Project 2: Modeling and Evaluation

CSE6242 - Data and Visual Analytics - Spring 2017 Due: Friday ,22, 2017 at 11:59 PM UTC-12:00 on T-Square Author : Melisande Zonta Roudes GT account name : mzr3

Contents

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.

```
library(ggplot2)
library(stringr)
library(caret)
library(tidytext)
library(dplyr)
library(ModelMetrics)
library(readr)
library(grid)
library(gridExtra)
library(stringr)
library(gtools)
library(qdap)
library(tm)
```

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

Non-standard packages used: None

Data Preprocessing

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

1. Remove non-movie rows

```
# TODO: Remove all rows from df that do not correspond to movies`
dim(df)
```

```
## [1] 40789    39
```

```
df1 = subset(df,Type == "movie")
dim(df1)
```

```
## [1] 40000    39
```

```
df = df1
```

2. Drop rows with missing `Gross` value

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

```
# TODO: Remove rows with missing Gross value

print(dim(df))
```

```
## [1] 40000    39
```

```
df = df[is.na(df$Gross) == FALSE,]
print(dim(df))
```

```
## [1] 4558   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).

```
# TODO: Exclude movies released prior to 2000

print(dim(df))
```

```
## [1] 4558   39
```

```
df = df[df$Year >= 2000,]
print(dim(df))
```

```
## [1] 3332   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.*

```
# TODO: Remove mismatched rows

# Function to select the year in the released year
released.in.year = function(x){
if (is.na(x) == FALSE){
   x = as.character(x)
   return(as.numeric(str_sub(x,start = 1,end = 4)))
}
  else {
    return(x)
  }
}

# Function to select only the month in the released year
released.in.month = function(x){
if (is.na(x) == FALSE){
   x = as.character(x)
   return(as.numeric(str_sub(x,start = 6,end = 7)))
}
  else {
    return(x)
  }
}

# Create lists where the previous were applied
new_year = c()
new_month = c()
for (i in 1:length(df$Released)){
    new_year = c(new_year,released.in.year(df$Released[i]))
    new_month = c(new_month,released.in.month(df$Released[i]))
}

# Select the index of the lists (the films) which don't abide by the condition
index_remove = c()
for (i in seq(1,length(new_year))){
  if (is.na(new_year[i])== FALSE){
    if ((((new_year[i] == (df$Year[i]+1)) && (new_month[i] > 3))||
         (new_year[i] > (df$Year[i]+1)))
      || (((new_year[i] == (df$Year[i]-1)) && (new_month[i] < 10))||
          (new_year[i] < (df$Year[i]-1))))
{
        index_remove = c(index_remove,i)
```

```
}
  }
}
# Remove the rows
df4 = df[-index_remove,]



cat("Before removal the number of rows was", dim(df)[1], "and after it is", dim(df4)[1], ".", "The perc
```

```
## Before removal the number of rows was 3332 and after it is 3004 . The percentage of removed rows is
```

```
df = df4
```

5. Drop `Domestic_Gross` column

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

```
# TODO: Exclude the `Domestic_Gross` column

df$Domestic_Gross = NULL
```

6. Process `Runtime` column

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

# Remove_min function allowing to delete the 'mins' and convert the hours in mins

remove_min <- function(x){
if (x != 'NA'){
   x = as.character(x)
   hours = as.numeric(str_match(x, "(\\d*) (h)")[,2])
   hours[is.na(hours) == TRUE] = 0
   minutes = as.numeric(str_match(x, "(\\d*) (m)")[,2])
   minutes[is.na(minutes) == TRUE] = 0
   if (hours > 0){
     return(str_sub(60*hours + minutes))
   }
   else {
     return(str_sub(minutes))
   }
}
}

# Apply the previous function and Conversion to numeric

new = lapply(df$Runtime,remove_min)
df$Runtime = as.numeric(unlist(new))
```

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

```
# TODO(optional): Additional preprocessing

# Will be done after
```

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

Final preprocessed dataset

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

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

dim(df)
```

```
## [1] 3004   38
```

```
names(df)
```

```
##  [1] "Title"             "Year"              "Rated"
##  [4] "Released"          "Runtime"           "Genre"
##  [7] "Director"          "Writer"            "Actors"
## [10] "Plot"              "Language"          "Country"
## [13] "Awards"            "Poster"            "Metascore"
## [16] "imdbRating"        "imdbVotes"         "imdbID"
## [19] "Type"              "tomatoMeter"       "tomatoImage"
## [22] "tomatoRating"      "tomatoReviews"     "tomatoFresh"
## [25] "tomatoRotten"      "tomatoConsensus"   "tomatoUserMeter"
## [28] "tomatoUserRating"  "tomatoUserReviews" "tomatoURL"
## [31] "DVD"               "BoxOffice"         "Production"
## [34] "Website"           "Response"          "Budget"
## [37] "Gross"             "Date"
```

Evaluation Strategy

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

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

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

Tasks

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

1. Numeric variables

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

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

# Function allowing to select the numeric variables

  selection_numeric <- function(df){
  cls <- sapply(df, class)
  numdf <- df %>% select(which(cls == "numeric" | cls == 'double' | cls == 'integer'))
  return(numdf)
  }

  df1 = selection_numeric(df)
  print(dim(df1))
```

```
## [1] 3004   15
```

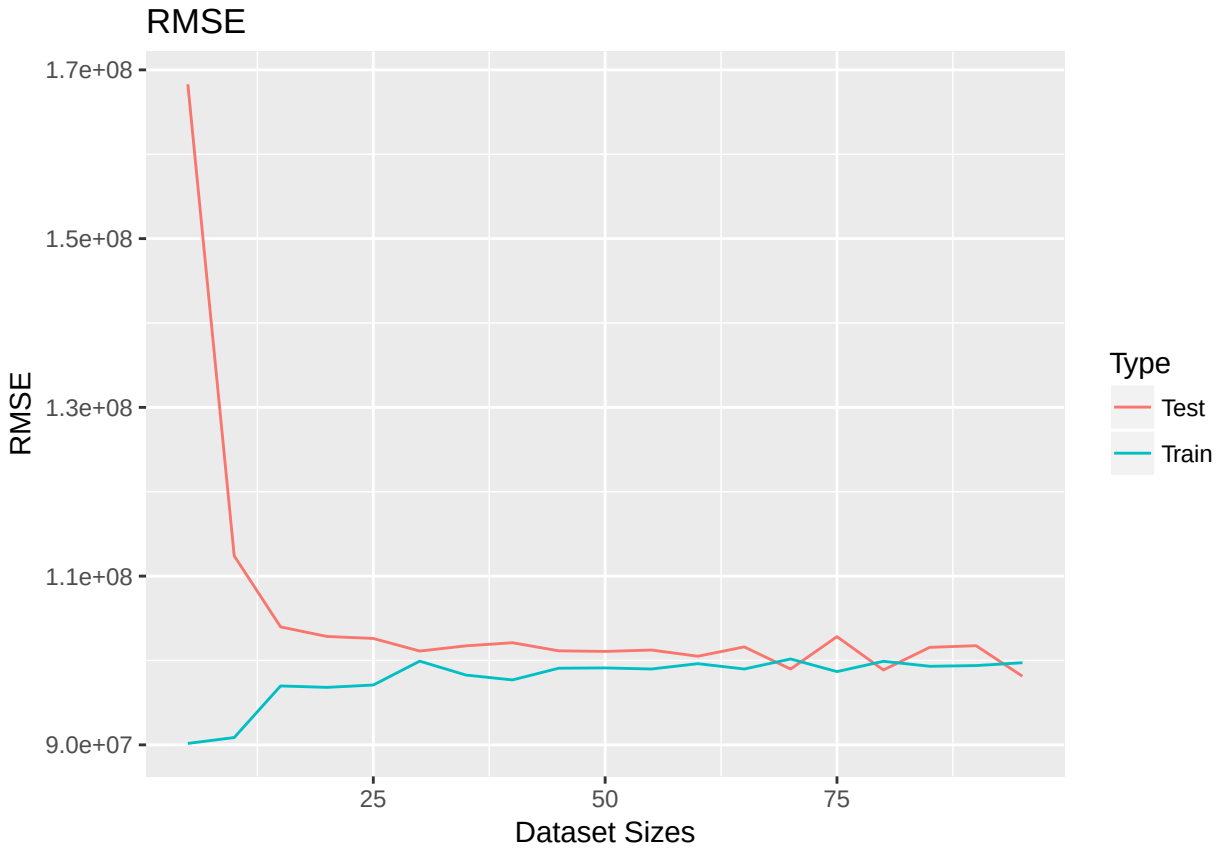
```
  df1 = na.omit(df1)
  print(dim(df1))
```

```
## [1] 2641   15
```

```
  splitdataframe <- function(dataframe,percent){
  index =  1:nrow(dataframe)
  training_index = sample(index, trunc((percent/100)*dim(dataframe)[1]))
  trainset = dataframe[training_index,]
  testset = dataframe[-training_index,]
  return(list(train = trainset,test = testset))
  }

  prediction <- function(numdf,attribute){
  a = numdf[[attribute]]
  numdf[[attribute]] = NULL
  numdf[[attribute]] = a

  car = names(numdf)[1:length(names(numdf))-1]

  car = as.list(car)
  car1 = str_c(car,collapse=' + ')
  car1 = paste(attribute,"~",car1)
  car1 = paste(car1,"+0")
  M = lm(as.formula(car1),numdf)
  return(M)
}
```

```
## 10 times test (change in 100 to smooth the curve)
ten_times_training_rmse <- function(percent_split,df,attribute,prediction){
```

```
  rmse_train = c()
  rmse_test = c()
  for (i in seq(1,100)){
    a = c()
    data = splitdataframe(df,percent_split)
    train = data$train
    test = data$test
    a = test[["Gross"]]
    test[["Gross"]] = NULL
    M_train = prediction(train,"Gross")
    residual_train = resid(M_train)
    RMSE_train = sqrt(mean(residual_train^2))
    rmse_train = c(rmse_train,RMSE_train)
    test_predict = predict(M_train,test)
    RMSE_test = sqrt(mean((a - test_predict)^2))
    rmse_test = c(rmse_test, RMSE_test)
    rmse = rbind(rmse_train,rmse_test)
  }
  return(rmse)
}

