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

submitted by: Frank Hahn ; gtid: fhahn3

Data

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

Objective

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

Instructions

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

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

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

Setup

Same as Project 1, load the dataset into memory:

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

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

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

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

```
colnames(df)
```

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

Load R packages

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

```
library(ggplot2)
library(GGally)
library(tm)
```

```
## Loading required package: NLP
```

```
##
## Attaching package: 'NLP'
```

```
## The following object is masked from 'package:ggplot2':
##
##     annotate
```

```
library(psych)
```

```
##
## Attaching package: 'psych'
```

```
## The following objects are masked from 'package:ggplot2':
##
##     %+%, alpha
```

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

Non-standard packages used: None

Data Preprocessing

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

1. Remove non-movie rows

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

2. Drop rows with missing `Gross` value

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

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

3. Exclude movies released prior to 2000

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

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

4. Eliminate mismatched rows

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

```
# TODO: Remove mismatched rows

# convert
df$Released_year<- as.numeric(format(df$Released,'%Y'))

#need to compare Year and Released_year
sum(is.na(df$Released)&!is.na(df$Gross))
```

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

```
sum(df$Year==df$Released_year, na.rm=TRUE)
```

```
## [1] 2379
```

```
# df[is.na(df$Released)&!is.na(df$Gross),]
mismatchexpression <- (abs(df$Year-df$Released_year)<2)
# df$Gross[!is.na(df$Gross)]

df_mismatch <- subset(df, mismatchexpression)

DFwithgross <-sum(!is.na(df$Gross))
DFMMwithgross<- sum(!is.na(df_mismatch$Gross))
deleted <- (DFwithgross-DFMMwithgross)/DFwithgross*100
cat("I deleted ", deleted,"% of the rows with gross")
```

```
## I deleted  2.349689 % of the rows with gross
```

```
df <- df_mismatch
```

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 <-df[ , !(names(df) %in% "Domestic_Gross")]
```

6. Process `Runtime` column

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

converttoMin <- function(movietime){
  #if contains h then convert to minutes
  if (grepl("h", movietime)) {
    # cat("Contains h")
    timelist =strsplit(movietime, " ")
    hour = strtoi (timelist[[1]][1])
    min = strtoi (timelist[[1]][3])
    min = hour*60+min
    #contains min, strip to the front, capture number
  } else if (grepl("min", movietime)) {
    # cat("contains just min")
    timelist =strsplit(movietime, " ")
    min = strtoi (timelist[[1]][1])
    #else "N/A"
  } else {
    # cat("N/A")
    min="N/A"
  }
  return(min)
}
# converttoMin(movietime)

suppressWarnings(df$Runtime <- lapply(df$Runtime, converttoMin ))
suppressWarnings(df$Runtime <- as.numeric(df$Runtime))
```

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

```
# TODO(optional): Additional preprocessing

df <-df[ , !(names(df) %in% "tomatoRotten")]
```

***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
print(dim(df))
```

```
## [1] 2826   38
```

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

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 (%).

```
# 1. Randomly divide the rows into two sets of sizes 5% and 95%.

numeric.only <- function(df.train){
  nums <- sapply(df.train, is.numeric)
  df.train   <- df.train[,nums]
  return(df.train)
}



sampling.splits<- seq(.05,.95, .05)

sample.train.test.data <- function(random=.05, df.tosample){
  # random <- .05
  sample.count <- nrow(df.tosample)
```

```r
  sample.index <- sample(sample.count, sample.count*random)
  df.train <<-df.tosample[sample.index,]
  df.test <<-df.tosample[-sample.index,]
}

sample.train.test.data(random=.05, df)

# 2. Use the first set for training and the second for testing.
# df.train<-numeric.only(df.train)
# mod.test <- lm(df.train$Gross~., data=df.train)
# summary(mod.test)

# 3. Compute the Root Mean Squared Error (RMSE) on the train and test sets.
get.RMSE <- function(lm.model, df.to.predict){
  predicted.value <- predict.lm(lm.model, df.to.predict)
  difference <- predicted.value-df.to.predict$Gross
  difference <- as.matrix(difference)
  count.of.good.predictions<-sum(!is.na(difference))
  RMSE <-sqrt(sum(difference^2, na.rm = TRUE)/count.of.good.predictions)
  return(RMSE)
}

getRMSE.ten.times <- function(df, model.cmd="df.train$Gross~."){
  iteration=20
  df.rmse <- data.frame(numeric(iteration*19),numeric(iteration*19),numeric(iteration
*19), stringsAsFactors = FALSE )
  colnames(df.rmse) <- c("random","rmse.train", "rmse.test")
  count=0

  for (random in sampling.splits){

    for (i in 1:iteration){
      sample.train.test.data(random, numeric.only(df))
      mod.test <- lm(as.formula(model.cmd), data=df.train)
      RMSE.train <-get.RMSE(mod.test, df.train)
      RMSE.test <-get.RMSE(mod.test, df.test)
      df.rmse[i+count,]<- c(random, RMSE.train, RMSE.test)
    }
    count=count+iteration
  }
  print(summary(mod.test))
  return(df.rmse)
}

# 4. Repeat the above data partition and model training and evaluation 10 times and a
verage 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)
df.q1 <- numeric.only(df)
df.rmse <- getRMSE.ten.times(numeric.only(df.q1), model.cmd="df.train$Gross~.")
```

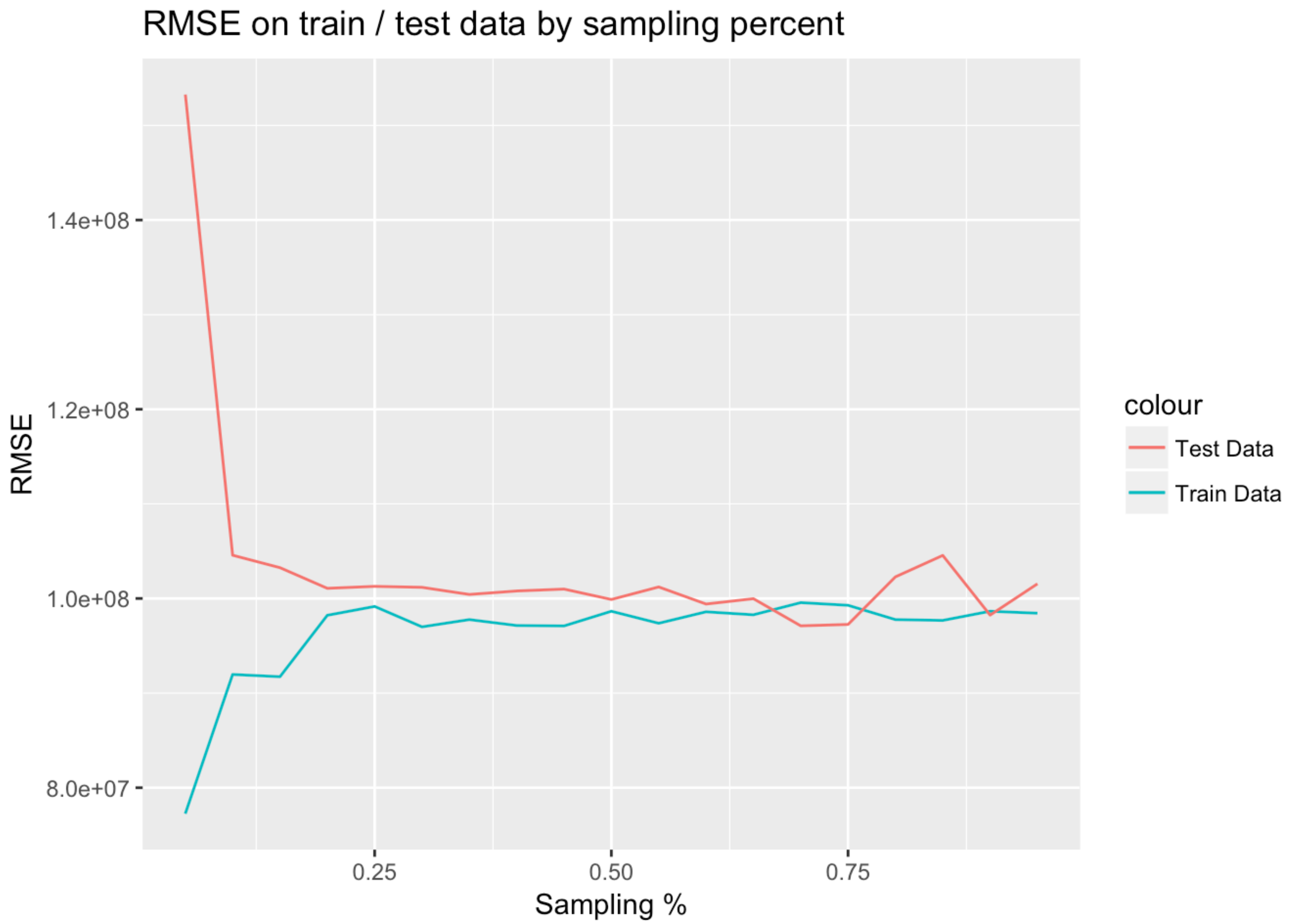
```
##
## Call:
## lm(formula = as.formula(model.cmd), data = df.train)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -441090095  -35893030   -1028531   28583292 1359900353
##
## Coefficients:
##                     Estimate Std. Error t value Pr(>|t|)
## (Intercept)       -1.309e+09  1.190e+09  -1.100   0.2714
## Year              -5.677e+06  6.350e+06  -0.894   0.3714
## Runtime           -5.894e+05  1.310e+05  -4.499 7.14e-06 ***
## imdbRating        -2.864e+07  4.229e+06  -6.771 1.59e-11 ***
## imdbVotes          4.899e+02  2.313e+01  21.185  < 2e-16 ***
## tomatoMeter        8.677e+05  3.455e+05   2.511   0.0121 *
## tomatoRating      -1.387e+07  6.857e+06  -2.023   0.0432 *
## tomatoReviews     -7.098e+04  8.604e+04  -0.825   0.4095
## tomatoFresh        1.370e+05  1.262e+05   1.086   0.2778
## tomatoUserMeter   -6.164e+05  3.619e+05  -1.703   0.0886 .
## tomatoUserRating   9.911e+07  1.418e+07   6.990 3.52e-12 ***
## tomatoUserReviews  4.356e+00  5.857e-01   7.438 1.40e-13 ***
## Budget             2.401e+00  5.826e-02  41.223  < 2e-16 ***
## Date               3.697e+06  5.520e+06   0.670   0.5031
## Released_year      2.606e+06  6.273e+06   0.415   0.6778
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 97440000 on 2502 degrees of freedom
##   (167 observations deleted due to missingness)
## Multiple R-squared:  0.7353, Adjusted R-squared:  0.7338
## F-statistic: 496.4 on 14 and 2502 DF,  p-value: < 2.2e-16
```

