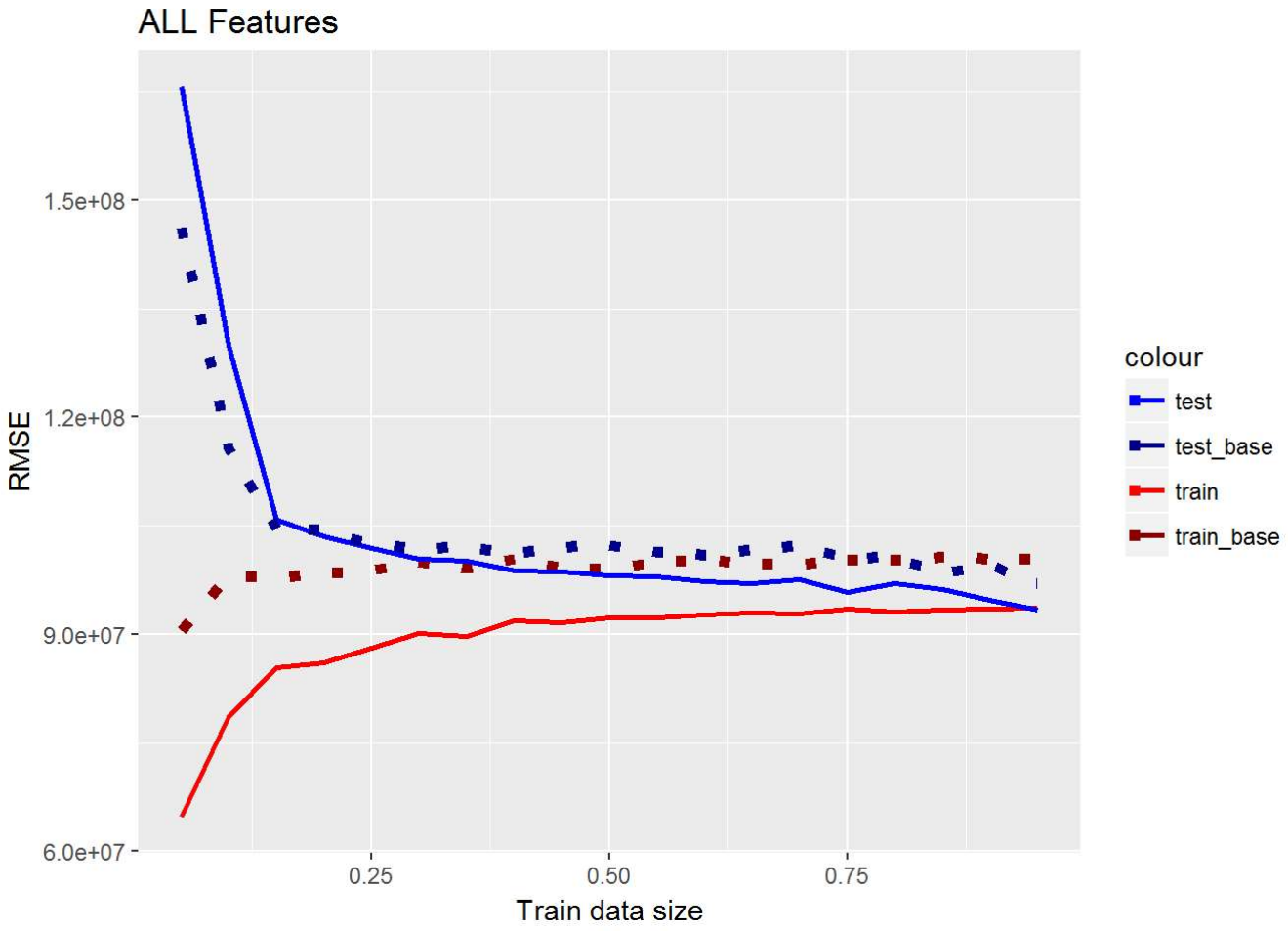
```
## [1]  71032838 397968959
```

```
#4. Repeat the above data partition and model training and evaluation 50 times and average the R
MSE results so the results stabilize.
train_rmse = c()
test_rmse = c()
for(i in 1:100){
    sets = get_sample_sets(new_df, 0.05)
    train_data = sets[[1]]
    test_data = sets[[2]]

    m1 = lm(Gross~imdbRating+I(imdbRating^2)+I(imdbRating^3)+
            tomatoUserMeter+I(tomatoUserMeter^2)+
            tomatoUserRating+I(tomatoUserRating^2)+I(tomatoUserRating^3)+
            I(imdbRating * tomatoUserRating) +
            I(G.Adventure * is_budget_great_than_50M) +
            I(G.Action * is_budget_great_than_50M) +
            I(G.Animation * is_budget_great_than_50M) +
            I(G.Fantasy * is_budget_great_than_50M) +
            I(d.c1 * a.c1) +
            I(d.c1 * a.c1 * is_budget_great_than_50M) +
            ., train_data)
    pred_train = predict(m1, train_data)
    pred_test = predict(m1, test_data)
    train_rmse = c(train_rmse, rmse(train_data$Gross, pred_train))
    test_rmse = c(test_rmse, rmse(test_data$Gross, pred_test))
  }
print(c(mean(train_rmse), mean(test_rmse)))
```