#ten_times_training_rmse(95,df3,"Gross",prediction)

percent_split = seq(5,95,by = 5)
rmse_train = c()
rmse_test = c()

for (i in seq(1,length(percent_split))){
  RMSE_train = ten_times_training_rmse(percent_split[i],df1,"Gross",prediction)
  rmse_train = c(rmse_train,mean(RMSE_train[1,]))
  rmse_test = c(rmse_test,mean(RMSE_train[2,]))
}
min_prediction = min(rmse_test)

x = c(percent_split,percent_split)
data_split = as.data.frame(c(Train = rmse_train, Test = rmse_test))
data_split$Type = c(rep("Train",1,length(percent_split)),rep("Test",1,length(percent_split)))
names(data_split) = c('data','Type')
ggplot(data_split,aes(x=x,y=data,col=Type))+geom_line()+ggtitle('RMSE')+xlab('Dataset Sizes') + ylab('RI
```



Q: List all the numeric variables you used.

A: By considering our dataframe, we see the presence of different types of variables as character or numeric ones. A first selection can be done on that,hence we selected the numeric variables which covers "integer", "double" and "numeric" ones. The list of the numeric variables is then : [1] "Year" "Runtime" "imdbRating"
[4] "imdbVotes" "tomatoMeter" "tomatoRating"
[7] "tomatoReviews" "tomatoFresh" "tomatoRotten"
[10] "tomatoUserMeter" "tomatoUserRating" "tomatoUserReviews" [13] "Budget" "Gross" "Date"

These 15 variables are used to compute the prediction model. This one is built according to the lm process of linear prediction. The function "prediction" allows to create the formula with the number of variables we want. We then follow the evaluation strategy described above. The parameter that will alow us to evaluate our model is the RMSE (root mean squared error) which represents the square of the variance. The MSE is a measure of the quality of an estimator and it is always non-negative, and values closer to zero are better. It assesses the quality of our predictor in our case. Indeed, as in the homework 3, we evaluate our model with the cross-validation model. Hence we split our dataframe between training dataset and testing one and we then evaluate the model by computing the average RMSE over multiple iterations, We first began by 10 iterations but because of the random behaviour, we increased the number of iterations until 100. We obtain smoother curves and repeatable ones. It gives us an insight on the behaviour of our training and testing data. As we can see on the results above, the training values are increasing and the testing values decreasing and it finally converge towards 1.0×10^8.

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`).

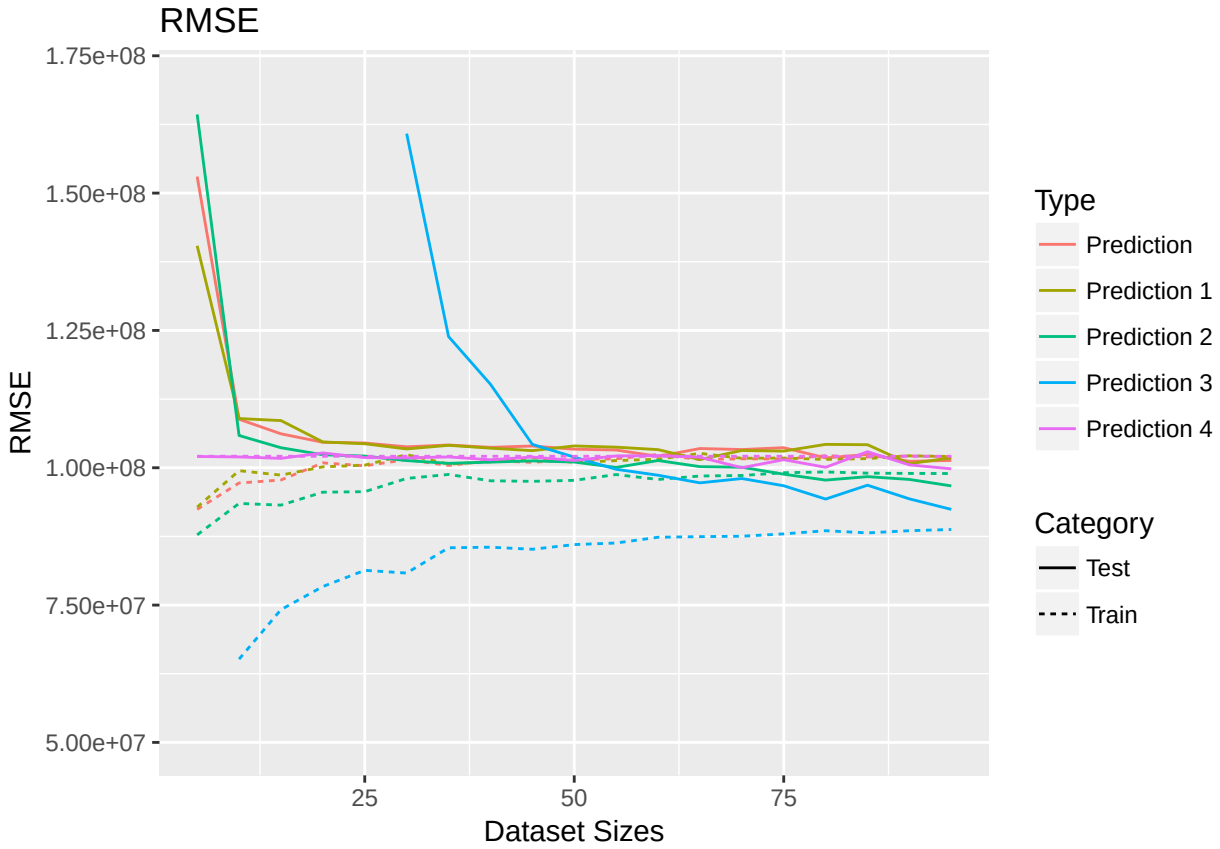
```r
# TODO: Build & evaluate model 2 (transformed numeric variables only)

  # Determination of the correlation between variables and removal of the ones that have the correlatio

  df2 = cor(df1)
  hc = findCorrelation(df2, cutoff = .90, verbose = FALSE)
  hc = sort(hc)
  reduced_data = df1[,-c(hc)]
  reduced_data$tomatoRotten = NULL
  reduced_data$imdbRating = (df1$imdbRating + df1$tomatoRating + df1$tomatoUserRating)/3
  df3 = reduced_data


  # Visualisation of the variable's importance

  M = prediction(df3,"Gross")
  varImp(M)
```

```
##                    Overall
## Runtime           5.466789
## imdbRating        4.067530
## imdbVotes        20.666077
## tomatoMeter       1.582544
## tomatoReviews     2.601577
## tomatoFresh       2.704603
## tomatoUserMeter   5.908537
## tomatoUserReviews 5.721345
## Budget           44.150723
```

```r
  # Prediction function with the sum of squared variables

  prediction1 <- function(numdf,attribute){
  a = numdf[[attribute]]
  numdf[[attribute]] = NULL
  numdf[[attribute]] = a

  car = names(numdf)[1:length(names(numdf))-1]

  car = as.list(car)
  car1 = str_c(car,collapse='^2 +')
  car1 = paste(car1,"^2", sep = "")
  car1 = paste(attribute,"~",car1)
  car1 = paste(car1,"+0")
  M = lm(as.formula(car1),numdf)
  return(M)
  }

  # Prediction function with the sum of some numeric variables and some where the logarithm was applied

  prediction2 <- function(numdf,attribute){
    M <- lm(as.formula("Gross ~ Year + Runtime + imdbRating + imdbVotes + tomatoMeter + tomatoReviews +
  return(M)
  }
```

```
  # Prediction function with the square of the sum of variables

  prediction3 <- function(numdf,attribute){
  a = numdf[[attribute]]
  numdf[[attribute]] = NULL
  numdf[[attribute]] = a

  car = names(numdf)[1:length(names(numdf))-1]

  car = as.list(car)
  car1 = str_c(car,collapse='+')
  car1 = paste(attribute,"~","(",car1,")","^2", sep = "")
  car1 = paste(car1,"+0")
  M = lm(as.formula(car1),numdf)
  return(M)
  }


  # Prediction function with the combinaison of variables and squared ones

  prediction4 <- function(numdf,attribute){
formula <- lm(as.formula("Gross ~ Year + Runtime + imdbRating + imdbVotes + tomatoMeter + tomatoReviews
  return(M)
  }


percent_split = seq(5,95,by = 5)
rmse_train_1 = c()
rmse_test_1 = c()
rmse_train_2 = c()
rmse_test_2 = c()
rmse_train_3 = c()
rmse_test_3 = c()
rmse_train_4 = c()
rmse_test_4 = c()
rmse_train_5 = c()
rmse_test_5 = c()

for (i in seq(1,length(percent_split))){
  RMSE_train_1 = ten_times_training_rmse(percent_split[i],df3,"Gross",prediction)
  rmse_train_1 = c(rmse_train_1,mean(RMSE_train_1[1,]))
  rmse_test_1 = c(rmse_test_1,mean(RMSE_train_1[2,]))
  RMSE_train_2 = ten_times_training_rmse(percent_split[i],df3,"Gross",prediction1)
  rmse_train_2 = c(rmse_train_2,mean(RMSE_train_2[1,]))
  rmse_test_2 = c(rmse_test_2,mean(RMSE_train_2[2,]))
  RMSE_train_3 = ten_times_training_rmse(percent_split[i],df3,"Gross",prediction2)
  rmse_train_3 = c(rmse_train_3,mean(RMSE_train_3[1,]))
  rmse_test_3 = c(rmse_test_3,mean(RMSE_train_3[2,]))
  RMSE_train_4 = ten_times_training_rmse(percent_split[i],df3,"Gross",prediction3)
  rmse_train_4 = c(rmse_train_4,mean(RMSE_train_4[1,]))
  rmse_test_4 = c(rmse_test_4,mean(RMSE_train_4[2,]))
  RMSE_train_5 = ten_times_training_rmse(percent_split[i],df3,"Gross",prediction4)
  rmse_train_5 = c(rmse_train_5,mean(RMSE_train_5[1,]))
```

```
  rmse_test_5 = c(rmse_test_5,mean(RMSE_train_5[2,]))
}

x = c(percent_split,percent_split,percent_split,percent_split,percent_split)
data_split_1 = as.data.frame(c(rmse_train_1,  rmse_test_1))
names(data_split_1)="data"
data_split_2 = as.data.frame(c( rmse_train_2,  rmse_test_2))
names(data_split_2)="data"
data_split_3 = as.data.frame(c( rmse_train_3,  rmse_test_3))
names(data_split_3)="data"
data_split_4 = as.data.frame(c( rmse_train_4,  rmse_test_4))
names(data_split_4)="data"
data_split_5 = as.data.frame(c( rmse_train_5,  rmse_test_5))
names(data_split_5)="data"