```
df.rmse.avg <- aggregate(df.rmse,list(df.rmse$random),data=df.rmse,FUN="mean")
```

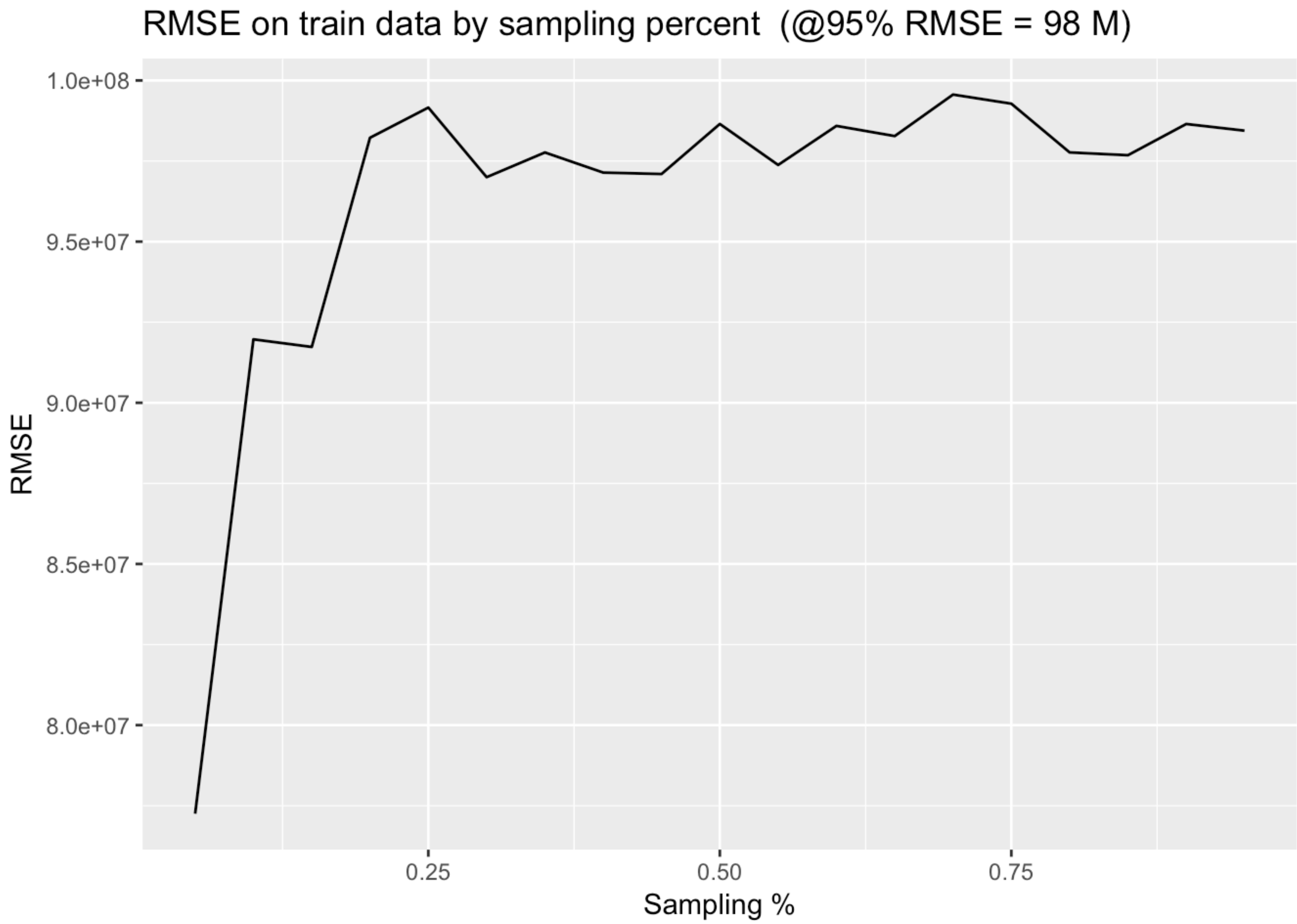
```
print(df.rmse.avg)
```

```
##    Group.1 random rmse.train rmse.test
## 1     0.05   0.05   77252200 153260668
## 2     0.10   0.10   91968865 104572383
## 3     0.15   0.15   91732182 103255966
## 4     0.20   0.20   98222910 101072510
## 5     0.25   0.25   99158936 101281753
## 6     0.30   0.30   96999694 101178277
## 7     0.35   0.35   97766167 100425057
## 8     0.40   0.40   97141979 100793793
## 9     0.45   0.45   97097764 100994013
## 10    0.50   0.50   98650213  99893478
## 11    0.55   0.55   97378868 101218557
## 12    0.60   0.60   98590363  99412562
## 13    0.65   0.65   98274653  99979514
## 14    0.70   0.70   99562011  97108162
## 15    0.75   0.75   99279674  97257132
## 16    0.80   0.80   97767256 102284210
## 17    0.85   0.85   97681166 104550984
## 18    0.90   0.90   98649584  98242979
## 19    0.95   0.95   98442679 101550640
```

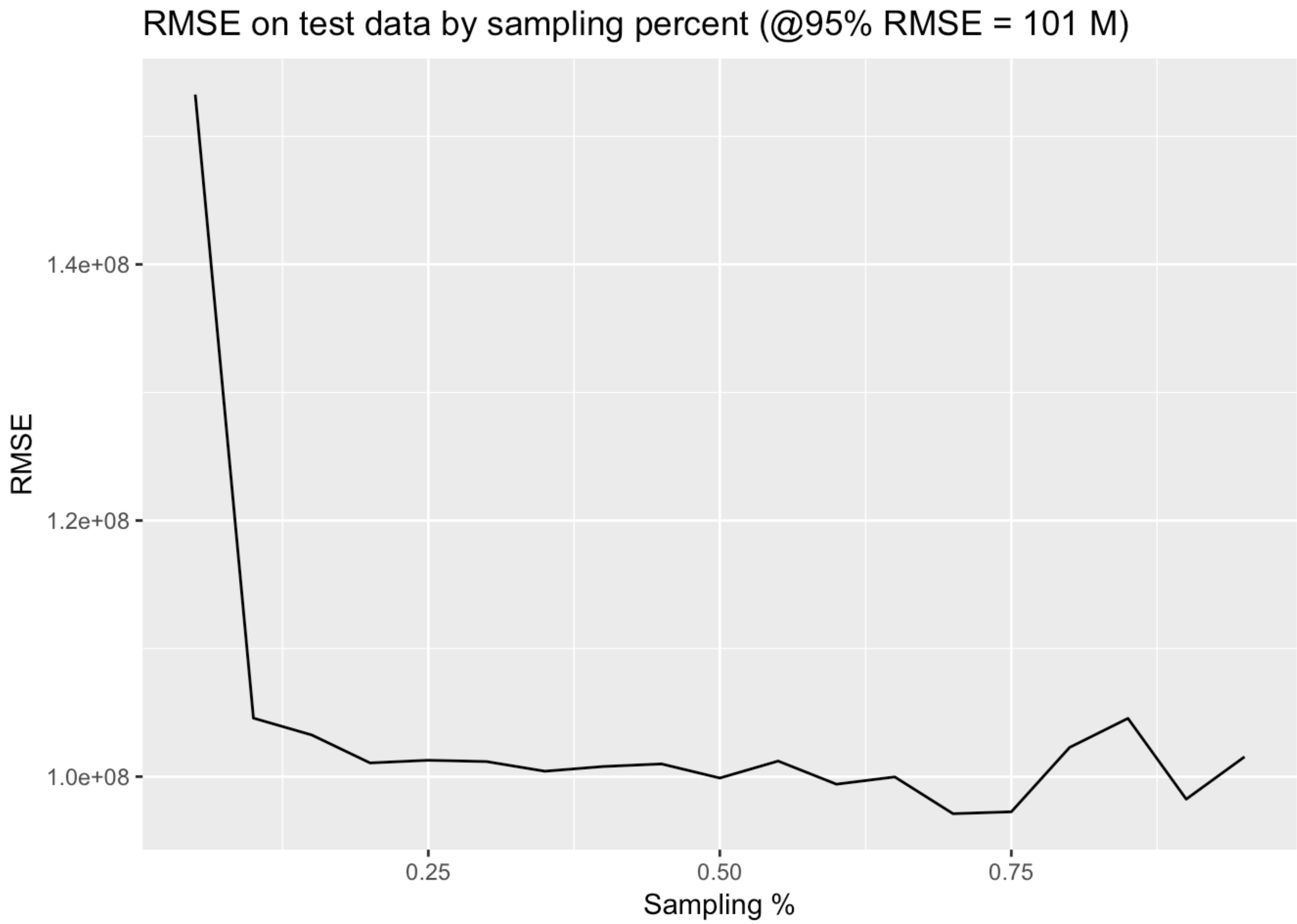
```
ggplot(df.rmse.avg, aes(x= df.rmse.avg$random))+geom_line(aes(y = df.rmse.avg$rmse.tr
ain, colour = 'Train Data')) + geom_line(aes(y = df.rmse.avg$rmse.test, colour = 'Tes
t Data')) +labs(title="RMSE on train / test data by sampling percent ", x="Sampling %
",y= "RMSE")
```




```
qplot(x= df.rmse.avg$random, y=df.rmse.avg$rmse.train, geom = "line")+ labs(title=pas
te("RMSE on train data by sampling percent ", "(@95% RMSE =", as.integer(df.rmse.avg$
rmse.train[19]/ 1e6) , "M)", sep=" "), x="Sampling %",y= "RMSE")
```




```
qplot(x= df.rmse.avg$random, y=df.rmse.avg$rmse.test, geom = "line")+labs(title=paste
("RMSE on test data by sampling percent", "(@95% RMSE =", as.integer(df.rmse.avg$rmse
.test[19]/ 1e6) , "M)", sep=" "), x="Sampling %",y= "RMSE")
```



Q: List all the numeric variables you used.

A:
[1] “Year” “Runtime” “imdbRating” “imdbVotes” “tomatoMeter”
[6] “tomatoRating” “tomatoReviews” “tomatoFresh” “tomatoUserMeter” “tomatoUserRating” [11] “tomatoUserReviews” “Budget” “Gross” “Date” “Released_year”

2. Feature transformations

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

```
# TODO: Build & evaluate model 2 (transformed numeric variables only)
df.q2 <- df.q1

df.q2$Budgetbw50_100[(df.q2$Budget>5e7)&(df.q2$Budget<1e8)]=1
df.q2$Budgetbw50_100[!(df.q2$Budget>5e7)&(df.q2$Budget<1e8)]=0



df.q2$imdbVotesless300k[(df.q2$imdbVotes<300000)]=1
df.q2$imdbVotesless300k[!(df.q2$imdbVotes<300000)]=0

df.q2$imdbVotessqrt<-log(df.q2$imdbVotes)
df.q2$imdbVotessqrtmore350[(df.q2$imdbVotessqrt<350)]=1
df.q2$imdbVotessqrtmore350[!(df.q2$imdbVotessqrt<350)]=0

