Project 2: Modeling and Evaluation

CSE6242 - Data and Visual Analytics

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

Name - ***Saad Khan***
GT Account Name - ***skhan315***
GT Email ID - skhan315@gatech.edu

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.

```r
library(ggplot2)
library(GGally)

# Additional packages used

# install.packages("splitstackshape")
# install.packages("reshape2")
# install.packages("gridExtra")
# install.packages("stringr")
# install.packages("VIM")
# install.packages("caret")
# install.packages("car")
# install.packages(c('tm', 'SnowballC', 'wordcloud', 'topicmodels'))

library(splitstackshape)
library(reshape2)
library(gridExtra)
library(stringr)
library(VIM)
library(caret)
library(car)
library(tm)
library(SnowballC)
library(wordcloud)
```

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:

Following packages (not been discussed in class videos) have been utilized as part of this project. All of these packages can simply be installed using the **install.packages()** command. All of the packages have been inlcluded in the 'Load R packages' r-chunk above and their respective usage tutorials are linked below.

- "splitstackshape" package - https://cran.r-project.org/web/packages/splitstackshape/splitstackshape.pdf
- "reshape2" package - https://cran.r-project.org/web/packages/reshape2/reshape2.pdf

- "gridExtra" package - ftp://cran.r-project.org/pub/R/web/packages/gridExtra/gridExtra.pdf
- "stringr" package - https://cran.r-project.org/web/packages/stringr/stringr.pdf
- "VIM" package - https://cran.r-project.org/web/packages/VIM/VIM.pdf
- "caret" package - https://cran.r-project.org/web/packages/caret/caret.pdf
- "car" pacakge - https://cran.r-project.org/web/packages/car/car.pdf
- "tm" package - https://cran.r-project.org/web/packages/tm/vignettes/tm.pdf
- "SnowballC" package - https://cran.r-project.org/web/packages/SnowballC/SnowballC.pdf
- "wordcloud" package - https://cran.r-project.org/web/packages/wordcloud/wordcloud.pdf
- "topicmodels" package - https://cran.r-project.org/web/packages/topicmodels/vignettes/topicmodels.pdf

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") # subsetting the data frame to retain only movies.

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

```
## Dataset has 40000 rows and 39 columns
```

After only retaining observation pertaining to movies, the dataset is left with the following dimensions:

Total Rows : 40000

Total Columns : 39

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 = df[!(is.na(df$Gross)), ] # subsetting the data frame to retain only rows with valid 'Gross' values


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

```
## Dataset has 4558 rows and 39 columns
```

```r
#write.csv(df, file='movies_gross_new.csv')
```

Further reduction in the observation in order to keep only valid 'Gross' rows, the dataset is left with the following dimensions:

Total Rows : 4558

Total Columns : 39

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

# subsetting the data frame to retain only movies after the year 2000. I used the Year column for this

pre_3_df = df # Assigning to an intermediate data frame

pre_3_df = subset(pre_3_df, (pre_3_df$Year >= 2000 ))

df = pre_3_df

#df = subset(df, select = -c(Released_Year) ) # now excluding the new column from the data frame

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

```
## Dataset has 3332 rows and 39 columns
```

```r
# write.csv(df, file='movies_gross_2000.csv')
```

Excluding movies prior to 2000 has left the dataset with the following dimensions:

Total Rows : 3332

Total Columns : 39

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

pre_4_df = df # Assigning to an intermediate data frame


# Creating 4 new columns for computation

pre_4_df$Released_Year = as.numeric(substring(df$Released,1,4))

pre_4_df$Released_Month = as.numeric(substring(df$Released,6,7))
```

```r
pre_4_df$Rel_Year_Diff = pre_4_df$Year - pre_4_df$Released_Year

pre_4_df$Date_Rel_Year_Diff = pre_4_df$Date - pre_4_df$Released_Year

pre_4_df = subset(pre_4_df, (pre_4_df$Rel_Year_Diff == -1 &
                               pre_4_df$Released_Month <= 3) |
                    pre_4_df$Rel_Year_Diff == 0 |
                    (pre_4_df$Date_Rel_Year_Diff == -1 &
                       pre_4_df$Released_Month <= 6))



df = pre_4_df

# now excluding the new columns from the data frame
df = subset(df, select = -c(Rel_Year_Diff,Released_Year,
                            Released_Month, Date_Rel_Year_Diff) )



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

```
## Dataset has 2999 rows and 39 columns
```

```r
cat("Rows removed : ", round((1 - (2999/3332)) * 100,3), "%")
```

```
## Rows removed :  9.994 %
```

Removal Process

For ease with the removal process, I first generated 4 new variables, which are as follows:

- **'Released_Year' :** a numeric version of the 'Released' variable only containing the year part.
- **'Released_Month' :** derived from 'Released', only containing the month part.
- **'Rel_Year_Diff' :** difference of 'Year' and 'Released_Year' variables.
- **'Date_Rel_Year_Diff' :** difference of 'Date' and 'Released_Year' variables.

Second thing was to keep track of the 'Gross' variable. Up until this point, the dataframe contained 3332 rows and to address the requirement of not loosing more than 10% of the rows with valid gross revenue values after attrition, I followed a 3 step process.

- **Step 1 :** First, I only kept the rows where the 'Rel_Year_Diff' was 0, i.e. 'Released_Year' was exactly equal to 'Year' as shown by the subset command (1) below. This way I was loosing a lot of rows.

```r
# subset(pre_4_df, (pre_4_df$Rel_Year_Diff == 0)) Command (1)
```

- **Step 2 :** To accomodate more rows with valid gross values I relaxed the criteria for the exact match and also decided to keep rows where 'Released_Month' was only within 3 months of the 'Year' variable. For example, I retained the rows if the 'Year' value for a movie was 2004 and the 'Released' value was uptil March, 2005. Examples of this are 2 movies released in 2004, Hotel Rwanda and Million Dollar Baby. The imdb pages for both these movies show the dates as 2004 but then underneath that it also mentions the release information date as 2/4/2005 and 1/28/2005 respectively. So there seems to be an overlap of dates in this case. In order to encompass rows such as these, I relaxed the subsetting criteria and instead used command (2) shown below.

```r
# subset(pre_4_df, (Rel_Year_Diff == -1 & Released_Month <= 3) | Rel_Year_Diff == 0) Command (2)
```

- **Step 3 :** To further relax the criteria and accomodate more rows in order to stay within the 10% threshold, I also included the rows where 'Released_Month' was only within 6 months of the 'Date' variable. This was done by using the 'Date_Rel_Year_Diff' created feature as shown by the command

(3) below.

```
# subset(pre_4_df, (pre_4_df$Rel_Year_Diff == -1 & pre_4_df$Released_Month <= 3) |   Command (3)
#                     pre_4_df$Rel_Year_Diff == 0 |
#                     (pre_4_df$Date_Rel_Year_Diff == -1 & pre_4_df$Released_Month <= 6))
```

Combining these commands, I was able to retain more rows with total row count being 2999 and rows with valid gross value which only led to removal of 9.994% of the rows.

Total Rows (before mismatch step) : 3332

Total Rows (after mismatch step) : 2999

Rows Removed : 333 (**Removal : 9.994%**)

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

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

```
## Dataset has 2999 rows and 38 columns
```

After removing the `Domestic_Gross` columns the dataset now has the following dimensions:

Total Rows : 2999

Total Columns : 38

6. Process `Runtime` column

```
# TODO: Replace df$Runtime with a numeric column containing the runtime in minutes

# Function to convert runtime from text to numeric.

runtime_conversion = function(rt){

  hour_part = 0
  min_part = 0

  if(grepl( "h" , rt ) & grepl( "min" , rt )){ # handling both 'h' and 'min'.

    rt_loc <- str_locate_all(rt, "[0-9]+")[[1]]
    converted_rt = as.numeric(str_sub(rt, rt_loc[, "start"], rt_loc[, "end"]))

    hour_part = converted_rt[1]
    min_part = converted_rt[2]
    total_runtime = (hour_part * 60) + min_part
```

```r
    return(total_runtime)

  } else if (grepl( "h" , rt )){ # handling the case where we only have 'h'.

    rt_loc <- str_locate_all(rt, "[0-9]+")[[1]]
    converted_rt = as.numeric(str_sub(rt, rt_loc[, "start"], rt_loc[, "end"]))

    hour_part = converted_rt[1]
    total_runtime = (hour_part * 60)

    return(total_runtime)

  } else if(grepl( "min" , rt )){ # handling the case where we only have 'min'.

    rt_loc <- str_locate_all(rt, "[0-9]+")[[1]]
    converted_rt = as.numeric(str_sub(rt, rt_loc[, "start"], rt_loc[, "end"]))

    min_part = converted_rt[1]
    total_runtime = min_part

    return(total_runtime)

  }  else { return(NA) } # handling the case where we N/A.

}

df$Runtime  = sapply(df$Runtime, function(x) runtime_conversion(x))

cat("Class of Runtime is now '", class(df$Runtime),"'")
```

```
## Class of Runtime is now ' numeric '
```

NOTE : Above method is taken from PR 1, that converts all 'character' runtime values to 'numeric'.

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
```

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

NOTE : Mostly, additional preprocessing is done while working on a spceific 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!)

```r
# TODO: Print the dimensions of the final preprocessed dataset and column names



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

```
## Dataset has 2999 rows and 38 columns
```

```r
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] "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.

Helper Functions for indexing, creating training/test sets, training/testing model and calculating RMSE

```r
# Helper Functions for indexing, creating training/test sets and calculating RMSE

indexing = function(df, set_size){

#Sampling Indexes as per training set percentage

indexes = sample(nrow(df), size = ceiling(set_size * nrow(df)), replace=FALSE)

return(indexes)

}

training_set = function(df, indexes) {

  tr = df[indexes,]

  return(tr)
}
```

```r
testing_set = function(df, indexes) {

  ts = df[-indexes,]

  return(ts)
}

rmse = function(actual_y, pred_y){

  #rmse = sqrt(sum((actual_y - pred_y)^2)/length(actual_y))

  rmse = sqrt(mean((actual_y - pred_y)^2))

  return(rmse)

}

train_test = function(df, iter, set_size){

  TRAIN_RMSE = rep(0,iter)
  TEST_RMSE = rep(0,iter)

  for (i in 1:iter){

  # generating indexes randomly
  set.seed(i)
  indices = indexing(df, set_size)

  # generating training and test sets based on set size
  train_set = training_set(df, indices)
  test_set = testing_set(df, indices)

  # training linear regression model using the training set
  lmlm = lm(Gross ~ ., data = train_set)

  # training and test samples (without labels)
  pred_train_set = train_set
  pred_train_set$Gross = NULL
  pred_test_set = test_set
  pred_test_set$Gross = NULL

  # predicitons based on training and test data
  prediction_train = predict(lmlm, pred_train_set)
  prediction_test = predict(lmlm, pred_test_set)

  # rmse based on training and test data
  rmse_train = rmse(train_set$Gross, prediction_train)
  rmse_test = rmse(test_set$Gross, prediction_test)

  TRAIN_RMSE[i] = rmse_train
  TEST_RMSE[i] = rmse_test

   }
```

```
  return(list(TRAIN_RMSE=mean(TRAIN_RMSE),TEST_RMSE=mean(TEST_RMSE)))

}


movetolast <- function(data, move) {
  data[c(setdiff(names(data), move), move)]
}


df_top_corr = function(df, low_limit, up_limit){

  df_check = as.data.frame(as.table(cor(df)))

  df_check = subset(df_check, Var2 =='Gross')

  df_check = subset(df_check, Freq > low_limit & Freq < up_limit)

  df_check = df[, c(df_check$Var1, unique(df_check$Var2))]

  return(df_check)

}

combine_columns = function(df, start, end){

  return(do.call(paste, c(df[,start:end], sep = "")))
}
```

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.

1.1 Using only numeric variables

1.1.1 Creating a DF containing only numeric variables

```
df_task_1 = df[, c('Year','Runtime', 'imdbRating', 'imdbVotes', 'tomatoMeter',
                   'tomatoRating', 'tomatoReviews', 'tomatoFresh','tomatoRotten',
                   'tomatoUserMeter', 'tomatoUserRating', 'tomatoUserReviews',
                   'Budget','Date','Gross')]

cat("Dataset has", dim(df_task_1)[1], "rows and", dim(df_task_1)[2], "columns", end="\n", file="")
```

```
## Dataset has 2999 rows and 15 columns
```

```r
colnames(df_task_1)
```

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

1.1.2 Preprocessing [handling columns with missing data]

```r
aggr(df_task_1, col=c('navyblue','red'), numbers=TRUE, sortVars=TRUE, labels=names(df_task_1), cex.axis=
```

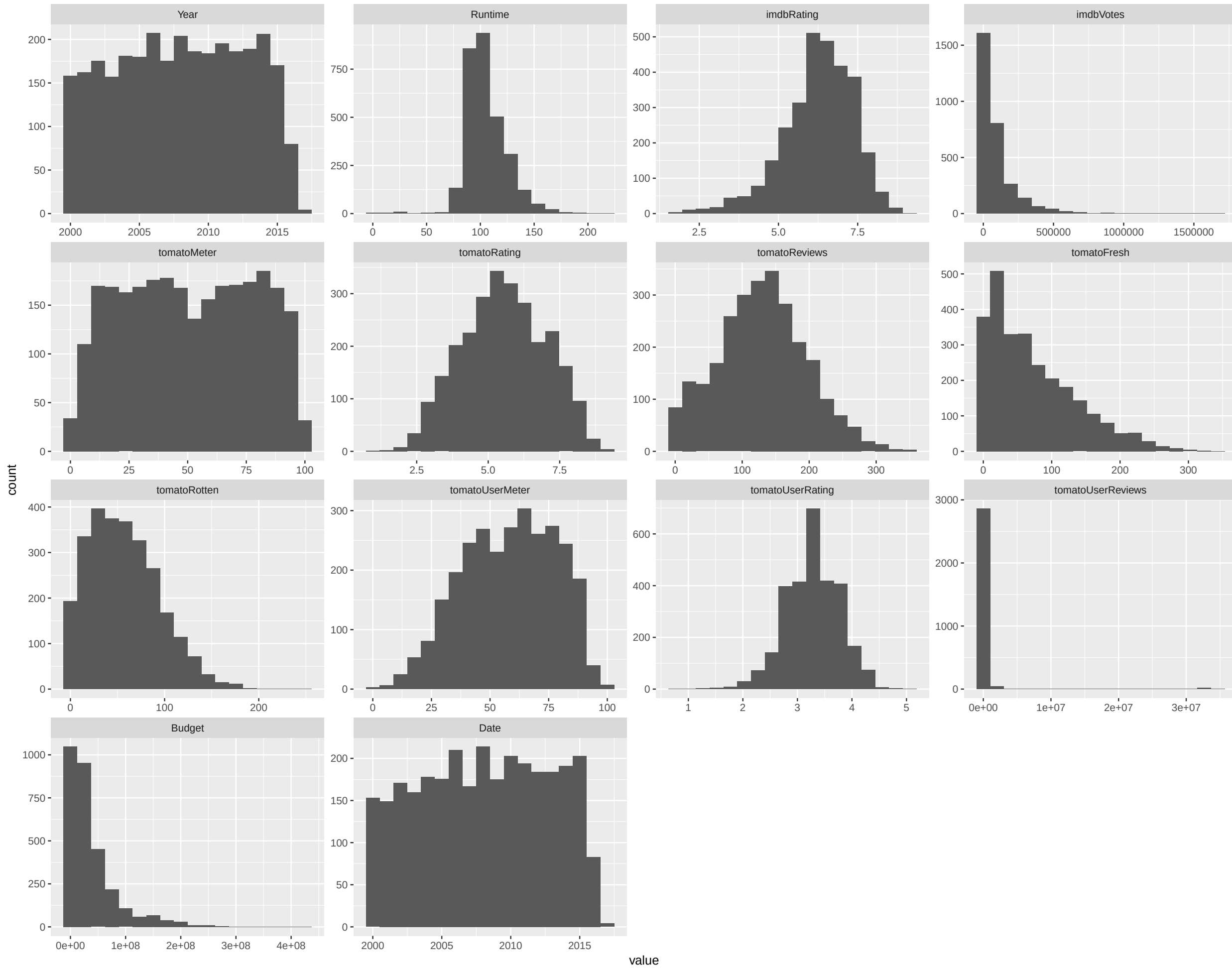
```
##
##  Missings in variables:
##           Variable Count
##            Runtime    24
##         imdbRating    26
##          imdbVotes    26
##        tomatoMeter   326
##       tomatoRating   326
##      tomatoReviews   326
##        tomatoFresh   326
##       tomatoRotten   326
##    tomatoUserMeter   153
##   tomatoUserRating   152
##  tomatoUserReviews    62
```

Comment : Using VIM package, one can see the number of missing values for the numberical variables. Now we plot the distributions of the variables using ggpairs.

1.1.3 Visualizing data to make sense of how to imputate missing values

```r
# http://stackoverflow.com/questions/13035834/plot-every-column-in-a-data-frame-as-a-histogram-on-one-p

d <- melt(df_task_1[,c(1:14)])
ggplot(d,aes(x = value)) +
    facet_wrap(~variable, scales = "free") +
    geom_histogram(bins = 18) +
  ggtitle("Figure 0 (a) : Distribution of Numeric Variables") +
  theme(plot.title = element_text(face = "bold",color = 'darkblue',
                                  hjust = 0.5))
```

Figure 0 (a) : Distribution of Numeric Variables



Comment : Most of the variables in Figure 0(a) have skewed distributions so I used median imputation in order to fill in the missing values for the columns that had NAs in them.

1.1.4 Preprocessing [handling highly correlated columns]

```r
cor_df_task_1 = na.omit(df_task_1)

cor_matrix = round(cor(cor_df_task_1),3)

# http://stackoverflow.com/questions/7074246/show-correlations-as-an-ordered-list-not-as-a-large-matrix

zdf = as.data.frame(as.table(cor_matrix))

subset(zdf, Freq > 0.9 & Freq < 1.0)
```

```
##                  Var1              Var2  Freq
## 14               Date              Year 0.997
## 66       tomatoRating       tomatoMeter 0.972
## 80        tomatoMeter      tomatoRating 0.972
## 146 tomatoUserRating   tomatoUserMeter 0.912
## 160  tomatoUserMeter tomatoUserRating 0.912
## 196              Year              Date 0.997
```

Comment : This code chunk shows the 3 most correlated numeric variables, i.e. corr. coeff. greater than 0.9. These are as follows:

- **'Year' and 'Date'** : r^2 = 0.997
- **'tomatoRating' and 'tomatoUserRating'** : r^2 = 0.972
- **'tomatoUserRating' and 'tomatoUserMeter'** : r^2 = 0.912

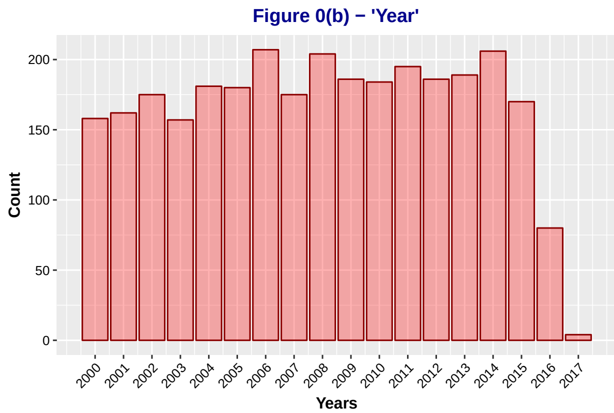
The following few sections discuss the removal process, i.e. which of these were kept and which ones were excluded from the model.

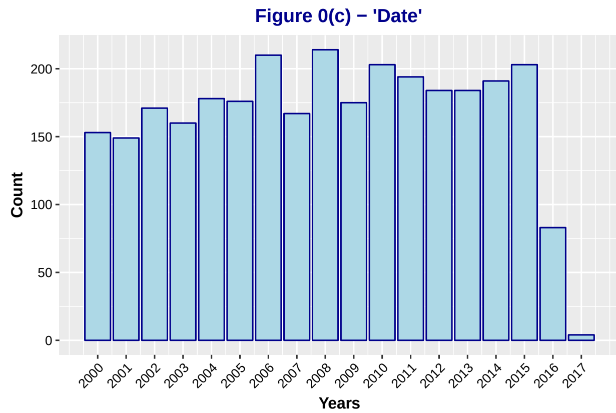
1.1.5 'Year' and 'Date'

```r
year_task_1 = ggplot(data = df_task_1) +
  geom_bar(aes(x = Year), fill = "red", alpha = 0.3, colour = "darkred") +
  scale_x_continuous(breaks = seq(2000, 2017,1)) +
  xlab('Years') + ylab('Count') +                  # Setting axes labels

  ggtitle("Figure 0(b) - 'Year'") +     # Setting the title

  theme(plot.title = element_text(face = "bold",color = 'darkblue',
                                  hjust = 0.5)) +
  theme(axis.text = element_text(colour = "black")) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  theme(axis.title.x = element_text(face = "bold", color='black'),
        axis.title.y = element_text(face = "bold", color='black'))

date_task_1 = ggplot(data = df_task_1) +
  geom_bar(aes(x = Date), fill = "lightblue", colour = "darkblue") +
  scale_x_continuous(breaks = seq(2000, 2017,1)) +
  xlab('Years') + ylab('Count') +                  # Setting axes labels

  ggtitle("Figure 0(c) - 'Date'") +     # Setting the title

  theme(plot.title = element_text(face = "bold",color = 'darkblue',
                                  hjust = 0.5)) +
  theme(axis.text = element_text(colour = "black")) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  theme(axis.title.x = element_text(face = "bold", color='black'),
        axis.title.y = element_text(face = "bold", color='black'))

cor(df_task_1$Year, df_task_1$Date)
```

```
## [1] 0.9952786
```

```r
grid.arrange(year_task_1, date_task_1, nrow = 1)
```





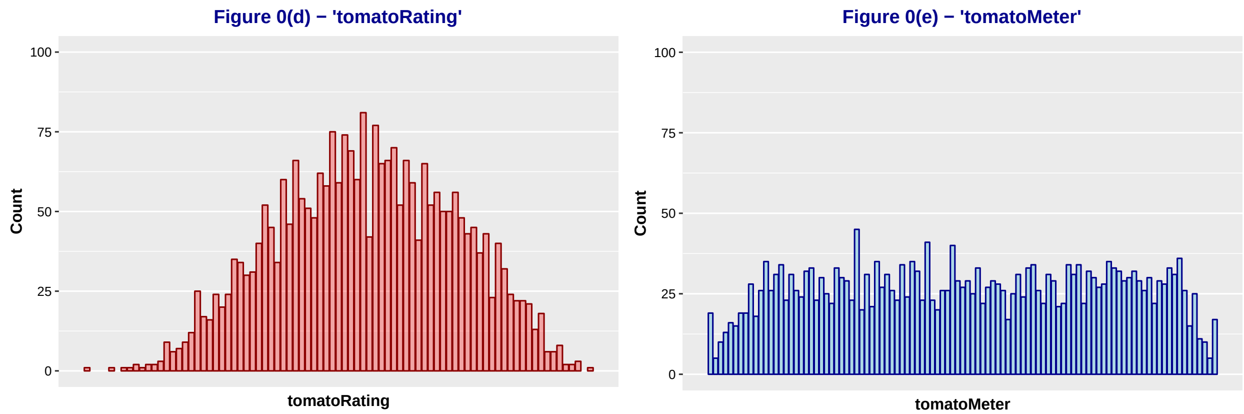
Observation : ‘Year’ and ‘Date’ columns are highly correlated as evident by correlation coeff. of 0.995. As I used ‘Year’ to remove movies prior to the year 2000 so I kept ‘Year’ as numerical variable and removed the ‘Date’ column from the data set.

1.1.6 ‘tomatoRating’ and ‘tomatoMeter’

```
tomatoRating_task_1 = ggplot(data = df_task_1) +
  geom_bar(aes(x = tomatoRating), fill = "red", alpha = 0.3, colour = "darkred") +
  scale_x_continuous(breaks = seq(2000, 2017,1)) +
  xlab('tomatoRating') + ylab('Count') +                   # Setting axes labels

  ggtitle("Figure 0(d) - 'tomatoRating'") +      # Setting the title

  theme(plot.title = element_text(face = "bold",color = 'darkblue',
                                  hjust = 0.5)) +
  theme(axis.text = element_text(colour = "black")) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  theme(axis.title.x = element_text(face = "bold", color='black'),
        axis.title.y = element_text(face = "bold", color='black')) +
  ylim(0,100)

tomatoMeter_task_1 = ggplot(data = df_task_1) +
  geom_bar(aes(x = tomatoMeter), fill = "lightblue", colour = "darkblue") +
  scale_x_continuous(breaks = seq(2000, 2017,1)) +
  xlab('tomatoMeter') + ylab('Count') +                    # Setting axes labels

  ggtitle("Figure 0(e) - 'tomatoMeter'") +      # Setting the title

  theme(plot.title = element_text(face = "bold",color = 'darkblue',
                                  hjust = 0.5)) +
  theme(axis.text = element_text(colour = "black")) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  theme(axis.title.x = element_text(face = "bold", color='black'),
        axis.title.y = element_text(face = "bold", color='black')) +
  ylim(0,100)

df_task_1_check = subset(df_task_1, select = c(tomatoRating, tomatoMeter))
df_task_1_check = na.omit(df_task_1_check)

cor(df_task_1_check)[2]
```

```
## [1] 0.9710884
```

```r
grid.arrange(tomatoRating_task_1, tomatoMeter_task_1, nrow = 1)
```



Observation : The correlation coeff. value of 0.971 for 'tomatoRating' and 'tomatoMeter' shows that these columns are highly correlated and it can also be seen that 'tomatoRating' has close to normal distribution where as 'tomatoMeter' has close to a uniform distribution. Also, as per the piazza post 757 'tomatoMeter' is derived from 'tomatoRotten' and 'tomatoFresh', so I excluded 'tomatoMeter' from the model.

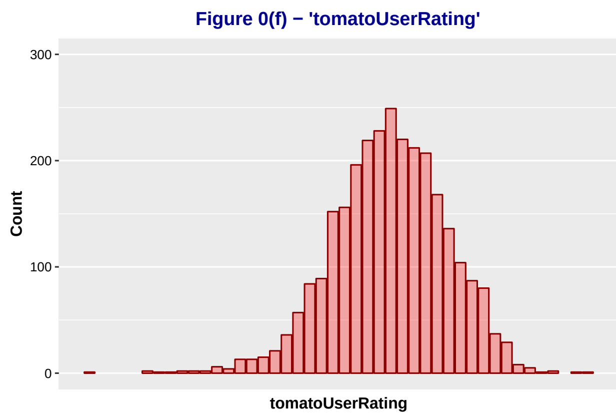
1.1.7 'tomatoUserRating' and 'tomatoUserMeter'

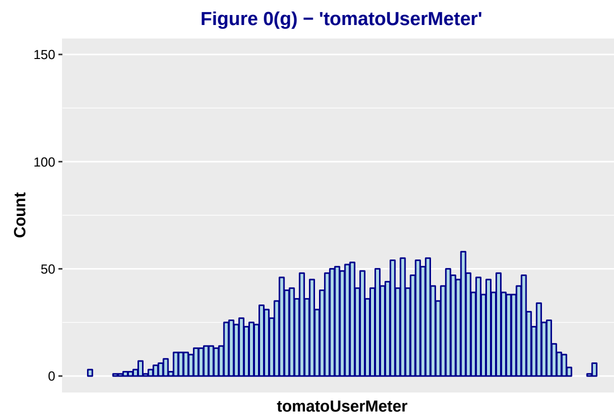
```r
tomatoUserRating_task_1 = ggplot(data = df_task_1) +
  geom_bar(aes(x = tomatoUserRating), fill = "red", alpha = 0.3, colour = "darkred") +
  scale_x_continuous(breaks = seq(2000, 2017,1)) +
  xlab('tomatoUserRating') + ylab('Count') +                     # Setting axes labels

  ggtitle("Figure 0(f) - 'tomatoUserRating'") +      # Setting the title

  theme(plot.title = element_text(face = "bold",color = 'darkblue',
                                  hjust = 0.5)) +
  theme(axis.text = element_text(colour = "black")) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  theme(axis.title.x = element_text(face = "bold", color='black'),
        axis.title.y = element_text(face = "bold", color='black')) +
  ylim(0,300)

tomatoUserMeter_task_1 = ggplot(data = df_task_1) +
  geom_bar(aes(x = tomatoUserMeter), fill = "lightblue", colour = "darkblue") +
  scale_x_continuous(breaks = seq(2000, 2017,1)) +
  xlab('tomatoUserMeter') + ylab('Count') +                     # Setting axes labels

  ggtitle("Figure 0(g) - 'tomatoUserMeter'") +      # Setting the title

  theme(plot.title = element_text(face = "bold",color = 'darkblue',
                                  hjust = 0.5)) +
  theme(axis.text = element_text(colour = "black")) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  theme(axis.title.x = element_text(face = "bold", color='black'),
        axis.title.y = element_text(face = "bold", color='black')) +
  ylim(0,150)
```