data_split = rbind(data_split_1,data_split_2,data_split_3,data_split_4,data_split_5)
data_split$Type = c(rep("Prediction",2*length(percent_split)),rep("Prediction 1",2*length(percent_split)
data_split$Category =  rep(c(rep("Train",length(percent_split)),rep("Test",length(percent_split))),5)
data_split$x=x
ggplot(data_split,aes(x=x,y=data,color=Type, linetype = Category))+geom_line()+ggtitle('RMSE')+xlab('Da
ylim(c(0.5e8,1.7e8))
```

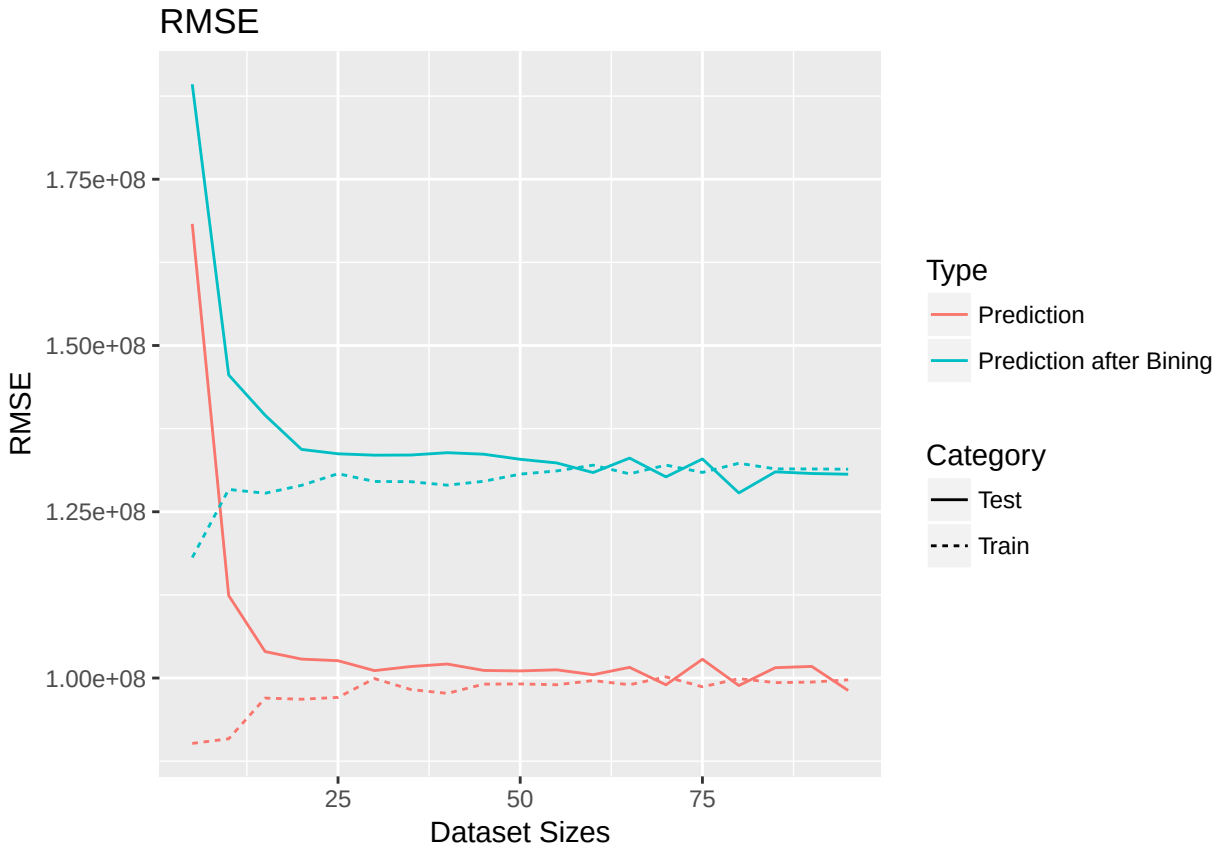



```
# Bining of the budget variable

df4 = df3
df4$binBudget=as.numeric(cut(df4$Budget, c(-Inf,median(df4$Budget),Inf), labels=1:2))
df4$binBudget[df4$binBudget==1]=0
```

```r
df4$binBudget[df4$binBudget==2]=1
df4$Budget=NULL

# Bining of the Runtime variable

variation=cut(df4$Runtime, c(-Inf,90,Inf), labels=1:2)
df4$binRuntime=as.numeric(variation)
df4$binRuntime[df4$binRuntime==1]=0
df4$binRuntime[df4$binRuntime==2]=1
df4$Runtime=NULL

M = prediction(df4,"Gross")
varImp(M)
```

```
##                     Overall
## imdbRating         5.327251
## imdbVotes         26.643898
## tomatoMeter        3.512548
## tomatoReviews      6.119949
## tomatoFresh        2.134052
## tomatoUserMeter    3.975871
## tomatoUserReviews  3.785220
## binBudget         11.330500
## binRuntime         2.567186
```

```r
percent_split = seq(5,95,by = 5)

rmse_train_5 = c()
rmse_test_5 = c()

for (i in seq(1,length(percent_split))){
  RMSE_train_5 = ten_times_training_rmse(percent_split[i],df4,"Gross",prediction)
  rmse_train_5 = c(rmse_train_5,mean(RMSE_train_5[1,]))
  rmse_test_5 = c(rmse_test_5,mean(RMSE_train_5[2,]))
}

x = c(percent_split,percent_split)
data_split_5 = as.data.frame(c(rmse_train_5,  rmse_test_5))
names(data_split_5)="data"
x = c(percent_split,percent_split)
data_split_6 = as.data.frame(c(rmse_train,  rmse_test))
names(data_split_6)="data"

data_split = rbind(data_split_5,data_split_6)
data_split$Type = c(rep("Prediction after Bining",2*length(percent_split)),rep("Prediction",2*length(pe

data_split$Category =  c(rep("Train",length(percent_split)),rep("Test",length(percent_split)),rep("Trai
data_split$x=x
#names(data_split) = c('data','Type')
ggplot(data_split,aes(x=x,y=data,color=Type, linetype = Category))+geom_line()+ggtitle('RMSE')+xlab('Da
```




```
ylim(c(0.5e8,1.7e8))
```

```
## <ScaleContinuousPosition>
##  Range:
##  Limits: 5e+07 -- 1.7e+08
```

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

A: Given our results in question 1 and the fact that we considered all the numeric variables, the first step was to preprocess the data by removing the correlated variables. To do that we used the correlation coefficient and we setted a threshold at 90% of correlation, above that value we removed the columns. It gives us :
[1] "Year" "Runtime" "imdbRating"
[4] "imdbVotes" "tomatoMeter" "tomatoReviews"
[7] "tomatoFresh" "tomatoRotten" "tomatoUserMeter"
[10] "tomatoUserReviews" "Budget" "Gross" By observing the results of this preprocessing, we decided to remove the "tomatoRotten" column because "tomatoReviews" = "tomatoFresh" + "tomatoRotten", as we have the three present in the above list and as the result of the application of the prediction function gives NA for tomatoRotten. Another thing is that we know that tomatoRating, tomatoUserRating and imdbRating are correlated, and that's the reason why we only have one of them after removing the highly correlated variables. However, we judged that the three of them are almost the same, so we prefered to calculate the average over the three of them instead of letting only one variable. Finally the variables we will use in our prediction functions is : [1] "Year" "Runtime" "imdbRating"
[4] "imdbVotes" "tomatoMeter" "tomatoReviews"
[7] "tomatoFresh" "tomatoUserMeter" "tomatoUserReviews" [10] "Budget" "Gross"
In order to see the impact of our selected vaiables on the prediction, we use varImp which shows the importance of each variable. Hence we can see some logical results as the importance of the Budget variable (the more you invest in a film, the more you will earn money). ImdbVotes has also a large weight, indeed the more people vote for a film, the more it is reknown and then provide money. Now after having done a first selection

of our variables, the second step is to find the best prediction model, we tested 5 different power transform :

- Prediction 1 : Sum of the squared variables. The first test is to combine squared variables, the result is really close to the sum of simple variables which is logical since we give the same weight to all variables. This a way of comparing the next transformations we will obtain.
- Prediction 4 : For this model, we take advantage of the results we get about the importance of each variable. That is, We used the sum of all variables + the sum of the square of the most important variables that are in our case: Budget, imdbVotes, tomatoUserMeter, imdbRating, tomatoUserReviews. This is meant for giving the important variables more impact on the regression task
- Prediction 3 : After several tests over the classical transformations as the identity (I(sum(variables))) or the product of all the variables (variable1:variable2:variable3...), which are not really indicated in our situation since it provide only one column. We tried the square of the sum of all the variables, which provide a combination of different variables and then create interaction between them.
- Prediction 2 : This model is similar to the third one,apart from the fact of instead of adding the square of the important variables to the formula, we added the log of these variables.
- Prediction Bining : Another type of prediction was tested as non-numeric transformations of the numeric variables. The aim is to transform numerical variables into binary columns, the confidence in that type of prediction is not huge since linear regression use continuous variables, hence turning the variables which have the most important impact into quite meaningless variables won't improve at all the results we had in the first question. The variables that are the more adapted to bining are Budget and Runtime because there are well spread on the range as we saw when we visualized some scatter plots in the project 1 and again the summary of the columns confirm that. So we whose our threshold for budget at the median and for runtime we selected the one that seemed to be the most important break. As we said before the results are disappointing. Indeed, by comparing the importance we obtained in the previous question on the budget, we can see that the importance of the binary column budget has been divided by 4 : (Budget = 44.150723 vs binBudget = 11.330500). This is the same for the Runtime column, since it is divided by 2 : (Runtime : 5.466789 vs binRuntime = 2.567186). So it does not increase the weight of these variables but reduce it a lot.

Analyse of the results : We did the prediction task with all these models, and plotted their RMSE on the same graph shown above so that we could compare between them and see which one improves the best our model from task 1. However, in order to compare between all of them, we were constrained to do a zoom on our graph. So we only see the interesting part of it. We can observe that except from the prediction model 3, all models are almost similar to the model from task 1 since their training RMSEs all increase with percentage of split and their testing RMSEs decrease with the increase of percentage of split until they converge towards almost 1.0×10^8. Concerning prediction 3, It improves our model since it converges toward 8.8×10^7. Another thing we can notice about this model is that it need a higher percentage of split to start to converge.

To confirm all what we stated, we used this model for the prediction of "Gross", and again, we plotted its RMSE along with the RMSE obtained in **task 1**, and we can clearly observe that this model worsens the prediction as it only converges toward almost 1.32×10^8.

3. Non-numeric variables

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

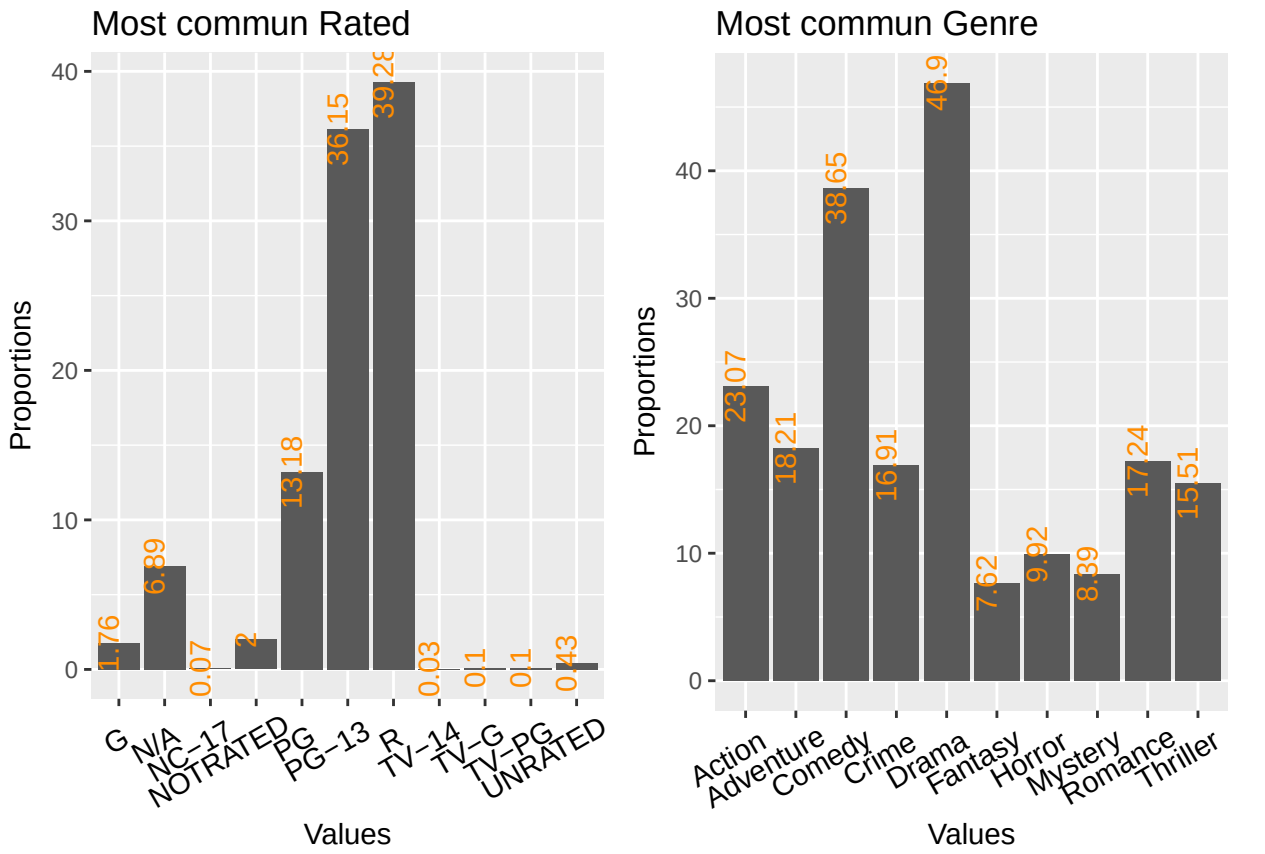
```
# TODO: Build & evaluate model 3 (converted non-numeric variables only)