# df.q2$imdbVotesless300k[(df.q2$imdbVotes<300000)&(df.q2$imdbVotes>100000)]=1
# df.q2$imdbVotesless300k[!(df.q2$imdbVotes<300000)&(df.q2$imdbVotes>100000)]=0

suppressWarnings(
  df.rmse <- getRMSE.ten.times(df.q2, model.cmd="Gross~.+I(Budget^2)+I(Budget^3)+I(im
dbVotes^2)+I(imdbVotes^3)"))
```

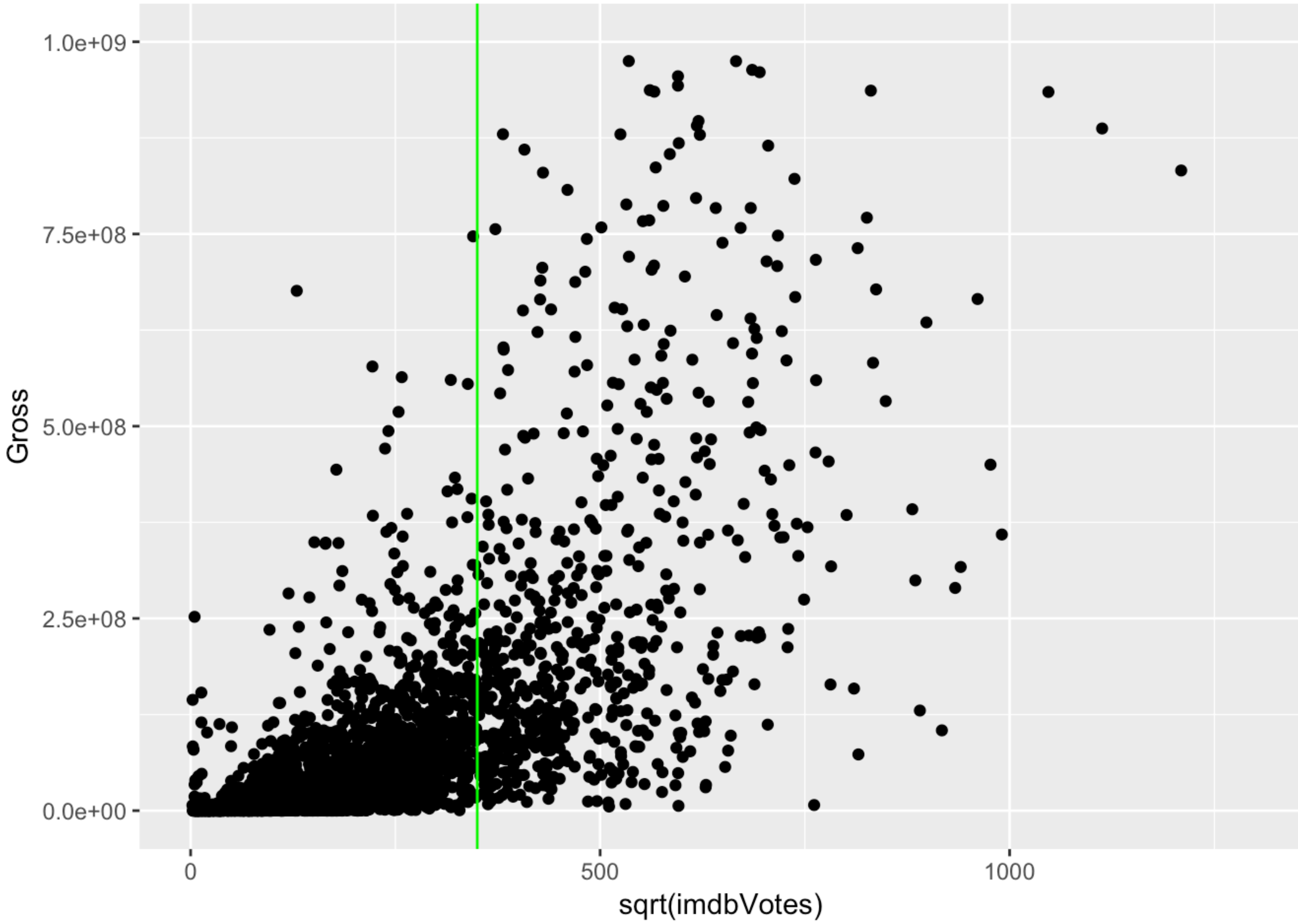
```
##
## Call:
## lm(formula = as.formula(model.cmd), data = df.train)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -259855251  -29874262   -4213339   19264123  939090399
##
## Coefficients: (1 not defined because of singularities)
##                         Estimate Std. Error t value Pr(>|t|)
## (Intercept)           -1.802e+09  8.899e+08  -2.025 0.042974 *
## Year                   4.274e+06  4.610e+06   0.927 0.353969
## Runtime               -7.723e+05  1.059e+05  -7.294 4.17e-13 ***
## imdbRating            -2.658e+07  3.048e+06  -8.720  < 2e-16 ***
## imdbVotes              8.725e+02  8.280e+01  10.538  < 2e-16 ***
## tomatoMeter            7.149e+05  2.566e+05   2.787 0.005372 **
## tomatoRating          -3.576e+06  5.095e+06  -0.702 0.482781
## tomatoReviews          3.512e+05  7.675e+04   4.576 5.00e-06 ***
## tomatoFresh           -3.741e+05  1.035e+05  -3.614 0.000308 ***
## tomatoUserMeter       -1.348e+05  2.658e+05  -0.507 0.612173
## tomatoUserRating       6.389e+07  1.052e+07   6.076 1.45e-09 ***
## tomatoUserReviews      4.022e+00  4.811e-01   8.360  < 2e-16 ***
## Budget                 1.883e+00  5.529e-01   3.405 0.000673 ***
## Date                   8.260e+05  4.022e+06   0.205 0.837288
## Released_year         -4.181e+06  4.561e+06  -0.917 0.359455
## Budgetbw50_100         7.732e+06  9.486e+06   0.815 0.415114
## imdbVotesless300k     -1.976e+06  1.330e+07  -0.149 0.881879
## imdbVotessqrt         -5.326e+06  2.280e+06  -2.336 0.019596 *
## imdbVotessqrtmore350         NA         NA      NA       NA
## I(Budget^2)           -2.160e-08  1.616e-08  -1.336 0.181584
## I(Budget^3)            2.473e-16  1.170e-16   2.115 0.034569 *
## I(imdbVotes^2)        -1.412e-03  2.464e-04  -5.732 1.13e-08 ***
## I(imdbVotes^3)         1.078e-09  1.753e-10   6.151 9.12e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 68740000 on 2230 degrees of freedom
##   (432 observations deleted due to missingness)
## Multiple R-squared:  0.5913, Adjusted R-squared:  0.5875
## F-statistic: 153.6 on 21 and 2230 DF,  p-value: < 2.2e-16
```

```
suppressWarnings(
  df.rmse.avg <- aggregate(df.rmse,list(df.rmse$random),data=df.rmse,FUN="mean") )
print(df.rmse.avg)
```

```
##    Group.1 random rmse.train rmse.test
## 1     0.05   0.05   53868757 224875982
## 2     0.10   0.10   59714055 114899497
## 3     0.15   0.15   61816767  81292882
## 4     0.20   0.20   64532754  79436387
## 5     0.25   0.25   67052270  76864046
## 6     0.30   0.30   66326077  77759718
## 7     0.35   0.35   65543959  72767892
## 8     0.40   0.40   66937659  74281694
## 9     0.45   0.45   67604634  74813801
## 10    0.50   0.50   68712670  72061015
## 11    0.55   0.55   66362911  73674030
## 12    0.60   0.60   66843832  72105133
## 13    0.65   0.65   67748809  69167880
## 14    0.70   0.70   67996046  68865589
## 15    0.75   0.75   68985514  64149867
## 16    0.80   0.80   66890150  71991099
## 17    0.85   0.85   67220094  69690809
## 18    0.90   0.90   67966921  65303871
## 19    0.95   0.95   67913740  63102681
```

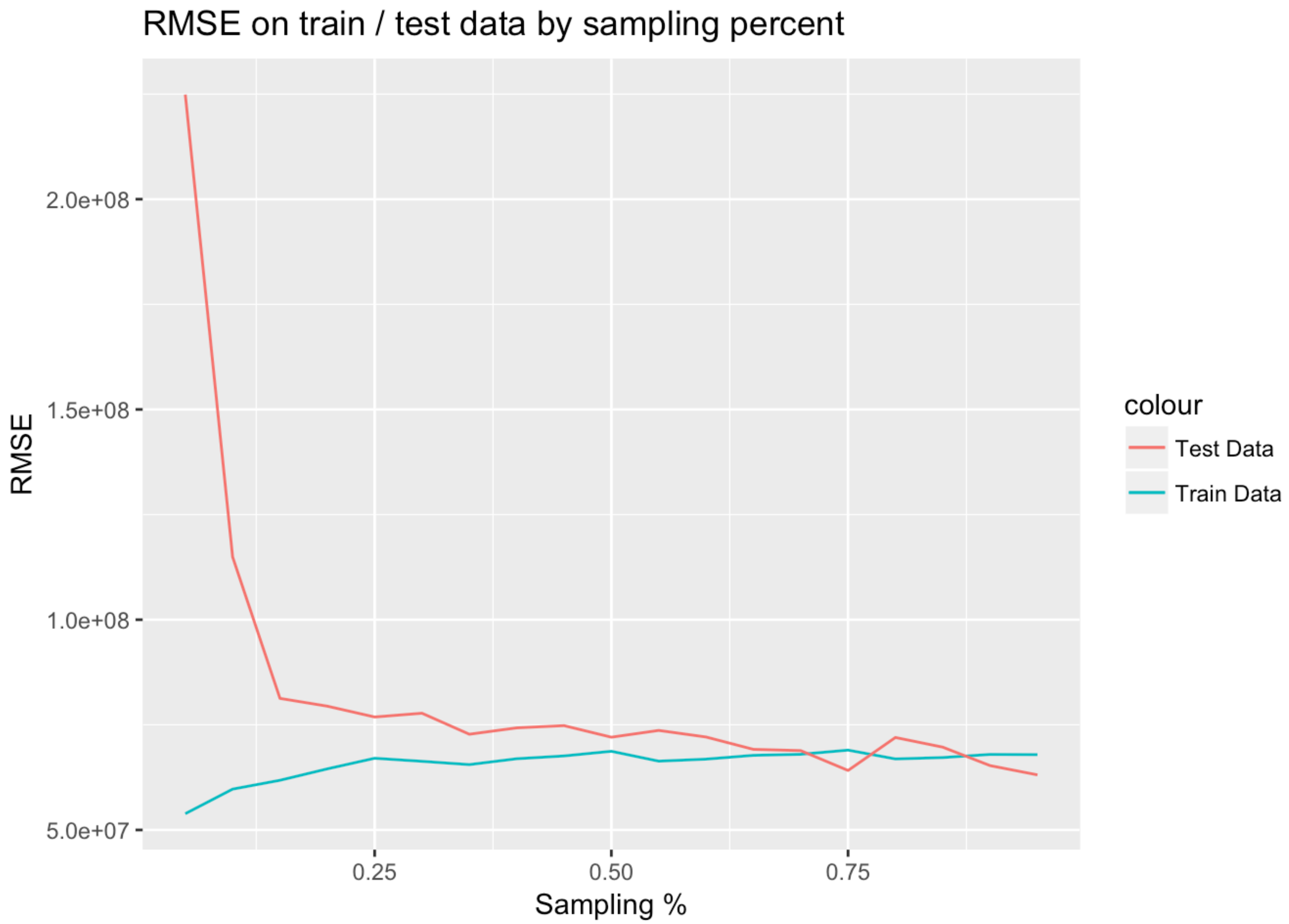
```
ggplot(df.q1, aes(sqrt(imdbVotes), Gross))+geom_point()+ylim(0,1e9)+
  geom_vline(xintercept = 350, color="green")
```