```
df_task_1_check_2 = subset(df_task_1, select = c(tomatoUserRating, tomatoUserMeter))
df_task_1_check_2 = na.omit(df_task_1_check_2)

cor(df_task_1_check_2)[2]
```

```
## [1] 0.9138545
```

```
grid.arrange(tomatoUserRating_task_1, tomatoUserMeter_task_1, nrow = 1)
```





Observation : Again, for 'tomatoUserRating' and 'tomatoUserMeter', high correlation coeff. of 0.912 suggested to use only one of these columns for modeling. Distribution for 'tomatoUserRating' was close to normal while 'tomatoUserMeter' was more spread out so I decided to retain 'tomatoUserRating'.

1.1.8 Creating a DF containing only 11 numeric variables (applying median imputation to replace NAs)

```
# https://stackoverflow.com/questions/43123462/how-to-obtain-rmse-out-of-lm-result

# http://stats.stackexchange.com/questions/107643/how-to-get-the-value-of-mean-
# squared-error-in-a-linear-regression-in-r

# http://stats.stackexchange.com/questions/110999/r-confused-on-residual-terminology

# http://stackoverflow.com/questions/9026383/r-numeric-envir-arg-not-of-length-one-in-predict

# https://ragrawal.wordpress.com/2012/01/14/dividing-data-into-training-and-testing-dataset-in-r/

# http://stackoverflow.com/questions/20643166/set-a-data-frame-column-as-the-
# index-of-r-data-frame-object


df_task_1 = df[, c('Year','Runtime', 'imdbRating', 'imdbVotes', 'tomatoRating',
                   'tomatoReviews', 'tomatoFresh','tomatoRotten', 'tomatoUserRating',
                   'tomatoUserReviews', 'Budget','Gross')]


for(i in 1:ncol(df_task_1)){
  df_task_1[is.na(df_task_1[,i]), i] <- median(df_task_1[,i], na.rm = TRUE)
}
```

```r
cat("Dataset has", dim(df_task_1)[1], "rows and", dim(df_task_1)[2], "columns", end="\n", file="")
```

```
## Dataset has 2999 rows and 12 columns
```

```r
colnames(df_task_1)
```

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

Comment : After deciding on which numeric features to include for Task 1, I then replaced the missing values for each column with median value for that column. This left me with the dataset having the following dimensions:

Total Rows : 2999

Total Columns : 12 (including 'Gross')

The numeric columns that were exlcuded are as follows:

- **Date**
- **tomatoMeter**
- **tomatoUserMeter**

1.2 Evaluation Strategy - Task 1 [Original Numeric variables only]

NOTE : Throughout the project, I have followed 2 step strategy for evaluating the model after each major change.

- **1.** I will computed Train/Test RMSE only based on Evalaution Strategy step 1, 2, 3 and 4 (as shown in the r-chunk below), i.e only reporting Train / Test RMSE for (a) training data 5% and testing data 95% and (b) training data 95% and testing data 5% . This way I will able to compare the initial and final RMSE values as it would be difficult to read off exact RMSE values from the plots.

- **2.** For the 2nd part I will created the actual RMSE curves for all 19 training / test size buckets (shown in subsequent r-chunk) displaying RMSE for both training and test set for a paritcular task or sub-task.

1.2.1 [RMSE values - Original Numeric variables only]

- **No. of Iterations :** 1000
- **Scenario 1 :** 5% Train data / 95% Test data
- **Scenario 2 :** 95% Train data / 5% Test data

```r
# TODO: Build & evaluate model 1 (numeric variables only)



rmse_1.4 = train_test(df_task_1, 1000, 0.05)

paste("Multiple run 5% Train data / 95% Test data - Average Training RMSE : ",
      round(rmse_1.4$TRAIN_RMSE,0))
```

```
## [1] "Multiple run 5% Train data / 95% Test data - Average Training RMSE :  85706183"
```

```r
paste("Multiple run 5% Train data / 95% Test data - Average Test RMSE     : ",
      round(rmse_1.4$TEST_RMSE,0))
```

```
## [1] "Multiple run 5% Train data / 95% Test data - Average Test RMSE     :  145485544"
```

```r
rmse_1.4 = train_test(df_task_1, 1000, 0.95)

print("")
```

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

```r
paste("Multiple run 95% Train data / 5% Test data - Average Training RMSE : ",
      round(rmse_1.4$TRAIN_RMSE,0))
```

```
## [1] "Multiple run 95% Train data / 5% Test data - Average Training RMSE :  95576815"
```

```r
paste("Multiple run 95% Train data / 5% Test data - Average Test RMSE     : ",
      round(rmse_1.4$TEST_RMSE,0))
```

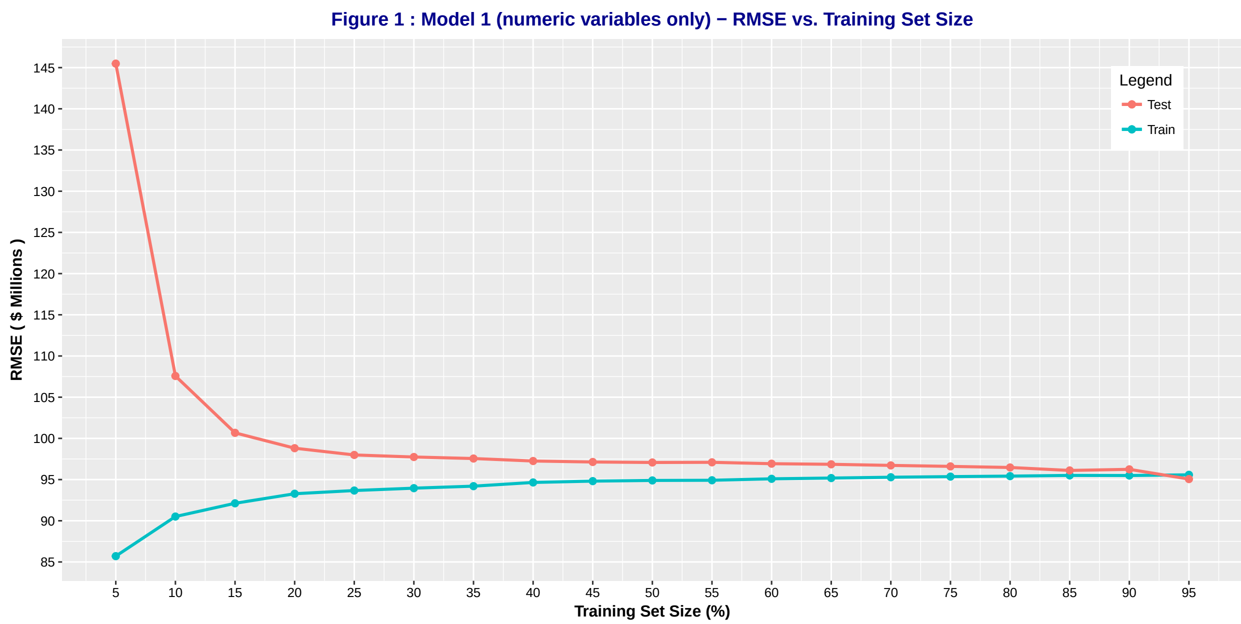
```
## [1] "Multiple run 95% Train data / 5% Test data - Average Test RMSE     :  95061453"
```

1.2.2 [RMSE Curves - Original Numeric variables only]

- **No. of Iterations :** 1000

```r
set_sizes = seq(0.05, 0.95, by = 0.05)

TRAIN_RMSE_1.5 = rep(0,19)
TEST_RMSE_1.5  = rep(0,19)

for (s in 1:length(set_sizes)){
  #print (s)

  rmse_1.5 = train_test(df_task_1, 1000, set_sizes[s])

  TRAIN_RMSE_1.5[s] = rmse_1.5$TRAIN_RMSE
  TEST_RMSE_1.5[s] = rmse_1.5$TEST_RMSE

}

#TRAIN_RMSE_1.5
#TEST_RMSE_1.5

RMSE_task_1 = data.frame(TrainSet_Size = set_sizes * 100,
                                Train = TRAIN_RMSE_1.5,
                                Test = TEST_RMSE_1.5)

ggplot(data=RMSE_task_1) +                                                   # Initializing the plo

  geom_line(size = 1, aes(x = TrainSet_Size, y = Train, color = "Train")) +      # Line Plot

  geom_line(size = 1, aes(x = TrainSet_Size, y = Test, color = "Test")) +        # Line Plot

  geom_point(size = 2, aes(x = TrainSet_Size, y = Train, color = "Train")) +     # Point

  geom_point(size = 2, aes(x = TrainSet_Size, y = Test, color = "Test")) +       # Point

  labs(colour="Legend") +                                                    # Legend

  theme(legend.position = c(0.95, 0.95), legend.justification = c(1, 1)) +       # Legend position
```

```
scale_y_continuous(breaks = seq(85000000, 155000000, 5000000),
                   labels = seq(85000000, 155000000, 5000000)/1000000) +       # Setting y-axis scale

scale_x_continuous(breaks = seq(0, 100, 5)) +                                  # Setting x-axis scale

xlab('Training Set Size (%)') + ylab('RMSE ( $ Millions )') +                  # Setting axes labels

ggtitle("Figure 1 : Model 1 (numeric variables only) - RMSE vs. Training Set Size") + # Setting the t

theme(plot.title = element_text(face = "bold",color = 'darkblue',              # Setting various them
                                hjust = 0.5)) +
theme(axis.text = element_text(colour = "black")) +
theme(axis.title.x = element_text(face = "bold", color='black'),
      axis.title.y = element_text(face = "bold", color='black')) +
theme(legend.key = element_rect(fill = "white"))
```



Observation : Figure 1 shows the RMSE curves only for the (11) numeric variables used for task 1 in their original forms. Test RMSE starts with a high value of around 145M for 95% test data and then slowly settles down to around 95M for smaller test set size beyond, i.e. less than 70%. Similarly, training set RMSE starts at the lowest value of around 85M and settles to around 95M for higher training set sizes.

```
paste(colnames(df_task_1))
```

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

Q: List all the numeric variables you used.

A: There were 11 numeric variables used to predict 'Gross' for this task.

- Year
- Runtime
- imdbRating

- imdbVotes
- tomatoRating
- tomatoReviews
- tomatoFresh
- tomatoRotten
- tomatoUserRating
- tomatoUserReviews
- Budget

There were 14 numeric variables out of which 3 of these: 'Date', 'tomatoMeter' and 'tomatoUserMeter' were excluded because of having high correlation with 'Year', 'tomatoRating' and 'tomatoUserRating' respectively and rest were used to build the model for this task. Following is the corr. coeff. of all thes 11 variables against 'Gross'.

```
t(cor(df_task_1$Gross, subset(df_task_1, select = -c(Gross))))
```

```
##                          [,1]
## Year              0.09173318
## Runtime           0.32419249
## imdbRating        0.24790390
## imdbVotes         0.68702224
## tomatoRating      0.20862274
## tomatoReviews     0.54547288
## tomatoFresh       0.46341497
## tomatoRotten      0.19601099
## tomatoUserRating  0.27935942
## tomatoUserReviews 0.21643324
## Budget            0.77686777
```

Here it can be seen that all the variables used to build the model have positive correlation with 'Gross' where 'Budget' is highly correlated having the highest r^2 of 0.77 while 'Year' with an r^2 of 0.092 is the least correlated.

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`).

Helper Function for performing numeric transformations only

19 different numeric transforms were tried for each numeric feature and the one providing best correlation with 'Gross' was chosen.

```
power_transform = function(Gross, Feature){

  transform_check = rep(0, 20)

  transform = list("Original" = round(cor(Gross, Feature), 6),
                         "power(1/9)" = round(cor(Gross, Feature^(1/9)), 6),
                         "power(1/4)" = round(cor(Gross, Feature^0.25), 6),
                         "power(1/3)" = round(cor(Gross, Feature^(1/3)), 6),
                         "power(1/2)" = round(cor(Gross, Feature^0.5), 6),
                         "power(2)" = round(cor(Gross, Feature^2), 6),
                         "power(3)" = round(cor(Gross, Feature^3), 6),
                         "power(4)" = round(cor(Gross, Feature^4), 6),
```

```
                         "power(5)" = round(cor(Gross, Feature^5), 6),
                         "power(6)" = round(cor(Gross, Feature^6), 6),
                         "power(7)" = round(cor(Gross, Feature^7), 6),
                         "power(8)" = round(cor(Gross, Feature^8), 6),
                         "power(9)" = round(cor(Gross, Feature^9), 6),
                         "power(10)" = round(cor(Gross, Feature^10), 6),
                         "Log" = round(cor(Gross, log(Feature + 1)), 6),
                         "Exp" = round(cor(Gross, exp(Feature)), 6),
                         "Log10" = round(cor(Gross, log10(Feature + 1)), 6),
                         "Log2" = round(cor(Gross, log2(Feature + 1)), 6),
                         "Sin" = round(cor(Gross, sin(Feature)), 6),
                         "1/x" = round(cor(Gross, 1/(Feature)), 6))

  for (i in 1:20) {
    transform_check[i] = transform[[i]]
  }

  return(paste("Best numeric transform =",
               names(transform[which.max(transform_check)]),
               "that gives r^2 of",
               transform[[which.max(transform_check)]], "with 'Gross'"))
}
```

2.1 Applying only Transforms

2.1.1 Checking which transformation is the best

```
numeric_features = c('Runtime', 'imdbRating', 'imdbVotes', 'tomatoRating',
                   'tomatoReviews', 'tomatoFresh','tomatoRotten', 'tomatoUserRating',
                   'tomatoUserReviews', 'Budget')

for (i in 1:length(numeric_features)){

  print(paste(numeric_features[i], power_transform(df_task_1$Gross,
                                                    df_task_1[[numeric_features[i]]])))
}
```

```
## [1] "Runtime Best numeric transform = power(3) that gives r^2 of 0.350134 with 'Gross'"
## [1] "imdbRating Best numeric transform = Exp that gives r^2 of 0.29085 with 'Gross'"
## [1] "imdbVotes Best numeric transform = Original that gives r^2 of 0.687022 with 'Gross'"
## [1] "tomatoRating Best numeric transform = power(4) that gives r^2 of 0.222436 with 'Gross'"
## [1] "tomatoReviews Best numeric transform = power(3) that gives r^2 of 0.633653 with 'Gross'"
## [1] "tomatoFresh Best numeric transform = power(2) that gives r^2 of 0.4876 with 'Gross'"
## [1] "tomatoRotten Best numeric transform = power(3) that gives r^2 of 0.229463 with 'Gross'"
## [1] "tomatoUserRating Best numeric transform = power(4) that gives r^2 of 0.291313 with 'Gross'"
## [1] "tomatoUserReviews Best numeric transform = power(1/4) that gives r^2 of 0.484947 with 'Gross'"
## [1] "Budget Best numeric transform = Original that gives r^2 of 0.776868 with 'Gross'"
```

2.1.2 Applying 'numeric transformations only' to the data frame

The data frame created after applying numeric transforms, only contains the tranformed variables based on the above recommendations. The only features which were included in their original form were 'imdbVotes' and 'Budget' as no numeric transform seemed to have improved corr. coeff. with 'Gross' for these 2 features.

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

df_task_2_pt = data.frame(Year = df_task_1$Year,
                          Runtime = (df_task_1$Runtime)^3,
                          imdbRating = exp(df_task_1$imdbRating),
                          imdbVotes = df_task_1$imdbVotes,
                          tomatoRating = (df_task_1$tomatoRating)^4,
                          tomatoReviews = (df_task_1$tomatoReviews)^3,
                          tomatoFresh = (df_task_1$tomatoFresh)^2,
                          tomatoRotten = (df_task_1$tomatoRotten)^3,
                          tomatoUserRating = (df_task_1$tomatoUserRating)^4,
                          tomatoUserReviews = (df_task_1$tomatoUserReviews)^0.25,
                          Budget = df_task_1$Budget,
                          Gross = df_task_1$Gross)
```

2.2 Evaluation Strategy - Task 2 [Only Numeric Transforms]

2.2.1 [RMSE values - Only Numeric Transforms]

- **No. of Iterations :** 1000
- **Scenario 1 :** 5% Train data / 95% Test data
- **Scenario 2 :** 95% Train data / 5% Test data

```r
rmse_2.4_pt  = train_test(df_task_2_pt , 1000, 0.05)

paste("Multiple run 5% Train data / 95% Test data - Average Training RMSE : ",
      round(rmse_2.4_pt $TRAIN_RMSE,0))
```

```
## [1] "Multiple run 5% Train data / 95% Test data - Average Training RMSE :  83257858"
```

```r
paste("Multiple run 5% Train data / 95% Test data - Average Test RMSE     : ",
      round(rmse_2.4_pt $TEST_RMSE,0))
```

```
## [1] "Multiple run 5% Train data / 95% Test data - Average Test RMSE     :  106266990"
```

```r
rmse_2.4_pt  = train_test(df_task_2_pt , 1000, 0.95)

print("")
```

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

```r
paste("Multiple run 95% Train data / 5% Test data - Average Training RMSE : ",
      round(rmse_2.4_pt $TRAIN_RMSE,0))
```

```
## [1] "Multiple run 95% Train data / 5% Test data - Average Training RMSE :  95060934"
```

```r
paste("Multiple run 95% Train data / 5% Test data - Average Test RMSE     : ",
      round(rmse_2.4_pt $TEST_RMSE,0))
```

```
## [1] "Multiple run 95% Train data / 5% Test data - Average Test RMSE     :  94808912"
```

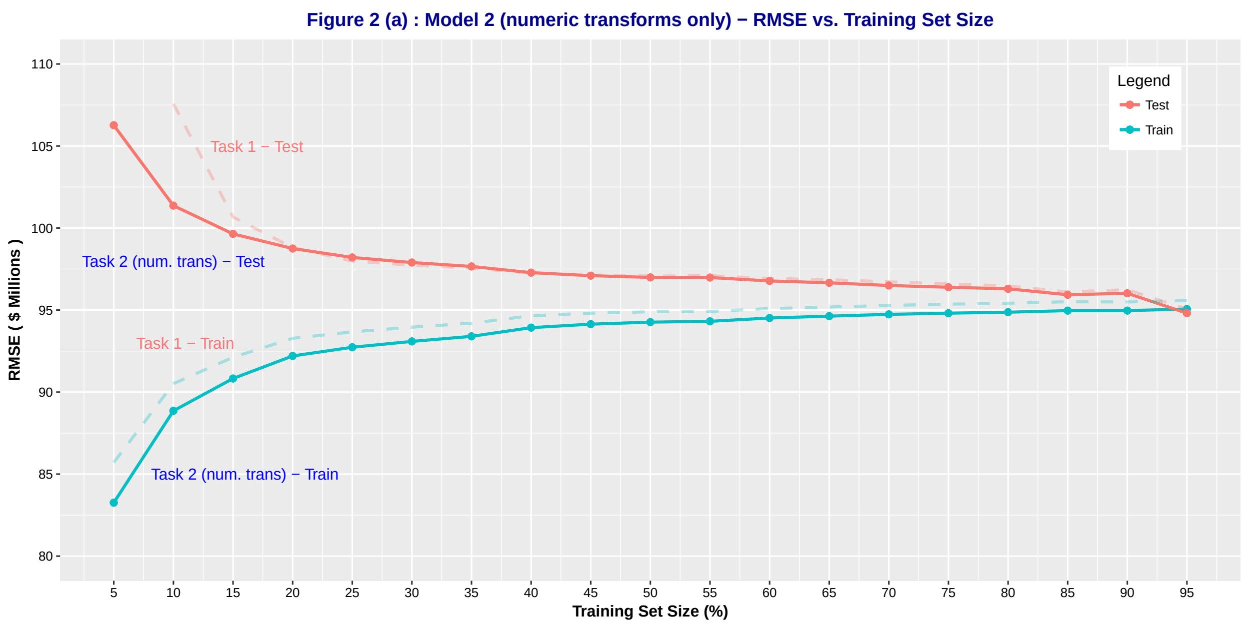
2.2.2 [RMSE Curves - Only Numeric Transforms]

- **No. of Iterations :** 1000

```r
set_sizes = seq(0.05, 0.95, by = 0.05)
```

```r
TRAIN_RMSE_2.5_pt = rep(0,19)
TEST_RMSE_2.5_pt  = rep(0,19)

for (s in 1:length(set_sizes)){

  #print (s)

  rmse_2.5_pt = train_test(df_task_2_pt, 1000, set_sizes[s])

  TRAIN_RMSE_2.5_pt[s] = rmse_2.5_pt$TRAIN_RMSE
  TEST_RMSE_2.5_pt[s] = rmse_2.5_pt$TEST_RMSE

  }

#TRAIN_RMSE_2.5_pt
#TEST_RMSE_2.5_pt

RMSE_task_2_pt = data.frame(TrainSet_Size = set_sizes * 100,
                            Train_2_pt = TRAIN_RMSE_2.5_pt,
                            Test_2_pt = TEST_RMSE_2.5_pt,
                            Train_1 = TRAIN_RMSE_1.5,
                            Test_1 = TEST_RMSE_1.5)

ggplot(data=RMSE_task_2_pt) +                                                # Initializing the pl

  geom_line(size = 1, aes(x = TrainSet_Size, y = Train_2_pt, color = "Train")) +  # Line Plot

  geom_line(size = 1, aes(x = TrainSet_Size, y = Test_2_pt, color = "Test")) +    # Line Plot

  geom_point(size = 2, aes(x = TrainSet_Size, y = Train_2_pt, color = "Train")) + # Point

  geom_point(size = 2, aes(x = TrainSet_Size, y = Test_2_pt, color = "Test")) +   # Point

  geom_line(size = 1, linetype = 2, alpha = 0.3,
          aes(x = TrainSet_Size,y = Train_1, color = "Train")) +               # Line Plot

  geom_line(size = 1, linetype = 2, alpha = 0.3,
          aes(x = TrainSet_Size, y = Test_1, color = "Test")) +               # Line Plot

  labs(colour="Legend") +                                                    # Legend

  theme(legend.position = c(0.95, 0.95), legend.justification = c(1, 1)) +      # Legend position

  scale_y_continuous(breaks = seq(80000000, 110000000, 5000000),
                     labels = seq(80000000, 110000000, 5000000)/1000000,
                     limits = c(80000000, 110000000)) +                        # Setting y-axis scal

  scale_x_continuous(breaks = seq(0, 100, 5)) +                                 # Setting x-axis scal

  xlab('Training Set Size (%)') + ylab('RMSE ( $ Millions )') +                  # Setting axes labels

  # Setting the title
  ggtitle("Figure 2 (a) : Model 2 (numeric transforms only) - RMSE vs. Training Set Size") +
```

```
  theme(plot.title = element_text(face = "bold",color = 'darkblue',              # Setting various the
                                   hjust = 0.5)) +
  theme(axis.text = element_text(colour = "black")) +
  theme(axis.title.x = element_text(face = "bold", color='black'),
        axis.title.y = element_text(face = "bold", color='black')) +
  theme(legend.key = element_rect(fill = "white")) +
  geom_text(aes(label= paste("Task 1 - Test"),
                x = 17, y = 105000000), check_overlap = TRUE,
            color = "red", size = 4, alpha = 0.5) +
  geom_text(aes(label= paste("Task 1 - Train"),
                x = 11, y = 93000000), check_overlap = TRUE,
            color = "red", size = 4, alpha = 0.5) +
  geom_text(aes(label= paste("Task 2 (num. trans) - Test"),
                x = 10, y = 98000000), check_overlap = TRUE,
            color = "blue", size = 4) +
  geom_text(aes(label= paste("Task 2 (num. trans) - Train"),
                x = 16, y = 85000000), check_overlap = TRUE,
            color = "blue", size = 4)
```



Observation : In Figure 2(a), it can clearly be seen that training the model with only numeric transforms, shows a little improvment in the RMSE curves for both train and test data. The dim-dashed curves in the background are shown for reference taken from Task 1 - (original numeric variables only).

Now lets combine the best of both worlds, i.e. (binning and numeric transforms) and see the collective affect.