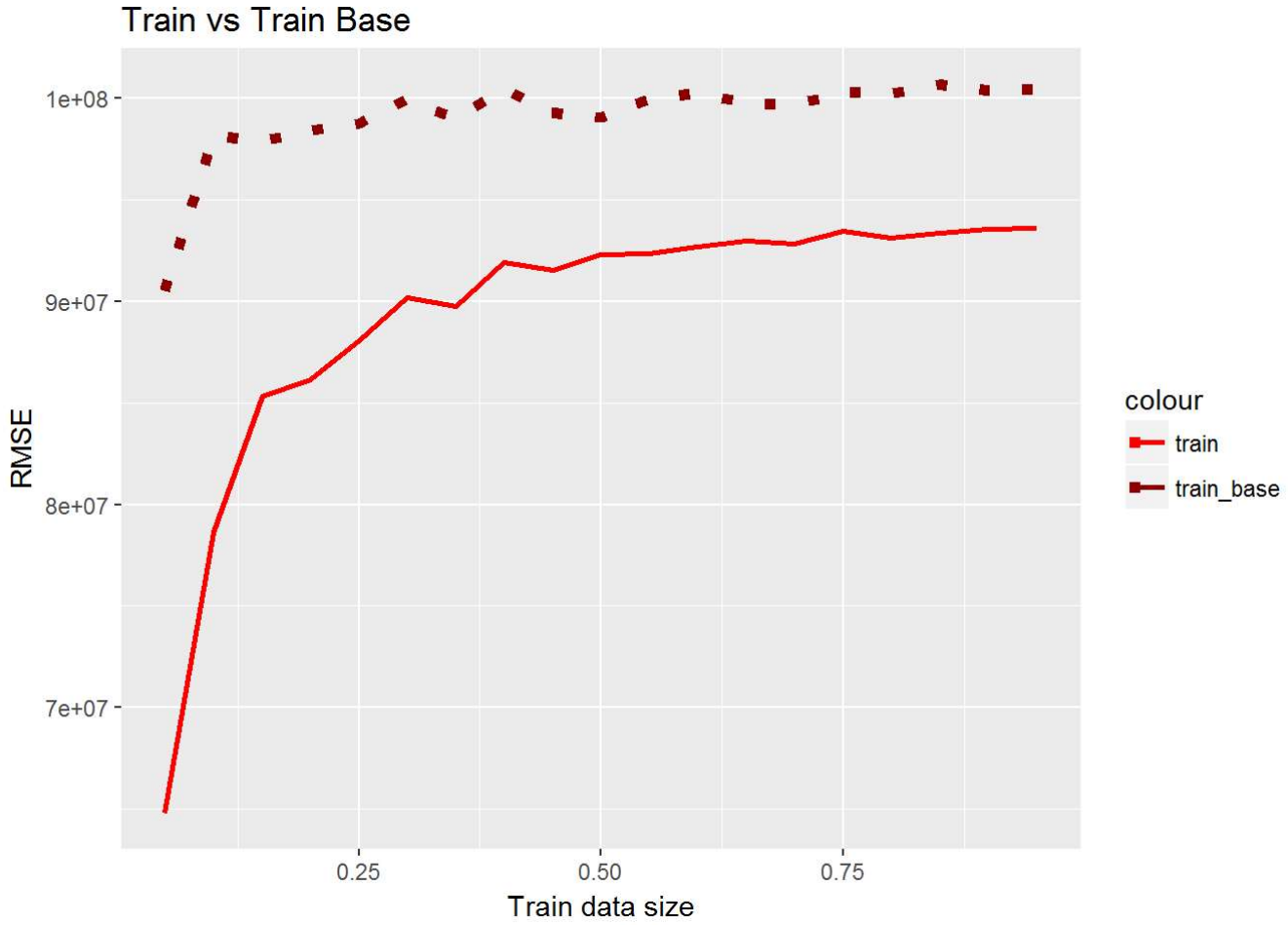
```
## [1]  61774836 179423512
```

```
#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%).
train_rmse = c()
test_rmse = c()
for(i in 1:length(perc)){
  train_rmse_perc = c()
  test_rmse_perc = c()
  for(j in 1:100){
    sets = get_sample_sets(new_df, perc[i])
    train_data = sets[[1]]
    test_data = sets[[2]]
    m1 = lm(Gross~imdbRating+I(imdbRating^2)+I(imdbRating^3)+
            tomatoUserMeter+I(tomatoUserMeter^2)+
            tomatoUserRating+I(tomatoUserRating^2)+I(tomatoUserRating^3)+
            I(imdbRating * tomatoUserRating) +
            I(G.Adventure * is_budget_great_than_50M) +
            I(G.Action * is_budget_great_than_50M) +
            I(G.Animation * is_budget_great_than_50M) +
            I(G.Fantasy * is_budget_great_than_50M) +
            I(d.c1 * a.c1) +
            I(d.c1 * a.c1 * is_budget_great_than_50M) +
            ., train_data)
    pred_train = predict(m1, train_data)
    pred_test = predict(m1, test_data)
    train_rmse_perc = c(train_rmse_perc, rmse(train_data$Gross, pred_train))
    test_rmse_perc = c(test_rmse_perc, rmse(test_data$Gross, pred_test))
  }
  train_rmse = c(train_rmse, mean(train_rmse_perc))