```
## Warning: Removed 34 rows containing missing values (geom_point).
```

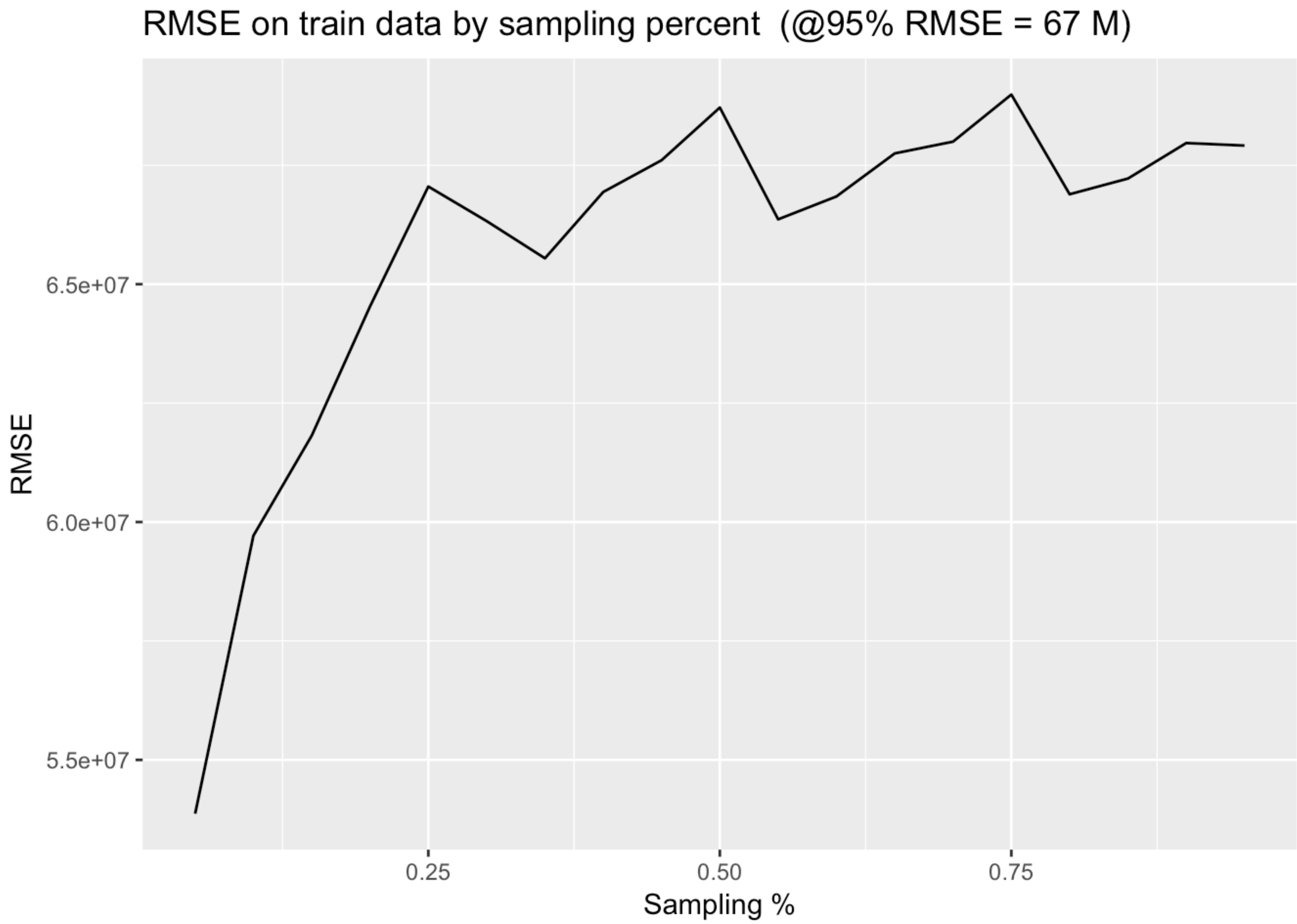



```
# geom_vline(xintercept = 300000, color="green")
```

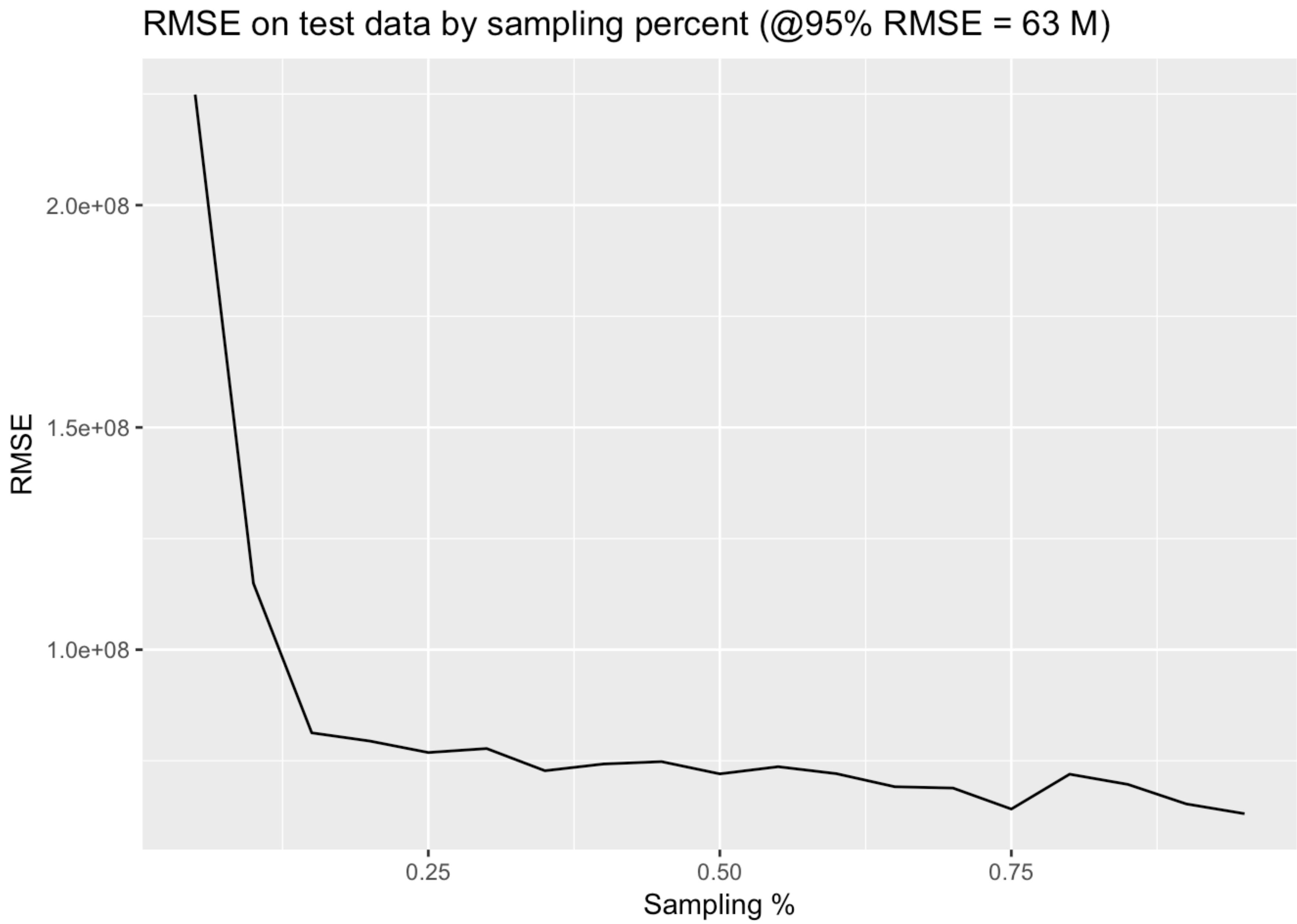
```
ggplot(df.rmse.avg, aes(x= df.rmse.avg$random))+geom_line(aes(y = df.rmse.avg$rmse.tr
ain, colour = 'Train Data')) + geom_line(aes(y = df.rmse.avg$rmse.test, colour = 'Tes
t Data')) +labs(title="RMSE on train / test data by sampling percent ", x="Sampling %
",y= "RMSE")
```




```
qplot(x= df.rmse.avg$random, y=df.rmse.avg$rmse.train, geom = "line")+ labs(title=pas
te("RMSE on train data by sampling percent ", "(@95% RMSE =", as.integer(df.rmse.avg$
rmse.train[19]/ 1e6) , "M)", sep=" "), x="Sampling %",y= "RMSE")
```




```
qplot(x= df.rmse.avg$random, y=df.rmse.avg$rmse.test, geom = "line")+labs(title=paste
("RMSE on test data by sampling percent", "(@95% RMSE =", as.integer(df.rmse.avg$rmse
.test[19]/ 1e6) , "M)", sep=" "), x="Sampling %",y= "RMSE")
```



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

A: I focussed on two variables that had higher t-values when I ran my regression model: budget and imdbVotes. Initially I conducted some exploratory analysis where I tried log, x^2, and sqrt(x), and these did not result in significant improvement in decreased RMSE.

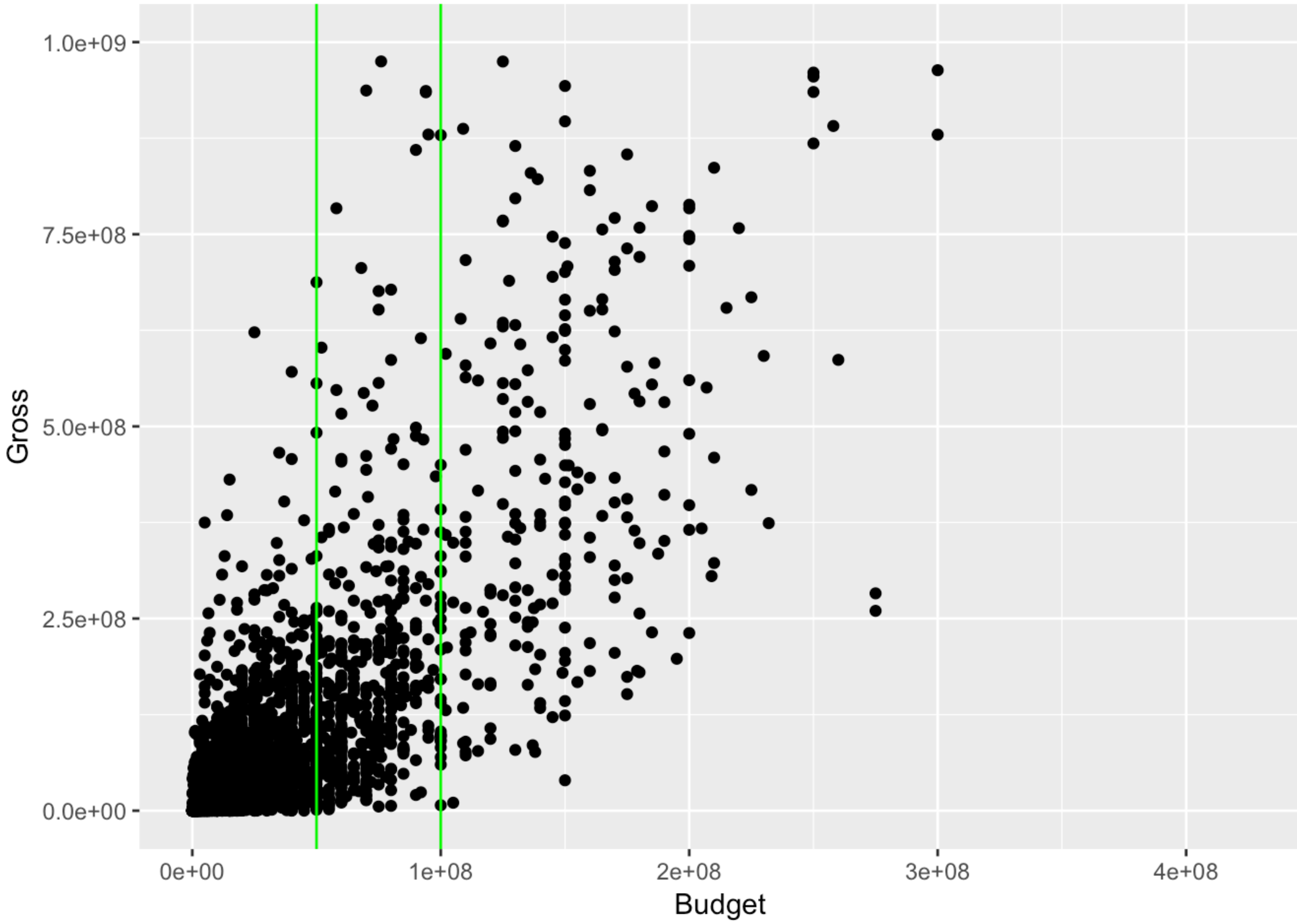
However when using binning on budget I iterated and found that there improvements with RMSE. When viewing the plot of Budget vs Gross it seems there is differentiation between normal budgeted movies and the ones that had high gross. After expermienting with budget > 1e8 and other permutations, I settled on the budget range of 50 to 100m. I hypothesized that budget within this range seemed to have greater gross. When plotting a histogram of the gross of movies within the range and those outside of the range, the plots show a significant difference.

I was unable to extract improvements from imdbVotes despite trying multiple transformations to include sqrt. I hypothesized that this transformation made the data more linear; however, it didn't result in lower RMSE.

Ultimately, with a baseline (Q1) of ~ $95M RMSE on 95% sampling of test data, I was able to improve the model by 31M.