Helper Function for performing power transformations and binning together

This method runs all 19 different numeric transforms and binning configurations for all numeric variables and suggests the numeric tranform/bins that has the best correlation with 'Gross'.

```
transformation = function(Gross, Feature, bin_limit){

  # 1. This part works on getting the numeric transformation

  transform_check = rep(0, 20)
```

```
  transform = list("Original" = round(cor(Gross, Feature), 6),
                        "power(1/9)" = round(cor(Gross, Feature^(1/9)), 6),
                        "power(1/4)" = round(cor(Gross, Feature^0.25), 6),
                        "power(1/3)" = round(cor(Gross, Feature^(1/3)), 6),
                        "power(1/2)" = round(cor(Gross, Feature^0.5), 6),
                        "power(2)" = round(cor(Gross, Feature^2), 6),
                        "power(3)" = round(cor(Gross, Feature^3), 6),
                        "power(4)" = round(cor(Gross, Feature^4), 6),
                        "power(5)" = round(cor(Gross, Feature^5), 6),
                        "power(6)" = round(cor(Gross, Feature^6), 6),
                        "power(7)" = round(cor(Gross, Feature^7), 6),
                        "power(8)" = round(cor(Gross, Feature^8), 6),
                        "power(9)" = round(cor(Gross, Feature^9), 6),
                        "power(10)" = round(cor(Gross, Feature^10), 6),
                        "Log" = round(cor(Gross, log(Feature + 1)), 6),
                        "Exp" = round(cor(Gross, exp(Feature)), 6),
                        "Log10" = round(cor(Gross, log10(Feature + 1)), 6),
                        "Log2" = round(cor(Gross, log2(Feature + 1)), 6),
                        "Sin" = round(cor(Gross, sin(Feature)), 6),
                        "1/x" = round(cor(Gross, 1/(Feature)), 6))

  for (i in 1:20) {
    transform_check[i] = transform[[i]]
  }

  # 2. This part works on binning

  bin_check = rep(0, bin_limit)

  bin_check[1] = round(cor(Gross, Feature), 6)

  for (i in 2:bin_limit){

    #print(paste("Bins", i, ":", round(cor(var1, as.numeric(cut(var2, i, labels = c(1:i)))), 6)))
    bin_check[i] = round(cor(Gross, as.numeric(cut(Feature, i, labels = c(1:i)))), 6)
  }

  if (max(bin_check) > transform[[which.max(transform_check)]]){

    return(paste("Best Transform --> Binning : No. of bins =",
                 which.max(bin_check), "that gives r^2 of",
                 max(bin_check), "with 'Gross'"))
  }
  else {

    return(paste("Best Transform --> Numeric Transform =",
               names(transform[which.max(transform_check)]),
               "that gives r^2 of",
               transform[[which.max(transform_check)]], "with 'Gross'"))
  }
}
```

```
# https://www.youtube.com/watch?v=MvPd0vSotWM

# https://www.youtube.com/watch?v=20K6fGuTBgw
```

2.3 Applying both Numeric Tranforms and Binning

2.3.1 Checking which Numeric Tranforms or Bins work the best for each numeric feature

```
numeric_features = c('Year','Runtime', 'imdbRating', 'imdbVotes', 'tomatoRating',
                    'tomatoReviews', 'tomatoFresh','tomatoRotten', 'tomatoUserRating',
                    'tomatoUserReviews', 'Budget')

for (i in 1:length(numeric_features)){

  print(paste(numeric_features[i], transformation(df_task_1$Gross, df_task_1[[numeric_features[i]]], 10
}
```

```
## [1] "Year Best Transform --> Binning : No. of bins = 6 that gives r^2 of 0.093234 with 'Gross'"
## [1] "Runtime Best Transform --> Numeric Transform = power(3) that gives r^2 of 0.350134 with 'Gross'
## [1] "imdbRating Best Transform --> Numeric Transform = Exp that gives r^2 of 0.29085 with 'Gross'"
## [1] "imdbVotes Best Transform --> Numeric Transform = Original that gives r^2 of 0.687022 with 'Gros
## [1] "tomatoRating Best Transform --> Numeric Transform = power(4) that gives r^2 of 0.222436 with 'G
## [1] "tomatoReviews Best Transform --> Numeric Transform = power(3) that gives r^2 of 0.633653 with '
## [1] "tomatoFresh Best Transform --> Numeric Transform = power(2) that gives r^2 of 0.4876 with 'Gros
## [1] "tomatoRotten Best Transform --> Numeric Transform = power(3) that gives r^2 of 0.229463 with 'G
## [1] "tomatoUserRating Best Transform --> Numeric Transform = power(4) that gives r^2 of 0.291313 wit
## [1] "tomatoUserReviews Best Transform --> Numeric Transform = power(1/4) that gives r^2 of 0.484947
## [1] "Budget Best Transform --> Binning : No. of bins = 92 that gives r^2 of 0.77829 with 'Gross'"
```

2.3.2 Applying 'Numeric Tranforms and Binning' to the data frame

A data frame with best (bin, numeric transform or original) configuration of each variable was created to evaluate the improvement in RMSE for different training / test sizes.

```
df_task_2 = data.frame(Year = as.numeric(cut(df_task_1$Year, 6, labels = c(1:6))),
                       Runtime = (df_task_1$Runtime)^3,
                       imdbRating = exp(df_task_1$imdbRating),
                       imdbVotes = df_task_1$imdbVotes,
                       tomatoRating = (df_task_1$tomatoRating)^4,
                       tomatoReviews = (df_task_1$tomatoReviews)^3,
                       tomatoFresh = (df_task_1$tomatoFresh)^2,
                       tomatoRotten = (df_task_1$tomatoRotten)^3,
                       tomatoUserRating = (df_task_1$tomatoUserRating)^4,
                       tomatoUserReviews = (df_task_1$tomatoUserReviews)^0.25,
                       Budget = as.numeric(cut(df_task_1$Budget, 92, labels = c(1:92))),
                       Gross = df_task_1$Gross)
```

2.4 Evaluation Strategy - Task 2 [Numeric Transforms + Binning]

2.4.1 [RMSE values - Numeric Transforms + Binning]

- **No. of Iterations :** 1000
- **Scenario 1 :** 5% Train data / 95% Test data

- **Scenario 2 :** 95% Train data / 5% Test data

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

rmse_2.4 = train_test(df_task_2, 1000, 0.05)

paste("Multiple run 5% Train data / 95% Test data - Average Training RMSE : ",
      round(rmse_2.4$TRAIN_RMSE,0))
```

```
## [1] "Multiple run 5% Train data / 95% Test data - Average Training RMSE :  83038769"
```

```
paste("Multiple run 5% Train data / 95% Test data - Average Test RMSE     : ",
      round(rmse_2.4$TEST_RMSE,0))
```

```
## [1] "Multiple run 5% Train data / 95% Test data - Average Test RMSE     :  105965040"
```

```
rmse_2.4 = train_test(df_task_2, 1000, 0.95)

print("")
```

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

```
paste("Multiple run 95% Train data / 5% Test data - Average Training RMSE : ",
      round(rmse_2.4$TRAIN_RMSE,0))
```

```
## [1] "Multiple run 95% Train data / 5% Test data - Average Training RMSE :  94800052"
```

```
paste("Multiple run 95% Train data / 5% Test data - Average Test RMSE     : ",
      round(rmse_2.4$TEST_RMSE,0))
```

```
## [1] "Multiple run 95% Train data / 5% Test data - Average Test RMSE     :  94568269"
```

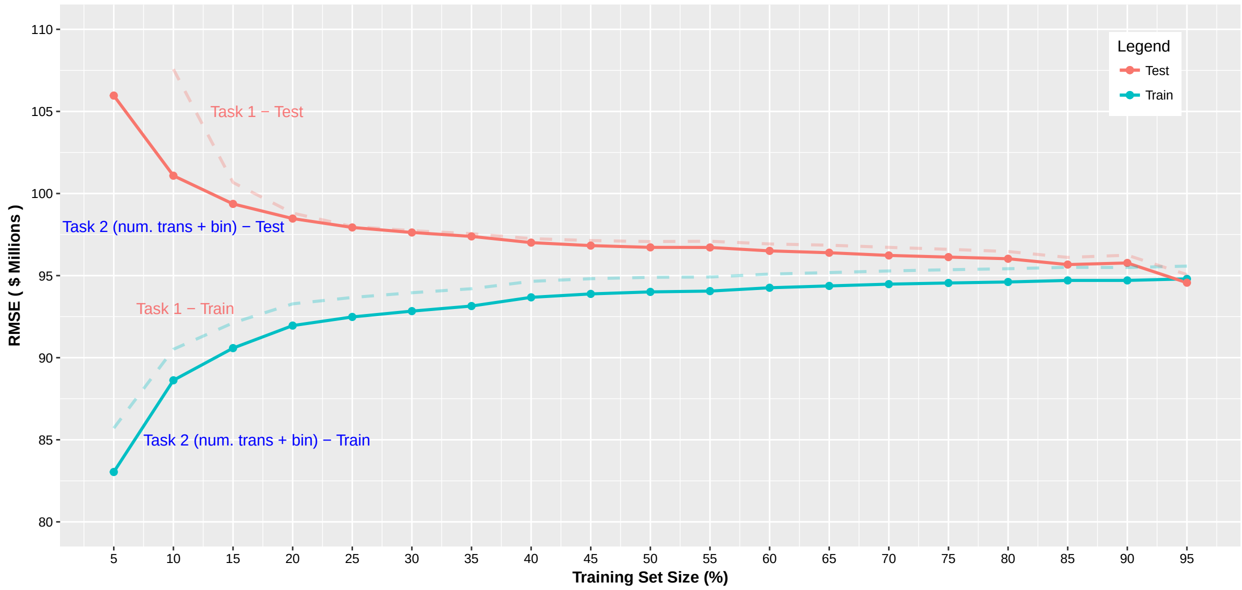
2.4.2 [RMSE Curves - Numeric Transforms + Binning]

- **No. of Iterations :** 1000

```
set_sizes = seq(0.05, 0.95, by = 0.05)


TRAIN_RMSE_2.5 = rep(0,19)
TEST_RMSE_2.5  = rep(0,19)

for (s in 1:length(set_sizes)){

  #print (s)

  rmse_2.5 = train_test(df_task_2, 1000, set_sizes[s])

  TRAIN_RMSE_2.5[s] = rmse_2.5$TRAIN_RMSE
  TEST_RMSE_2.5[s] = rmse_2.5$TEST_RMSE

  }

#TRAIN_RMSE_2.5
#TEST_RMSE_2.5

RMSE_task_2 = data.frame(TrainSet_Size = set_sizes * 100,
                                Train_2 = TRAIN_RMSE_2.5,
```

```r
                               Test_2 = TEST_RMSE_2.5,
                               Train_1 = TRAIN_RMSE_1.5,
                               Test_1 = TEST_RMSE_1.5)

ggplot(data=RMSE_task_2) +                                                           # Initializing the p

  geom_line(size = 1, aes(x = TrainSet_Size, y = Train_2, color = "Train")) +      # Line Plot

  geom_line(size = 1, aes(x = TrainSet_Size, y = Test_2, color = "Test")) +        # Line Plot

  geom_point(size = 2, aes(x = TrainSet_Size, y = Train_2, color = "Train")) +     # Point

  geom_point(size = 2, aes(x = TrainSet_Size, y = Test_2, color = "Test")) +       # Point

  geom_line(size = 1, linetype = 2, alpha = 0.3,
            aes(x = TrainSet_Size, y = Train_1, color = "Train")) +                # Line Plot

  geom_line(size = 1, linetype = 2, alpha = 0.3,
            aes(x = TrainSet_Size, y = Test_1, color = "Test")) +                  # Line Plot

  labs(colour="Legend") +                                                           # Legend

  theme(legend.position = c(0.95, 0.95), legend.justification = c(1, 1)) +         # Legend position

  scale_y_continuous(breaks = seq(80000000, 110000000, 5000000),
                     labels = seq(80000000, 110000000, 5000000)/1000000,
                     limits = c(80000000, 110000000)) +                             # Setting y-axis sca

  scale_x_continuous(breaks = seq(0, 100, 5)) +                                     # Setting x-axis sca

  xlab('Training Set Size (%)') + ylab('RMSE ( $ Millions )') +                     # Setting axes label

  # Setting the title
  ggtitle("Figure 2 (b) : Model 2 (Binned + Numeric Trasformed variables only) - RMSE vs. Training Set S

  theme(plot.title = element_text(face = "bold",color = 'darkblue',                 # Setting various th
                                  hjust = 0.5)) +
  theme(axis.text = element_text(colour = "black")) +
  theme(axis.title.x = element_text(face = "bold", color='black'),
        axis.title.y = element_text(face = "bold", color='black')) +
  theme(legend.key = element_rect(fill = "white")) +
  geom_text(aes(label= paste("Task 1 - Test"),
                x = 17, y = 105000000), check_overlap = TRUE,
            color = "red", size = 4, alpha = 0.5) +
  geom_text(aes(label= paste("Task 1 - Train"),
                x = 11, y = 93000000), check_overlap = TRUE,
            color = "red", size = 4, alpha = 0.5) +
  geom_text(aes(label= paste("Task 2 (num. trans + bin) - Test"),
                x = 10, y = 98000000), check_overlap = TRUE,
            color = "blue", size = 4) +
  geom_text(aes(label= paste("Task 2 (num. trans + bin) - Train"),
                x = 17, y = 85000000), check_overlap = TRUE,
            color = "blue", size = 4)
```

Figure 2 (b) : Model 2 (Binned + Numeric Trasformed variables only) − RMSE vs. Training Set Size



Observation : By combining both numeric transforms and binned features, there is improvement in both train and test RMSE compared to task 1. This is quite evident towards the larger train set sizes, i.e. training set larger than 40%

2.5 Evaluation Strategy - Task 2 [Showing improvement progressively]

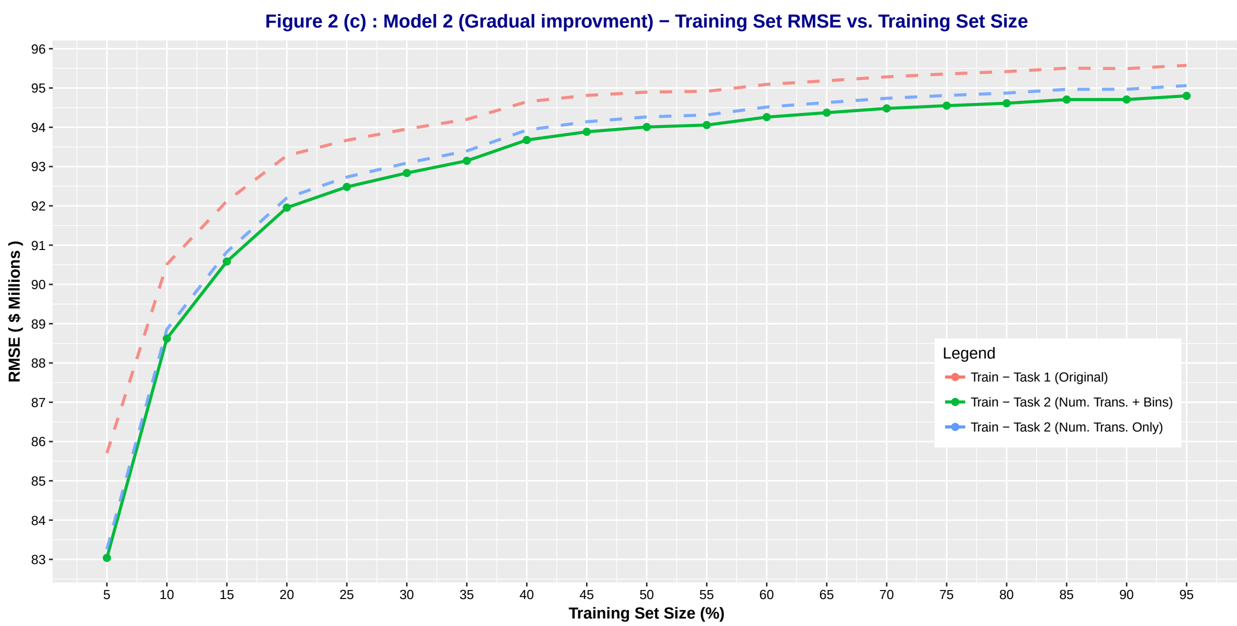
This section shows seperate training and testing RMSE curves focusing on improvement seen between task 1 and task 2.

2.5.1 [RMSE TRAINING Curves - Showing improvement progressively]

- **No. of Iterations :** 1000

```
RMSE_task_2_train = data.frame(TrainSet_Size = set_sizes * 100,
                               Train_2 = TRAIN_RMSE_2.5,
                               Train_2a = TRAIN_RMSE_2.5_pt,
                               Train_1 = TRAIN_RMSE_1.5)

ggplot(data=RMSE_task_2_train) +                                           # Initializing the 

  geom_line(size = 1,
            aes(x = TrainSet_Size,
                y = Train_2, color = "Train - Task 2 (Num. Trans. + Bins)")) +      # Line Plot

  geom_point(size = 2,
             aes(x = TrainSet_Size, y = Train_2,
                 color = "Train - Task 2 (Num. Trans. + Bins)")) +                 # Point

  geom_line(size = 1, linetype = 2,
            alpha = 0.8, aes(x = TrainSet_Size, y = Train_2a,
                             color = "Train - Task 2 (Num. Trans. Only)")) +        # Line Plot

  geom_line(size = 1, linetype = 2,
            alpha = 0.8, aes(x = TrainSet_Size, y = Train_1,
```

```
                              color = "Train - Task 1 (Original)")) +                # Line Plot

  labs(colour="Legend") +                                                            # Legend

  theme(legend.position = c(0.95, 0.45), legend.justification = c(1, 1)) +          # Legend position

  scale_y_continuous(breaks = seq(75000000, 96000000, 1000000),
                     labels = seq(75000000, 96000000, 1000000)/1000000) +           # Setting y-axis sc

  scale_x_continuous(breaks = seq(0, 100, 5)) +                                      # Setting x-axis sc

  xlab('Training Set Size (%)') + ylab('RMSE ( $ Millions )') +                      # Setting axes labe

  # Setting the title
  ggtitle("Figure 2 (c) : Model 2 (Gradual improvment) - Training Set RMSE vs. Training Set Size") +

  theme(plot.title = element_text(face = "bold",color = 'darkblue',                  # Setting various t
                                  hjust = 0.5)) +
  theme(axis.text = element_text(colour = "black")) +
  theme(axis.title.x = element_text(face = "bold", color='black'),
        axis.title.y = element_text(face = "bold", color='black')) +
  theme(legend.key = element_rect(fill = "white"))
```



Observation : Figure 2 (c) shows the gradual improvement in Train RMSE after task 1. It can clearly be seen that by combining best of binning and numeric transforms there has been significant improvement in training RMSE.

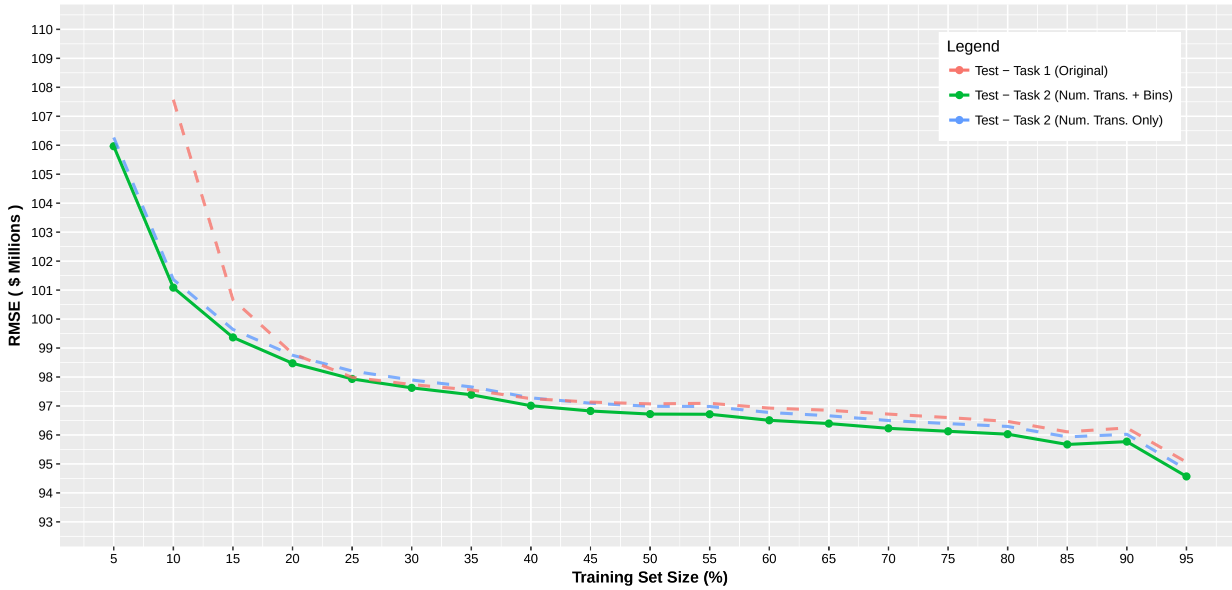
2.5.2 [RMSE TEST Curves - Showing improvement progressively]

- **No. of Iterations :** 1000

```
RMSE_task_2_test = data.frame(TrainSet_Size = set_sizes * 100,
                              Test_2 = TEST_RMSE_2.5,
                              Test_2a = TEST_RMSE_2.5_pt,
```

```
                              Test_1 = TEST_RMSE_1.5)

ggplot(data=RMSE_task_2_test) +                                                # Initializing the plo

  geom_line(size = 1,
            aes(x = TrainSet_Size,
                y = Test_2, color = "Test - Task 2 (Num. Trans. + Bins)")) +     # Line Plot

  geom_point(size = 2,
             aes(x = TrainSet_Size,
                 y = Test_2, color = "Test - Task 2 (Num. Trans. + Bins)")) +    # Point


  geom_line(size = 1, linetype = 2,
            alpha = 0.8, aes(x = TrainSet_Size, y = Test_2a,
                             color = "Test - Task 2 (Num. Trans. Only)")) +      # Line Plot



  geom_line(size = 1, linetype = 2,
            alpha = 0.8, aes(x = TrainSet_Size, y = Test_1,
                             color = "Test - Task 1 (Original)")) +              # Line Plot

  labs(colour="Legend") +                                                      # Legend

  theme(legend.position = c(0.95, 0.95), legend.justification = c(1, 1)) +       # Legend position

  scale_y_continuous(breaks = seq(93000000, 110000000, 1000000),
                     labels = seq(93000000, 110000000, 1000000)/1000000,
                     limits = c(93000000, 110000000)) +                          # Setting y-axis scale

  scale_x_continuous(breaks = seq(0, 100, 5)) +                                  # Setting x-axis scale

  xlab('Training Set Size (%)') + ylab('RMSE ( $ Millions )') +                  # Setting axes labels

  # Setting the title
  ggtitle("Figure 2 (d) : Model 2 (Gradual improvment) - Test Set RMSE vs. Training Set Size") +

  theme(plot.title = element_text(face = "bold",color = 'darkblue',              # Setting various them
                                  hjust = 0.5)) +
  theme(axis.text = element_text(colour = "black")) +
  theme(axis.title.x = element_text(face = "bold", color='black'),
        axis.title.y = element_text(face = "bold", color='black')) +
  theme(legend.key = element_rect(fill = "white"))
```

Figure 2 (d) : Model 2 (Gradual improvment) − Test Set RMSE vs. Training Set Size



Observation : Here Figure 2(d) shows progressive improvement in Test RMSE after task 1. Although, not as much as the training RMSE (Figure 2(c)), final Test RMSE curve with best of both binning and numeric transforms combined shows slight improvement compared to Test RMSE form task 1.

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

A: For the purpose of this task, I created a function which would take a feature vector as input and output the best possible transform for that feature based on its correlation coefficient with 'Gross', i.e. the function suggests the transform or binning configuration that has the highest r^2 with 'Gross'. This transform can be a numeric transform (x^2, x^4, log(x), exp(x), etc) or form of binning (dividing a feature vector into N equal numeric factors using the 'cut' function).

Numeric Transforms: There were 19 different numeric transforms tried out to identify which one would work best for a particular feature. They are as follows:

- "power(1/9)" = Feature^(1/9)
- "power(1/4)" = Feature^0.25
- "power(1/3)" = Feature^(1/3)
- "power(1/2)" = Feature^0.5
- "power(2)" = Feature^2
- "power(3)" = Feature^3
- "power(4)" = Feature^4
- "power(5)" = Feature^5
- "power(6)" = Feature^6
- "power(7)" = Feature^7
- "power(8)" = Feature^8
- "power(9)" = Feature^9
- "power(10)" = Feature^10
- "Log" = log(Feature + 1)
- "Exp" = exp(Feature)
- "Log10" = log10(Feature + 1)
- "Log2" = log2(Feature + 1)
- "Sin" = sin(Feature)
- "1/x" = 1/(Feature)

Binning: The cut function was run on each feature using as a FOR loop with i from 2 to 100, i.e. checking from 2 bins to 100 bins and the best bin configuration that gave highest corr. coeff. with 'Gross' was chosen. Code chunk for applying 'cut' function is shown below:

- as.numeric(cut(Feature, i, labels = c(1:i))), where i is 2 to 100

Final Transforms used: Following is the list of the best transform (either numeric or binning) used for each of the 11 numeric features.

- 1. "Year" Best Transform –> **Binning : No. of bins = 6**
- 2. "Runtime" Best Transform –> **Numeric Transform = power(3)**
- 3. "imdbRating" Best Transform –> **Numeric Transform = Exp**
- 4. "imdbVotes" Best Transform –> **Numeric Transform = Original**
- 5. "tomatoRating" Best Transform –> **Numeric Transform = power(4)**
- 6. "tomatoReviews" Best Transform –> **Numeric Transform = power(3)**
- 7. "tomatoFresh" Best Transform –> **Numeric Transform = power(2)**
- 8. "tomatoRotten" Best Transform –> **Numeric Transform = power(3)**
- 9. "tomatoUserRating" Best Transform –> **Numeric Transform = power(4)**
- 10. "tomatoUserReviews" Best Transform –> **Numeric Transform = power(1/4)**
- 11. "Budget" Best Transform –> **Binning : No. of bins = 92**

Out of the 11 numeric features, only 2 performed best with binning ('Year' and 'Budget'), 1 feature performed best in original form ('imdbVotes'), while rest of them gave best corr. coeff. with a numeric transform.