  test_rmse = c(test_rmse, mean(test_rmse_perc))
}

errors$train_rmse_07 = train_rmse
errors$test_rmse_07 = test_rmse
plot1 = ggplot(errors, aes(train_size, train_rmse_07, color = "train")) +
  geom_line(size = 1) +
  geom_line(aes(train_size, train_rmse_01, color = "train_base"), size = 2, linetype="dotted")+
  geom_line(aes(train_size, test_rmse_07, color = "test"), size = 1)+
  geom_line(aes(train_size, test_rmse_01, color = "test_base"), size = 2, linetype="dotted")+
  scale_color_manual(values=c("train"="red", "train_base"="red4",
                              "test"="blue", "test_base"="blue4")) +
  ggtitle("ALL Features") +
  labs(x="Train data size", y="RMSE")

plot2 = ggplot(errors, aes(train_size, train_rmse_07, color = "train")) +
  geom_line(size = 1) +
  geom_line(aes(train_size, train_rmse_01, color = "train_base"), size = 2, linetype="dotted")+
  scale_color_manual(values=c("train"="red", "train_base"="red4"))+
  ggtitle("Train vs Train Base") +
  labs(x="Train data size", y="RMSE")

plot3 = ggplot(errors, aes(train_size, test_rmse_07, color = "test")) +
  geom_line(size = 1) +
  geom_line(aes(train_size, test_rmse_01, color = "test_base"), size = 2, linetype="dotted")+
```