```
ggplot(df.q1, aes(Budget, Gross))+geom_point()+ylim(0,1e9) +
  geom_vline(xintercept = 5e7, colour =("green"))+
  geom_vline(xintercept = 1e8, colour =("green"))
```

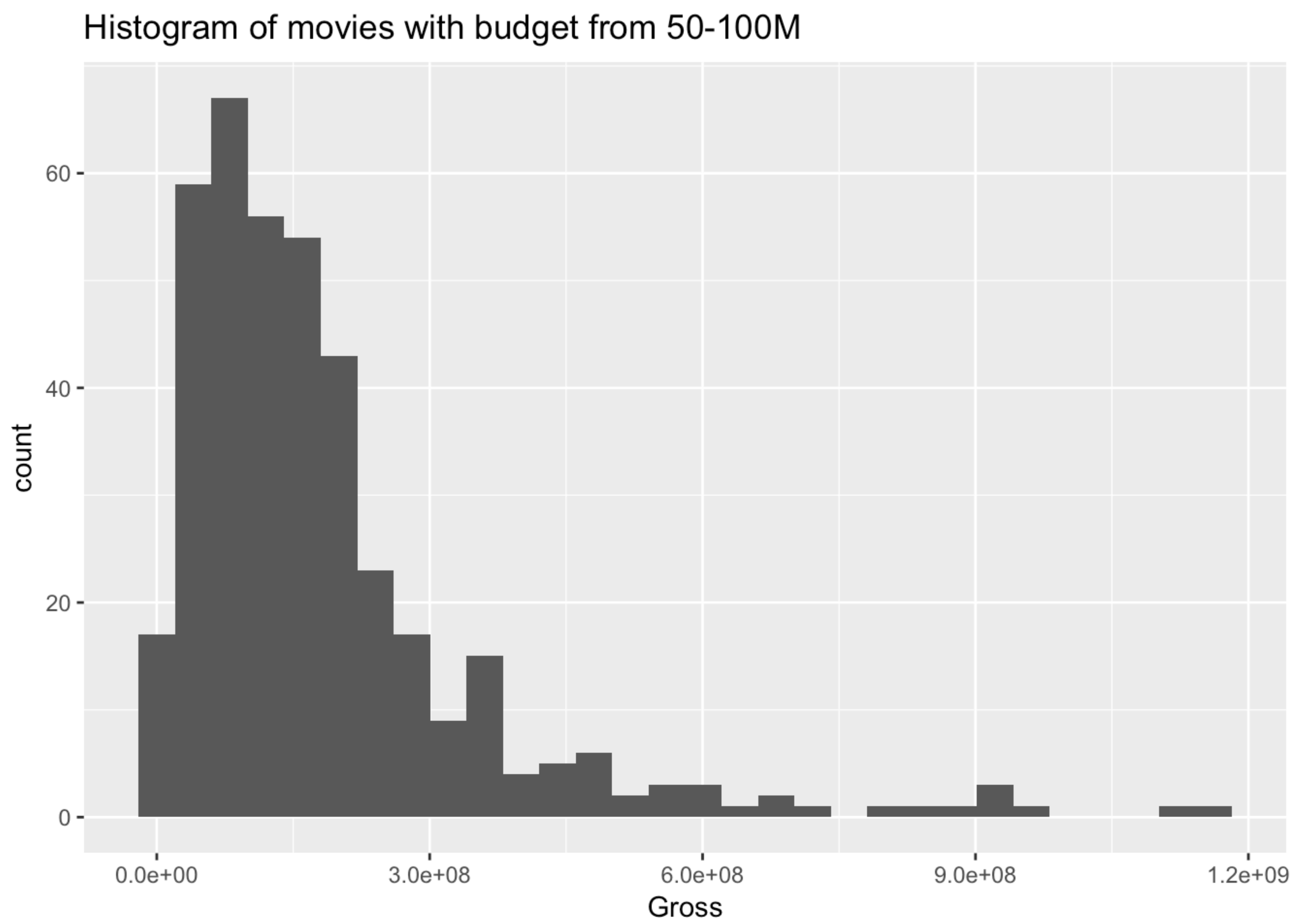
```
## Warning: Removed 20 rows containing missing values (geom_point).
```




```
qplot(df.q2$Gross[df.q2$Budgetbw50_100==1])+ggtitle("Histogram of movies with budget
from 50-100M")+xlab("Gross")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

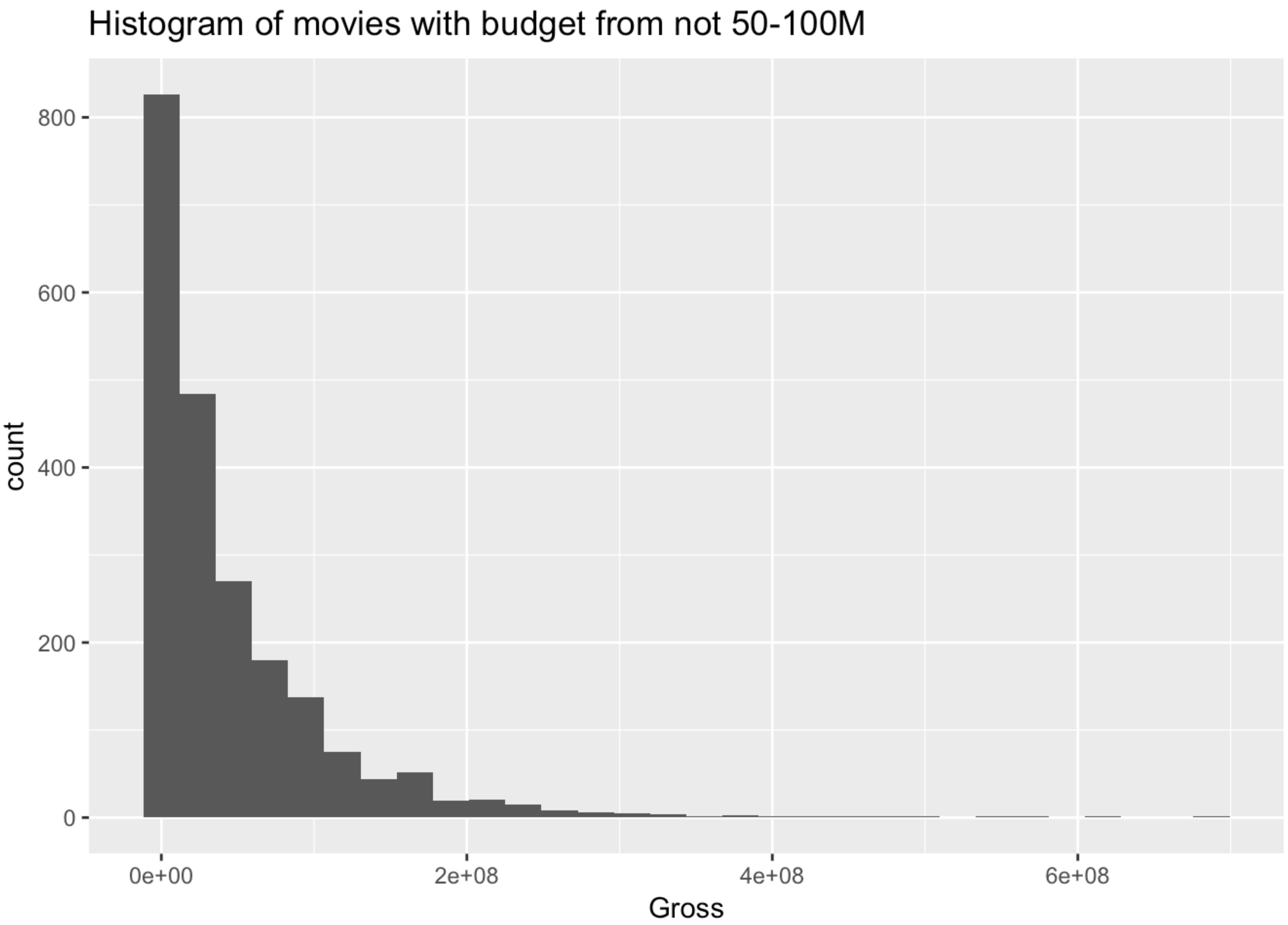
```
## Warning: Removed 271 rows containing non-finite values (stat_bin).
```




```
qplot(df.q2$Gross[df.q2$Budgetbw50_100==0])+ggtitle("Histogram of movies with budget
from not 50-100M")+xlab("Gross")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## Warning: Removed 271 rows containing non-finite values (stat_bin).
```