```
t1_t2_compare = data.frame(Task_1 = t(cor(df_task_1$Gross, subset(df_task_1, select = -c(Gross)))),
                Task_2 = t(cor(df_task_2$Gross, subset(df_task_2, select = -c(Gross)))),
                Delta = t(cor(df_task_2$Gross, subset(df_task_2, select = -c(Gross)))) -
                  t(cor(df_task_1$Gross, subset(df_task_1, select = -c(Gross)))))

names(t1_t2_compare) = c("Task_1", "Task_2", "Delta")

t1_t2_compare
```

```
##                       Task_1     Task_2       Delta
## Year              0.09173318 0.09323378 0.001500600
## Runtime           0.32419249 0.35013362 0.025941122
## imdbRating        0.24790390 0.29085029 0.042946388
## imdbVotes         0.68702224 0.68702224 0.000000000
## tomatoRating      0.20862274 0.22243649 0.013813744
## tomatoReviews     0.54547288 0.63365276 0.088179874
## tomatoFresh       0.46341497 0.48759977 0.024184796
## tomatoRotten      0.19601099 0.22946290 0.033451910
## tomatoUserRating  0.27935942 0.29131316 0.011953742
## tomatoUserReviews 0.21643324 0.48494674 0.268513501
## Budget            0.77686777 0.77829046 0.001422689
```

Table above shows the improvement in r^2 with 'Gross' after the best transform/bin was applied (comparing Task 1 r^2 to Task 2 r^2). 'tomatoUserReviews' showed the most improvement where the transform was power of 4 ('tomatoUserReviews'^4) while the least improvement was seen for 'Budget' where the transform was binning.

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.

3.1 Approach 1 [Top N approach]

3.1.1 Converting non-numeric variables to useful numeric variables [Top N approach]

I first started by creating numeric versions of the non-numeric variables based on one hot coding. For some of the variables where columns were plentiful (such as 'Director', 'Production', 'Writer', etc) after performing one hot coding I limited the number of columns to top N (5, 10 or 20, etc) most popular (highest colSums) to be used for modeling.

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

df_task_3 = df


#################### RATED #####################################################

df_task_3_rated_1 = df_task_3[, c('Rated'), drop=FALSE]

df_task_3_rated_1 = concat.split.expanded(df_task_3_rated_1, "Rated",
                                        sep=",", fill = 0,
                                        drop = TRUE, type = "character")

df_task_3_rated_1$`Rated_N/A` <- NULL

# SORT # http://stackoverflow.com/questions/36411338/using-ordercolsums-in-r

df_task_3_rated_1_dash =
  df_task_3_rated_1[,order(colSums(-df_task_3_rated_1,na.rm=TRUE))]


# --- Logical OR to combine UNRATED and NOT RATED

df_task_3_rated_1_dash$Rated_UNRATED =
  df_task_3_rated_1_dash$`Rated_NOT RATED` |
  df_task_3_rated_1_dash$Rated_UNRATED

df_task_3_rated_1_dash$Rated_UNRATED[df_task_3_rated_1_dash$Rated_UNRATED] = 1

df_task_3_rated_1_dash$`Rated_NOT RATED` = NULL

df_task_3_rated_1_dash =
  df_task_3_rated_1_dash[,order(colSums(-df_task_3_rated_1_dash,na.rm=TRUE))]


# --- Based on Top N

df_task_3_rated_1_dash = df_task_3_rated_1_dash[,1:3]
```

```r
#################### METASCORE ################################################

df_task_3_meta_1 = df_task_3[, c('Metascore'), drop=FALSE]

df_task_3_meta_1$Metascore[df_task_3_meta_1$Metascore == 'N/A'] = NA


df_task_3_meta_1$Metascore = as.numeric(df_task_3_meta_1$Metascore)

df_task_3_meta_1$Metascore[is.na(df_task_3_meta_1$Metascore)] =
  median(as.numeric(df_task_3_meta_1$Metascore), na.rm = TRUE)

#################### tomatoImage ##############################################

df_task_3_tomatoIm_1 = df_task_3[, c('tomatoImage'), drop=FALSE]

df_task_3_tomatoIm_1 = concat.split.expanded(df_task_3_tomatoIm_1,
                                             "tomatoImage", sep=",", fill = 0,
                                             drop = TRUE, type = "character")

df_task_3_tomatoIm_1$`tomatoImage_N/A` <- NULL

#################### PRODUCTION ###############################################

df_task_3_prod_1 = df_task_3[, c('Production'), drop=FALSE]

df_task_3_prod_1 = concat.split.expanded(df_task_3_prod_1, "Production",
                                         sep=",", fill = 0, drop = TRUE,
                                         type = "character")


df_task_3_prod_1$`Production_N/A` <- NULL

df_task_3_prod_1_dash =
  df_task_3_prod_1[,order(colSums(-df_task_3_prod_1,na.rm=TRUE))]

# Combining top 15 Production houses based on same major corporations

prod_combine = function(df, prod){

  combined = Reduce('|', df[ , grepl( prod , names( df) ) ])

  combined[combined] = 1
  return(combined)
}

#1
Warner_prod = prod_combine(df_task_3_prod_1_dash, "Warner")

#2
Universal_prod = prod_combine(df_task_3_prod_1_dash, "Universal")

#3
```

```r
Fox_prod = prod_combine(df_task_3_prod_1_dash, "Fox")

#4
Paramount_prod = prod_combine(df_task_3_prod_1_dash, "Paramount")

#5
Sony_prod = prod_combine(df_task_3_prod_1_dash, "Sony")

#6
NewLine_prod = prod_combine(df_task_3_prod_1_dash, "New Line")

#7
Walt_prod = prod_combine(df_task_3_prod_1_dash, "Disney")

#8
Columbia_prod = prod_combine(df_task_3_prod_1_dash, "Columbia")

#9
Focus_prod = prod_combine(df_task_3_prod_1_dash, "Focus")

#10
Miramax_prod = prod_combine(df_task_3_prod_1_dash, "Miramax")

#11
Lion_prod = prod_combine(df_task_3_prod_1_dash, "Lion")

#12
Weinstein_prod = prod_combine(df_task_3_prod_1_dash, "Weinstein")

#13
MGM_prod = prod_combine(df_task_3_prod_1_dash, "MGM")

#14
Summit_prod = prod_combine(df_task_3_prod_1_dash, "Summit")

#15
Buena_prod = prod_combine(df_task_3_prod_1_dash, "Buena")

# Combining all Production houses

df_task_3_prod_1_dash = data.frame(Production_Warner = Warner_prod,
                                 Production_Universal = Universal_prod,
                                 Production_Fox = Fox_prod,
                                 Production_Paramount = Paramount_prod,
                                 Production_Sony = Sony_prod,
                                 Production_NewLine = NewLine_prod,
                                 Production_WaltDisney = Walt_prod,
                                 Production_Columbia = Columbia_prod,
                                 Production_Focus = Focus_prod,
                                 Production_Miramax = Miramax_prod,
                                 Production_LionsGate = Lion_prod,
                                 Production_Weinstein = Weinstein_prod,
                                 Production_MGM = MGM_prod,
```

```
                                 Production_Summit = Summit_prod,
                                 Production_Buena = Buena_prod
                                 )

df_task_3_prod_1_dash =
  df_task_3_prod_1_dash[,order(colSums(-df_task_3_prod_1_dash,na.rm=TRUE))]

# --- Based on Top N

df_task_3_prod_1_dash = df_task_3_prod_1_dash[,1:5]

#################### GENRES ####################################################

df_task_3_genre_1 = df_task_3[, c('Genre'), drop=FALSE]

df_task_3_genre_1 =
  concat.split.expanded(df_task_3_genre_1, "Genre", sep=",",
                                    fill = 0, drop = TRUE, type = "character")

df_task_3_genre_1$`Genre_N/A` <- NULL

df_task_3_genre_1_dash =
  df_task_3_genre_1[,order(colSums(-df_task_3_genre_1,na.rm=TRUE))]

# Selecting Top N Genres

df_task_3_genre_1_dash = df_task_3_genre_1_dash[,1:5]

#################### DIRECTOR ##################################################

# Based on https://piazza.com/class/ixpif8oc7gi1vr?cid=1342

df_task_3_dir_1 = df_task_3[, c('Director'), drop=FALSE]

df_task_3_dir_1 = concat.split.expanded(df_task_3_dir_1, "Director", sep=",",
                                  fill = 0, drop = TRUE, type = "character")

df_task_3_dir_1$`Director_N/A` <- NULL

df_task_3_dir_1_dash =
  df_task_3_dir_1[,order(colSums(-df_task_3_dir_1,na.rm=TRUE))]

# Selecting Top N directors that gave best correlation with 'Gross'

Top_dir = Reduce('|', df_task_3_dir_1_dash[ , 1:400 ])
Top_dir[Top_dir] = 1

df_task_3_dir_1_dash = data.frame(Top_Directors = Top_dir )


#################### Writer ####################################################

df_task_3_writer_1 = df_task_3[, c('Writer'), drop=FALSE]
```

```r
df_task_3_writer_1 = concat.split.expanded(df_task_3_writer_1, "Writer",
                                          sep=",", fill = 0, drop = TRUE,
                                          type = "character")

df_task_3_writer_1$`Writer_N/A` <- NULL

df_task_3_writer_1_dash =
  df_task_3_writer_1[,order(colSums(-df_task_3_writer_1,na.rm=TRUE))]

# Selecting Top N writers that gave best correlation with 'Gross'

Top_writer = Reduce('|', df_task_3_writer_1_dash[ , 1:450 ])
Top_writer[Top_writer] = 1

df_task_3_writer_1_dash = data.frame(Top_Writers = Top_writer )


#################### ACTOR ###################################################

df_task_3_actor_1 = df_task_3[, c('Actors'), drop=FALSE]

df_task_3_actor_1 = concat.split.expanded(df_task_3_actor_1, "Actors",
                                          sep=",", fill = 0, drop = TRUE,
                                          type = "character")

df_task_3_actor_1$`Actors_N/A` <- NULL

df_task_3_actor_1_dash =
  df_task_3_actor_1[,order(colSums(-df_task_3_actor_1,na.rm=TRUE))]

# Selecting Top N Actors that gave best correlation with 'Gross'

Top_actor = Reduce('|', df_task_3_actor_1_dash[ , 1:400 ])
Top_actor[Top_actor] = 1

df_task_3_actor_1_dash = data.frame(Top_Actors = Top_actor )


#################### LANGUAGE ################################################

df_task_3_lang_1 = df_task_3[, c('Language'), drop=FALSE]

df_task_3_lang_1 = concat.split.expanded(df_task_3_lang_1, "Language",
                                          sep=",", fill = 0, drop = TRUE,
                                          type = "character")

df_task_3_lang_1$`Language_N/A` <- NULL

df_task_3_lang_1_dash =
  df_task_3_lang_1[,order(colSums(-df_task_3_lang_1,na.rm=TRUE))]

# Selecting Last N Languages that gave best correlation with 'Gross'
```

```r
Other_lang = Reduce('|', df_task_3_lang_1_dash[ , 5:117])
Other_lang[Other_lang] = 1

df_task_3_lang_1_dash = data.frame(Other_Languages = Other_lang )


#################### COUNTRY ##################################################

df_task_3_country_1 = df_task_3[, c('Country'), drop=FALSE]

df_task_3_country_1 = concat.split.expanded(df_task_3_country_1, "Country",
                                            sep=",", fill = 0, drop = TRUE,
                                            type = "character")

df_task_3_country_1$`Country_N/A` <- NULL

df_task_3_country_1_dash =
  df_task_3_country_1[,order(colSums(-df_task_3_country_1,na.rm=TRUE))]

# Selecting Top N most popular countries

df_task_3_country_1_dash = df_task_3_country_1_dash[ , 1:5]

#################### AWARDS ####################################################


award_check = function(num){

  winnings = 0
  nominations = 0

  if(grepl( "nom" , num ) & grepl( "win" , num ) &
     grepl( "Nominated for" , num )){

    num_loc <- str_locate_all(num, "[0-9]+")[[1]]
    converted = as.numeric(str_sub(num, num_loc[, "start"], num_loc[, "end"]))

    winnings = converted[2]
    nominations = converted[1] + converted[3]
    return(paste(winnings, nominations))


  } else if(grepl( "nom" , num ) & grepl( "win" , num ) &
            grepl( "Won" , num ) ){

    num_loc <- str_locate_all(num, "[0-9]+")[[1]]
    converted = as.numeric(str_sub(num, num_loc[, "start"], num_loc[, "end"]))

    winnings = converted[1] + converted[2]
    nominations = converted[3]
    return(paste(winnings, nominations))
```

```
} else if(grepl( "nom" , num ) & grepl( "Nominated for" , num ) ){

  num_loc <- str_locate_all(num, "[0-9]+")[[1]]
  converted = as.numeric(str_sub(num, num_loc[, "start"], num_loc[, "end"]))

  nominations = converted[1] + converted[2]
  return(paste(winnings, nominations))


} else if(grepl( "win" , num ) & grepl( "Nominated for" , num )){

  num_loc <- str_locate_all(num, "[0-9]+")[[1]]
  converted = as.numeric(str_sub(num, num_loc[, "start"], num_loc[, "end"]))

  winnings = converted[2]
  nominations = converted[1]
  return(paste(winnings, nominations))


}else if(grepl( "nom" , num ) & grepl( "Won" , num )){

  num_loc <- str_locate_all(num, "[0-9]+")[[1]]
  converted = as.numeric(str_sub(num, num_loc[, "start"], num_loc[, "end"]))

  winnings = converted[1]
  nominations = converted[2]
  return(paste(winnings, nominations))


}else if(grepl( "win" , num ) & grepl( "Won" , num )  ){

  num_loc <- str_locate_all(num, "[0-9]+")[[1]]
  converted = as.numeric(str_sub(num, num_loc[, "start"], num_loc[, "end"]))

  winnings = converted[1] + converted[2]
  return(paste(winnings, nominations))


}else if(grepl( "nom" , num ) & grepl( "win" , num ) ){

  num_loc <- str_locate_all(num, "[0-9]+")[[1]]
  converted = as.numeric(str_sub(num, num_loc[, "start"], num_loc[, "end"]))

  winnings = converted[1]
  nominations = converted[2]
  return(paste(winnings, nominations))


}else if(grepl( "win" , num ) ){

  num_loc <- str_locate_all(num, "[0-9]+")[[1]]
  converted = as.numeric(str_sub(num, num_loc[, "start"], num_loc[, "end"]))
```

```r
    winnings = converted[1]
    return(paste(winnings, nominations))


  }else if(grepl( "nom" , num ) ){

    num_loc <- str_locate_all(num, "[0-9]+")[[1]]
    converted = as.numeric(str_sub(num, num_loc[, "start"], num_loc[, "end"]))

    nominations = converted[1]
    return(paste(winnings, nominations))


  } else if(num == "N/A"){
    #print("NA")
    return(paste(NA, NA))
  }


  else {return(paste(0, 0))}



}

result_awards = sapply(df_task_3$Awards, function(x) award_check(x))

split_awards = read.table(text =
                            result_awards, sep = " ", colClasses = "numeric")

names(split_awards) =  c('Wins', 'Nominations')

df_task_3_awards_1 = df_task_3[, c('Awards'), drop=FALSE]

df_task_3_awards_1$Wins = split_awards$Wins
df_task_3_awards_1$Nominations = split_awards$Nominations

df_task_3_awards_1$Awards = NULL

# Creatin Awards and nominatinos numerical features

df_task_3_awards_1$Wins[is.na(df_task_3_awards_1$Wins)] =
  median(df_task_3_awards_1$Wins, na.rm = TRUE)

df_task_3_awards_1$Nominations[is.na(df_task_3_awards_1$Nominations)] =
  median(df_task_3_awards_1$Nominations, na.rm = TRUE)

#################### RELEASED ################################################

df_task_3_released =  df_task_3[, c('Released'), drop=FALSE]

df_task_3_released$Released_Year =
  as.numeric(substring(df_task_3_released$Released,1,4))
```

```r
df_task_3_released$Released_Month =
  as.numeric(substring(df_task_3_released$Released,6,7))

df_task_3_released$Released_Day =
  as.numeric(substring(df_task_3_released$Released,9,10))

df_task_3_released$Released = NULL
```

3.1.2 Creating the dataframe based on Approach 1

```r
df_task_3_a = cbind(df_task_3_rated_1_dash,
                    df_task_3_meta_1,
                    df_task_3_tomatoIm_1,
                    df_task_3_prod_1_dash,
                    df_task_3_genre_1_dash,
                    df_task_3_dir_1_dash,
                    df_task_3_writer_1_dash,
                    df_task_3_actor_1_dash,
                    df_task_3_lang_1_dash,
                    df_task_3_country_1_dash,
                    df_task_3_awards_1,
                    df_task_3_released[, c('Released_Month'), drop=FALSE],
                    df_task_3[, c('Gross'), drop=FALSE])
```

3.2 Evaluation Strategy - Task 3 [Approach 1]

3.2.1 [RMSE values - Approach 1]

- **No. of Iterations :** 1000
- **Scenario 1 :** 5% Train data / 95% Test data
- **Scenario 2 :** 95% Train data / 5% Test data

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

rmse_3.4_a = train_test(df_task_3_a, 1000, 0.05)

paste("Multiple run 5% Train data / 95% Test data - Average Training RMSE : ",
      round(rmse_3.4_a$TRAIN_RMSE,0))
```

```
## [1] "Multiple run 5% Train data / 95% Test data - Average Training RMSE :  120565768"
```

```r
paste("Multiple run 5% Train data / 95% Test data - Average Test RMSE     : ",
      round(rmse_3.4_a$TEST_RMSE,0))
```

```
## [1] "Multiple run 5% Train data / 95% Test data - Average Test RMSE     :  161790458"
```

```r
rmse_3.4_a = train_test(df_task_3_a, 1000, 0.95)

print("")
```

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

```r
paste("Multiple run 95% Train data / 5% Test data - Average Training RMSE : ",
      round(rmse_3.4_a$TRAIN_RMSE,0))
```

```
## [1] "Multiple run 95% Train data / 5% Test data - Average Training RMSE :  139548243"
```

```
paste("Multiple run 95% Train data / 5% Test data - Average Test RMSE     : ",
      round(rmse_3.4_a$TEST_RMSE,0))
```

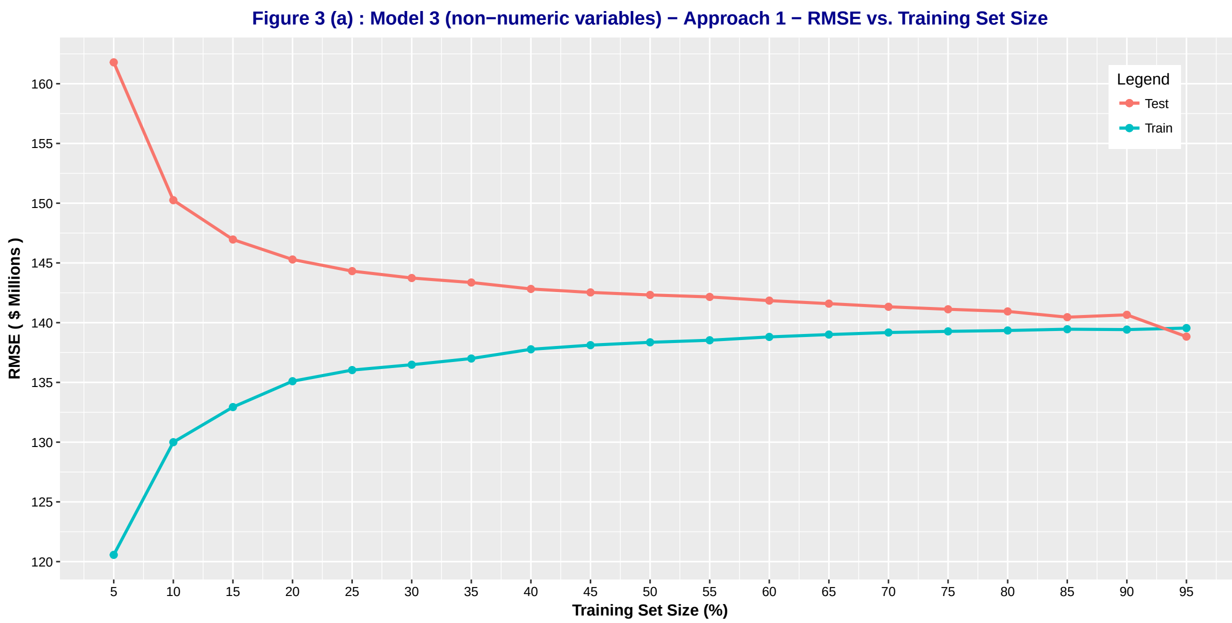
```
## [1] "Multiple run 95% Train data / 5% Test data - Average Test RMSE     :  138832167"
```

3.2.2 [RMSE Curves - Approach 1]

- **No. of Iterations :** 1000

```
set_sizes = seq(0.05, 0.95, by = 0.05)


TRAIN_RMSE_3.5_a = rep(0,19)
TEST_RMSE_3.5_a  = rep(0,19)

for (s in 1:length(set_sizes)){

  #print (s)

  rmse_3.5_a = train_test(df_task_3_a, 1000, set_sizes[s])

  TRAIN_RMSE_3.5_a[s] = rmse_3.5_a$TRAIN_RMSE
  TEST_RMSE_3.5_a[s] = rmse_3.5_a$TEST_RMSE

  }

#TRAIN_RMSE_3.5_a
#TEST_RMSE_3.5_a

RMSE_task_3_a = data.frame(TrainSet_Size = set_sizes * 100,
                                Train = TRAIN_RMSE_3.5_a,
                                Test = TEST_RMSE_3.5_a)

ggplot(data=RMSE_task_3_a) +                                                  # Initializing the plo

  geom_line(size = 1, aes(x = TrainSet_Size, y = Train, color = "Train")) +      # Line Plot

  geom_line(size = 1, aes(x = TrainSet_Size, y = Test, color = "Test")) +        # Line Plot

  geom_point(size = 2, aes(x = TrainSet_Size, y = Train, color = "Train")) +     # Point

  geom_point(size = 2, aes(x = TrainSet_Size, y = Test, color = "Test")) +       # Point

  labs(colour="Legend") +                                                      # Legend

  theme(legend.position = c(0.95, 0.95), legend.justification = c(1, 1)) +       # Legend position

  scale_y_continuous(breaks = seq(120000000, 165000000, 5000000),
                     labels = seq(120000000, 165000000, 5000000)/1000000) +      # Setting y-axis scale

  scale_x_continuous(breaks = seq(0, 100, 5)) +                               # Setting x-axis scale

  xlab('Training Set Size (%)') + ylab('RMSE ( $ Millions )') +               # Setting axes labels
```

```
# Setting the title
ggtitle("Figure 3 (a) : Model 3 (non-numeric variables) - Approach 1 - RMSE vs. Training Set Size") +

theme(plot.title = element_text(face = "bold",color = 'darkblue',              # Setting various them
                                hjust = 0.5)) +
theme(axis.text = element_text(colour = "black")) +
theme(axis.title.x = element_text(face = "bold", color='black'),
      axis.title.y = element_text(face = "bold", color='black')) +
theme(legend.key = element_rect(fill = "white"))
```



Observation : Figure 3(a) shows the RMSE curves for approach 1 and it can be seen that using numeric versions (binary and numeric) of the non-numeric features results in much higher RMSE values for both training and test sets.

3.2 Approach 2 [Most Correlated N approach]

3.2.1 Converting non-numeric variables to useful numeric variables [Most Correlated N approach]

For this apporach, to reduce RMSE further, I used a combination of some of the previously encoded variables from approach 1 and rest of them using this new approch. Instead of using top N most popular dicrectors, actors, etc (based on colSums/Frequency), I used top N most correlated approach for directors, actors, etc with 'Gross'.

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

df_task_3 = df


#################### RATED #####################################################


df_task_3_rated_2_dash = df_task_3_rated_1_dash
```

```r
#################### METASCORE ################################################

df_task_3_meta_2 = df_task_3_meta_1


#################### tomatoImage ###############################################

df_task_3_tomatoIm_2 = df_task_3_tomatoIm_1


#################### PRODUCTION ################################################

df_task_3_prod_2_dash =
  df_task_3_prod_1[,order(colSums(-df_task_3_prod_1,na.rm=TRUE))]

# Top N most correlated production houses

Top_prod_2 = Reduce('|',
      df_task_3_prod_2_dash[,cor(df_task_3$Gross, df_task_3_prod_2_dash) > 0.0])

Top_prod_2[Top_prod_2] = 1

df_task_3_prod_2_dash = data.frame(Top_Prod_2 = Top_prod_2)


#################### GENRES ####################################################


df_task_3_genre_2_dash =
  df_task_3_genre_1[,order(colSums(-df_task_3_genre_1,na.rm=TRUE))]

# only highly correlated top N

df_task_3_genre_2_dash =
  data.frame(Genre_Action_2 = df_task_3_genre_2_dash$Genre_Action,
             Genre_Adventure_2 = df_task_3_genre_2_dash$Genre_Adventure,
             Genre_Fantasy_2 = df_task_3_genre_2_dash$Genre_Fantasy,
             Genre_Sci_Fi_2 = df_task_3_genre_2_dash$'Genre_Sci-Fi',
             Genre_Animation_2 = df_task_3_genre_2_dash$Genre_Animation)


#################### DIRECTOR ##################################################

df_task_3_dir_2_dash = df_task_3_dir_1

# Top N most correlated directors

Top_dir_2 = Reduce('|',
    df_task_3_dir_2_dash[,cor(df_task_3$Gross, df_task_3_dir_2_dash) > 0.02])

Top_dir_2[Top_dir_2] = 1

df_task_3_dir_2_dash = data.frame(Top_Directors_2 = Top_dir_2 )
```

```r
#################### Writer ###################################################

df_task_3_writer_2_dash = df_task_3_writer_1

Top_writer_2 =
  Reduce('|', df_task_3_writer_2_dash[, cor(df_task_3$Gross,
                                             df_task_3_writer_2_dash) > 0.04])


Top_writer_2[Top_writer_2] = 1

df_task_3_writer_2_dash = data.frame(Top_Writers_2 = Top_writer_2 )


#################### ACTOR ###################################################

df_task_3_actor_2_dash = df_task_3_actor_1

Top_actor_2 =
  Reduce('|', df_task_3_actor_2_dash[,cor(df_task_3$Gross,
                                           df_task_3_actor_2_dash) > 0.07])


Top_actor_2[Top_actor_2] = 1

df_task_3_actor_2_dash = data.frame(Top_Actors_2 = Top_actor_2 )


#################### LANGUAGE ##############################################

df_task_3_lang_2_dash = df_task_3_lang_1

Other_lang_2 =
  Reduce('|', df_task_3_lang_2_dash[,cor(df_task_3$Gross,
                                          df_task_3_lang_2_dash) < 0.05])


Other_lang_2[Other_lang_2] = 1

df_task_3_lang_2_dash = data.frame(Other_Languages_2 = Other_lang_2 )


#################### COUNTRY ###############################################

df_task_3_country_2_dash = df_task_3_country_1

Other_Country_2 =
  Reduce('|', df_task_3_country_2_dash[,cor(df_task_3$Gross,
                                           df_task_3_country_2_dash) > 0.1])
```

```r
Other_Country_2[Other_Country_2] = 1

df_task_3_country_2_dash = data.frame(Other_Country_2 = Other_Country_2 )


#################### AWARDS ################################################


df_task_3_awards_2 = df_task_3_awards_1


#################### RELEASED ################################################

df_task_3_released_2 = df_task_3_released
```

3.2.2 Creating the dataframe based on Approach 2

```r
df_task_3_b = cbind(df_task_3_rated_2_dash,
                    df_task_3_meta_2,
                    df_task_3_tomatoIm_2,
                    df_task_3_prod_2_dash,
                    df_task_3_genre_2_dash,
                    df_task_3_dir_2_dash,
                    df_task_3_writer_2_dash,
                    df_task_3_actor_2_dash,
                    df_task_3_lang_2_dash,
                    df_task_3_country_2_dash,
                    df_task_3_awards_2,
                    df_task_3_released_2[, c('Released_Month'), drop=FALSE],
                    df_task_3[, c('Gross'), drop=FALSE])
```

3.3 Evaluation Strategy - Task 3 [Approach 2]

3.3.1 [RMSE Values - Approach 2]

- **No. of Iterations :** 1000
- **Scenario 1 :** 5% Train data / 95% Test data
- **Scenario 2 :** 95% Train data / 5% Test data

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

rmse_3.4_b = train_test(df_task_3_b, 1000, 0.05)

paste("Multiple run 5% Train data / 95% Test data - Average Training RMSE : ",
      round(rmse_3.4_b$TRAIN_RMSE,0))
```

```
## [1] "Multiple run 5% Train data / 95% Test data - Average Training RMSE :  92507888"
```

```r
paste("Multiple run 5% Train data / 95% Test data - Average Test RMSE     : ",
      round(rmse_3.4_b$TEST_RMSE,0))
```

```
## [1] "Multiple run 5% Train data / 95% Test data - Average Test RMSE     :  123018445"
```

```r
rmse_3.4_b = train_test(df_task_3_b, 1000, 0.95)

print("")
```

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

```r
paste("Multiple run 95% Train data / 5% Test data - Average Training RMSE : ",
      round(rmse_3.4_b$TRAIN_RMSE,0))
```

```
## [1] "Multiple run 95% Train data / 5% Test data - Average Training RMSE :  106838120"
```