# Function selecting the n most frequent classes of a categorical variable
```

```
select_ten_best_values <- function(df,n){

  list = c()
  # Sum all the 1 in each column
  list = lapply(df,sum)
  data_frame.to.be.ordered = data.frame(list)
  names(data_frame.to.be.ordered) = names(df)
  # Order the list in decreasing sum
  data_frame.ordered = data_frame.to.be.ordered[rev(order(sapply(list,'[[',1)))]
  # Create the data frame for the most common types
  data.frame.10.first.values = unlist(data_frame.ordered[,1:n], use.names=TRUE)
  data.frame.10.first = df[,names(data.frame.10.first.values)]

return(data.frame.10.first)
}

# Binary conversion of a categorical variable

binary_conversion <- function(df,attribute,n,choice = 1){

    df[[attribute]] = gsub(" ", "", df[[attribute]])

    list_attributes = c()
    temp_attributes = c()

    for (i in seq(1,length(df[[attribute]]))) {
      temp_attributes = unlist(strsplit(df[[attribute]][i],","))
      list_attributes = c(list_attributes,temp_attributes)
    }

    duplicated.values = which(duplicated(list_attributes))
    list.attributes.bis = list_attributes[-duplicated.values]
    list.attributes.bis = sort(list.attributes.bis)

    map = matrix(0,length(df[[attribute]]),length(list.attributes.bis))

    for (j in seq(1,length(df[[attribute]]))){

      temp_attributes = unlist(strsplit(df[[attribute]][j],","))

      for (k in seq(1,length(temp_attributes))){
        index = which(temp_attributes[k] == list.attributes.bis)
        map[j,index] = 1
      }
    }
    dim_old = (dim(df)[2])+1
    data_frame = data.frame(map)
    names(data_frame) = list.attributes.bis
    data_frame_ten = select_ten_best_values(data_frame,n)
    if (choice == 1){
      df = cbind(df,data_frame_ten)
      return(df)}
    else{
```

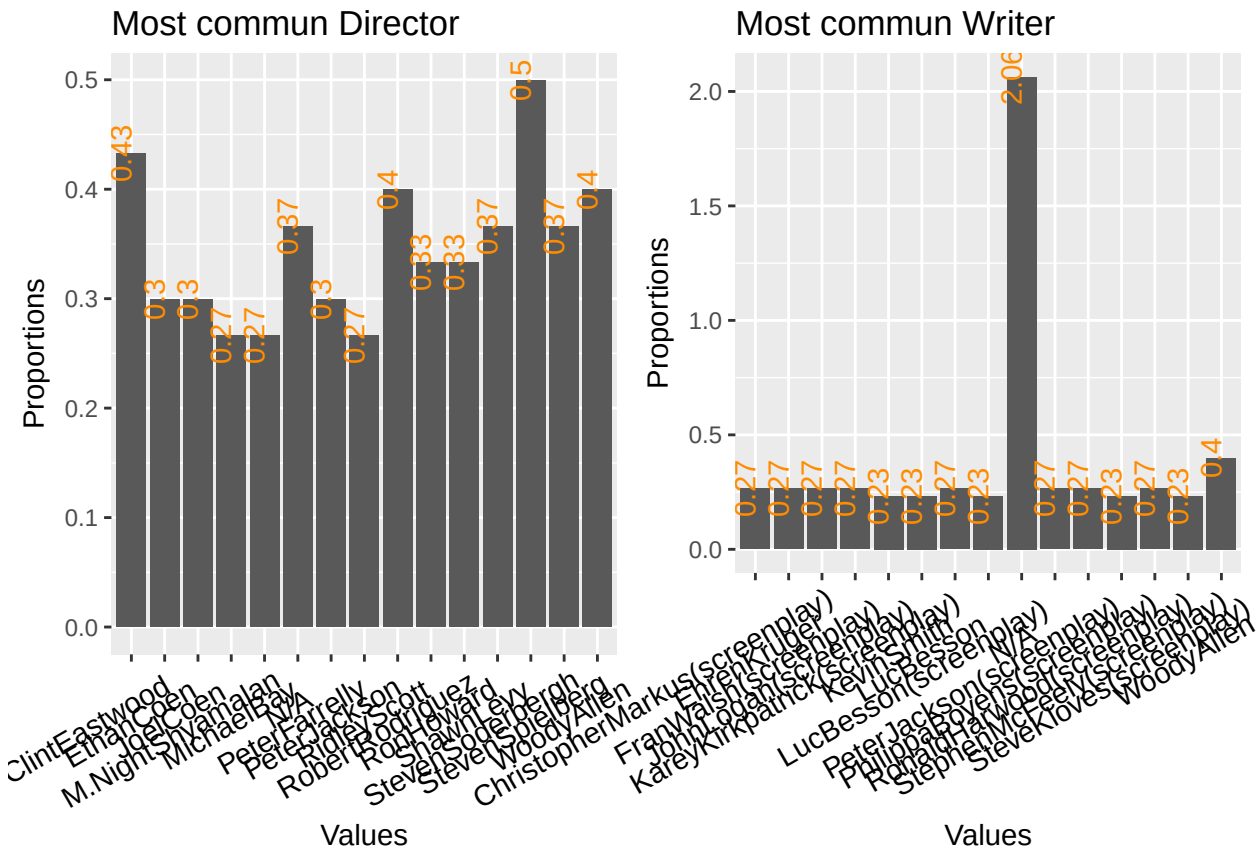
```
      return(data_frame_ten)
    }
}

# Function allowing to select non numeric variables

selection_non_numeric <- function(df){
  cls <- sapply(df, class)
  nonnumdf <- df %>% select(which(cls == "character"))
  return(nonnumdf)
}
```

```
# Construction of histograms

histogram_10_first_values <- function(df,type,n){
number_values = length(unique(df[[type]]))
if (number_values > 15){
  df1 = binary_conversion(df,type,n,choice = 0)
}
else{
  df1 = binary_conversion(df,type,number_values,choice = 0)
}
# Sum all the 1 in each column
list = lapply(df1,sum)
# Order the list in decreasing sum
list = list[rev(order(sapply(list,'[[',1)))]
# Create the data frame for the 10 most common types
list.10.first = head(list,n)
data.10.first = as.data.frame(list.10.first)
data_total = as.data.frame(list)
sum_total = dim(df1)[1]
data.10.first.relative = (data.10.first/sum_total)*100
temp = c()
for (i in seq(1,length(data.10.first.relative))){
  temp = c(temp,data.10.first.relative[,i])
}
data_frame.10.first.relative = data.frame(attribute_type = names(list.10.first),proportions = temp)

p <- ggplot(data=data_frame.10.first.relative, aes(x = data_frame.10.first.relative$attribute_type, y=da
geom_bar(stat="identity")+
geom_text(aes(label=round(data_frame.10.first.relative$proportions,digits = 2)),angle = 90, vjust=0, col
theme(axis.text.x=element_text(color = "black", size=11, angle=30, vjust=.8, hjust=0.8))+
xlab('Values')+
ylab('Proportions')
return(p)
}

list_representation = c("Rated","Genre","Director","Writer","Actors","Language","Country", "Production"

p1 <- histogram_10_first_values(df,"Rated",15) + ggtitle("Most commun Rated")
p2 <- histogram_10_first_values(df,"Genre",10) + ggtitle("Most commun Genre")
p3 <- histogram_10_first_values(df,"Director",15) + ggtitle("Most commun Director")
p4 <- histogram_10_first_values(df,"Writer",15) + ggtitle("Most commun Writer")
p5 <- histogram_10_first_values(df,"Actors",15) + ggtitle("Most commun Actors")
```

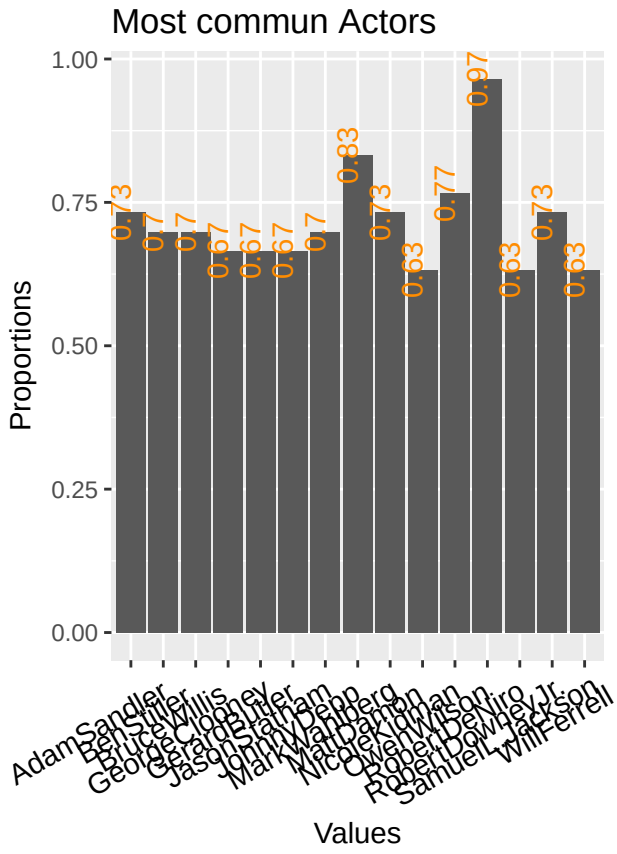
```
p6 <- histogram_10_first_values(df,"Language",15) + ggtitle("Most commun Language")
p7 <- histogram_10_first_values(df,"Country",15) + ggtitle("Most commun Country")
p8 <- histogram_10_first_values(df,"Production",15) + ggtitle("Most commun Production")