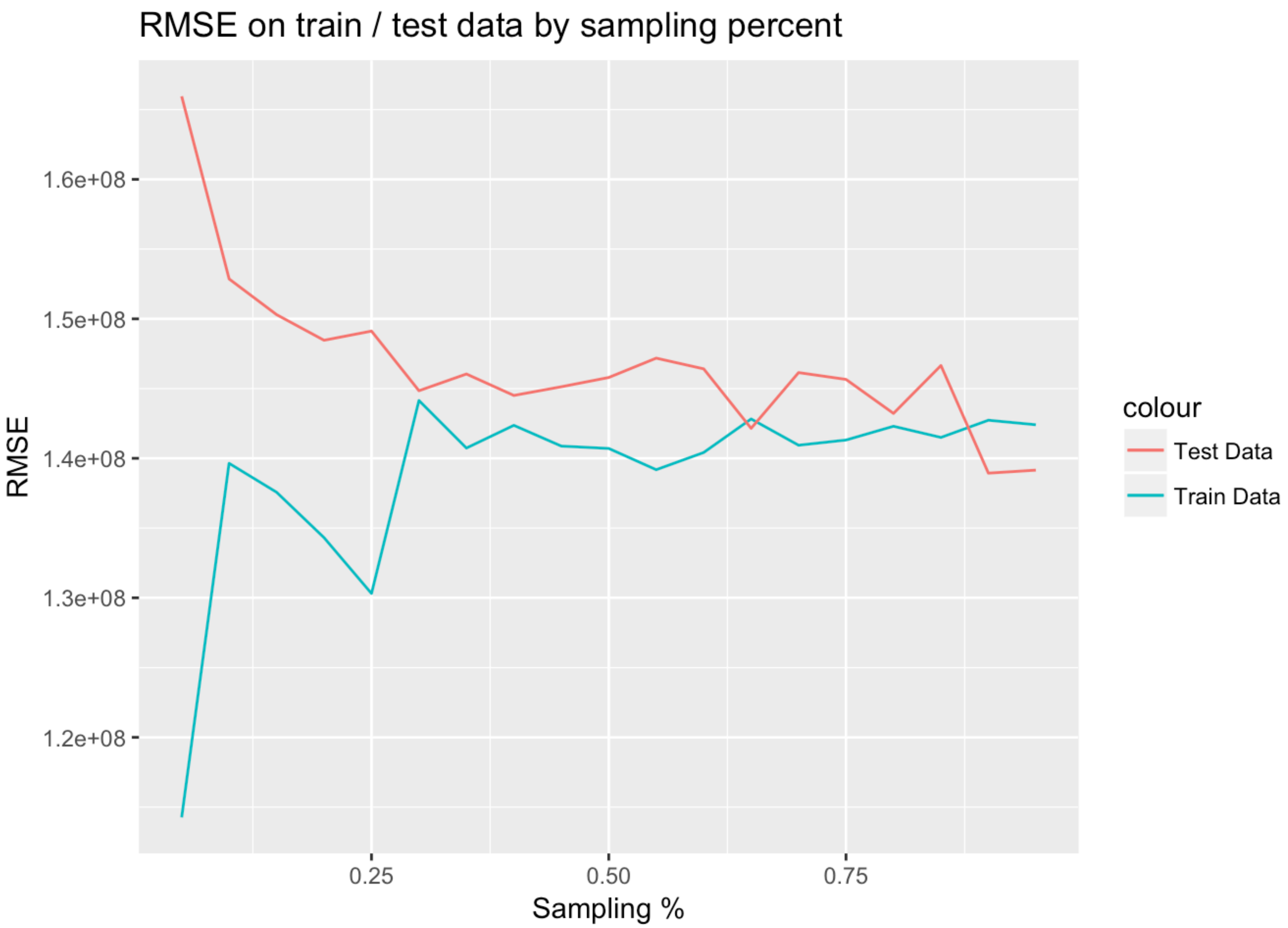


3. Non-numeric variables

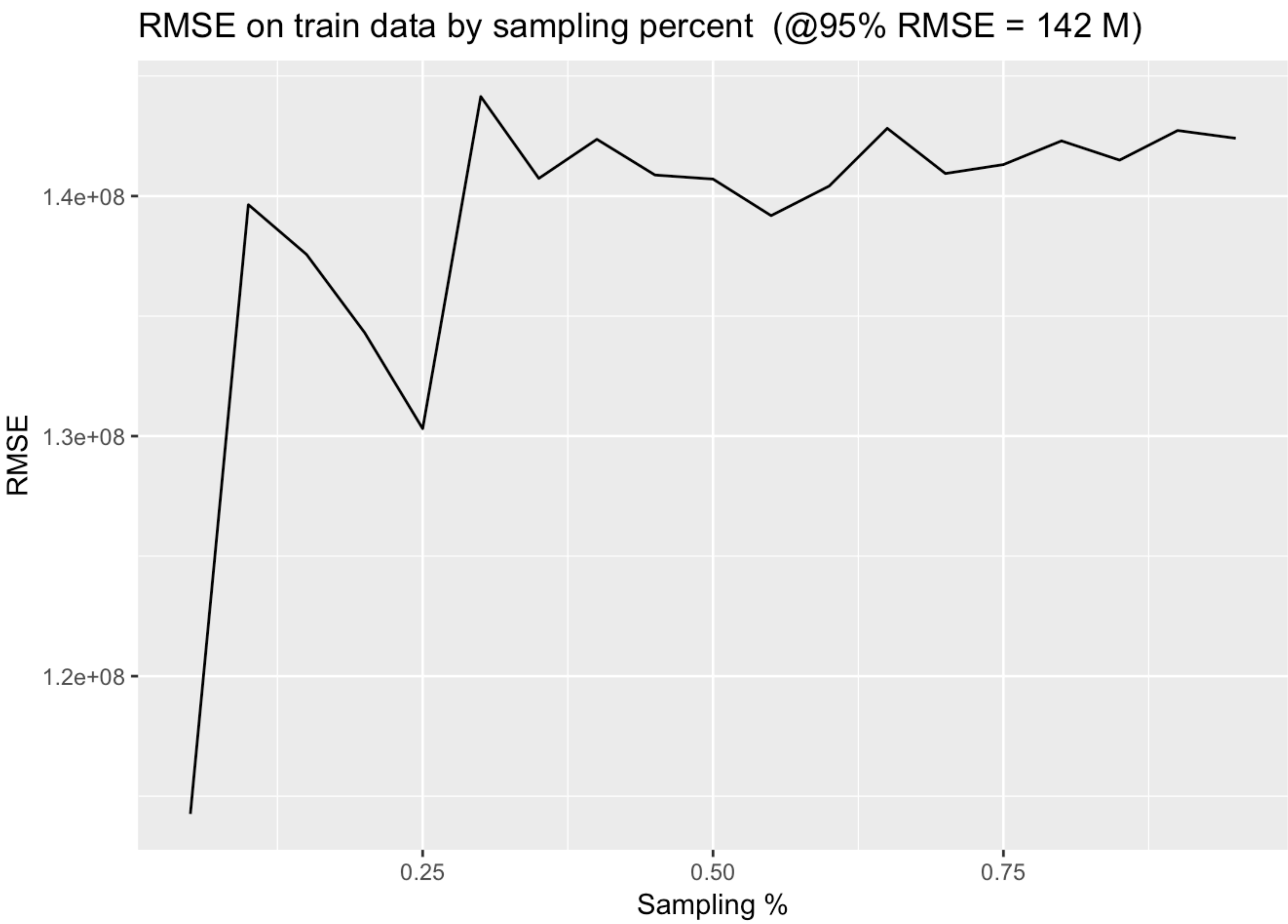
Write code that converts genre, actors, directors, and other categorical variables to columns that can be used for regression (e.g. binary columns as you did in Project 1). Also process variables such as awards into more useful columns (again, like you did in Project 1). Now use these converted columns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

```
##
## Call:
## lm(formula = as.formula(model.cmd), data = df.train)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -481709154  -66608033  -12799525   43110874 2191637609
##
## Coefficients: (1 not defined because of singularities)
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)      -592512   20416559  -0.029 0.976850
## horror         -31614224   11335729  -2.789 0.005326 **
## sci             55383788   11594056   4.777 1.88e-06 ***
## adventure      103002501    9146228  11.262  < 2e-16 ***
## comedy         -22044608    7950519  -2.773 0.005598 **
## family          11343500   11435922   0.992 0.321328
## crime          -18460245    8376599  -2.204 0.027625 *
## music             223568   14792894   0.015 0.987943
## drama          -47788412    7663410  -6.236 5.21e-10 ***
## mystery          1926262   10742028   0.179 0.857700
## thriller        11617591    9140103   1.271 0.203820
## romance         14693663    8341552   1.762 0.078269 .
## sport           -9198461   17055735  -0.539 0.589713
## fantasy         65277702   11345150   5.754 9.73e-09 ***
## action          58231880    8282962   7.030 2.61e-12 ***
## biography      -19459298   12192561  -1.596 0.110608
## documentary    -57033677   17376738  -3.282 0.001043 **
## history         -1088523   17180674  -0.063 0.949487
## animation       74716216   13745476   5.436 5.96e-08 ***
## musical        -10675948   32518570  -0.328 0.742708
## western        -94549664   35101849  -2.694 0.007113 **
## war            -33521398   21750752  -1.541 0.123397
## short           32655091   42817844   0.763 0.445739
## news                  NA         NA      NA       NA
## Released_month   -868676    1026538  -0.846 0.397507
## winter          56044826   10077283   5.562 2.94e-08 ***
## summer          51224211    7085827   7.229 6.33e-13 ***
## is_english      42245832   16701343   2.529 0.011480 *
## is_fresh        28179570    8068495   3.493 0.000486 ***
## is_certified    74843798    6889254  10.864  < 2e-16 ***
## Director_top   203651458   14278578  14.263  < 2e-16 ***
## Actors_top      86329951    8220254  10.502  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 142200000 on 2653 degrees of freedom
## Multiple R-squared:  0.4182, Adjusted R-squared:  0.4117
## F-statistic: 63.57 on 30 and 2653 DF,  p-value: < 2.2e-16
```