```r
paste("Multiple run 95% Train data / 5% Test data - Average Test RMSE     : ",
      round(rmse_3.4_b$TEST_RMSE,0))
```

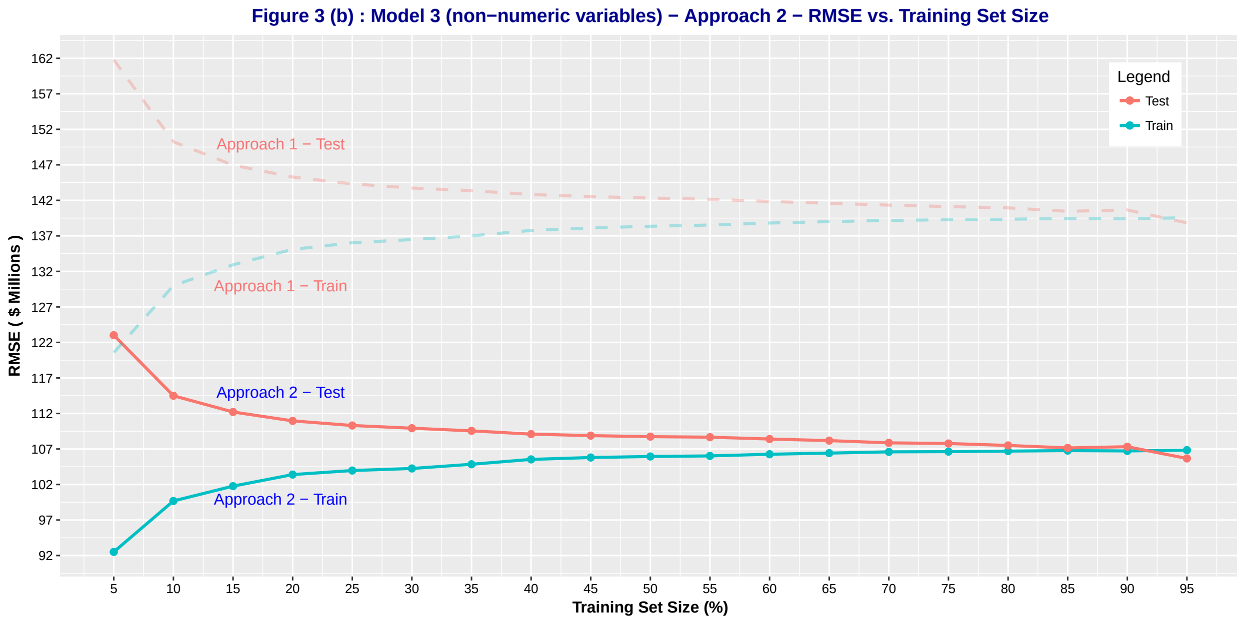
```
## [1] "Multiple run 95% Train data / 5% Test data - Average Test RMSE     :  105668002"
```

3.3.2 [RMSE Curves - Approach 2]

- **No. of Iterations :** 1000

```r
set_sizes = seq(0.05, 0.95, by = 0.05)


TRAIN_RMSE_3.5_b = rep(0,19)
TEST_RMSE_3.5_b  = rep(0,19)

for (s in 1:length(set_sizes)){

  #print (s)

  rmse_3.5_b = train_test(df_task_3_b, 1000, set_sizes[s])

  TRAIN_RMSE_3.5_b[s] = rmse_3.5_b$TRAIN_RMSE
  TEST_RMSE_3.5_b[s] = rmse_3.5_b$TEST_RMSE

  }

#TRAIN_RMSE_3.5_b
#TEST_RMSE_3.5_b

RMSE_task_3_b = data.frame(TrainSet_Size = set_sizes * 100,
                               Train = TRAIN_RMSE_3.5_b,
                               Test = TEST_RMSE_3.5_b,
                               Train_1 = TRAIN_RMSE_3.5_a,
                               Test_1 = TEST_RMSE_3.5_a)

ggplot(data=RMSE_task_3_b) +                                              # Initializing the plo

  geom_line(size = 1, aes(x = TrainSet_Size, y = Train, color = "Train")) +      # Line Plot

  geom_line(size = 1, aes(x = TrainSet_Size, y = Test, color = "Test")) +        # Line Plot

  geom_point(size = 2, aes(x = TrainSet_Size, y = Train, color = "Train")) +     # Point

  geom_point(size = 2, aes(x = TrainSet_Size, y = Test, color = "Test")) +       # Point
```

```r
geom_line(size = 1, linetype = 2,
          alpha = 0.3, aes(x = TrainSet_Size, y = Train_1, color = "Train")) + # Line Plot

geom_line(size = 1, linetype = 2,
          alpha = 0.3, aes(x = TrainSet_Size, y = Test_1, color = "Test")) +   # Line Plot


labs(colour="Legend") +                                                        # Legend

theme(legend.position = c(0.95, 0.95), legend.justification = c(1, 1)) +       # Legend position

scale_y_continuous(breaks = seq(92000000, 165000000, 5000000),
                   labels = seq(92000000, 165000000, 5000000)/1000000) +       # Setting y-axis scale

scale_x_continuous(breaks = seq(0, 100, 5)) +                                  # Setting x-axis scale

xlab('Training Set Size (%)') + ylab('RMSE ( $ Millions )') +                  # Setting axes labels

# Setting the title
ggtitle("Figure 3 (b) : Model 3 (non-numeric variables) - Approach 2 - RMSE vs. Training Set Size") +

theme(plot.title = element_text(face = "bold",color = 'darkblue',              # Setting various them
                                hjust = 0.5)) +
theme(axis.text = element_text(colour = "black")) +
theme(axis.title.x = element_text(face = "bold", color='black'),
      axis.title.y = element_text(face = "bold", color='black')) +
theme(legend.key = element_rect(fill = "white")) +
geom_text(aes(label= paste("Approach 1 - Test"),
              x = 19, y = 150000000), check_overlap = TRUE,
          color = "red", size = 4, alpha = 0.5) +
geom_text(aes(label= paste("Approach 1 - Train"),
              x = 19, y = 130000000), check_overlap = TRUE,
          color = "red", size = 4, alpha = 0.5) +
geom_text(aes(label= paste("Approach 2 - Test"),
              x = 19, y = 115000000), check_overlap = TRUE,
          color = "blue", size = 4) +
geom_text(aes(label= paste("Approach 2 - Train"),
              x = 19, y = 100000000), check_overlap = TRUE,
          color = "blue", size = 4)
```



Observation : It can clearly be seen that using feature construction based on approach 2 improved the RMSE for both the training and test sets drastically. The improvement is more than 38M for the test set RMSE while for the training set RMSE is around 32M.

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

A: There were 12 categorical variables used for this task and they are as follows:

- "Rated"
- "MetaScore"
- "tomatoImage"
- "Production"
- "Genre"
- "Director"
- "Writer"
- "Actors"
- "Language"
- "Country"
- "Awards"
- "Released"

I did not use "DVD" and "BoxOffice" as "DVD" had dates with 219 rows having NAs while "BoxOffice" only had 17 valid entries out of the total 2999 observations in the data frame. Complex variables such as "Plot", "Title", etc were also left out for task 5.

This task was conducted in 2 steps.

- Approach 1 - Top N approach
- Approach 2 - Most Correlated N approach

APPROACH 1

In this approach, most of the categorical variables used were based on one hot coding (using the 'concat.split.expanded' method from the "splitstackshape" package) and the top N most popular approach. As some of the variables (such as 'Director', 'Production', 'Writer', etc) had a lot columns after applying one hot coding, I limited the number of columns to top N (5, 10 or 20, etc) most popular (highest colSums) to be used for modeling. Following are the procedures used to encode these variables.

- **Rated :** After applying one hot coding, I created separate binary columns (just like columns created for genres in project 1) for all the different ratings. Further, I combined 'UNRATED' and 'NOT RATED' in a single column using logical OR. and then sorted the columns based on 'colSums' in descending order. This way I was able to pick top 3 columns (based on highest colSums) for model construction. These top 3 columns were "Rated_R", "Rated_PG-13" and "Rated_PG".
- **MetaScore :** This variable was compartively easier to encode as I first replaced the NAs with the median values of the rest of the valid entries in the feature vector and then I just converted the vector to numeric.
- **tomatoImage :** For this feature, I used one hot coding ('concat.split.expanded' method) same as "Rated" which left me with 3 encoded binary columns for this feature namely "tomatoImage_certified", "tomatoImage_fresh" and "tomatoImage_rotten".
- **Production :** Using one hot coding ('concat.split.expanded' method) I ended up with 485 binary columns for all the different production houses. In order to make the modeling more meaningful, I first sorted these columns from left to right based on 'colSums' (left most being the one with highest colSums). This way I could choose the top N easily but the problem faced here was that in many cases the main production house had used different names for different movies (for example : Warner Bros., Warner Bros. Pictures, Warners Bros. Pictures, WB, etc). To capture the correct information, I created a small function called 'prod_combine' which would take input a data frame and the short name of a production house and output the logical OR (using 'Reduce' function) of all the columns that represented that particular production house (for example : combing Warner Bros. and Warner Bros. Pictures into a single binary column where 1's represent the presenece of Warner Brothers). This way I combined the top 15 production houses into their respective main production house binary column. This gave me the correct count of the number of movies a particular production house produced. After sorting these 15 production houses by colSums, I decided to keep only top 5 that gave highest corr. coeff. with 'Gross'. These were "Warner Brothers", "20th Century Fox", "Sony", "Universal" and "Paramount".
- **Genre :** This feature was encoded the same way it was done for project 1. Again, one hot coding using ('concat.split.expanded' method) was performed and the binary columns were sorted based on highest colSums in descending order (from left to right). Then top 5 genres, namely "Drama", "Comedy", "Action", "Adventure" and "Romance" based highest frequency/colSums were chosen.
- **Director, Writer, Actors and Country :** All 4 features were encoded in the same fashion. For each feature, I first used one hot coding ('concat.split.expanded' method) just like I did for most of the features mentioned above, then I sorted the columns in descending order (left to right) based on 'colSums' and finally used logical OR to combine the top N directors, writers, Actors and Countries in each case into a single binary column. This was done using the 'Reduce' command : Reduce('|', df_task_3_country_1[, 1:5]). This way I was able to represent the presence of the top N most popular directors, writers, etc in a single binary column. To be specific, the binary columns I created combined top 400 directors into one binary column, top 450 writers in one binary column, top 400 actors in one binary column and top 5 countries in one binary column respectively. The number 'N' for the top 'N' was chosen based on best corr. coeff. I got with 'Gross' for that particular combination and I arrived at this trying different 'N's and honing onto the one that gave best r^2 with 'Gross'.
- **Language :** This feature was also converted to single binary column based on almost the same approach used to combine the 4 features above. The only difference was that after applying one hot coding ('concat.split.expanded' method) I used bottom N instead of top N as bottom N gave me better correlation with 'Gross'. After one hot coding I had 117 different columns representing different languages so I combined all the columns using the 'Reduce' command : Reduce('|', df_task_3_lang_1_dash[, 5:117]), i.e. all languages except the top 4. At leas tthis way I got positive r^2 with 'Gross' while using the top 4 or 5 languages was giving me negative r^2.
- **Awards :** 2 features were created of this variable, namely "Wins" and "Nominations". These were created using the same methodology as project 1. In order to convert all the textual based rows in

the ‘Awards’ column, I utilized ‘grepl()’ method and the ‘str_locate_all()’ method from the ‘stringr’ package. By performing quick analysis in MS Excel after removing duplicates I determined that entries in the ‘Awards’ column can be divide into 1 of 10 categories, which are as follows:

- ‘Nominated For’ + Other ‘Wins’ and ‘Nominations’ (eg: Nominated for 1 BAFTA Film Award. Another 1 win & 3 nominations.)
- ‘Won’ + Other ‘Wins’ and ‘Nominations’ (eg: Won 1 BAFTA Film Award. Another 3 wins & 6 nominations.)
- ‘Nominated For’ + Other ‘Nominations’ (eg: Nominated for 1 BAFTA Film Award. Another 1 nomination.)
- ‘Nominated For’ + Other ‘Wins’ (eg: Nominated for 1 Golden Globe. Another 2 wins.)
- ‘Won’ + Other ‘Nominations’ (eg: Won 1 BAFTA Film Award. Another 2 nominations.)
- ‘Won’ + Other ‘Wins’ (eg: Won 1 BAFTA Film Award. Another 4 wins.)
- ‘Wins’ and ‘Nominations’ (eg: 10 wins & 10 nominations.)
- ‘Wins’ (eg: 1 win.)
- ‘Nominations’ (eg: 10 nominations.)
- N/A

I created a function to handle all of the above (if-else) cases. In each case block, grepl() was used to identify the row with a particular case by looking for the key words such as ‘Nominated For’, ‘Wins’, ‘Nominations’, etc, then str_locate_all() was used to strip out the digit part followed by addition of appropriate category (Wins or Nominations) and then return them in string pairs. Examples: “4 5” for 4 Wins and 5 Nominations and respective median (Wins or Nominations) values for N/A values. Then using sapply(), I converted the complete Awards column into such string pairs. Then using the read.table command with space (" “) seperator, I converted this column containing the Win / Nomination string pairs into a dataframe with 2 numeric columns one for Wins and the other for Nominations.

Released : This variable was only used to create the ‘Released Month’ variable. I used simple substring command : as.numeric(substring(df_task_3_released$Released,6,7)) to strip out the month and converted that to numeric in order to utilize this as a feature to see if released month had an affect in particular on the RMSE.

APPROACH 2

In order to reduce RMSE further, instead of just using top N most popular directors, actors, etc and combining them into one binary column, I used top N most correlated directors, actors, etc with ‘Gross’ and combined them in a single binary column. Most of the features were still kept the same as approach 1 while rest of them were encoded using this new approach.

- **Rated, MetaScore, tomatoImage, Awards (Wins, Nominations) and Released(Released Month) :** All of these features were kept same as approach 1.
- **Production, Director, Writer, Actors and Country :** After applying one hot coding I used the ‘Reduce’ command [example from production encoding : Reduce(‘|’, df_task_3_prod[,cor(df_task_3$Gross, df_task_3_prod) > 0.0])] to combine all production, director, etc columns that had corr. coeff. greater than a certain threshold with ‘Gross’ into a single binary column based on logical OR. This binary column had better corr. coeff. with ‘Gross’ in comparison to the top N production, director, etc houses columns used in approach 1. This way I ended up with 5 binary columns one for each Production, Director, Writer, Actors and Country. The thresholds used for corr. coeff. were 0, 0.02, 0.04, 0.07 and 0.1 respectively. I arrived at these thresholds by staring of with 0 and then honing on to the threshold gave best corr. coeff. with ‘Gross’. The specific ‘Reduce’ commands were as follows:
- Reduce(‘|’, df_task_3_prod_2_dash[,cor(df_task_3$Gross, df_task_3_prod_2_dash) > 0.0])

- Reduce('|', df_task_3_dir_2_dash[,cor(df_task_3$Gross, df_task_3_dir_2_dash) > 0.02])
- Reduce('|', df_task_3_writer_2_dash[, cor(df_task_3$Gross, df_task_3_writer_2_dash) > 0.04])
- Reduce('|', df_task_3_actor_2_dash[,cor(df_task_3$Gross, df_task_3_actor_2_dash) > 0.07])
- Reduce('|', df_task_3_country_2_dash[,cor(df_task_3$Gross, df_task_3_country_2_dash) > 0.1])

And the number of most correlated binary columns combined in each case are as follows:

- **Production :** 73
- **Director :** 152
- **Writer :** 364
- **Actors :** 71
- **Country :** 2
- **Genre :** For Genre, I took a similar approach too. After one hot coding instead of choosing the top 5 most popular genres as I did in approach 1, I chose the top 5 most correlated genres with 'Gross'. These were "Action", "Adventure", "Fantasy", "Sci Fi" and "Animation". Based on on better corr. coeff. with 'Gross', "Fantasy", "Sci Fi" and "Animation" replaced "Drama", "Comedy" and "Romance" from approach 1 while "Action" and "Adventure" remained consistent.
- **Language :** As for approach 1 I had used bottom N least popular languages to be combined in a single binary column using 'Reduce'. Here again I used the bottom N languages that gave corr. coeff. less than 0.05 with 'Gross'. This approach gave a low but positive r^2 so I decided to use this as a feature. The number of different lanuages that got reduced to a single binary column were 103.

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**.

4.1 Using only task 2 and task 3 features

4.1.1 Creating a data frame using only task 2 and task 3 features

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

df_task_4_a = cbind(subset(df_task_3_b, select = -c(Gross)),
                    subset(df_task_2, select = -c(tomatoRotten)))
```

4.2 Evaluation Strategy - Task 4 [Combining only task 2 and task 3 features]

4.2.1 [RMSE Values - task 2 and task 3 features]

- **No. of Iterations :** 1000
- **Scenario 1 :** 5% Train data / 95% Test data
- **Scenario 2 :** 95% Train data / 5% Test data

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

rmse_4.4_a = train_test(df_task_4_a, 1000, 0.05)
```

```r
paste("Multiple run 5% Train data / 95% Test data - Average Training RMSE : ",
      round(rmse_4.4_a$TRAIN_RMSE,0))
```

```
## [1] "Multiple run 5% Train data / 95% Test data - Average Training RMSE :  65150083"
```

```r
paste("Multiple run 5% Train data / 95% Test data - Average Test RMSE     : ",
      round(rmse_4.4_a$TEST_RMSE,0))
```

```
## [1] "Multiple run 5% Train data / 95% Test data - Average Test RMSE     :  104280756"
```

```r
rmse_4.4_a = train_test(df_task_4_a, 1000, 0.95)

print("")
```

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

```r
paste("Multiple run 95% Train data / 5% Test data - Average Training RMSE : ",
      round(rmse_4.4_a$TRAIN_RMSE,0))
```

```
## [1] "Multiple run 95% Train data / 5% Test data - Average Training RMSE :  84063221"
```

```r
paste("Multiple run 95% Train data / 5% Test data - Average Test RMSE     : ",
      round(rmse_4.4_a$TEST_RMSE,0))
```

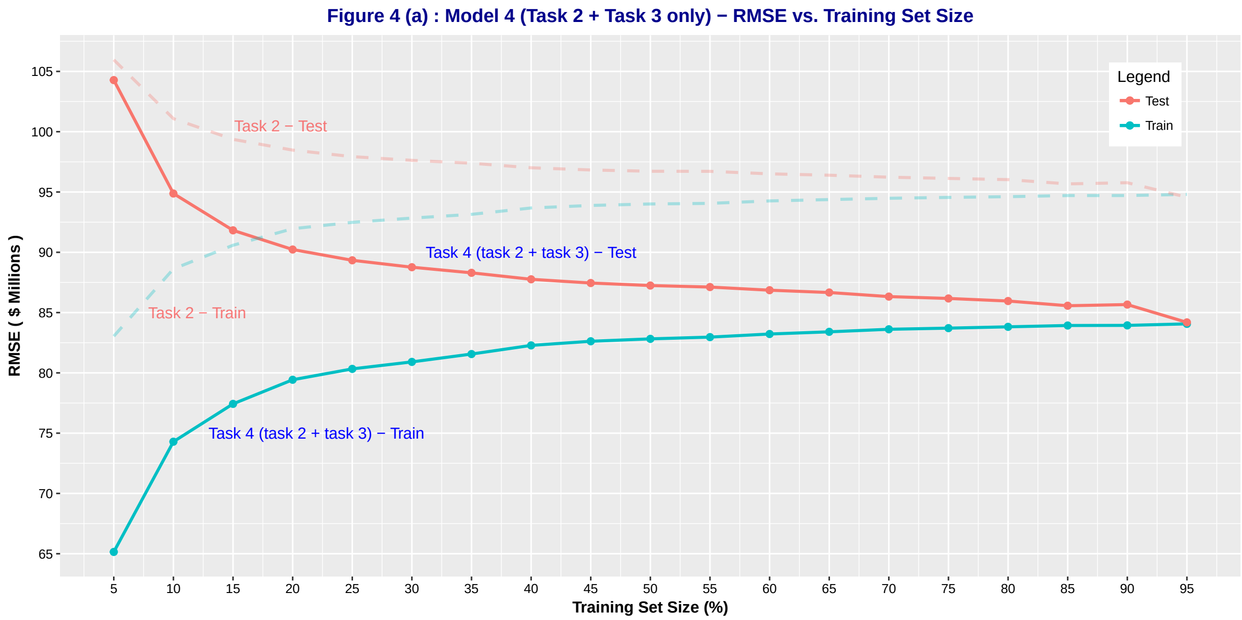
```
## [1] "Multiple run 95% Train data / 5% Test data - Average Test RMSE     :  84187606"
```

4.2.2 [RMSE Curves - task 2 and task 3 features]

- **No. of Iterations :** 1000

```r
set_sizes = seq(0.05, 0.95, by = 0.05)


TRAIN_RMSE_4.5_a = rep(0,19)
TEST_RMSE_4.5_a  = rep(0,19)

for (s in 1:length(set_sizes)){

  #print (s)

  rmse_4.5_a = train_test(df_task_4_a, 1000, set_sizes[s])

  TRAIN_RMSE_4.5_a[s] = rmse_4.5_a$TRAIN_RMSE
  TEST_RMSE_4.5_a[s] = rmse_4.5_a$TEST_RMSE

  }

#TRAIN_RMSE_4.5_a
#TEST_RMSE_4.5_a

RMSE_task_4_a = data.frame(TrainSet_Size = set_sizes * 100,
                               Train = TRAIN_RMSE_4.5_a,
                               Test = TEST_RMSE_4.5_a,
                               Train_1 = TRAIN_RMSE_2.5,
                               Test_1 = TEST_RMSE_2.5)

ggplot(data=RMSE_task_4_a) +                                          # Initializing the plo
```

```r
geom_line(size = 1, aes(x = TrainSet_Size, y = Train, color = "Train")) +      # Line Plot

geom_line(size = 1, aes(x = TrainSet_Size, y = Test, color = "Test")) +        # Line Plot

geom_point(size = 2, aes(x = TrainSet_Size, y = Train, color = "Train")) +     # Point

geom_point(size = 2, aes(x = TrainSet_Size, y = Test, color = "Test")) +       # Point

geom_line(size = 1, linetype = 2,
          alpha = 0.3, aes(x = TrainSet_Size, y = Train_1, color = "Train")) + # Line Plot

geom_line(size = 1, linetype = 2,
          alpha = 0.3, aes(x = TrainSet_Size, y = Test_1, color = "Test")) +   # Line Plot

labs(colour="Legend") +                                                        # Legend

theme(legend.position = c(0.95, 0.95), legend.justification = c(1, 1)) +       # Legend position

scale_y_continuous(breaks = seq(65000000, 110000000, 5000000),
                   labels = seq(65000000, 110000000, 5000000)/1000000) +       # Setting y-axis scale

scale_x_continuous(breaks = seq(0, 100, 5)) +                                  # Setting x-axis scale

xlab('Training Set Size (%)') + ylab('RMSE ( $ Millions )') +                  # Setting axes labels

# Setting the title
ggtitle("Figure 4 (a) : Model 4 (Task 2 + Task 3 only) - RMSE vs. Training Set Size") +

theme(plot.title = element_text(face = "bold",color = 'darkblue',              # Setting various them
                                hjust = 0.5)) +
theme(axis.text = element_text(colour = "black")) +
theme(axis.title.x = element_text(face = "bold", color='black'),
      axis.title.y = element_text(face = "bold", color='black')) +
theme(legend.key = element_rect(fill = "white"))+
geom_text(aes(label= paste("Task 2 - Test"),
              x = 19, y = 100500000), check_overlap = TRUE,
          color = "red", size = 4, alpha = 0.5) +
geom_text(aes(label= paste("Task 2 - Train"),
              x = 12, y = 85000000), check_overlap = TRUE,
          color = "red", size = 4, alpha = 0.5) +
geom_text(aes(label= paste("Task 4 (task 2 + task 3) - Test"),
              x = 40, y = 90000000), check_overlap = TRUE,
          color = "blue", size = 4) +
geom_text(aes(label= paste("Task 4 (task 2 + task 3) - Train"),
              x = 22, y = 75000000), check_overlap = TRUE,
          color = "blue", size = 4)
```



Observation : In Figure 4(a), it can clearly be seen that just by combining features used in task 2 (Binned + Numeric Transforms) and task 3 (Approach 2) there is significant improvement in RMSE for both training and test sets. Dashed curves from task 2 are also shown for reference.

4.3 Adding transformations + few original features to improve RMSE

4.3.1 Performing transformation on non-binary features [MetaScore, Wins, etc]

```
#head(df_task_4_2)

selected_features = c('Metascore', 'Wins', 'Nominations', 'Released_Month')

for (i in 1:length(selected_features)){

  print(paste(selected_features[i], transformation(df_task_1$Gross, df_task_4_a[[selected_features[i]]]
}
```

```
## [1] "Metascore Best Transform --> Binning : No. of bins = 7 that gives r^2 of 0.198602 with 'Gross'"
## [1] "Wins Best Transform --> Numeric Transform = power(1/2) that gives r^2 of 0.316988 with 'Gross'"
## [1] "Nominations Best Transform --> Numeric Transform = Log that gives r^2 of 0.403966 with 'Gross'"
## [1] "Released_Month Best Transform --> Numeric Transform = Exp that gives r^2 of 0.097355 with 'Gros
```

NOTE : Here I have used transformations for 4 of the numeric (non-binary) features that I encoded (which were originally not numeric) in task 3 namely 'MetaScore', 'Wins', 'Nominations' and 'Released_Month'

4.3.2 Renaming features in order to avoid colnames clash while combining into a single data frame.