grid.newpage()
pushViewport(viewport(layout = grid.layout(1, 2)))
vplayout <- function(x, y) viewport(layout.pos.row = x, layout.pos.col = y)
print(p1, vp = vplayout(1, 1))
print(p2, vp = vplayout(1, 2))
```

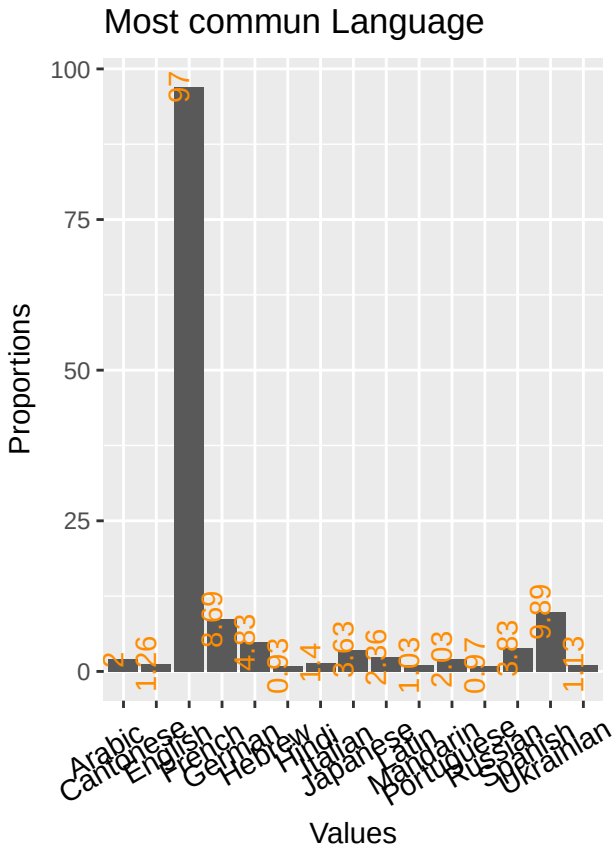



```
grid.newpage()
pushViewport(viewport(layout = grid.layout(1, 2)))
vplayout <- function(x, y) viewport(layout.pos.row = x, layout.pos.col = y)
print(p3, vp = vplayout(1, 1))
print(p4, vp = vplayout(1, 2))
```

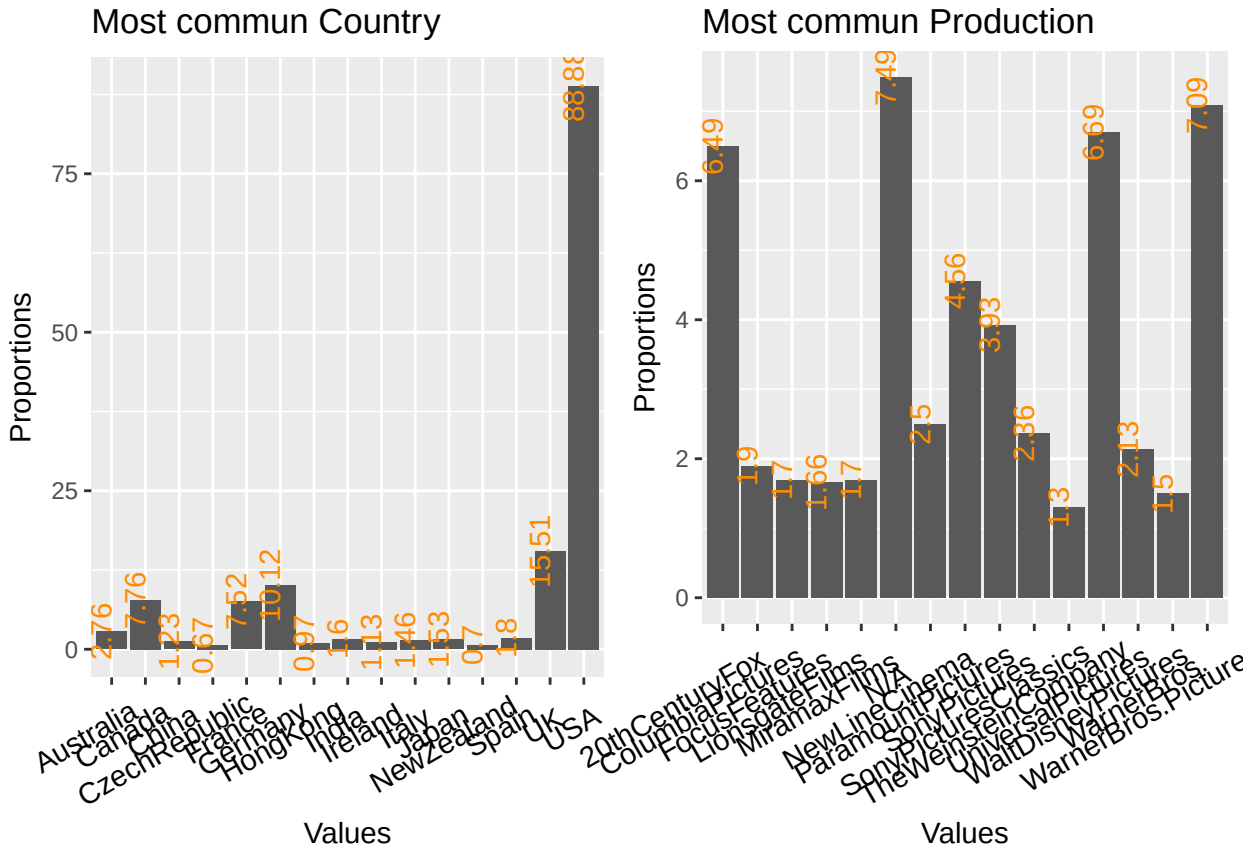



```
grid.newpage()
pushViewport(viewport(layout = grid.layout(1, 2)))
vplayout <- function(x, y) viewport(layout.pos.row = x, layout.pos.col = y)
print(p5, vp = vplayout(1, 1))
print(p6, vp = vplayout(1, 2))
```






```
grid.newpage()
pushViewport(viewport(layout = grid.layout(1, 2)))
vplayout <- function(x, y) viewport(layout.pos.row = x, layout.pos.col = y)
print(p7, vp = vplayout(1, 1))
print(p8, vp = vplayout(1, 2))
```




```
list_conversion = names(selection_non_numeric(df))
print(list_conversion)
```

```
##  [1] "Title"           "Rated"           "Genre"
##  [4] "Director"        "Writer"          "Actors"
##  [7] "Plot"            "Language"        "Country"
## [10] "Awards"          "Poster"          "Metascore"
## [13] "imdbID"          "Type"            "tomatoImage"
## [16] "tomatoConsensus" "tomatoURL"       "BoxOffice"
## [19] "Production"      "Website"         "Response"
```

```
list_conversion = c("Rated","Genre", "tomatoImage", "Production")

df_cat = c()
df_cat = selection_non_numeric(df)
df_cat = df_cat[list_conversion]

for (i in seq(1, length(list_conversion))){
  if (list_conversion[i] == 'tomatoImage'){
    df_cat = binary_conversion(df_cat,list_conversion[i],3,choice = 1)
    df_cat[[list_conversion[i]]] = NULL
    }
  if (list_conversion[i] == 'Rated'){
    df_cat = binary_conversion(df_cat,list_conversion[i],4,choice = 1)
    df_cat[[list_conversion[i]]] = NULL
  }
  if (list_conversion[i] == 'Genre'){
  df_cat = binary_conversion(df_cat,list_conversion[i],10,choice = 1)
```

```
  df_cat[[list_conversion[i]]] = NULL
  }
  if (list_conversion[i] == 'Production'){
  df_cat = binary_conversion(df_cat,list_conversion[i],6,choice = 1)
  df_cat[[list_conversion[i]]] = NULL
  }
  df_cat[["N/A"]] = NULL
  }

# Adapting the variables names to the application of the prediction function

transform_variables <- function(numdf,attribute){
  a = numdf[[attribute]]
  numdf[[attribute]] = NULL
  numdf[[attribute]] = a

  car = names(numdf)[1:length(names(numdf))-1]
  car <- gsub('[[:punct:]]','',car)
  names(numdf)[1:length(names(numdf))-1] = car
  return(numdf)
}

# Prediction function adpated to the character variables

prediction_char <- function(numdf,attribute){
  a = numdf[[attribute]]
  numdf[[attribute]] = NULL
  numdf[[attribute]] = a

  car = names(numdf)[1:length(names(numdf))-1]
  pbs1 =  as.numeric(gsub("\\D", "", car))
  car <- gsub('[[:punct:]]','',car)
  names(numdf)[1:length(names(numdf))-1] = car
  car[which(pbs1 != 'NA')] = paste("`",car[which(pbs1 != 'NA')] ,"`", sep ="")
  car = as.list(car)
  car1 = str_c(car,collapse=' + ')
  car1 = paste(attribute,"~",car1)
  car1 = paste(car1,"+0")
  M = lm(car1,numdf)
return(M)
  }


ten_times_training_rmse_char <- function(percent_split,df,attribute,prediction,transform = 1){

  rmse_train = c()
  rmse_test = c()
  for (i in seq(1,100)){
    a = c()
    data = splitdataframe(df,percent_split)
    train = data$train
    test = data$test
    if (transform == 1){
```

```
      test = transform_variables(test,attribute)
    }
    a = test[[attribute]]
    test[[attribute]] = NULL
    M_train = prediction(train,"Gross")
    residual_train = resid(M_train)
    RMSE_train = sqrt(mean(residual_train^2))
    rmse_train = c(rmse_train,RMSE_train)
    test_predict = predict(M_train,test)
    RMSE_test = sqrt(mean((a - test_predict)^2))
    rmse_test = c(rmse_test, RMSE_test)
    rmse = rbind(rmse_train,rmse_test)
  }
  return(rmse)
}
```

```
df_cat$Gross = df$Gross

# Conversion of awards to wins and nominations

# Convert to lowercase
df$Awards = tolower(df$Awards)

# Select all the patterns "win" and "wins" and return the number associated
wins = as.numeric(str_match(df$Awards, "(\\d*) (w)")[,2])
wins[is.na(wins) == TRUE] = 0