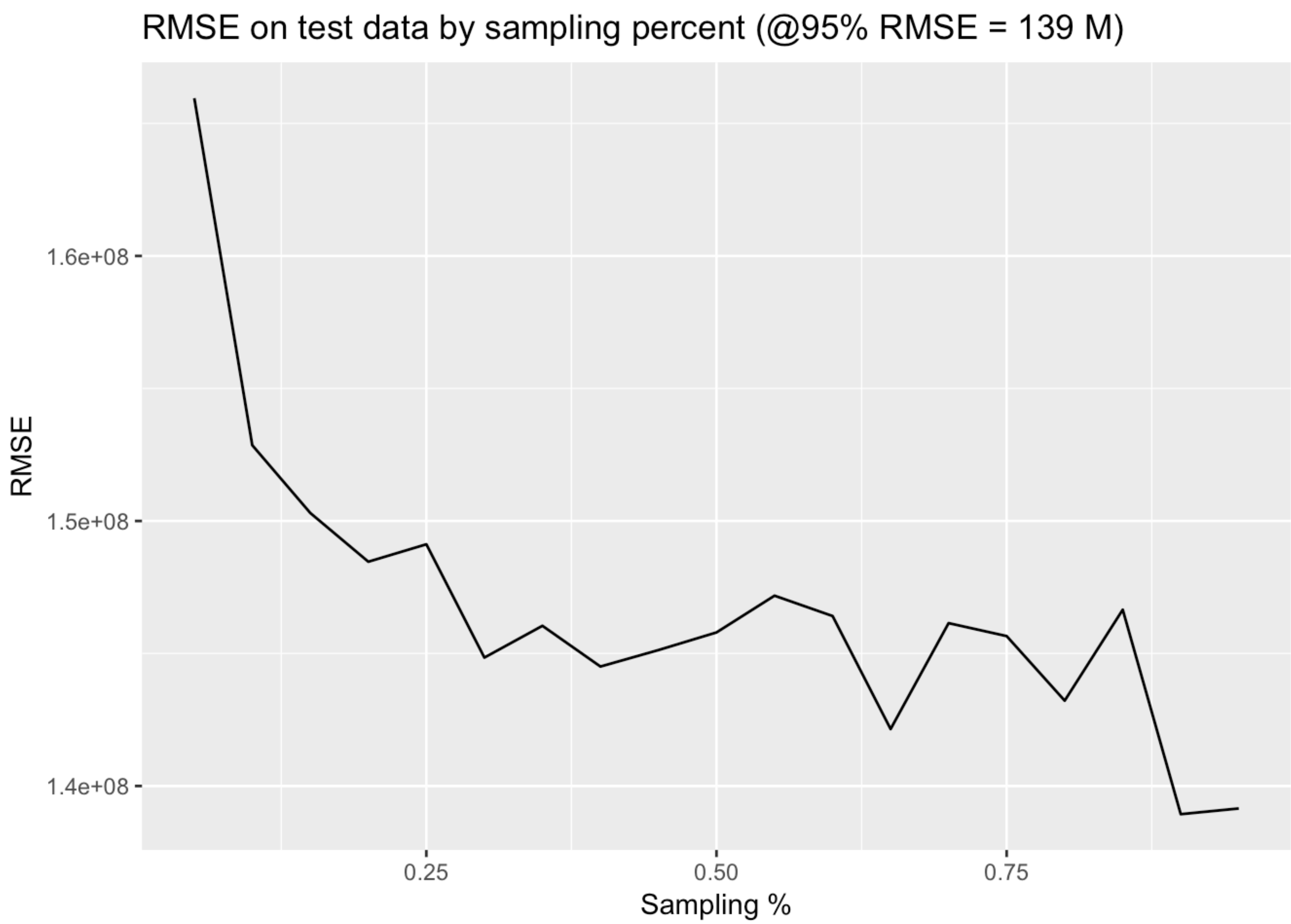
```
##    Group.1 random rmse.train rmse.test
## 1     0.05   0.05  114259923 165952100
## 2     0.10   0.10  139640409 152858990
## 3     0.15   0.15  137569004 150301535
## 4     0.20   0.20  134313165 148463025
## 5     0.25   0.25  130309996 149120716
## 6     0.30   0.30  144145728 144845748
## 7     0.35   0.35  140736349 146044921
## 8     0.40   0.40  142365501 144507958
## 9     0.45   0.45  140877439 145130235
## 10    0.50   0.50  140707423 145795028
## 11    0.55   0.55  139184893 147181725
## 12    0.60   0.60  140417113 146414215
## 13    0.65   0.65  142822553 142149792
## 14    0.70   0.70  140939426 146145967
## 15    0.75   0.75  141311610 145656954
## 16    0.80   0.80  142298925 143220778
## 17    0.85   0.85  141497188 146652284
## 18    0.90   0.90  142731595 138937874
## 19    0.95   0.95  142407382 139153601
```



```
qplot(x= df.rmse.avg$random, y=df.rmse.avg$rmse.train, geom = "line")+ labs(title=paste("RMSE on train data by sampling percent ", "(@95% RMSE =", as.integer(df.rmse.avg$rmse.train[19]/ 1e6) , "M)", sep=" "), x="Sampling %",y= "RMSE")
```




```
qplot(x= df.rmse.avg$random, y=df.rmse.avg$rmse.test, geom = "line")+labs(title=paste("RMSE on test data by sampling percent", "(@95% RMSE =", as.integer(df.rmse.avg$rmse.test[19]/ 1e6) , "M)", sep=" "), x="Sampling %",y= "RMSE")
```



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

A: My approach for including categorical variables intended to use hypothesises from project 1.

- Certain genres correlate with higher gross
- Summer and Winter months correlate with higher gross
- Rottentomatoes fresh and certified are more highly critically acclaimed so are higher gross
- Certain top actors and directors make higher gross (source IMDB)

For genre I used text recognition on the genre column, e.g., if drama, df$drama=1

For summer and winter months, I created comparisons for the summer and winter months against released month. winter=1

For rotten tomatoes image provides another source of info that is proxy for critical acclaim. I used grepl to detect whether the text is fresh and certfied and made them into 0 and 1.

Lastly, for those movies that have top directors or top actors, did a grepl search for IMDB top grossing actors and directors. If present Director_top and Actor_top is 1.

Ultimately the performance of the non-numeric model performed poorly compared to the numeric. Q1 ~ $95M vs non-numeric 144M

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)

drops <- c("Gross")
df.q4 <- cbind.data.frame(df.q2, df.q3[, !(names(df.q3) %in% drops)])

suppressWarnings(df.rmse <- getRMSE.ten.times(df.q4, model.cmd="Gross~."))
```

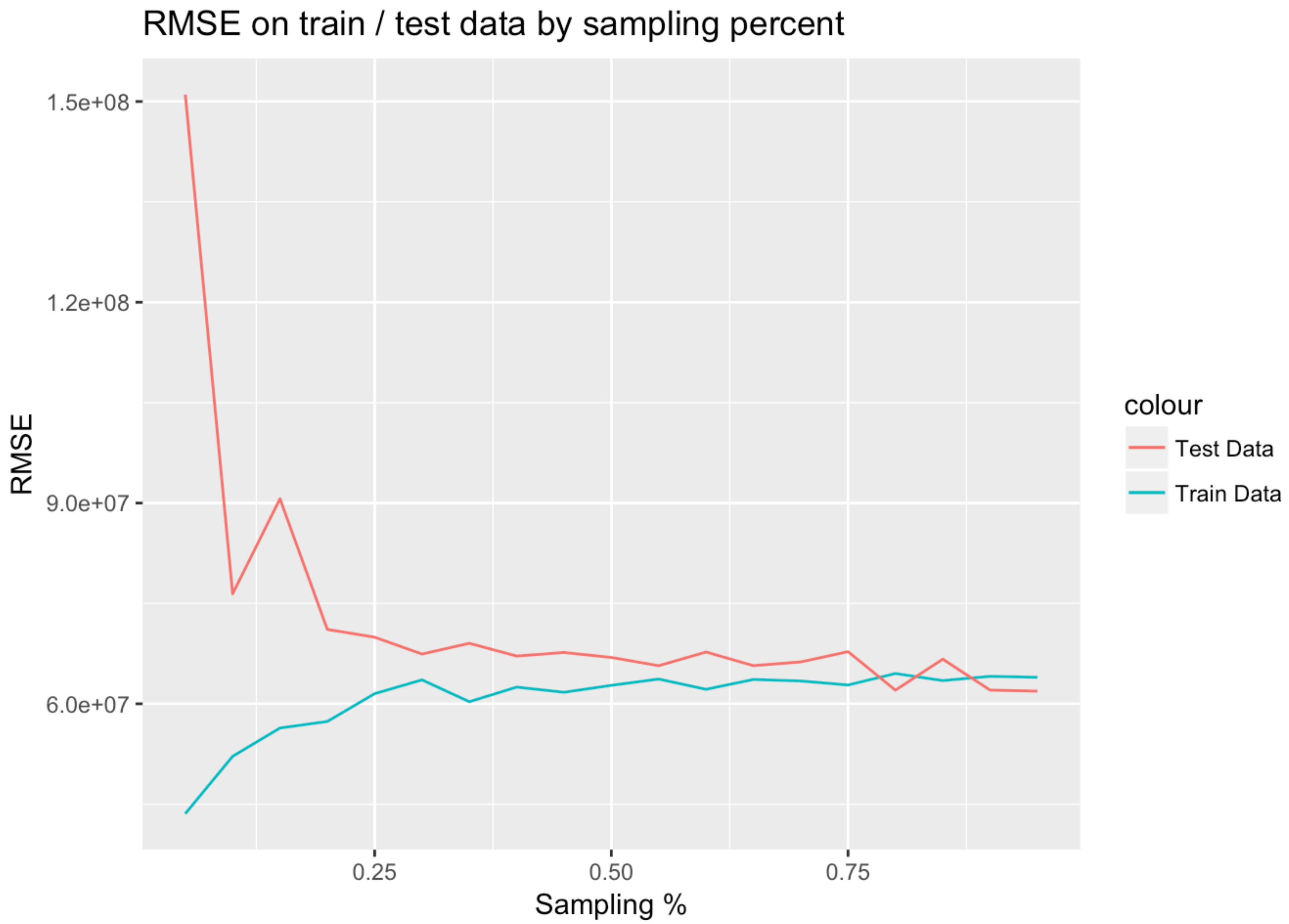
```
##
## Call:
## lm(formula = as.formula(model.cmd), data = df.train)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -327453061  -26826977   -1713515   21825860  835442165
##
## Coefficients: (1 not defined because of singularities)
##                         Estimate Std. Error t value Pr(>|t|)
## (Intercept)           -1.528e+09  8.475e+08  -1.802 0.071606 .
## Year                  -3.040e+06  4.638e+06  -0.656 0.512169
## Runtime               -1.976e+05  1.160e+05  -1.704 0.088584 .
## imdbRating            -2.407e+07  3.015e+06  -7.985 2.25e-15 ***
## imdbVotes              5.899e+02  3.119e+01  18.915  < 2e-16 ***
## tomatoMeter            1.082e+06  2.754e+05   3.928 8.83e-05 ***
## tomatoRating          -1.055e+07  5.001e+06  -2.110 0.034941 *
## tomatoReviews          3.817e+05  7.801e+04   4.893 1.07e-06 ***
## tomatoFresh           -3.617e+05  1.095e+05  -3.304 0.000969 ***
## tomatoUserMeter       -1.961e+05  2.583e+05  -0.759 0.447703
## tomatoUserRating       5.719e+07  9.952e+06   5.747 1.04e-08 ***
## tomatoUserReviews      3.705e+00  4.702e-01   7.879 5.11e-15 ***
## Budget                 1.290e+00  1.354e-01   9.524  < 2e-16 ***
## Date                   2.788e+06  3.885e+06   0.718 0.472960
## Released_year          9.860e+05  4.779e+06   0.206 0.836560
## Budgetbw50_100         1.054e+07  6.785e+06   1.553 0.120476
## imdbVotesless300k      4.220e+07  1.112e+07   3.795 0.000152 ***
## imdbVotessqrt         -1.063e+05  1.951e+06  -0.054 0.956560
## imdbVotessqrtmore350          NA         NA      NA       NA
## horror                 1.569e+07  5.665e+06   2.771 0.005643 **
## sci                   -2.838e+07  6.431e+06  -4.413 1.07e-05 ***
## adventure             -3.599e+06  4.997e+06  -0.720 0.471520
## comedy                 5.501e+06  4.076e+06   1.350 0.177298
## family                 2.479e+07  6.102e+06   4.063 5.02e-05 ***
## crime                 -1.260e+07  4.127e+06  -3.053 0.002295 **
## music                 -4.039e+06  7.276e+06  -0.555 0.578844
## drama                  2.146e+06  4.025e+06   0.533 0.593961
## mystery               -3.465e+06  5.300e+06  -0.654 0.513277
## thriller              -2.711e+06  4.588e+06  -0.591 0.554605
```