```
df_task_1_dash = df_task_1

names(df_task_1_dash) = c("Year_o", "Runtime_o", "imdbRating_o", "imdbVotes_o",
                          "tomatoRating_o", "tomatoReviews_o", "tomatoFresh_o",
                       "tomatoRotten_o", "tomatoUserRating_o", "tomatoUserReviews_o",
                          "Budget_o", "Gross")
```

4.3.3 Creating a data frame combing few task 1 features, complete task 2 and task 3 features and 4 recently transformed features

```
df_task_4_b = cbind(df_task_4_a,
                    Metascore_t = as.numeric(cut(df_task_4_a$Metascore, 7, labels = c(1:7))),
                    Wins_t = (df_task_4_a$Wins)^0.5,
                    Nominations_t = log(df_task_4_a$Nominations+ 1),
                    Released_Month_t = exp(df_task_4_a$Released_Month),
                    Runtime_o = df_task_1_dash$Runtime_o,
                    Budget_o = df_task_1_dash$Budget_o,
                    imdbRating_o = df_task_1_dash$imdbRating_o,
                    tomatoReviews_o = df_task_1_dash$tomatoReviews_o)
```

4.4 Evaluation Strategy - Task 4 [task 2 + task 3 + few transformed features + few original numeric features]

4.4.1 [RMSE Values - task 2 + task 3 + few transformed features + few original numeric features]

- **No. of Iterations :** 1000
- **Scenario 1 :** 5% Train data / 95% Test data
- **Scenario 2 :** 95% Train data / 5% Test data

```
rmse_4.4_b = train_test(df_task_4_b, 1000, 0.05)

paste("Multiple run 5% Train data / 95% Test data - Average Training RMSE : ",
      round(rmse_4.4_b$TRAIN_RMSE,0))
```

```
## [1] "Multiple run 5% Train data / 95% Test data - Average Training RMSE :  61714710"
```

```
paste("Multiple run 5% Train data / 95% Test data - Average Test RMSE     : ",
      round(rmse_4.4_b$TEST_RMSE,0))
```

```
## [1] "Multiple run 5% Train data / 95% Test data - Average Test RMSE     :  107300894"
```

```
rmse_4.4_b = train_test(df_task_4_b, 1000, 0.95)

print("")
```

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

```
paste("Multiple run 95% Train data / 5% Test data - Average Training RMSE : ",
      round(rmse_4.4_b$TRAIN_RMSE,0))
```

```
## [1] "Multiple run 95% Train data / 5% Test data - Average Training RMSE :  82897752"
```

```
paste("Multiple run 95% Train data / 5% Test data - Average Test RMSE     : ",
      round(rmse_4.4_b$TEST_RMSE,0))
```

```
## [1] "Multiple run 95% Train data / 5% Test data - Average Test RMSE     :  83371052"
```

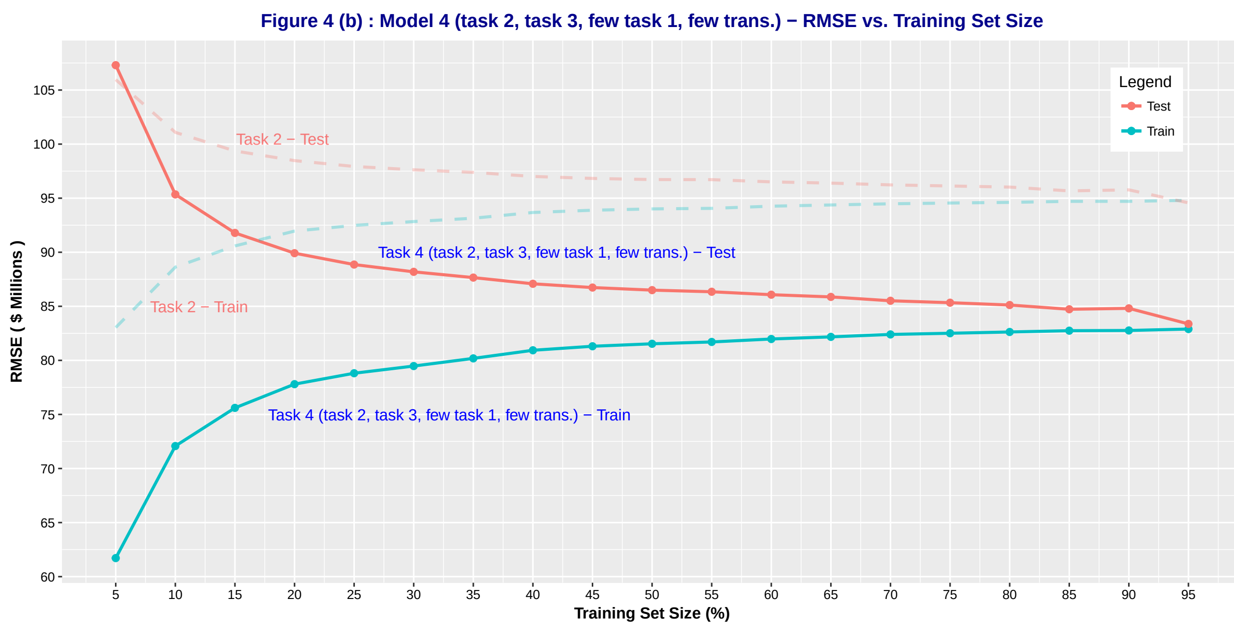
4.4.2 [RMSE Curves - task 2 + task 3 + few transformed features + few original numeric features]

- **No. of Iterations :** 1000

```
set_sizes = seq(0.05, 0.95, by = 0.05)
```

```r
TRAIN_RMSE_4.5_b = rep(0,19)
TEST_RMSE_4.5_b  = rep(0,19)

for (s in 1:length(set_sizes)){

  #print (s)

  rmse_4.5_b = train_test(df_task_4_b, 1000, set_sizes[s])

  TRAIN_RMSE_4.5_b[s] = rmse_4.5_b$TRAIN_RMSE
  TEST_RMSE_4.5_b[s] = rmse_4.5_b$TEST_RMSE

  }

#TRAIN_RMSE_4.5_b
#TEST_RMSE_4.5_b

RMSE_task_4_b = data.frame(TrainSet_Size = set_sizes * 100,
                                Train = TRAIN_RMSE_4.5_b,
                                Test = TEST_RMSE_4.5_b,
                                Train_1 = TRAIN_RMSE_2.5,
                                Test_1 = TEST_RMSE_2.5)

ggplot(data=RMSE_task_4_b) +                                                 # Initializing the plo

  geom_line(size = 1, aes(x = TrainSet_Size, y = Train, color = "Train")) +      # Line Plot

  geom_line(size = 1, aes(x = TrainSet_Size, y = Test, color = "Test")) +        # Line Plot

  geom_point(size = 2, aes(x = TrainSet_Size, y = Train, color = "Train")) +     # Point

  geom_point(size = 2, aes(x = TrainSet_Size, y = Test, color = "Test")) +       # Point

  geom_line(size = 1, linetype = 2,
            alpha = 0.3, aes(x = TrainSet_Size, y = Train_1, color = "Train")) + # Line Plot

  geom_line(size = 1, linetype = 2,
            alpha = 0.3, aes(x = TrainSet_Size, y = Test_1, color = "Test")) +   # Line Plot

  labs(colour="Legend") +                                                    # Legend

  theme(legend.position = c(0.95, 0.95), legend.justification = c(1, 1)) +       # Legend position

  scale_y_continuous(breaks = seq(60000000, 110000000, 5000000),
                     labels = seq(60000000, 110000000, 5000000)/1000000) +       # Setting y-axis scale

  scale_x_continuous(breaks = seq(0, 100, 5)) +                              # Setting x-axis scale

  xlab('Training Set Size (%)') + ylab('RMSE ( $ Millions )') +              # Setting axes labels

  # Setting the title
  ggtitle("Figure 4 (b) : Model 4 (task 2, task 3, few task 1, few trans.) - RMSE vs. Training Set Size
```

```
  theme(plot.title = element_text(face = "bold",color = 'darkblue',                # Setting various them
                                  hjust = 0.5)) +
  theme(axis.text = element_text(colour = "black")) +
  theme(axis.title.x = element_text(face = "bold", color='black'),
        axis.title.y = element_text(face = "bold", color='black')) +
  theme(legend.key = element_rect(fill = "white"))+
  geom_text(aes(label= paste("Task 2 - Test"),
                x = 19, y = 100500000), check_overlap = TRUE,
            color = "red", size = 4, alpha = 0.5) +
  geom_text(aes(label= paste("Task 2 - Train"),
                x = 12, y = 85000000), check_overlap = TRUE,
            color = "red", size = 4, alpha = 0.5) +
  geom_text(aes(label= paste("Task 4 (task 2, task 3, few task 1, few trans.) - Test"),
                x = 42, y = 90000000), check_overlap = TRUE,
            color = "blue", size = 4) +
  geom_text(aes(label= paste("Task 4 (task 2, task 3, few task 1, few trans.) - Train"),
                x = 33, y = 75000000), check_overlap = TRUE,
            color = "blue", size = 4)
```



Observation : For further improvement in RMSE, I included few of the original numeric features namely "Runtime", "Budget", "imdbRating" and "tomatoReviews" to the data frame used for Figure 4(a). Along with these original features I transformed few of the non-binary numeric features used in task 3 namely transformed "Metascore" with 7 bins, square root of "Wins", log transformed "Nominations" and exp of "Released Month". This improved the RMSE values for both training and test sets as can be seen in Figure 4(b) above.

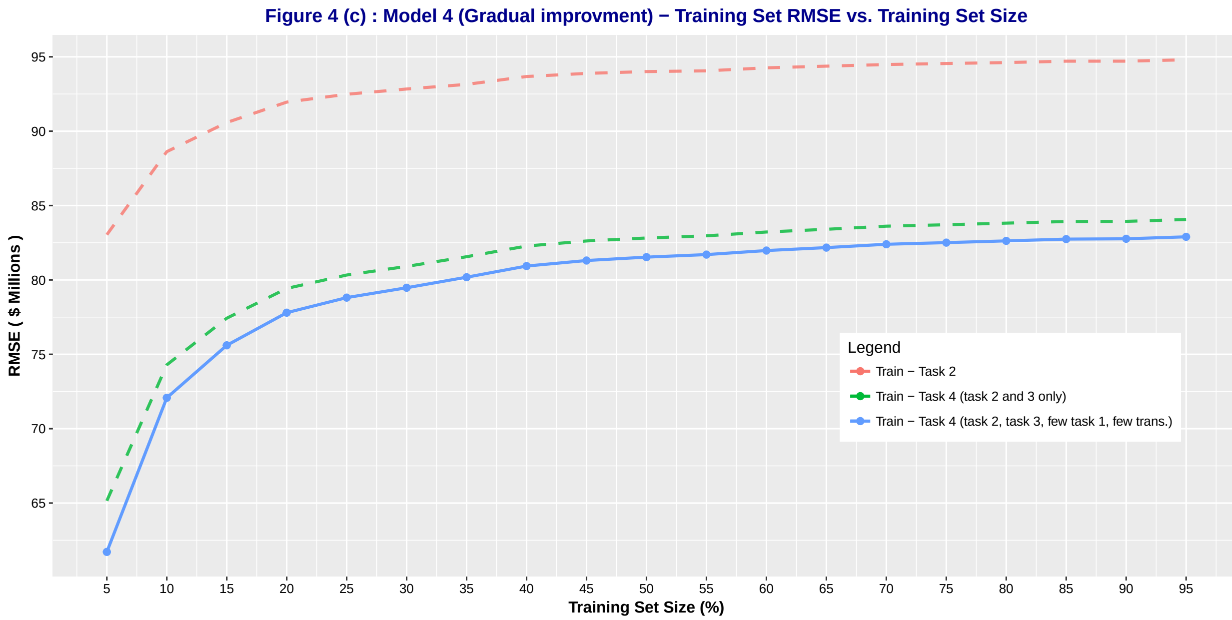
4.5 Evaluation Strategy - Task 4 [Showing improvement progressively]

4.5.1 [RMSE TRAINING Curves - Showing improvement progressively]

- **No. of Iterations :** 1000

```
RMSE_task_4_train = data.frame(TrainSet_Size = set_sizes * 100,
                                Train = TRAIN_RMSE_4.5_b,
```

```
                                Train_1 = TRAIN_RMSE_4.5_a,
                                Train_2 = TRAIN_RMSE_2.5)

ggplot(data=RMSE_task_4_train) +                                                # Initializing the plo

  geom_line(size = 1,
            aes(x = TrainSet_Size, y = Train,
                color = "Train - Task 4 (task 2, task 3, few task 1, few trans.)")) +  # Line Plot

  geom_point(size = 2,
             aes(x = TrainSet_Size, y = Train,
                 color = "Train - Task 4 (task 2, task 3, few task 1, few trans.)")) + # Point

  geom_line(size = 1, linetype = 2,
            alpha = 0.8, aes(x = TrainSet_Size, y = Train_1,
                             color = "Train - Task 4 (task 2 and 3 only)")) +    # Line Plot

  geom_line(size = 1, linetype = 2,
            alpha = 0.8, aes(x = TrainSet_Size, y = Train_2,
                             color = "Train - Task 2")) +                        # Line Plot


  labs(colour="Legend") +                                                       # Legend

  theme(legend.position = c(0.95, 0.45), legend.justification = c(1, 1)) +      # Legend position

  scale_y_continuous(breaks = seq(60000000, 95000000, 5000000),
                     labels = seq(60000000, 95000000, 5000000)/1000000) +       # Setting y-axis scale

  scale_x_continuous(breaks = seq(0, 100, 5)) +                                 # Setting x-axis scale

  xlab('Training Set Size (%)') + ylab('RMSE ( $ Millions )') +                 # Setting axes labels

  # Setting the title
  ggtitle("Figure 4 (c) : Model 4 (Gradual improvment) - Training Set RMSE vs. Training Set Size") +

  theme(plot.title = element_text(face = "bold",color = 'darkblue',             # Setting various them
                                  hjust = 0.5)) +
  theme(axis.text = element_text(colour = "black")) +
  theme(axis.title.x = element_text(face = "bold", color='black'),
        axis.title.y = element_text(face = "bold", color='black')) +
  theme(legend.key = element_rect(fill = "white"))
```



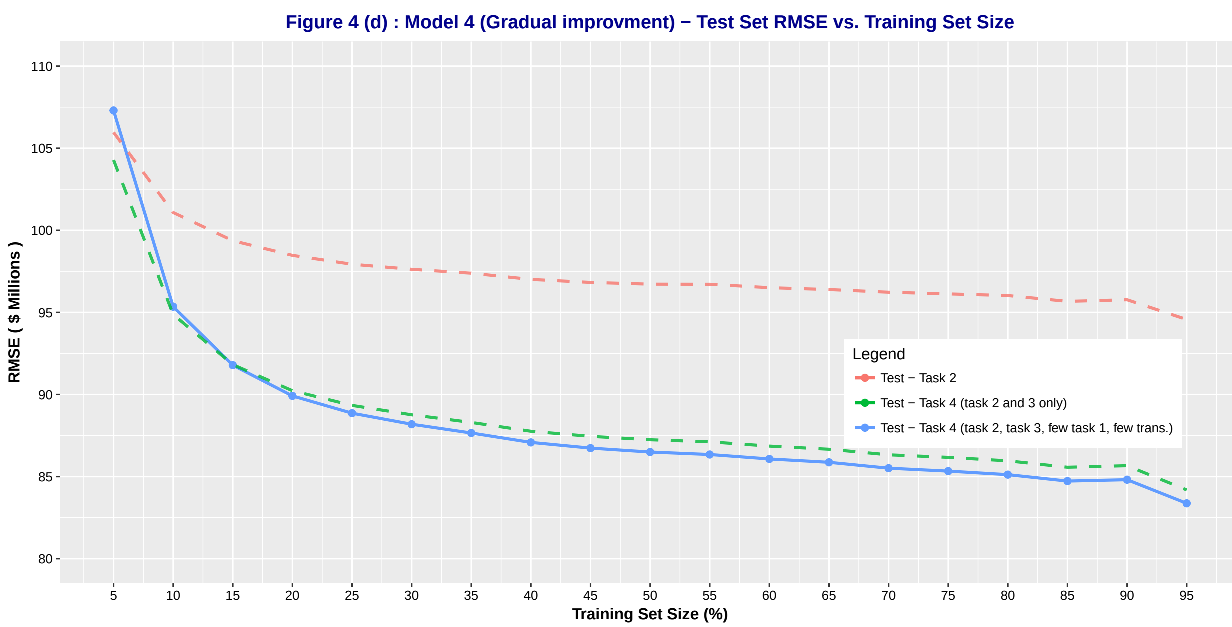
Observation : It can clearly be seen that adding few original numeric features from task 1 and few of the transforming non-binary numeric features from task 3, RMSE for the training set was reduced for all training set sizes.

4.5 and 4.6 [RMSE TEST Curves - Gradual Improvement]

- **No. of Iterations :** 1000

```
RMSE_task_4_test = data.frame(TrainSet_Size = set_sizes * 100,
                                Test = TEST_RMSE_4.5_b,
                                Test_1 = TEST_RMSE_4.5_a,
                                Test_2 = TEST_RMSE_2.5)

ggplot(data=RMSE_task_4_test) +                                                    # Initializing the pl

  geom_line(size = 1,
            aes(x = TrainSet_Size, y = Test,
                color = "Test - Task 4 (task 2, task 3, few task 1, few trans.)")) +  # Line Plot

  geom_point(size = 2,
             aes(x = TrainSet_Size, y = Test,
                 color = "Test - Task 4 (task 2, task 3, few task 1, few trans.)")) + # Point

  geom_line(size = 1, linetype = 2,
            alpha = 0.8, aes(x = TrainSet_Size, y = Test_1,
                             color = "Test - Task 4 (task 2 and 3 only)")) +        # Line Plot

  geom_line(size = 1, linetype = 2,
            alpha = 0.8, aes(x = TrainSet_Size, y = Test_2,
                             color = "Test - Task 2")) +           # Line Plot

  labs(colour="Legend") +                                                           # Legend

  theme(legend.position = c(0.95, 0.45), legend.justification = c(1, 1)) +        # Legend position
```

```
scale_y_continuous(breaks = seq(80000000, 110000000, 5000000),
                   labels = seq(80000000, 110000000, 5000000)/1000000,
                   limits = c(80000000, 110000000)) +                    # Setting y-axis scal

scale_x_continuous(breaks = seq(0, 100, 5)) +                              # Setting x-axis scal

xlab('Training Set Size (%)') + ylab('RMSE ( $ Millions )') +              # Setting axes labels

# Setting the title
ggtitle("Figure 4 (d) : Model 4 (Gradual improvment) - Test Set RMSE vs. Training Set Size") +

theme(plot.title = element_text(face = "bold",color = 'darkblue',          # Setting various the
                                hjust = 0.5)) +
theme(axis.text = element_text(colour = "black")) +
theme(axis.title.x = element_text(face = "bold", color='black'),
      axis.title.y = element_text(face = "bold", color='black')) +
theme(legend.key = element_rect(fill = "white"))
```



Observation : Similar to Figure 4(c), RMSE for testing set was also reduced when a combination of features were used to train the model. Although, the RMSE for larger test set sizes (i.e. for test set size greater than 85%) did deteriorate but overall for smaller set sizes (i.e. less than 80%) the test RMSE decreased considerably.

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`).

5.1 Working with Full Text Columns

5.1.1 Extracting useful features using from full text columns

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

# https://rstudio-pubs-static.s3.amazonaws.com/132792_864e3813b0ec47cb95c7e1e2e2ad83e7.html

df_task_5_plot = subset(df, select = c(Plot))
df_task_5_title = subset(df, select = c(Title))
df_task_5_consensus = subset(df, select = c(tomatoConsensus))


plot_corpus = Corpus(VectorSource(df_task_5_plot$Plot))
title_corpus = Corpus(VectorSource(df_task_5_title$Title))
consensus_corpus = Corpus(VectorSource(df_task_5_consensus$tomatoConsensus))

# pre-processing steps:

# 1. Switch to lower case
# 2. Remove numbers
# 3. Remove punctuation marks and stopwords
# 4. Remove extra whitespaces

plot_corpus = tm_map(plot_corpus, content_transformer(tolower))
plot_corpus = tm_map(plot_corpus, removeNumbers)
plot_corpus = tm_map(plot_corpus, removePunctuation)
plot_corpus = tm_map(plot_corpus, removeWords,
                     c("the", "and", stopwords("english")))
plot_corpus = tm_map(plot_corpus, stripWhitespace)

title_corpus = tm_map(title_corpus, content_transformer(tolower))
title_corpus = tm_map(title_corpus, removeNumbers)
title_corpus = tm_map(title_corpus, removePunctuation)
title_corpus = tm_map(title_corpus, removeWords,
                      c("the", "and", stopwords("english")))
title_corpus = tm_map(title_corpus, stripWhitespace)

consensus_corpus = tm_map(consensus_corpus, content_transformer(tolower))
consensus_corpus = tm_map(consensus_corpus, removeNumbers)
consensus_corpus = tm_map(consensus_corpus, removePunctuation)
consensus_corpus = tm_map(consensus_corpus, removeWords,
                          c("the", "and", stopwords("english")))
consensus_corpus = tm_map(consensus_corpus, stripWhitespace)

# using a Document-Term Matrix (DTM) representation
# usinf tf-idf to remove less frequent terms such that the sparsity is less than 0.99

plot_dtm_tfidf = DocumentTermMatrix(plot_corpus,
                                    control = list(weighting = weightTfIdf))
plot_dtm_tfidf = removeSparseTerms(plot_dtm_tfidf, 0.99)

title_dtm_tfidf = DocumentTermMatrix(title_corpus,
                                     control = list(weighting = weightTfIdf))
title_dtm_tfidf = removeSparseTerms(title_dtm_tfidf, 0.999)

consensus_dtm_tfidf = DocumentTermMatrix(consensus_corpus,
```

```
                                          control = list(weighting = weightTfIdf))
consensus_dtm_tfidf = removeSparseTerms(consensus_dtm_tfidf, 0.99)


#################### PLOT ######################################################

plot_dtm_tfidf = as.matrix(plot_dtm_tfidf)
df_5_plot = data.frame(plot_dtm_tfidf)

top_df_plot = Reduce('|',
      df_5_plot[,cor(df_task_3$Gross, df_5_plot) > 0.065])

top_df_plot[top_df_plot] = 1

plot_prop = colMeans(df_5_plot[,cor(df_task_3$Gross, df_5_plot) > 0.065])
plot_prop = data.frame(plot_prop)
plot_prop$Word = rownames(plot_prop)

gg_plot = ggplot(plot_prop,aes(Word,plot_prop)) +
  geom_bar(stat="identity", alpha = 0.8) + coord_flip() +

  xlab('Words') + ylab('Average relative improtance (based on tf-idf)') +

  ggtitle("Figure 5 (a) : Avg. relative improtance of words in 'Plot' column") +

  theme(plot.title = element_text(face = "bold",color = 'darkblue',
                                  hjust = 0.5)) +
  theme(axis.text = element_text(colour = "black")) +
  theme(axis.title.x = element_text(face = "bold", color='black'),
        axis.title.y = element_text(face = "bold", color='black')) +
  theme(legend.key = element_rect(fill = "white"))


#################### TITLE #####################################################

title_dtm_tfidf = as.matrix(title_dtm_tfidf)
df_5_title = data.frame(title_dtm_tfidf)

top_df_title = Reduce('|',
      df_5_title[,cor(df_task_3$Gross, df_5_title) > 0.061])

top_df_title[top_df_title] = 1

title_prop = colMeans(df_5_title[,cor(df_task_3$Gross, df_5_title) > 0.061])
title_prop = data.frame(title_prop)
title_prop$Word = rownames(title_prop)

gg_title = ggplot(title_prop,aes(Word,title_prop)) +
  geom_bar(stat="identity", alpha = 0.8) + coord_flip() +

  xlab('Words') + ylab('Average relative improtance (based on tf-idf)') +

  ggtitle("Figure 5 (c) : Avg. relative improtance of words in 'Title' column") +
```

```r
  theme(plot.title = element_text(face = "bold",color = 'darkblue',
                                  hjust = 0.5)) +
  theme(axis.text = element_text(colour = "black")) +
  theme(axis.title.x = element_text(face = "bold", color='black'),
        axis.title.y = element_text(face = "bold", color='black')) +
  theme(legend.key = element_rect(fill = "white"))


#################### CONCENSUS #################################################

consensus_dtm_tfidf = as.matrix(consensus_dtm_tfidf)
df_5_consensus = data.frame(consensus_dtm_tfidf)

top_df_consensus = Reduce('|',
      df_5_consensus[,cor(df_task_3$Gross, df_5_consensus) > 0.08])

top_df_consensus[top_df_consensus] = 1

consensus_prop =
  colMeans(df_5_consensus[,cor(df_task_3$Gross, df_5_consensus) > 0.08])
consensus_prop = data.frame(consensus_prop)
consensus_prop$Word = rownames(consensus_prop)

gg_concensus = ggplot(consensus_prop,aes(Word,consensus_prop)) +
  geom_bar(stat="identity", alpha = 0.8) + coord_flip() +

  xlab('Words') + ylab('Average relative improtance (based on tf-idf)') +

  ggtitle("Figure 5 (b) : Avg. relative improtance of words in 'Concensus' column") +

  theme(plot.title = element_text(face = "bold",color = 'darkblue',
                                  hjust = 0.5)) +
  theme(axis.text = element_text(colour = "black")) +
  theme(axis.title.x = element_text(face = "bold", color='black'),
        axis.title.y = element_text(face = "bold", color='black')) +
  theme(legend.key = element_rect(fill = "white"))

grid.arrange(grid.arrange(gg_plot, gg_concensus, nrow = 2), gg_title, heights=c(1/2, 1/2), nrow = 2)
```

Figure 5 (a) : Avg. relative improtance of words in 'Plot' column

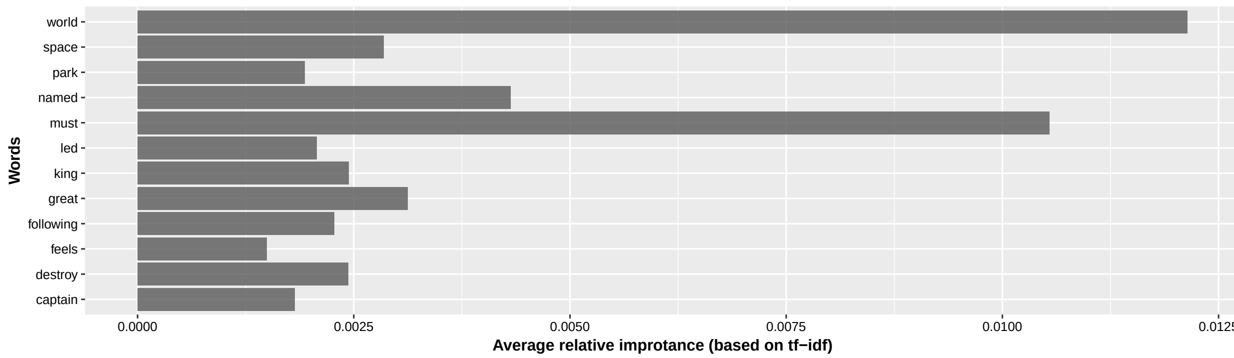


Figure 5 (b) : Avg. relative improtance of words in 'Concensus' column

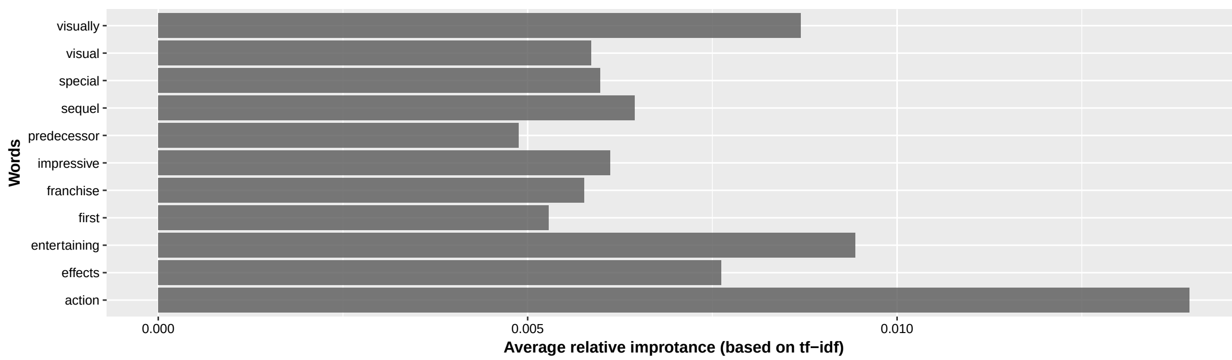
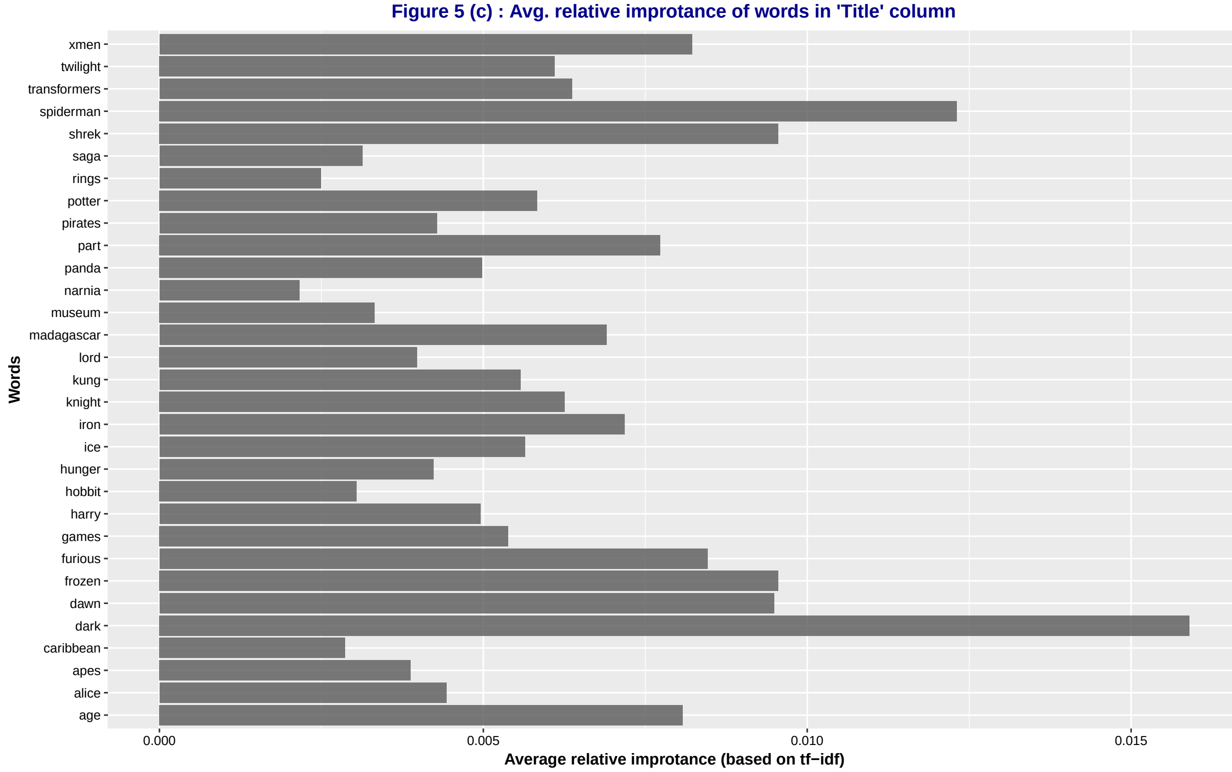


Figure 5 (c) : Avg. relative improtance of words in 'Title' column



Observation : Figures 5 (a, b and c) show the average relative importance of the words that were used to create the binary columns from the 'Plot', 'Concesnus' and the 'Title' columns respectively. For the 'Plot' column, words 'world' and 'must' seemed to be relativeley most important. For the 'Concensus' column, words 'action', 'visually' and 'entertaining' seemed to be relatively important while for the 'Tile' column, words ‘dark’ and ‘spiderman’ seemed to be relatively most important on average.