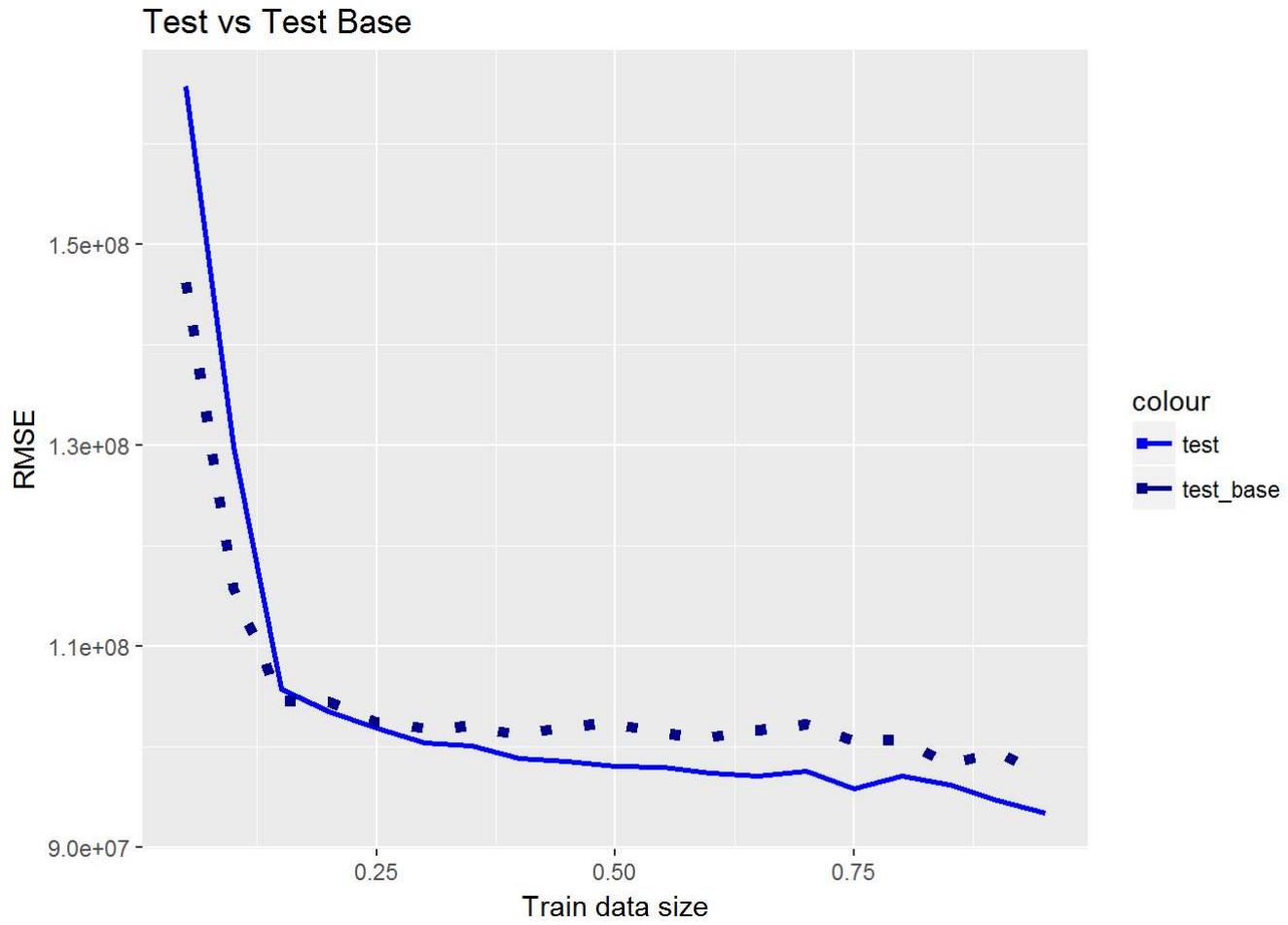
```
  scale_color_manual(values=c("test"="blue", "test_base"="blue4")) +
  ggtitle("Test vs Test Base") +
  labs(x="Train data size", y="RMSE")

print(plot1)
```




```
print(plot2)
```


Train vs Train Base
1e+08
9e+07
8e+07
7e+07
RMSE
0.25
0.50
0.75
Train data size
colour
train
train_base

```
print(plot3)
```



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

A: First, I believe high budget Adventure, Action, Fantasy or Animation movies are popular, thus I add four new features as:(G.Adventure x is_budget_great_than_50M), (G.Action x is_budget_great_than_50M), (G.Animation x is_budget_great_than_50M), (G.Fantasy x is_budget_great_than_50M).

Second, I believe movies with good director and good actors are popular, thus I add a new feature (d.c1 x a.c1).

Third, I believe movies with both good director and actor and high budget are popular, thus I add a new feature (d.c1 x a.c1 x is_budget_great_than_50M).