# Select the pattern "won "and return the number associated
more_wins = as.numeric(str_match(df$Awards, "won (\\d*)")[,2])
more_wins[is.na(more_wins) == TRUE] = 0

# Sum the two previous lists
wins = wins + more_wins
length_wins = length(wins[which(wins != 0)])
sprintf("The number of valid/non zeros wins is %i",length_wins)
```

```
## [1] "The number of valid/non zeros wins is 1805"
```

```
# Select all the patterns "nomination" and "nominations" and return the number associated
nominations = as.numeric(str_match(df$Awards, "(\\d*) (n)")[,2])
nominations[is.na(nominations) == TRUE] = 0

# Select the pattern "nominated for "and return the number associated
more_nominations = as.numeric(str_match(df$Awards, "nominated for (\\d*)")[,2])
more_nominations[is.na(more_nominations) == TRUE] = 0

# Sum the two previous lists
nominations = nominations + more_nominations


length_nominations = length(nominations[which(nominations != 0)])
length_wins_nominations = length((which(nominations != 0 | wins != 0)))
sprintf("The number of valid/non zeros nominations is %i",length_nominations)
```

```
## [1] "The number of valid/non zeros nominations is 2336"
```

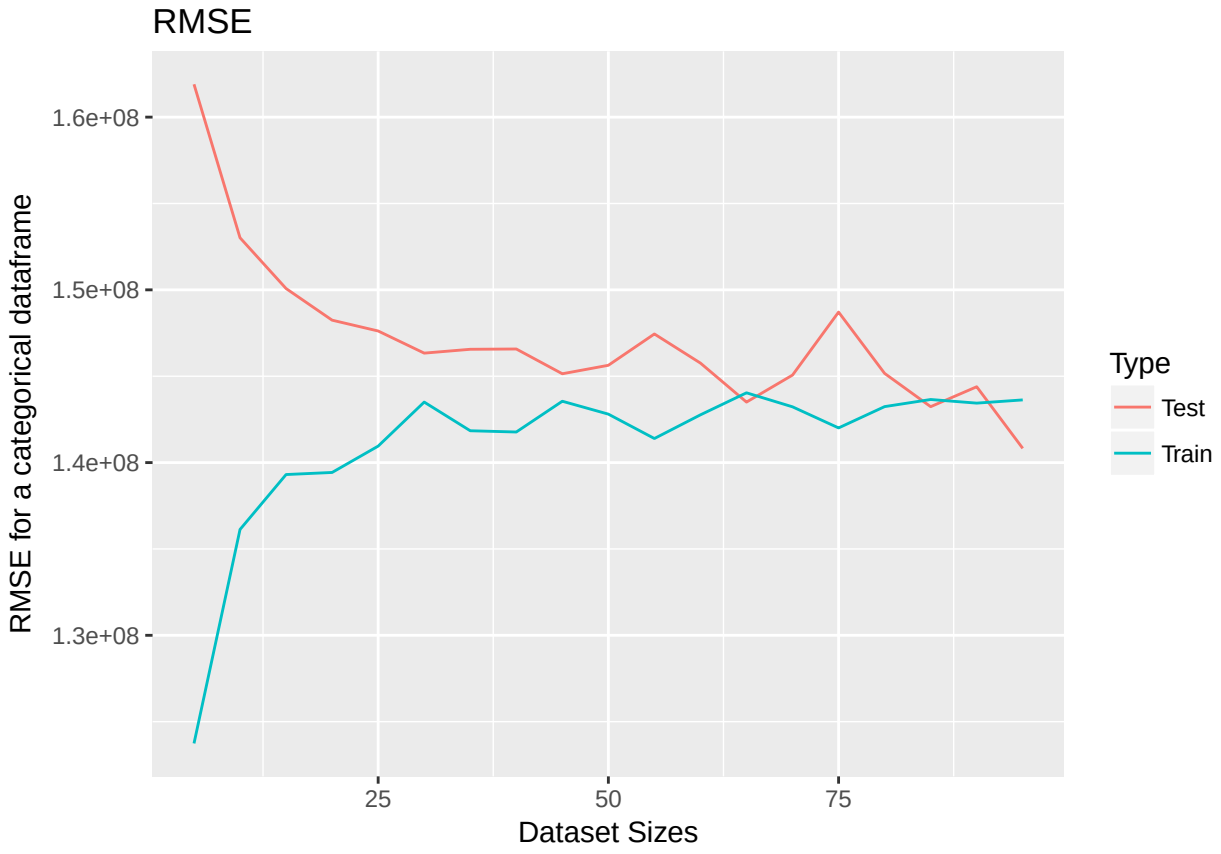
```
sprintf("The number of valid/non zeros nominations or wins is %i",length_wins_nominations)
```

```
## [1] "The number of valid/non zeros nominations or wins is 2447"
```

```
df_cat$Wins = wins
df_cat$Nominations = nominations

df_cat[df_cat== "N/A"]  <- NA
df_cat=na.omit(df_cat)

M = prediction_char(df_cat,"Gross")
varImp(M)
```

```
##                       Overall
## PG13                5.0419174
## PG                  3.4725351
## Drama               8.5485756
## Comedy              0.5133570
## Action              7.2962689
## Adventure          15.7369836
## Romance             0.1066721
## Crime               0.8969831
## Thriller            2.6683708
## Horror              1.2455905
## Mystery             0.4064884
## Fantasy             5.5519690
## rotten              1.8834502
## certified           4.7502759
## fresh               4.4117021
## WarnerBrosPictures  5.5218261
## UniversalPictures   4.8994671
## `20thCenturyFox`    4.5321486
## ParamountPictures   0.7793938
## SonyPictures        3.8847415
## Wins                1.7020116
## Nominations         9.0166311
```

```
percent_split = seq(5,95,by = 5)
rmse_train = c()
rmse_test = c()

for (i in seq(1,length(percent_split))){
  RMSE_train = ten_times_training_rmse_char(percent_split[i],df_cat,"Gross",prediction_char, 1)
  rmse_train = c(rmse_train,mean(RMSE_train[1,]))
  rmse_test = c(rmse_test,mean(RMSE_train[2,]))
}

x = c(percent_split,percent_split)
data_split = as.data.frame(c(Train = rmse_train, Test = rmse_test))
data_split$Type = c(rep("Train",1,length(percent_split)),rep("Test",1,length(percent_split)))
names(data_split) = c('data','Type')
ggplot(data_split,aes(x=x,y=data,col=Type))+geom_line()+ggtitle('RMSE')+xlab('Dataset Sizes') + ylab('RM
```



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

A: Now we have studied the numeric variables, let's see if we can use the character ones : [1] "Title" "Rated" "Genre" "Director" "Writer"
[6] "Actors" "Plot" "Language" "Country" "Awards"
[11] "Poster" "Metascore" "imdbID" "Type" "tomatoImage"
[16] "tomatoConsensus" "tomatoURL" "BoxOffice" "Production" "Website"
[21] "Response"
By looking at their meaning and their correlation, we removed a lot of them. Hence we end up with : "Rated" "Genre" "Director" "Writer" "Actors" "Language" "Country" "Production" This list will be completed by the tranformation of awards into wins and nominations. As it is impossible and more importantly as it is useless to use all the categories of those variables, we select the most important ones ie the ones that are the most freuqently appearing. After having done binary conversion by following the same model that we used in the project 1, we use a function that select all the classes if their number is less than 15 and let us choose the number of variables if it is superior. In order to visualize the most important categories, we represent some histogrames as we have done in the project 1 for genre. Just by visualizing the histograms and their values, we can eliminate "Writer","Actor" and "Director" because the fifteen first are really not representative since their frequency of appearing within all the movies is lower that 1% apart from some exceptions. Concerning "Language" and "Country" since only one category is present (USA for country and English for Language) is present at more than 80%. We thought that they would not be of any help in the regression task. This leaves us with : "Rated","Genre", "tomatoImage", "Production". For those, we selected the 3 categories of tomatoImage, the 4 most important categories of Rated and the 10 most common Genre as well as the 10 most frequent production.

After having created our new categorical dataframe, we again evaluate the importance of each variable to anticipate the results of our prediction. Before analyzing any curves results, we can say that it won't be good since the variable that has the most important weight is "Adventure" (15.7369836) and that the others are all less than 10.000000 which is for recall 4 times less than Budget (our most impactful variable so far). The

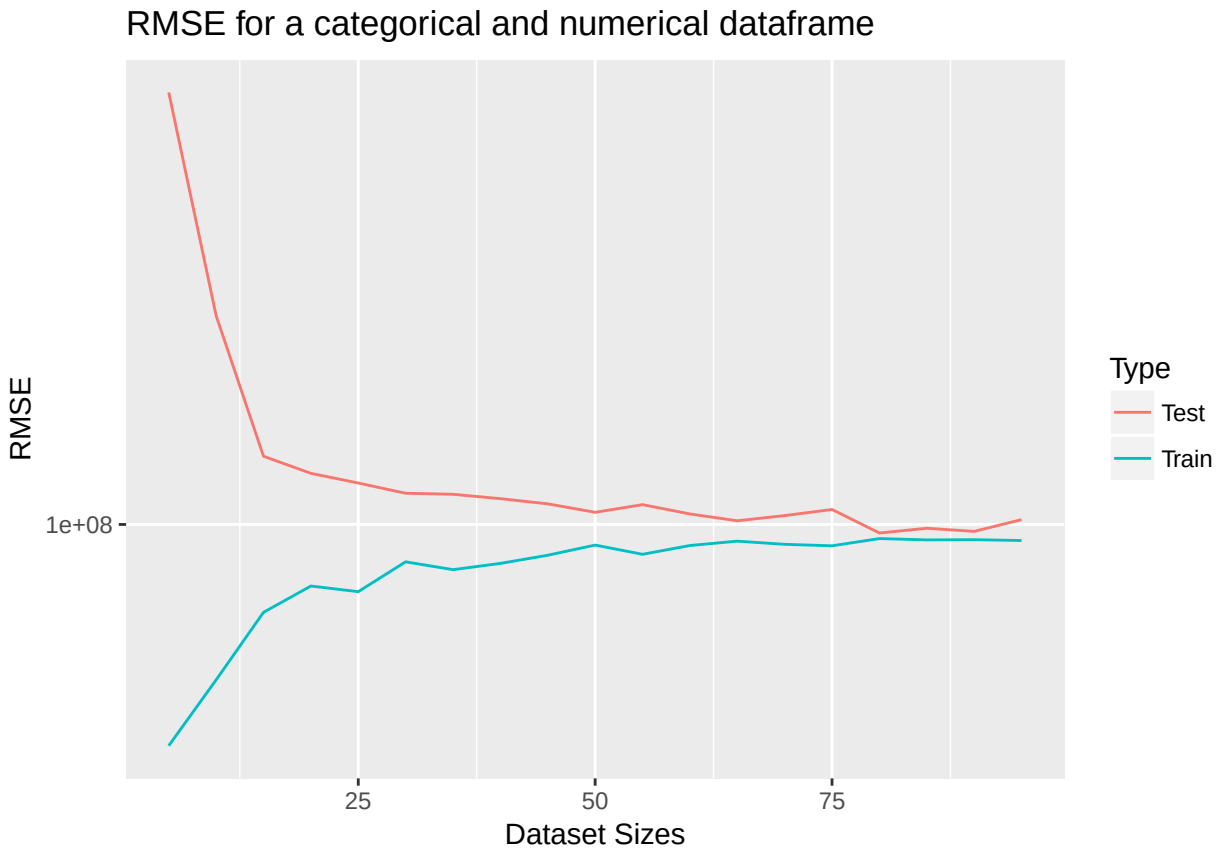
second most important variable here is nominations which is 9 times more powerful than wins.

The previous allegation is confirmed by our bad results on the curves indeed the training and testing curves converge around 1.45×10^8. Indeed by computing the percentage between the minimum of the rmse for the testing curve in this question and the one in the first question, we obtain an increasing 43%. So this part provides no improvement.

4. Numeric and categorical variables

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

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

prediction_char_1 <- function(numdf,attribute){
  a = numdf[[attribute]]
  numdf[[attribute]] = NULL
  numdf[[attribute]] = a

  car = names(numdf)[1:length(names(numdf))-1]
  pbs1 =  as.numeric(gsub("\\D", "", car))
  car <- gsub('[[:punct:]]','',car)
  names(numdf)[1:length(names(numdf))-1] = car
  car[which(pbs1 != 'NA')] = paste("`",car[which(pbs1 != 'NA')] ,"`", sep ="")
  car = as.list(car)
  car1 = str_c(car,collapse='+')
  car1 = paste(attribute,"~","(",car1,")","^2", sep = "")
  car1 = paste(car1,"+0")
  M = lm(as.formula(car1),numdf)
return(M)
}

df_num = df[names(df3)]
df_char_num = cbind(df_cat,df_num)
df_char_num = na.omit(df_char_num)
percent_split = seq(5,95,by = 5)
rmse_train = c()
rmse_test = c()

for (i in seq(1,length(percent_split))){
  RMSE_train = ten_times_training_rmse_char(percent_split[i],df_char_num,"Gross",prediction_char, 1)
  rmse_train = c(rmse_train,mean(RMSE_train[1,]))
  rmse_test = c(rmse_test,mean(RMSE_train[2,]))
}

x = c(percent_split,percent_split)
data_split = as.data.frame(c(Train = rmse_train, Test = rmse_test))
data_split$Type = c(rep("Train",1,length(percent_split)),rep("Test",1,length(percent_split)))
names(data_split) = c('data','Type')
ggplot(data_split,aes(x=x,y=data,col=Type))+geom_line()+ggtitle('RMSE for a categorical and numerical da
```