5.1.2 Creating data frame including these 3 columns and all the features used in task 4

```
df_task_5_a = cbind(df_task_4_b, top_df_plot, top_df_title, top_df_consensus)
```

5.2 Evaluation Strategy - Task 5 [Combining task 4 features + ‘Plot’ + ‘Title’ + ‘Concensus’]

5.2.1 [RMSE Values - Combining task 4 features + ‘Plot’ + ‘Title’ + ‘Concensus’]

- **No. of Iterations :** 1000
- **Scenario 1 :** 5% Train data / 95% Test data
- **Scenario 2 :** 95% Train data / 5% Test data

```
rmse_5.4_a = train_test(df_task_5_a, 1000, 0.05)

paste("Multiple run 5% Train data / 95% Test data - Average Training RMSE : ",
      round(rmse_5.4_a$TRAIN_RMSE,0))
```

```
## [1] "Multiple run 5% Train data / 95% Test data - Average Training RMSE :  58681596"
```

```
paste("Multiple run 5% Train data / 95% Test data - Average Test RMSE     : ",
      round(rmse_5.4_a$TEST_RMSE,0))
```

```
## [1] "Multiple run 5% Train data / 95% Test data - Average Test RMSE     :  107405930"
```

```
rmse_5.4_a = train_test(df_task_5_a, 1000, 0.95)

print("")
```

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

```
paste("Multiple run 95% Train data / 5% Test data - Average Training RMSE : ",
      round(rmse_5.4_a$TRAIN_RMSE,0))
```

```
## [1] "Multiple run 95% Train data / 5% Test data - Average Training RMSE :  81277003"
```

```
paste("Multiple run 95% Train data / 5% Test data - Average Test RMSE     : ",
      round(rmse_5.4_a$TEST_RMSE,0))
```

```
## [1] "Multiple run 95% Train data / 5% Test data - Average Test RMSE     :  81820373"
```

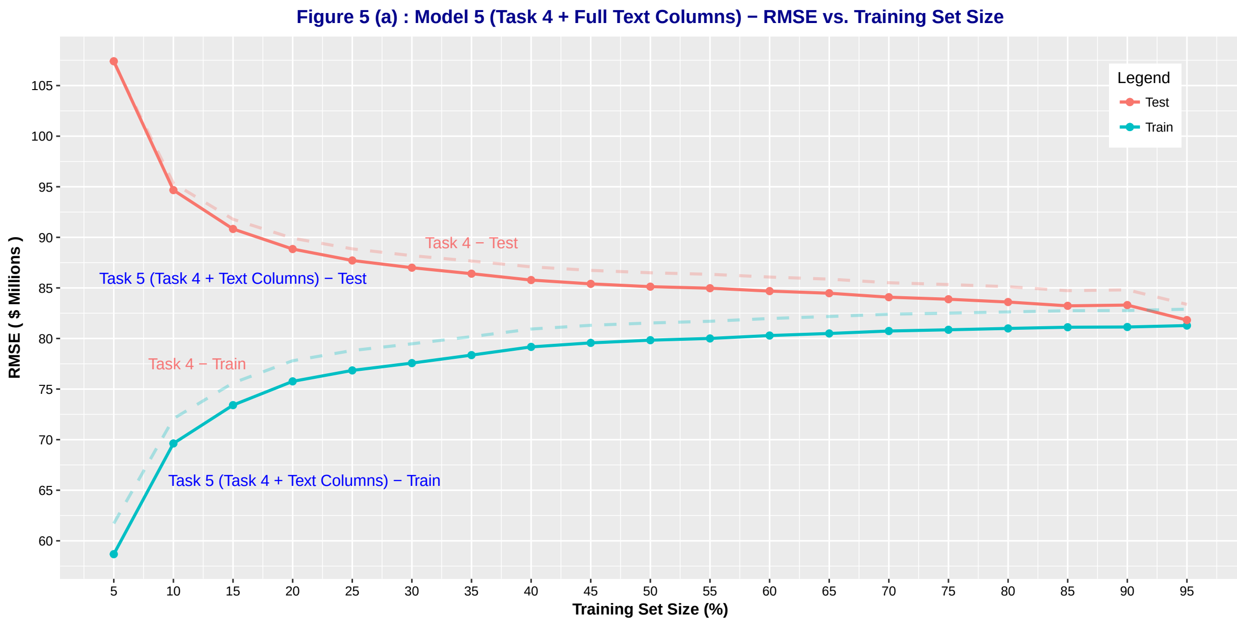
5.2.2 [RMSE Curves - Combining task 4 features + ‘Plot’ + ‘Title’ + ‘Concensus’]

- **No. of Iterations :** 1000

```
set_sizes = seq(0.05, 0.95, by = 0.05)


TRAIN_RMSE_5.5_a = rep(0,19)
TEST_RMSE_5.5_a  = rep(0,19)

for (s in 1:length(set_sizes)){

  #print (s)

  rmse_5.5_a = train_test(df_task_5_a, 1000, set_sizes[s])
```

```
  TRAIN_RMSE_5.5_a[s] = rmse_5.5_a$TRAIN_RMSE
  TEST_RMSE_5.5_a[s] = rmse_5.5_a$TEST_RMSE

  }

#TRAIN_RMSE_5.5_a
#TEST_RMSE_5.5_a

RMSE_task_5_a = data.frame(TrainSet_Size = set_sizes * 100,
                                Train = TRAIN_RMSE_5.5_a,
                                Test = TEST_RMSE_5.5_a,
                                Train_1 = TRAIN_RMSE_4.5_b,
                                Test_1 = TEST_RMSE_4.5_b)

ggplot(data=RMSE_task_5_a) +                                                  # Initializing the plo

  geom_line(size = 1, aes(x = TrainSet_Size, y = Train, color = "Train")) +      # Line Plot

  geom_line(size = 1, aes(x = TrainSet_Size, y = Test, color = "Test")) +        # Line Plot

  geom_point(size = 2, aes(x = TrainSet_Size, y = Train, color = "Train")) +     # Point

  geom_point(size = 2, aes(x = TrainSet_Size, y = Test, color = "Test")) +       # Point

  geom_line(size = 1, linetype = 2,
            alpha = 0.3, aes(x = TrainSet_Size, y = Train_1, color = "Train")) + # Line Plot

  geom_line(size = 1, linetype = 2,
            alpha = 0.3, aes(x = TrainSet_Size, y = Test_1, color = "Test")) +   # Line Plot

  labs(colour="Legend") +                                                      # Legend

  theme(legend.position = c(0.95, 0.95), legend.justification = c(1, 1)) +       # Legend position

  scale_y_continuous(breaks = seq(55000000, 110000000, 5000000),
                     labels = seq(55000000, 110000000, 5000000)/1000000) +       # Setting y-axis scale

  scale_x_continuous(breaks = seq(0, 100, 5)) +                                # Setting x-axis scale

  xlab('Training Set Size (%)') + ylab('RMSE ( $ Millions )') +                # Setting axes labels

  # Setting the title
  ggtitle("Figure 5 (a) : Model 5 (Task 4 + Full Text Columns) - RMSE vs. Training Set Size") +

  theme(plot.title = element_text(face = "bold",color = 'darkblue',              # Setting various them
                                  hjust = 0.5)) +
  theme(axis.text = element_text(colour = "black")) +
  theme(axis.title.x = element_text(face = "bold", color='black'),
        axis.title.y = element_text(face = "bold", color='black')) +
  theme(legend.key = element_rect(fill = "white")) +
  geom_text(aes(label= paste("Task 4 - Test"),
                x = 35, y = 89500000), check_overlap = TRUE,
            color = "red", size = 4, alpha = 0.5) +
```

```
  geom_text(aes(label= paste("Task 4 - Train"),
                x = 12, y = 77500000), check_overlap = TRUE,
            color = "red", size = 4, alpha = 0.5) +
  geom_text(aes(label= paste("Task 5 (Task 4 + Text Columns) - Test"),
                x = 15, y = 86000000), check_overlap = TRUE,
            color = "blue", size = 4) +
  geom_text(aes(label= paste("Task 5 (Task 4 + Text Columns) - Train"),
                x = 21, y = 66000000), check_overlap = TRUE,
            color = "blue", size = 4)
```



Observation : In Figure 5(a), it can clearly be seen that by encoding full text columns into meaningful features has helped reduce RMSE for both training and test sets. The full text columns used for this section were 'Plot', 'Title' and 'Concensus'.

5.3 Looking for useful interactions

5.3.1 Trying all interactions with Budget

As 'Budget' variable had the highest r^2 with 'Gross' (0.77) so I decided to focus on trying out interactions (that would reduce RMSE) based on 'Budget'. First I tried interactions with all the other features (excluding 'Gross' and 'Budget' itself) and looked at the summary of p values to check for statistically insignificant interaction terms and then in subsequent iterations I removed those terms from the model. **NOTE :** I ran the following r-chunk at-least 10 times to make sure I have confidence in removing specific interaction terms.

```
df_task_5_b = df_task_5_a

colnames(df_task_5_b)[2] = "Rated_PG_13"

df_5_b_index = indexing(df_task_5_b, 0.95)

df_5_b_index_training_set_new = training_set(df_task_5_b,df_5_b_index)

df_5_b_index_testing_set_new = testing_set(df_task_5_b, df_5_b_index)
```

```r
df_5_b_lm = lm(Gross ~ . + (Rated_R + Rated_PG_13 + Rated_PG + Metascore +
                              tomatoImage_certified + tomatoImage_fresh +
                              tomatoImage_rotten + Top_Prod_2 + Genre_Action_2 +
                              Genre_Adventure_2 + Genre_Fantasy_2 + Genre_Sci_Fi_2 +
                              Genre_Animation_2 + Top_Directors_2 + Top_Writers_2 +
                              Top_Actors_2 + Other_Languages_2 + Other_Country_2 +
                              Wins + Nominations + Released_Month + Year + Runtime +
                              imdbRating + imdbVotes + tomatoRating + tomatoReviews +
                              tomatoFresh + tomatoUserRating + tomatoUserReviews +
                              Metascore_t + Wins_t + Nominations_t + Released_Month_t +
                              Runtime_o + imdbRating_o + tomatoReviews_o):Budget_o,
               data = df_5_b_index_training_set_new)

# https://stat.ethz.ch/pipermail/r-help/2005-December/084308.html

#summary(df_5_b_lm)

p_value = as.data.frame(as.table(coef(summary(df_5_b_lm))))

p_value = subset(subset(p_value, Var2 == 'Pr(>|t|)'), select = c(Var1, Freq))[44:83,]

p_value = data.frame(feature = p_value$Var1, p_value = round(p_value$Freq,5))

subset(p_value, p_value > 0.05)
```

```
##                        feature p_value
## 1            Rated_R:Budget_o 0.19224
## 2        Rated_PG_13:Budget_o 0.21245
## 3           Rated_PG:Budget_o 0.56225
## 4          Metascore:Budget_o 0.89720
## 8         Top_Prod_2:Budget_o 0.43851
## 9     Genre_Action_2:Budget_o 0.41393
## 10 Genre_Adventure_2:Budget_o 0.14097
## 11   Genre_Fantasy_2:Budget_o 0.17981
## 12    Genre_Sci_Fi_2:Budget_o 0.13388
## 14   Top_Directors_2:Budget_o 0.12743
## 15     Top_Writers_2:Budget_o 0.10820
## 17 Other_Languages_2:Budget_o 0.72424
## 18   Other_Country_2:Budget_o 0.08646
## 21    Released_Month:Budget_o 0.83495
## 27     tomatoReviews:Budget_o 0.19516
## 31       Metascore_t:Budget_o 0.13310
## 32            Wins_t:Budget_o 0.53605
## 33     Nominations_t:Budget_o 0.05333
## 35         Runtime_o:Budget_o 0.33377
```

Comment : Interaction terms for the following features were removed from the model based on the results above:

- Rated_R, Rated_PG_13, Rated_PG
- Metascore, MetaScore_transform
- Production, Directors, Writers, Languages, Countries
- Genres : Action, Adventure, Fantasy, Sci_Fi, Animation
- Released_Month, Runtme_original, Year_transform
- tomatoReviews, Nominations_transform, tomatoRating

5.3.2 Only features that had stat. significance interactions with Budget after 1st iteration were kept

This r-chunk shows that the model only includes interactions that were statistically significant after the first run.

```
df_5_b_index = indexing(df_task_5_b, 0.95)

df_5_b_index_training_set_new = training_set(df_task_5_b,df_5_b_index)

df_5_b_index_testing_set_new = testing_set(df_task_5_b, df_5_b_index)

df_5_b_lm = lm(Gross ~ . + (  tomatoImage_certified + tomatoImage_fresh +
                              tomatoImage_rotten + Top_Actors_2 + Wins +
                              Nominations + Runtime + imdbRating +
                              imdbVotes + tomatoFresh + tomatoUserRating +
                              tomatoUserReviews + Released_Month_t +
                              imdbRating_o + tomatoReviews_o):Budget_o,
               data = df_5_b_index_training_set_new)

# https://stat.ethz.ch/pipermail/r-help/2005-December/084308.html

#summary(df_5_b_lm)

p_value = as.data.frame(as.table(coef(summary(df_5_b_lm))))

p_value = subset(subset(p_value, Var2 == 'Pr(>|t|)'), select = c(Var1, Freq))[44:66,]

p_value = data.frame(feature = p_value$Var1, p_value = round(p_value$Freq,5))

subset(p_value, p_value > 0.05)
```

```
## [1] feature p_value
## <0 rows> (or 0-length row.names)
```

Comment : At this point all of the statistically insignificant interaction terms are excluded from the model so went ahead with the evaluation strategy.

5.4 Evaluation Strategy - Task 5 [task 4 + Full-text columns + Interactions]

5.4.1 [RMSE Values - Combining task 4 + Full-text columns + Interactions]

- **No. of Iterations :** 1000
- **Scenario 1 :** 5% Train data / 95% Test data
- **Scenario 2 :** 95% Train data / 5% Test data

```
TRAIN_RMSE_5.4_b = rep(0,19)
TEST_RMSE_5.4_b  = rep(0,19)

for (i in 1:1000){

set.seed(i)

df_5_b_index = indexing(df_task_5_b, 0.05)

df_5_b_index_training_set_new = training_set(df_task_5_b,df_5_b_index)
```

```
df_5_b_index_testing_set_new = testing_set(df_task_5_b, df_5_b_index)

df_5_b_lm = lm(Gross ~ . + (  tomatoImage_certified + tomatoImage_fresh +
                              tomatoImage_rotten + Top_Actors_2 + Wins +
                              Nominations + Runtime + imdbRating +
                              imdbVotes + tomatoFresh + tomatoUserRating +
                              tomatoUserReviews + Released_Month_t +
                              imdbRating_o + tomatoReviews_o):Budget_o,
               data = df_5_b_index_training_set_new)

pred_train_set = subset(df_5_b_index_training_set_new, select = -c(Gross))
pred_test_set = subset(df_5_b_index_testing_set_new, select = -c(Gross))
# predicitons based on training and test data
prediction_train = predict(df_5_b_lm, pred_train_set)
prediction_test = predict(df_5_b_lm, pred_test_set)

# rmse based on training and test data
rmse_train = rmse(df_5_b_index_training_set_new$Gross, prediction_train)
rmse_test = rmse(df_5_b_index_testing_set_new$Gross, prediction_test)

TRAIN_RMSE_5.4_b[i] = rmse_train
TEST_RMSE_5.4_b[i] = rmse_test

}

paste("Multiple run 5% Train data / 95% Test data - Average Training RMSE : ",
      round(mean(TRAIN_RMSE_5.4_b),0))
```

```
## [1] "Multiple run 5% Train data / 95% Test data - Average Training RMSE :  40357928"
```

```
paste("Multiple run 5% Train data / 95% Test data - Average Test RMSE     : ",
      round(mean(TEST_RMSE_5.4_b),0))
```

```
## [1] "Multiple run 5% Train data / 95% Test data - Average Test RMSE     :  132258925"
```

```
TRAIN_RMSE_5.4_b = rep(0,19)
TEST_RMSE_5.4_b  = rep(0,19)

for (i in 1:1000){

set.seed(i)

df_5_b_index = indexing(df_task_5_b, 0.95)

df_5_b_index_training_set_new = training_set(df_task_5_b,df_5_b_index)

df_5_b_index_testing_set_new = testing_set(df_task_5_b, df_5_b_index)

df_5_b_lm = lm(Gross ~ . + (  tomatoImage_certified + tomatoImage_fresh +
                              tomatoImage_rotten + Top_Actors_2 + Wins +
                              Nominations + Runtime + imdbRating +
                              imdbVotes + tomatoFresh + tomatoUserRating +
                              tomatoUserReviews + Released_Month_t +
                              imdbRating_o + tomatoReviews_o):Budget_o,
```

```
                data = df_5_b_index_training_set_new)

pred_train_set = subset(df_5_b_index_training_set_new, select = -c(Gross))
pred_test_set = subset(df_5_b_index_testing_set_new, select = -c(Gross))
# predicitons based on training and test data
prediction_train = predict(df_5_b_lm, pred_train_set)
prediction_test = predict(df_5_b_lm, pred_test_set)

# rmse based on training and test data
rmse_train = rmse(df_5_b_index_training_set_new$Gross, prediction_train)
rmse_test = rmse(df_5_b_index_testing_set_new$Gross, prediction_test)

TRAIN_RMSE_5.4_b[i] = rmse_train
TEST_RMSE_5.4_b[i] = rmse_test

}

print("")
```

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

```
paste("Multiple run 95% Train data / 5% Test data - Average Training RMSE : ",
      round(mean(TRAIN_RMSE_5.4_b),0))
```

```
## [1] "Multiple run 95% Train data / 5% Test data - Average Training RMSE :  71027067"
```

```
paste("Multiple run 95% Train data / 5% Test data - Average Test RMSE     : ",
      round(mean(TEST_RMSE_5.4_b),0))
```

```
## [1] "Multiple run 95% Train data / 5% Test data - Average Test RMSE     :  75747269"
```

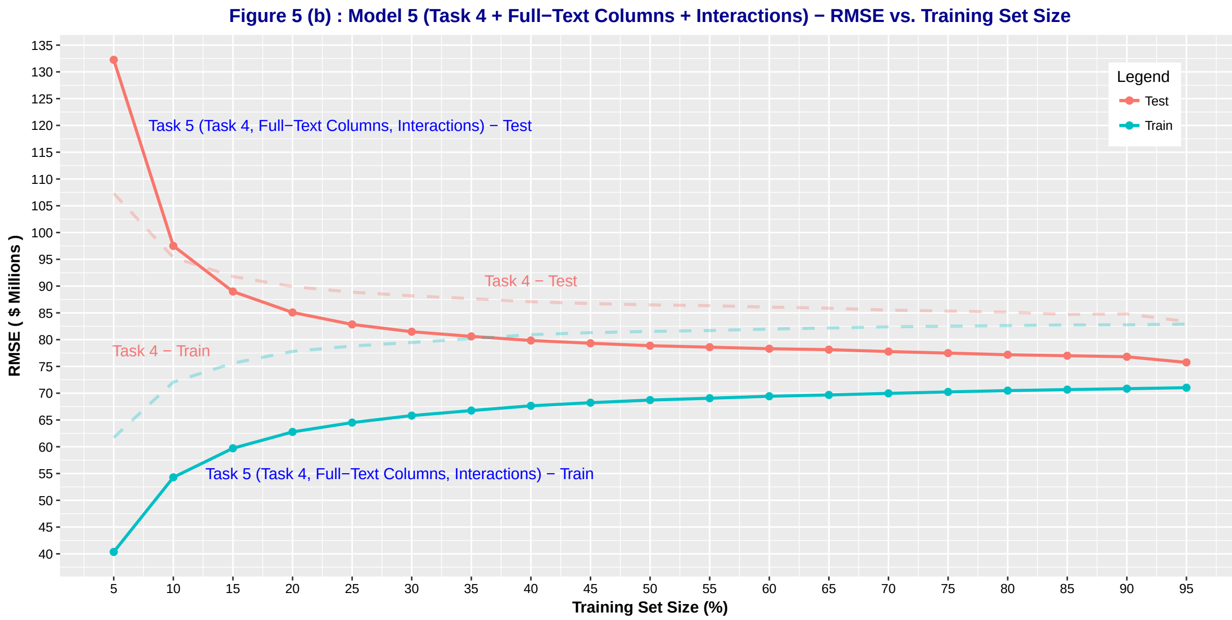
5.4.2 [RMSE Curves - Combining task 4 + Full-text columns + Interactions]

- **No. of Iterations :** 1000

```
set_sizes = seq(0.05, 0.95, by = 0.05)


TRAIN_RMSE_5.5_b = rep(0,19)
TEST_RMSE_5.5_b  = rep(0,19)

for (s in 1:length(set_sizes)){

  TRAIN_RMSE = rep(0,1000)
  TEST_RMSE = rep(0,1000)


  for (i in 1:1000){

        set.seed(i)

        df_5_b_index = indexing(df_task_5_b, set_sizes[s])

        df_5_b_index_training_set_new = training_set(df_task_5_b,df_5_b_index)
```

```
        df_5_b_index_testing_set_new = testing_set(df_task_5_b, df_5_b_index)

        df_5_b_lm = lm(Gross ~ . + (  tomatoImage_certified + tomatoImage_fresh +
                                      tomatoImage_rotten + Top_Actors_2 + Wins +
                                      Nominations + Runtime + imdbRating +
                                      imdbVotes + tomatoFresh + tomatoUserRating +
                                      tomatoUserReviews + Released_Month_t +
                                      imdbRating_o + tomatoReviews_o):Budget_o,
                       data = df_5_b_index_training_set_new)

        pred_train_set = subset(df_5_b_index_training_set_new, select = -c(Gross))
        pred_test_set = subset(df_5_b_index_testing_set_new, select = -c(Gross))
        # predicitons based on training and test data
        prediction_train = predict(df_5_b_lm, pred_train_set)
        prediction_test = predict(df_5_b_lm, pred_test_set)

        # rmse based on training and test data
        rmse_train = rmse(df_5_b_index_training_set_new$Gross, prediction_train)
        rmse_test = rmse(df_5_b_index_testing_set_new$Gross, prediction_test)


        TRAIN_RMSE[i] = rmse_train
        TEST_RMSE[i] = rmse_test

      }

  TRAIN_RMSE_5.5_b[s] = mean(TRAIN_RMSE)
  TEST_RMSE_5.5_b[s] = mean(TEST_RMSE)

  }

#TRAIN_RMSE_5.5_b
#TEST_RMSE_5.5_b

RMSE_task_5_b = data.frame(TrainSet_Size = set_sizes * 100,
                                Train = TRAIN_RMSE_5.5_b,
                                Test = TEST_RMSE_5.5_b,
                                Train_1 = TRAIN_RMSE_4.5_b,
                                Test_1 = TEST_RMSE_4.5_b)

ggplot(data=RMSE_task_5_b) +                                           # Initializing the plo

  geom_line(size = 1, aes(x = TrainSet_Size, y = Train, color = "Train")) +      # Line Plot

  geom_line(size = 1, aes(x = TrainSet_Size, y = Test, color = "Test")) +        # Line Plot

  geom_point(size = 2, aes(x = TrainSet_Size, y = Train, color = "Train")) +     # Point

  geom_point(size = 2, aes(x = TrainSet_Size, y = Test, color = "Test")) +       # Point

  geom_line(size = 1, linetype = 2,
          alpha = 0.3, aes(x = TrainSet_Size, y = Train_1, color = "Train")) + # Line Plot
```

```
geom_line(size = 1, linetype = 2,
          alpha = 0.3, aes(x = TrainSet_Size, y = Test_1, color = "Test")) +   # Line Plot

labs(colour="Legend") +                                                         # Legend

theme(legend.position = c(0.95, 0.95), legend.justification = c(1, 1)) +       # Legend position

scale_y_continuous(breaks = seq(40000000, 135000000, 5000000),
                   labels = seq(40000000, 135000000, 5000000)/1000000) +       # Setting y-axis scale

scale_x_continuous(breaks = seq(0, 100, 5)) +                                   # Setting x-axis scale

xlab('Training Set Size (%)') + ylab('RMSE ( $ Millions )') +                   # Setting axes labels

# Setting the title
ggtitle("Figure 5 (b) : Model 5 (Task 4 + Full-Text Columns + Interactions) - RMSE vs. Training Set Si

theme(plot.title = element_text(face = "bold",color = 'darkblue',              # Setting various them
                                hjust = 0.5)) +
theme(axis.text = element_text(colour = "black")) +
theme(axis.title.x = element_text(face = "bold", color='black'),
      axis.title.y = element_text(face = "bold", color='black')) +
theme(legend.key = element_rect(fill = "white")) +
geom_text(aes(label= paste("Task 4 - Test"),
              x = 40, y = 91000000), check_overlap = TRUE,
          color = "red", size = 4, alpha = 0.5) +
geom_text(aes(label= paste("Task 4 - Train"),
              x = 9, y = 78000000), check_overlap = TRUE,
          color = "red", size = 4, alpha = 0.5) +
geom_text(aes(label= paste("Task 5 (Task 4, Full-Text Columns, Interactions) - Test"),
              x = 24, y = 120000000), check_overlap = TRUE,
          color = "blue", size = 4) +
geom_text(aes(label= paste("Task 5 (Task 4, Full-Text Columns, Interactions) - Train"),
              x = 29, y = 55000000), check_overlap = TRUE,
          color = "blue", size = 4)
```



Observation : Including interactions in the model has significantly improved RMSE. There is a slight hint of overfitting towards increasing training set size but not that much, considering the imrpovement that is seen in RMSE.

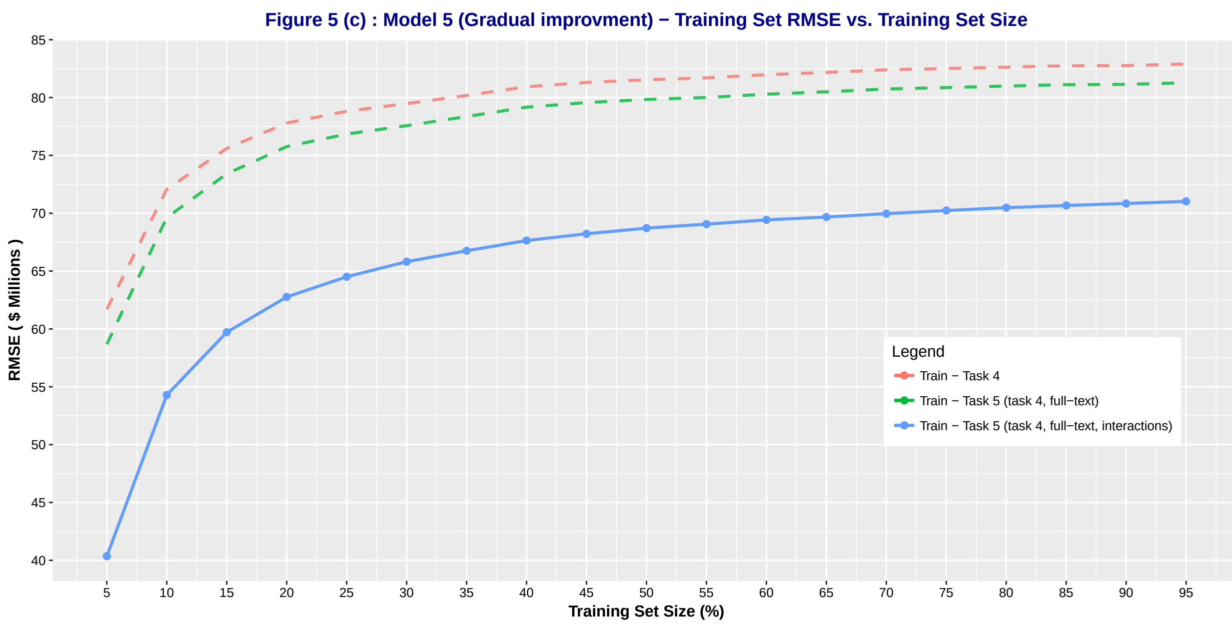
5.5 Evaluation Strategy - Task 5 [Showing improvement progressively]

5.5.1 [RMSE TRAINING Curves - Showing improvement progressively]

- **No. of Iterations :** 1000

```
RMSE_task_5_train = data.frame(TrainSet_Size = set_sizes * 100,
                                Train = TRAIN_RMSE_5.5_b,
                                Train_1 = TRAIN_RMSE_5.5_a,
                                Train_2 = TRAIN_RMSE_4.5_b)

ggplot(data=RMSE_task_5_train) +                                          # Initializing the plo

  geom_line(size = 1,
            aes(x = TrainSet_Size, y = Train,
                color = "Train - Task 5 (task 4, full-text, interactions)")) +  # Line Plot

  geom_point(size = 2,
             aes(x = TrainSet_Size, y = Train,
                 color = "Train - Task 5 (task 4, full-text, interactions)")) + # Point

  geom_line(size = 1, linetype = 2,
            alpha = 0.8, aes(x = TrainSet_Size, y = Train_1,
                             color = "Train - Task 5 (task 4, full-text)")) +     # Line Plot

  geom_line(size = 1, linetype = 2,
            alpha = 0.8, aes(x = TrainSet_Size, y = Train_2,
                             color = "Train - Task 4")) +                         # Line Plot
```

```
  labs(colour="Legend") +                                                # Legend

  theme(legend.position = c(0.95, 0.45), legend.justification = c(1, 1)) +      # Legend position

  scale_y_continuous(breaks = seq(40000000, 85000000, 5000000),
                     labels = seq(40000000, 85000000, 5000000)/1000000) +       # Setting y-axis scale

  scale_x_continuous(breaks = seq(0, 100, 5)) +                                 # Setting x-axis scale

  xlab('Training Set Size (%)') + ylab('RMSE ( $ Millions )') +                 # Setting axes labels

  # Setting the title
  ggtitle("Figure 5 (c) : Model 5 (Gradual improvment) - Training Set RMSE vs. Training Set Size") +

  theme(plot.title = element_text(face = "bold",color = 'darkblue',            # Setting various them
                                  hjust = 0.5)) +
  theme(axis.text = element_text(colour = "black")) +
  theme(axis.title.x = element_text(face = "bold", color='black'),
        axis.title.y = element_text(face = "bold", color='black')) +
  theme(legend.key = element_rect(fill = "white"))
```



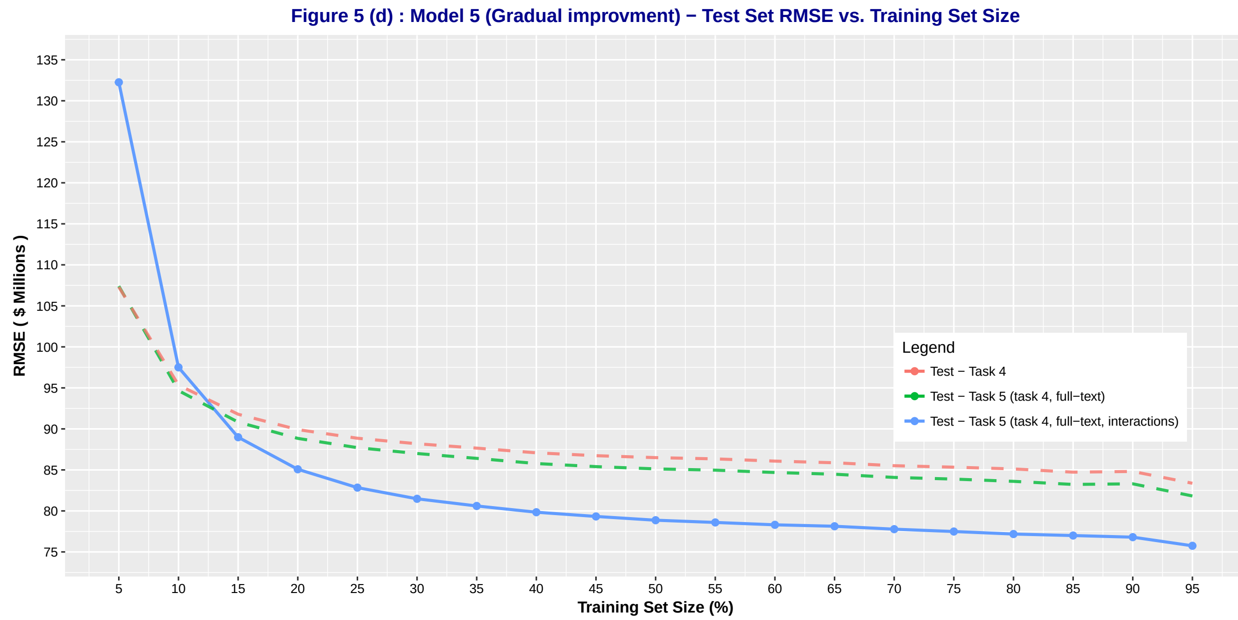
Observation : Here in Figure 5(c), it can clearly be seen that adding features extracted from full-text columns 'Plot', 'Title' and 'Concensus' along with interacion terms has reduced RMSE for the training set by a very big margin.

5.5.2 [RMSE TEST Curves - Gradual Improvement]

- **No. of Iterations :** 1000

```
RMSE_task_5_test = data.frame(TrainSet_Size = set_sizes * 100,
                              Test = TEST_RMSE_5.5_b,
                              Test_1 = TEST_RMSE_5.5_a,
                              Test_2 = TEST_RMSE_4.5_b)
```

```r
ggplot(data=RMSE_task_5_test) +                                                    # Initializing the pl

  geom_line(size = 1,
            aes(x = TrainSet_Size, y = Test,
                color = "Test - Task 5 (task 4, full-text, interactions)")) +  # Line Plot

  geom_point(size = 2,
             aes(x = TrainSet_Size, y = Test,
                 color = "Test - Task 5 (task 4, full-text, interactions)")) + # Point

  geom_line(size = 1, linetype = 2,
            alpha = 0.8, aes(x = TrainSet_Size, y = Test_1,
                             color = "Test - Task 5 (task 4, full-text)")) +      # Line Plot

  geom_line(size = 1, linetype = 2,
            alpha = 0.8, aes(x = TrainSet_Size, y = Test_2,
                             color = "Test - Task 4")) +           # Line Plot

  labs(colour="Legend") +                                                         # Legend

  theme(legend.position = c(0.95, 0.45), legend.justification = c(1, 1)) +        # Legend position

  scale_y_continuous(breaks = seq(75000000, 135000000, 5000000),
                     labels = seq(75000000, 135000000, 5000000)/1000000,
                     limits = c(75000000, 135000000)) +                           # Setting y-axis scal

  scale_x_continuous(breaks = seq(0, 100, 5)) +                                   # Setting x-axis scal

  xlab('Training Set Size (%)') + ylab('RMSE ( $ Millions )') +                   # Setting axes labels

  # Setting the title
  ggtitle("Figure 5 (d) : Model 5 (Gradual improvment) - Test Set RMSE vs. Training Set Size") +

  theme(plot.title = element_text(face = "bold",color = 'darkblue',              # Setting various the
                                  hjust = 0.5)) +
  theme(axis.text = element_text(colour = "black")) +
  theme(axis.title.x = element_text(face = "bold", color='black'),
        axis.title.y = element_text(face = "bold", color='black')) +
  theme(legend.key = element_rect(fill = "white"))
```



Observation : RMSE for testing set was also reduced when a combination of text features and interaction terms were used to train the model. Although, the RMSE for larger test set sizes again did show some deterioration, however, similar to Figure 5 (c) the improvement in RMSE for smaller test set sizes (i.e. less than 80%) compensates for that.

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

A:

FULL TEXT Columns

For this task, just like I combined (using logical OR) the highly correlated directors, actors, etc using 'Reduce' command in task 3, the same way I combined columns of highly correlated words (with 'Gross') into seperate binary columns for 'Plot', 'Title' and 'Concensus'. Following steps were performed in order to extract a binary feature column from a respective full text column. This is applicable to all 3 'Plot', 'Title' and 'concensus' columns.

Pre-processing steps (using 'tm' package)

- Switch to lower case (example : tm_map(plot_corpus, content_transformer(tolower)))
- Remove numbers (example : tm_map(plot_corpus, removeNumbers))
- Remove punctuation marks : (example : tm_map(plot_corpus, removePunctuation))
- Remove stopwords (example : tm_map(plot_corpus, removeWords, c("the", "and", stopwords("english"))))
- Remove extra whitespaces (example : tm_map(plot_corpus, stripWhitespace))
- Remove less frequent words using tf-idf (example : DocumentTermMatrix(plot_corpus, control = list(weighting = weightTfIdf))) (example : removeSparseTerms(plot_dtm_tfidf, 0.99))

Once I had processed these 3 columns in the above mentioned way, I then used the 'Reduce' command and created 3 binary columns for each of the variables.

specific 'Reduce' commands used were as follows

- Reduce('|', df_5_plot[,cor(df_task_3$Gross, df_5_plot) > 0.065])
- Reduce('|', df_5_title[,cor(df_task_3$Gross, df_5_title) > 0.061])
- Reduce('|', df_5_consensus[,cor(df_task_3$Gross, df_5_consensus) > 0.08])

The thresholds were chosen in a way to hone in on to a number of frequently used words that when combined into a single binary column would give good r^2 with 'Gross'. The number of frequency words chosen for

'Plot' were 12, for 'Title' were 31 and for 'Concensus' were 11. This led to 3 different encoded columns for 'Plot', 'Title' and 'concensus' and finally these were appended to the task 4 data frame to be further used to train the model. The results for this are shown above in Figure 5(a). All three encoded features had positive correlation with 'Gross', encoded variable for 'Plot' had an r^2 of 0.19, encoded variable for 'Title' had an r^2 of 0.45 while the encoded variable for 'Concensus' had an r^2 of 0.35.

INTERACTIONS

For exploring interactions, I focused on Budget as the conditional variable based on its highest r^2 of 0.77 with 'Gross'. I followed an eliminatation based approach to choose the terms for interactions. First I started off with training a model using interactions for complete set of features with 'Budget', i.e. lm(Gross ~ . + (all features): Budget, data = df)). This only excluded the 3 encoded columns for 'Plot', 'Title' and 'Concensus'. Based on this link and on this link I only focused on keeping the interaction terms that were statistically significant. The steps I followed are shown below:

Step 1 :

In order to select the appropriate interaction terms, I first trained a model with interaction terms for all the features. Thsi is shown by the lm command below:

```
'
lm(Gross ~ . + (Rated_R + Rated_PG_13 + Rated_PG + Metascore +
                              tomatoImage_certified + tomatoImage_fresh +
                              tomatoImage_rotten + Top_Prod_2 + Genre_Action_2 +
                              Genre_Adventure_2 + Genre_Fantasy_2 + Genre_Sci_Fi_2 +
                              Genre_Animation_2 + Top_Directors_2 + Top_Writers_2 +
                              Top_Actors_2 + Other_Languages_2 + Other_Country_2 +
                              Wins + Nominations + Released_Month + Year + Runtime +
                              imdbRating + imdbVotes + tomatoRating + tomatoReviews +
                              tomatoFresh + tomatoUserRating + tomatoUserReviews +
                              Metascore_t + Wins_t + Nominations_t + Released_Month_t +
                              Runtime_o + imdbRating_o + tomatoReviews_o):Budget_o,
               data = df_5_b_)
'
```

```
## [1] "\nlm(Gross ~ . + (Rated_R + Rated_PG_13 + Rated_PG + Metascore +\n
```

Step 2 :

Then in the next step, I viewed all the interacting features that had p-values that were statistically insignificant (greater than 5%). In the next iteration of the model training, I excluded all these interaction terms. Following features that had statistically insignificant interaction terms with 'Budget' were removed from the model before running the next iteration:

- Rated_R, Rated_PG_13, Rated_PG
- Metascore, MetaScore_transform
- Production, Directors, Writers, Languages, Countries
- Genres : Action, Adventure, Fantasy, Sci_Fi, Animation
- Released_Month, Runtme_original, Year_transform
- tomatoReviews, Nominations_transform, tomatoRating

The lm command for the next iteration is shown below:

```
'
lm(Gross ~ . + (  tomatoImage_certified + tomatoImage_fresh +
                              tomatoImage_rotten + Top_Actors_2 + Wins +
                              Nominations + Runtime + imdbRating +
                              imdbVotes + tomatoFresh + tomatoUserRating +
                              tomatoUserReviews + Released_Month_t +
```

```
                              imdbRating_o + tomatoReviews_o):Budget_o,
               data = df_5_b)
'
```

```
## [1] "\nlm(Gross ~ . + (  tomatoImage_certified + tomatoImage_fresh +\n
```

Step 3 :

Running the model with the interaction terms as shown above did not showany p-values that were greater than 0.05. **NOTE :** Models in both step 1 and 2 were run at-least 10 times in order to have confidence on what terms to keep in the model. Once this was finalized then next process of computing RMSE was performed the same way as it was done in all the previous tasks. The results are shown in Figure 5(b).

6 All Tasks - [Showing improvement progressively]

This section is not part of the project but I have add to show the gradual changes in the RMSE curves based on each task.

6.1 [RMSE TRAINING Curves - Showing improvement progressively]

- **No. of Iterations :** 1000

```
RMSE_task_6_train = data.frame(TrainSet_Size = set_sizes * 100,
                                Train = TRAIN_RMSE_5.5_b,
                                Train_1 = TRAIN_RMSE_4.5_b,
                                Train_2 = TRAIN_RMSE_3.5_b,
                                Train_3 = TRAIN_RMSE_2.5,
                                Train_4 = TRAIN_RMSE_1.5)

ggplot(data=RMSE_task_6_train) +                                            # Initializing the plo

  geom_line(size = 1,
            aes(x = TrainSet_Size, y = Train,
                color = "Train - Task 5")) +                                # Line Plot

  geom_point(size = 2,
             aes(x = TrainSet_Size, y = Train,
                 color = "Train - Task 5")) +                               # Point

  geom_line(size = 1, #linetype = 2,
            alpha = 0.8, aes(x = TrainSet_Size, y = Train_1,
                             color = "Train - Task 4")) +                   # Line Plot

  geom_point(size = 2,
             aes(x = TrainSet_Size, y = Train_1,
                 color = "Train - Task 4")) +                               # Point


  geom_line(size = 1, #linetype = 2,
            alpha = 0.8, aes(x = TrainSet_Size, y = Train_2,
                             color = "Train - Task 3")) +                   # Line Plot

  geom_point(size = 2,
             aes(x = TrainSet_Size, y = Train_2,
```

```r
            color = "Train - Task 3")) +                                # Point


geom_line(size = 1, #linetype = 2,
          alpha = 0.8, aes(x = TrainSet_Size, y = Train_3,
                           color = "Train - Task 2")) +                 # Line Plot

geom_point(size = 2,
           aes(x = TrainSet_Size, y = Train_3,
               color = "Train - Task 2")) +                             # Point


geom_line(size = 1, #linetype = 2,
          alpha = 0.8, aes(x = TrainSet_Size, y = Train_4,
                           color = "Train - Task 1")) +                 # Line Plot

geom_point(size = 2,
           aes(x = TrainSet_Size, y = Train_4,
               color = "Train - Task 1")) +                             # Point



labs(colour="Legend") +                                                 # Legend

theme(legend.position = c(0.95, 0.40), legend.justification = c(1, 1)) +      # Legend position

scale_y_continuous(breaks = seq(40000000, 110000000, 5000000),
                   labels = seq(40000000, 110000000, 5000000)/1000000) +      # Setting y-axis scale

scale_x_continuous(breaks = seq(0, 100, 5)) +                           # Setting x-axis scale

xlab('Training Set Size (%)') + ylab('RMSE ( $ Millions )') +           # Setting axes labels

# Setting the title
ggtitle("Figure 6 (a) : All Models (Gradual improvment) - Training Set RMSE vs. Training Set Size") +

theme(plot.title = element_text(face = "bold",color = 'darkblue',             # Setting various them
                                hjust = 0.5)) +
theme(axis.text = element_text(colour = "black")) +
theme(axis.title.x = element_text(face = "bold", color='black'),
      axis.title.y = element_text(face = "bold", color='black')) +
theme(legend.key = element_rect(fill = "white"))
```

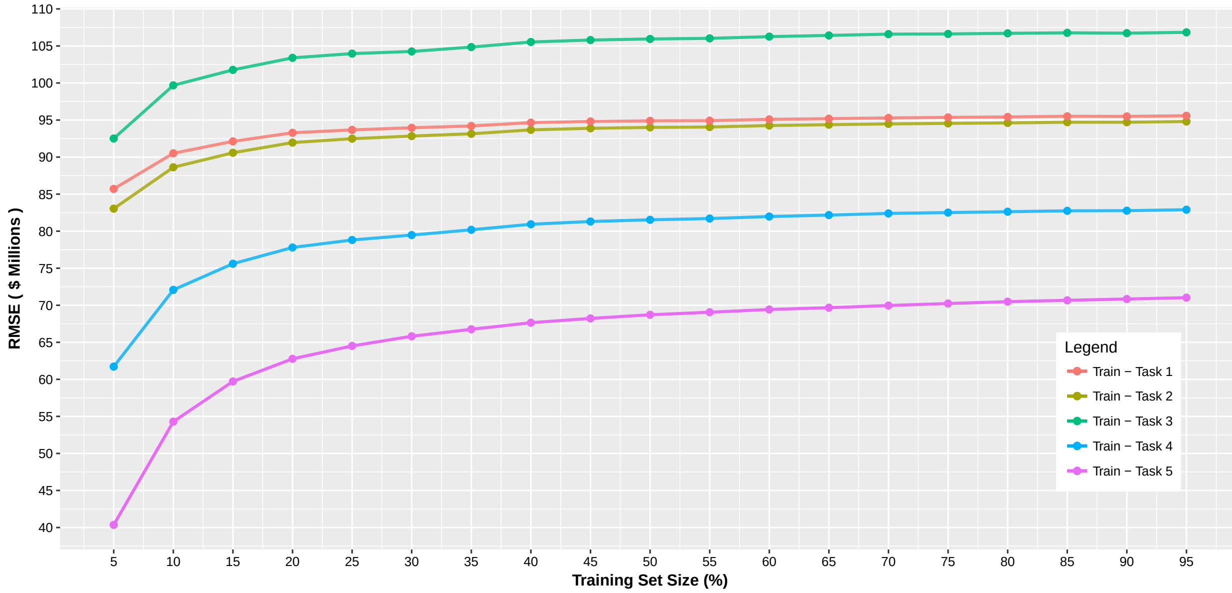


Figure 6 (a) : All Models (Gradual improvment) – Training Set RMSE vs. Training Set Size

6.2 [RMSE TEST Curves - Showing improvement progressively]

- **No. of Iterations :** 1000

```
RMSE_task_6_Test = data.frame(TestSet_Size = set_sizes * 100,
                              Test = TEST_RMSE_5.5_b,
                              Test_1 = TEST_RMSE_4.5_b,
                              Test_2 = TEST_RMSE_3.5_b,
                              Test_3 = TEST_RMSE_2.5,
                              Test_4 = TEST_RMSE_1.5)

ggplot(data=RMSE_task_6_Test) +                                     # Initializing the plot

  geom_line(size = 1,
            aes(x = TestSet_Size, y = Test,
                color = "Test - Task 5")) +                         # Line Plot

  geom_point(size = 2,
             aes(x = TestSet_Size, y = Test,
                 color = "Test - Task 5")) +                        # Point

  geom_line(size = 1, #linetype = 2,
            alpha = 0.8, aes(x = TestSet_Size, y = Test_1,
                             color = "Test - Task 4")) +            # Line Plot

  geom_point(size = 2,
             aes(x = TestSet_Size, y = Test_1,
                 color = "Test - Task 4")) +                        # Point


  geom_line(size = 1, #linetype = 2,
            alpha = 0.8, aes(x = TestSet_Size, y = Test_2,
                             color = "Test - Task 3")) +            # Line Plot
```

```r
geom_point(size = 2,
           aes(x = TestSet_Size, y = Test_2,
               color = "Test - Task 3")) +                                    # Point


geom_line(size = 1, #linetype = 2,
          alpha = 0.8, aes(x = TestSet_Size, y = Test_3,
                           color = "Test - Task 2")) +                        # Line Plot

geom_point(size = 2,
           aes(x = TestSet_Size, y = Test_3,
               color = "Test - Task 2")) +                                    # Point


geom_line(size = 1, #linetype = 2,
          alpha = 0.8, aes(x = TestSet_Size, y = Test_4,
                           color = "Test - Task 1")) +                        # Line Plot

geom_point(size = 2,
           aes(x = TestSet_Size, y = Test_4,
               color = "Test - Task 1")) +                                    # Point



labs(colour="Legend") +                                                       # Legend

theme(legend.position = c(0.95, 0.90), legend.justification = c(1, 1)) +      # Legend position

scale_y_continuous(breaks = seq(70000000, 150000000, 5000000),
                   labels = seq(70000000, 150000000, 5000000)/1000000) +      # Setting y-axis scales

scale_x_continuous(breaks = seq(0, 100, 5)) +                                 # Setting x-axis scales

xlab('Training Set Size (%)') + ylab('RMSE ( $ Millions )') +                 # Setting axes labels

# Setting the title
ggtitle("Figure 6 (b) : All Models (Gradual improvment) - Test Set RMSE vs. Training Set Size") +

theme(plot.title = element_text(face = "bold",color = 'darkblue',             # Setting various theme
                                hjust = 0.5)) +
theme(axis.text = element_text(colour = "black")) +
theme(axis.title.x = element_text(face = "bold", color='black'),
      axis.title.y = element_text(face = "bold", color='black')) +
theme(legend.key = element_rect(fill = "white"))
```

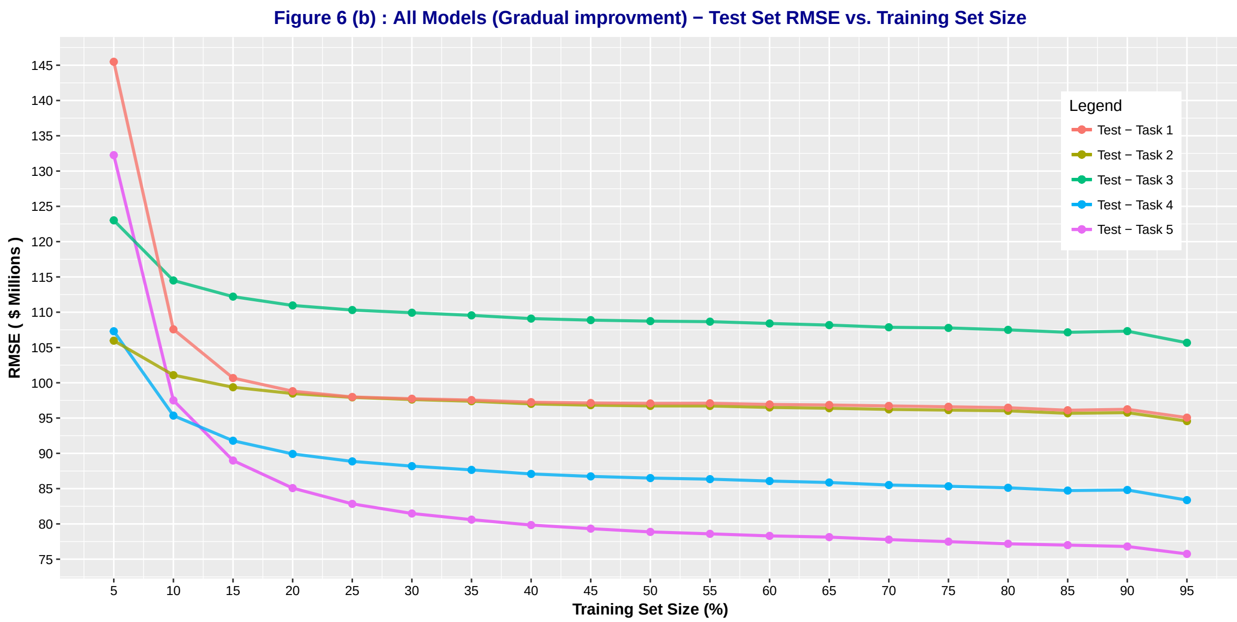


Figure 6 (b) : All Models (Gradual improvment) − Test Set RMSE vs. Training Set Size
RMSE ($ Millions)
145
140
135
130
125
120
115
110
105
100
95
90
85
80
75
5
10
15
20
25
30
35
40
45
50
55
60
65
70
75
80
85
90
95
Training Set Size (%)
Legend
Test − Task 1
Test − Task 2
Test − Task 3
Test − Task 4
Test − Task 5