For this question, we were asked to combine categorical variables with numeric variables. After that, we plotted the RMSE for this model, and we obtained a quite better result than **task1** as our training and testing curves converge toward 9.65×10^7.

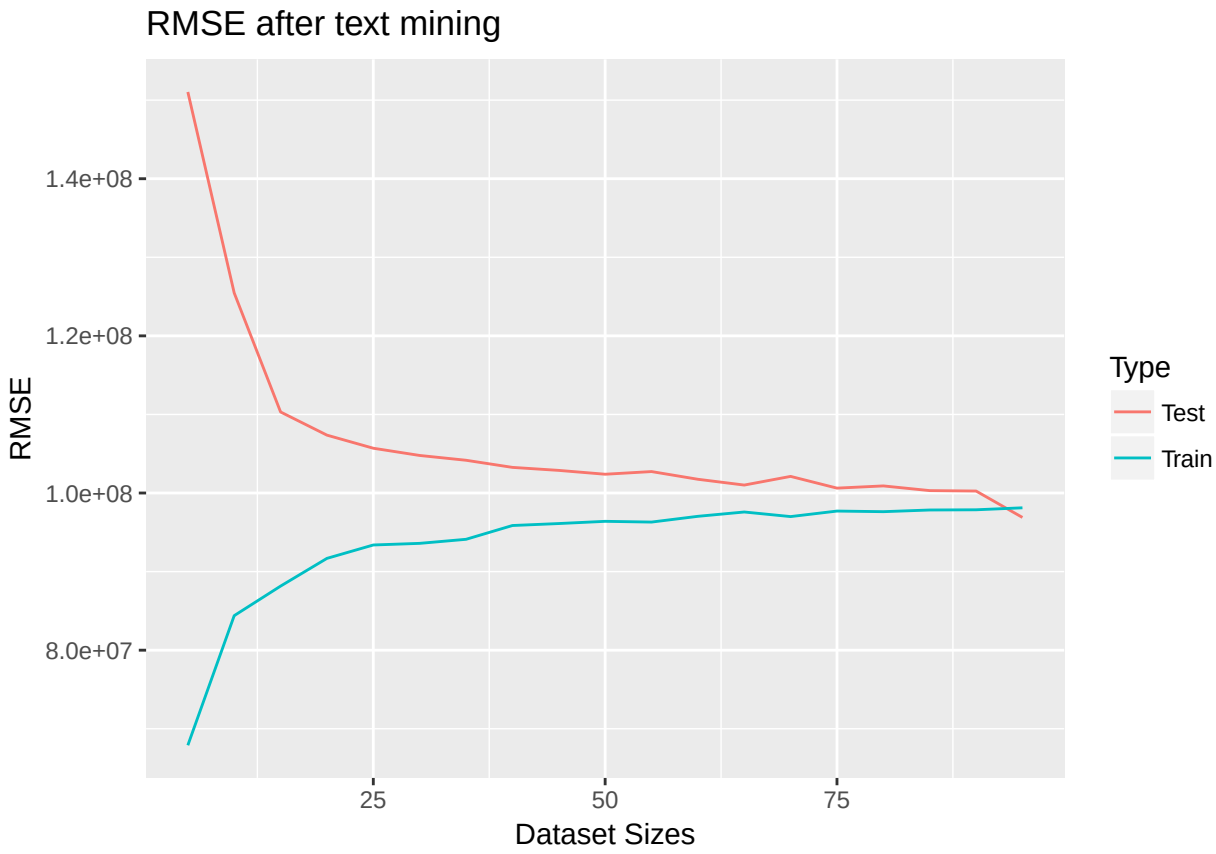
5. Additional features

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

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

# Analysis of tomatoConsensus

df$tomatoConsensus <- tolower(df$tomatoConsensus)
df$tomatoConsensus <- tm::removeNumbers(df$tomatoConsensus)
df$tomatoConsensus <- str_replace_all(df$tomatoConsensus, "  ", "") # replace double spaces with single
df$tomatoConsensus <- str_replace_all(df$tomatoConsensus, pattern = "[[:punct:]]", " ")

terms_tomatoConsensus = freq_terms(text.var = df$tomatoConsensus, top = 25) # find the 25 most frequent

# Analysis of plot

df$Plot <- tolower(df$Plot)
df$Plot <- tm::removeNumbers(df$Plot)
df$Plot <- str_replace_all(df$Plot, "  ", "") # replace double spaces with single space
df$Plot <- str_replace_all(df$Plot, pattern = "[[:punct:]]", " ")
```

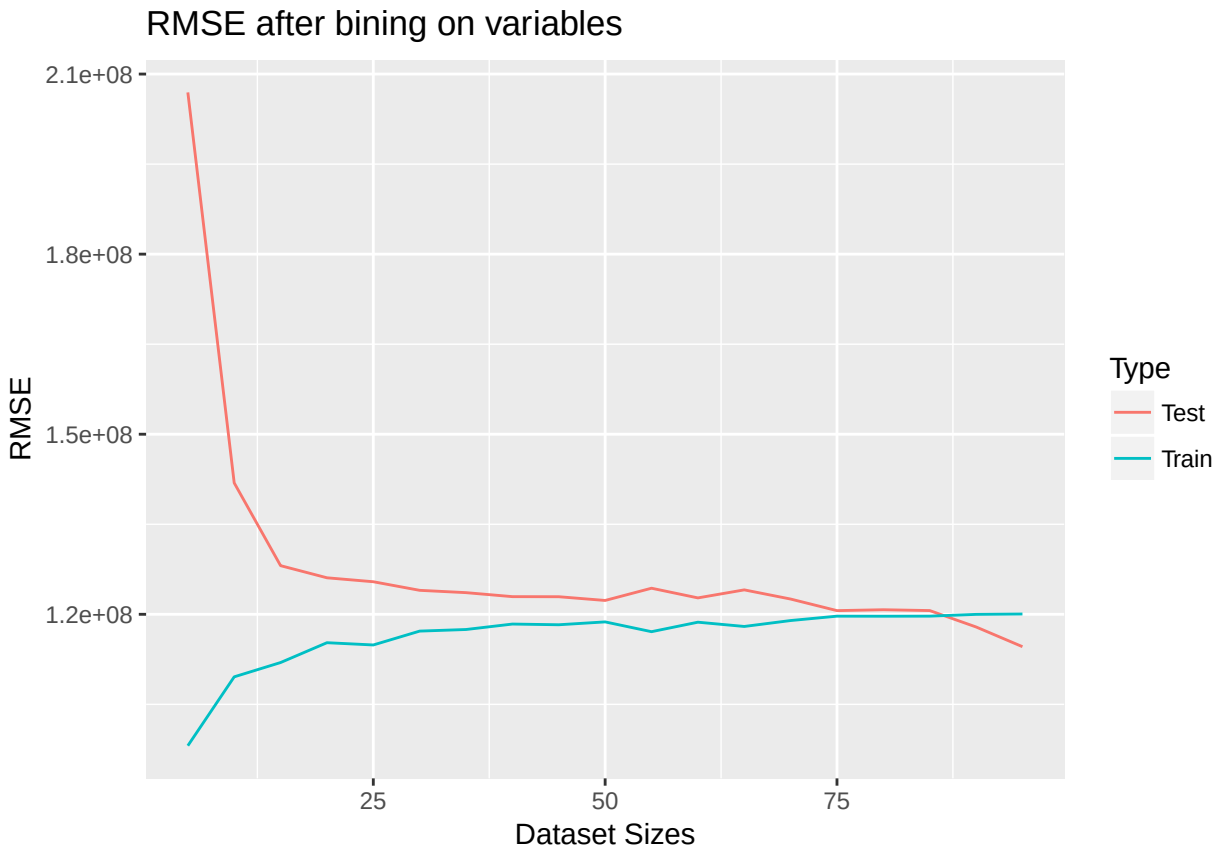
```
df$Plot <- tm::removeWords(x = df$Plot, stopwords(kind = "SMART"))

terms_Plot = freq_terms(text.var = df$Plot, top = 25) # find the 25 most frequent words

score.dictionnary = function(sentence, dic.words)
{    # clean up sentences with R's regex-driven global substitute, gsub():
    sentence = gsub('[[:punct:]]', '', sentence)
    sentence = gsub('[[:cntrl:]]', '', sentence)
    sentence = gsub('\\d+', '', sentence)
    # and convert to lower case:
    sentence = tolower(sentence)

    # split into words. str_split is in the stringr package
    word.list = str_split(sentence, '\\s+')
    # sometimes a list() is one level of hierarchy too much
    words = unlist(word.list)

    # compare our words to the dictionaries of positive & negative terms
    dic.matches = match(dic.words, words)

    # match() returns the position of the matched term or NA
    # we just want a TRUE/FALSE:
    dic.matches = !is.na(dic.matches)
    dic.matches = as.numeric(dic.matches)
    return(dic.matches)}

binary_dataframe_tomatoConsensus = c()
for (i in seq(1, length(df$tomatoConsensus))){
  binary_dataframe_tomatoConsensus = rbind(binary_dataframe_tomatoConsensus,score.dictionnary(df$tomato
}
binary_dataframe_tomatoConsensus = as.data.frame(binary_dataframe_tomatoConsensus)
names(binary_dataframe_tomatoConsensus) = terms_tomatoConsensus$WORD

binary_dataframe_plot = c()
for (i in seq(1, length(df$tomatoConsensus))){
  binary_dataframe_plot = rbind(binary_dataframe_plot,score.dictionnary(df$tomatoConsensus[i],terms_toma
}
binary_dataframe_plot = as.data.frame(binary_dataframe_plot)
names(binary_dataframe_plot) = terms_Plot$WORD


df_num = df[names(df3)]
df_char_num_bin = cbind(df_cat,df_num,binary_dataframe_plot,binary_dataframe_tomatoConsensus)
df_char_num_bin = na.omit(df_char_num_bin)
df_char_num_bin[["in"]] = NULL
df_char_num_bin[["for"]] = NULL

percent_split = seq(5,95,by = 5)
rmse_train = c()
rmse_test = c()

for (i in seq(1,length(percent_split))){
  RMSE_train = ten_times_training_rmse_char(percent_split[i],df_char_num_bin,"Gross",prediction_char, 1
```

```
  rmse_train = c(rmse_train,mean(RMSE_train[1,]))
  rmse_test = c(rmse_test,mean(RMSE_train[2,]))
}

x = c(percent_split,percent_split)
data_split = as.data.frame(c(Train = rmse_train, Test = rmse_test))
data_split$Type = c(rep("Train",1,length(percent_split)),rep("Test",1,length(percent_split)))
names(data_split) = c('data','Type')
ggplot(data_split,aes(x=x,y=data,col=Type))+geom_line()+ggtitle('RMSE after text mining')+xlab('Dataset
```

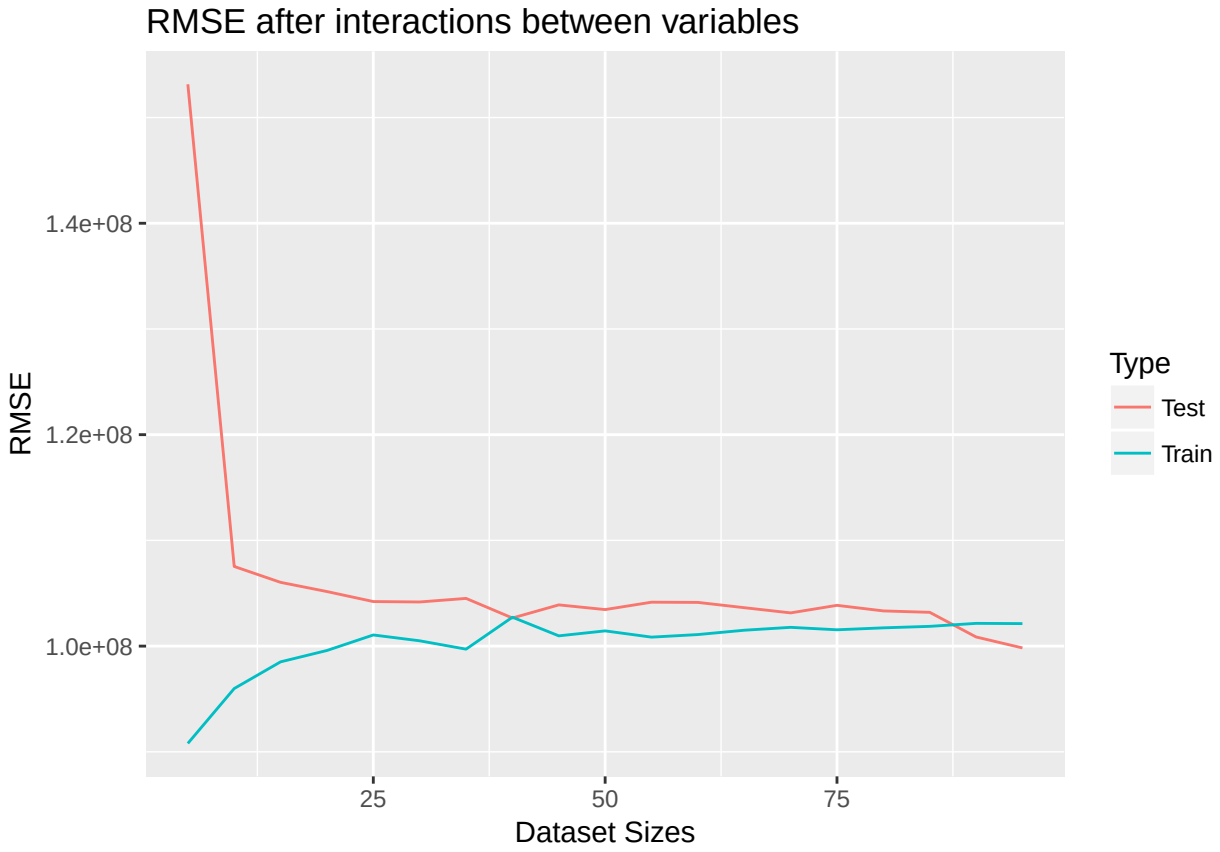



```
df_num = df[names(df3)]
df_char_num = cbind(df_cat,df_num)
df_char_num$Title=df$Title
df_char_num = na.omit(df_char_num)

# Bining of the budget

df_bin = df_char_num
df_bin$binBudget=as.numeric(cut(df_bin$Budget, c(-Inf,median(df_bin$Budget),Inf), labels=1:2))
df_bin$binBudget[df_bin$binBudget==1]=0
df_bin$binBudget[df_bin$binBudget==2]=1
df_bin$Budget=NULL

# Creation of a variable with the interaction of the binary column budget and the one for the comedy ge
df_bin$budget_comedy=df_bin$binBudget * df_bin$Comedy
df_bin$Comedy=NULL
```

```
df_bin$binBudget=NULL


# Creation of a column filled with the title's length

vect=sapply(gregexpr("\\S+", df_bin$Title), length)
df_bin$titlelength=vect
df_bin$Title=NULL

var=cut(df_bin$Runtime, c(-Inf,90,Inf), labels=1:2)
df_bin$binRuntime=as.numeric(var)
df_bin$binRuntime[df_bin$binRuntime==1]=0
df_bin$binRuntime[df_bin$binRuntime==2]=1
df_bin$Runtime=NULL

df_bin$runtime_certified=df_bin$binRuntime * df_bin$certified
df_bin$certified=NULL
df_bin$binRuntime=NULL


df_bin$title_rating=df_bin$imdbRating*1/df_bin$titlelength
df_bin$imdbRating=NULL
df_bin$titlelength=NULL

percent_split = seq(5,95,by = 5)
rmse_train = c()
rmse_test = c()

for (i in seq(1,length(percent_split))){
  RMSE_train = ten_times_training_rmse_char(percent_split[i],df_bin,"Gross",prediction_char, 1)
  rmse_train = c(rmse_train,mean(RMSE_train[1,]))
  rmse_test = c(rmse_test,mean(RMSE_train[2,]))
}

x = c(percent_split,percent_split)
data_split = as.data.frame(c(Train = rmse_train, Test = rmse_test))
data_split$Type = c(rep("Train",1,length(percent_split)),rep("Test",1,length(percent_split)))
names(data_split) = c('data','Type')
ggplot(data_split,aes(x=x,y=data,col=Type))+geom_line()+ggtitle('RMSE after bining on variables')+xlab(
```




```
  prediction6 <- function(numdf,attribute){
formula <- lm(as.formula("Gross ~ Year + Runtime + imdbRating + imdbVotes + tomatoMeter + tomatoReviews
  return(M)
  }

percent_split = seq(5,95,by = 5)
rmse_train = c()
rmse_test = c()

for (i in seq(1,length(percent_split))){
  RMSE_train = ten_times_training_rmse_char(percent_split[i],df3,"Gross",prediction, 1)
  rmse_train = c(rmse_train,mean(RMSE_train[1,]))
  rmse_test = c(rmse_test,mean(RMSE_train[2,]))
}

x = c(percent_split,percent_split)
data_split = as.data.frame(c(Train = rmse_train, Test = rmse_test))
data_split$Type = c(rep("Train",1,length(percent_split)),rep("Test",1,length(percent_split)))
names(data_split) = c('data','Type')
ggplot(data_split,aes(x=x,y=data,col=Type))+geom_line()+ggtitle('RMSE after interactions between variabl
```



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

A: The last variables we haven't used so far because they can't be exploited as there as more complicated are "Plot" and "tomatoConsensus". This analysis is text mining. Several approaches can be applied on those types of variables. We chose to move towards some frequency words and sentiment analysis. Unfortunately, the dictionnaries that we have in the tm package do not allow us to obtain results, since the vocabulary of "Plot" and "tomatoConsensus" is not at all the same. We hence proceed a frequency words analysis by picking the 25 most repeated words and by converting those two character variables into binary variables. The curves that we obtain are quite satisfactory since by computing the minimum of the RMSE test curve and by comparing it to the rmse of the first question, we obtain a decreasing of 2.8%. It converges faster that in the first question also.

For this part, we tried to create additional features such as interactions between variables.So, we tried the interaction is_Budget_greater> median(Budget)* is_genre_comedy, is_Runtime_greater>90 : is_certified. Another feature that seemed interesting is the combination of the inverse of length of the title with the Ratings. We added these features and exploited this model to do the regression task. As we talked before, bining does not work realy well, even with interaction on variables, so the percentage of "increasement" of our performances is not surprising since the RMSE increased by 16% so results are again really bad but better than question 3 where only binary columns were used and no interactions were done. It converges towards 115350199.

Finally, we tried to work on other combinaison of numeric variables which could improve the results we had in question 1 since besides text mining that provided better results, question 1 remains our best chance. By observing which variables could be related, we came up with a new formula. According to the work done in project 1, it seemed a good idea to create interaction between budget and runtime, budget and each rating or votes and rating. The curves does not provide better results since there is an increasing of 2.3% in the RMSE. But as this is slight evolution, we can say that it does not change much .