```
## romance              1.891e+05  4.041e+06   0.047 0.962669    
## sport               -1.486e+07  7.986e+06  -1.860 0.062959 .  
## fantasy             -1.526e+07  6.061e+06  -2.517 0.011902 *  
## action              -4.089e+06  4.514e+06  -0.906 0.365097    
## biography            3.694e+06  6.139e+06   0.602 0.547391    
## documentary          1.658e+07  9.627e+06   1.722 0.085186 .  
## history             -4.234e+06  8.705e+06  -0.486 0.626774    
## animation            1.076e+08  8.357e+06  12.879  < 2e-16 ***
## musical              2.312e+07  1.572e+07   1.470 0.141615    
## western              2.268e+05  1.668e+07   0.014 0.989152    
## war                 -2.001e+07  1.041e+07  -1.921 0.054848 .  
## short                2.399e+07  5.049e+07   0.475 0.634771    
## news                 7.732e+06  6.595e+07   0.117 0.906680    
## Released_month      -5.713e+05  5.350e+05  -1.068 0.285758    
## winter               8.942e+06  5.052e+06   1.770 0.076878 .  
## summer               1.931e+07  3.687e+06   5.239 1.77e-07 ***
## is_english          -1.952e+07  1.080e+07  -1.808 0.070803 .  
## is_fresh            -1.143e+07  5.803e+06  -1.969 0.049082 *  
## is_certified        -2.057e+07  7.234e+06  -2.843 0.004511 ** 
## Director_top         1.171e+07  9.139e+06   1.281 0.200188    
## Actors_top          -5.546e+06  4.553e+06  -1.218 0.223241    
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 65080000 on 2212 degrees of freedom
##   (423 observations deleted due to missingness)
## Multiple R-squared:  0.6423, Adjusted R-squared:  0.6345 
## F-statistic: 82.73 on 48 and 2212 DF,  p-value: < 2.2e-16
```

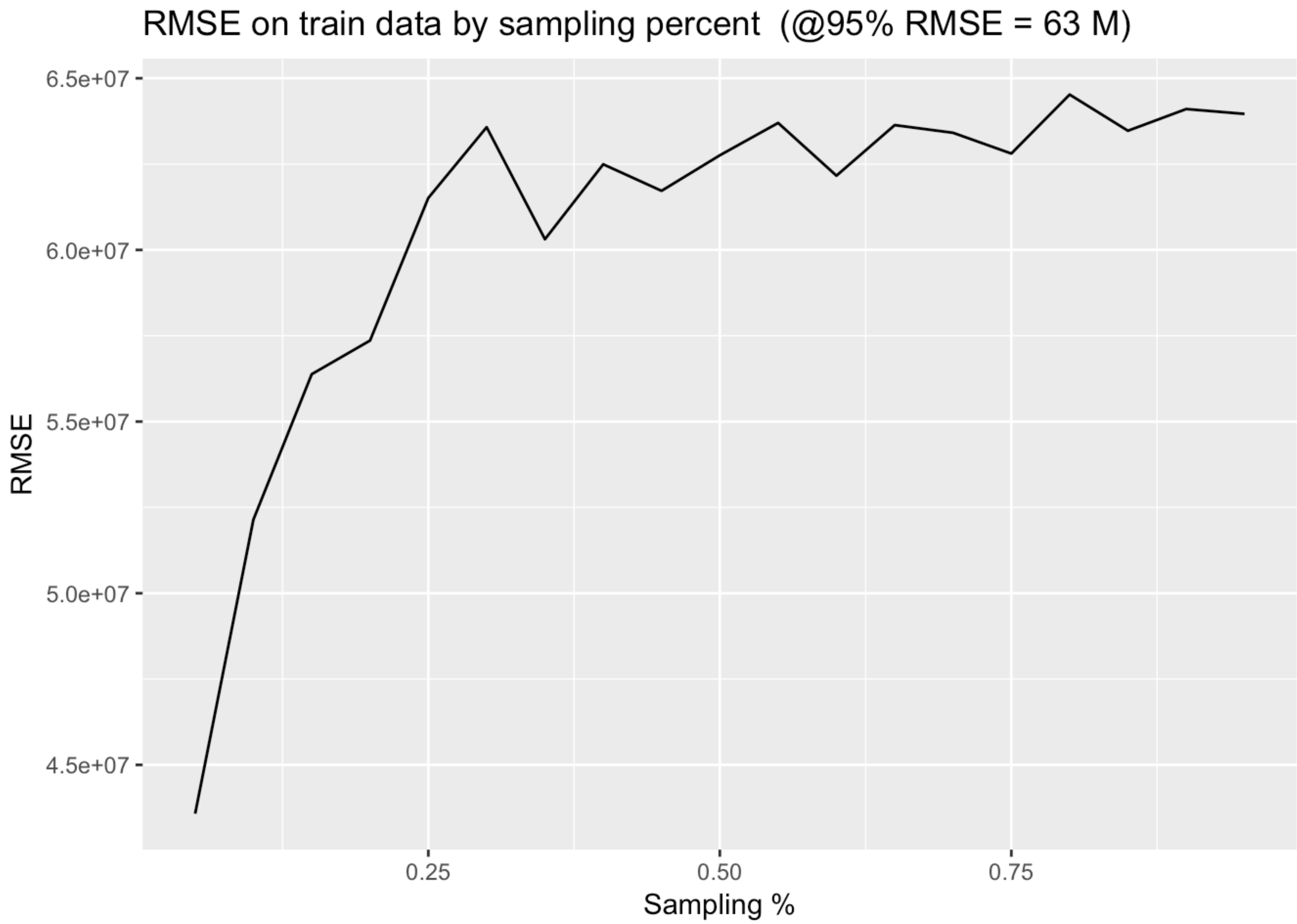
```
suppressWarnings(df.rmse.avg <- aggregate(df.rmse,list(df.rmse$random),data=df.rmse,F
UN="mean") )

print(df.rmse.avg)
```

```
##    Group.1 random rmse.train rmse.test
## 1     0.05   0.05   43578468 151035867
## 2     0.10   0.10   52147163  76441437
## 3     0.15   0.15   56380667  90628165
## 4     0.20   0.20   57360465  71117904
## 5     0.25   0.25   61514642  69950687
## 6     0.30   0.30   63574138  67432341
## 7     0.35   0.35   60311249  69039633
## 8     0.40   0.40   62493894  67133250
## 9     0.45   0.45   61721189  67668608
## 10    0.50   0.50   62756569  66924975
## 11    0.55   0.55   63699063  65691899
## 12    0.60   0.60   62162618  67735420
## 13    0.65   0.65   63635739  65700452
## 14    0.70   0.70   63411881  66258162
## 15    0.75   0.75   62807074  67786648
## 16    0.80   0.80   64525009  62036356
## 17    0.85   0.85   63471279  66670923
## 18    0.90   0.90   64104175  62045058
## 19    0.95   0.95   63962078  61891232
```

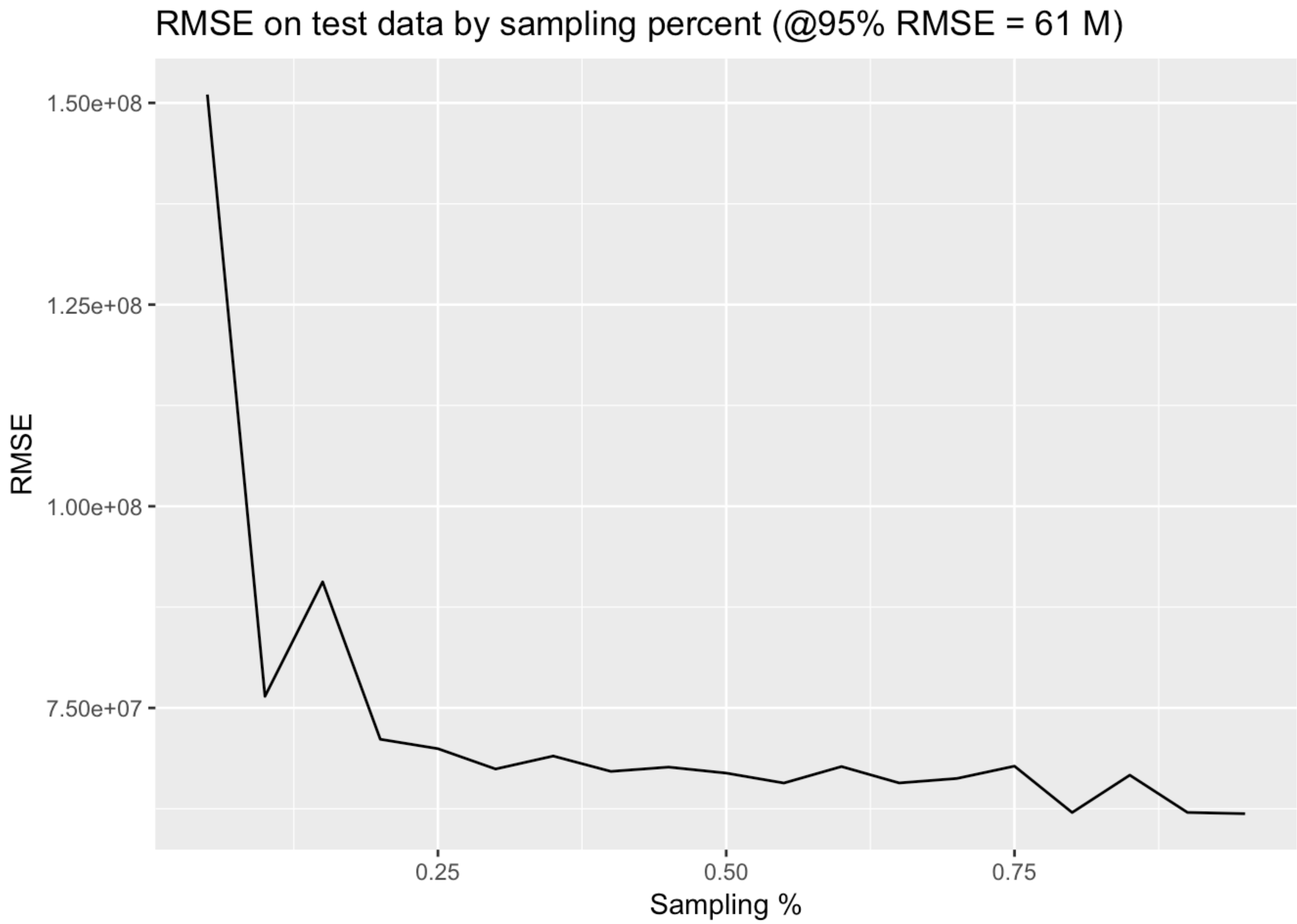
```
ggplot(df.rmse.avg, aes(x= df.rmse.avg$random))+geom_line(aes(y = df.rmse.avg$rmse.tr
ain, colour = 'Train Data')) + geom_line(aes(y = df.rmse.avg$rmse.test, colour = 'Tes
t Data')) +labs(title="RMSE on train / test data by sampling percent ", x="Sampling %
",y= "RMSE")
```




```
qplot(x= df.rmse.avg$random, y=df.rmse.avg$rmse.train, geom = "line")+ labs(title=pas
te("RMSE on train data by sampling percent ", "(@95% RMSE =", as.integer(df.rmse.avg$
rmse.train[19]/ 1e6) , "M)", sep=" "), x="Sampling %",y= "RMSE")
```




```
qplot(x= df.rmse.avg$random, y=df.rmse.avg$rmse.test, geom = "line")+labs(title=paste
("RMSE on test data by sampling percent", "(@95% RMSE =", as.integer(df.rmse.avg$rmse
.test[19]/ 1e6) , "M)", sep=" "), x="Sampling %",y= "RMSE")
```



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)
# df.rmse <- getRMSE.ten.times(df.q4, model.cmd="Gross~.+summer*action+winter*animati
on+ summer*animation+ winter*drama+ summer*drama")
df.rmse <- getRMSE.ten.times(df.q4, model.cmd="Gross~.+summer:action+ summer:animatio
n+imdbVotes:Budget")
```

```
##
## Call:
## lm(formula = as.formula(model.cmd), data = df.train)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -398548520  -26176945   -3369119   19540370  736025759
##
## Coefficients: (1 not defined because of singularities)
```

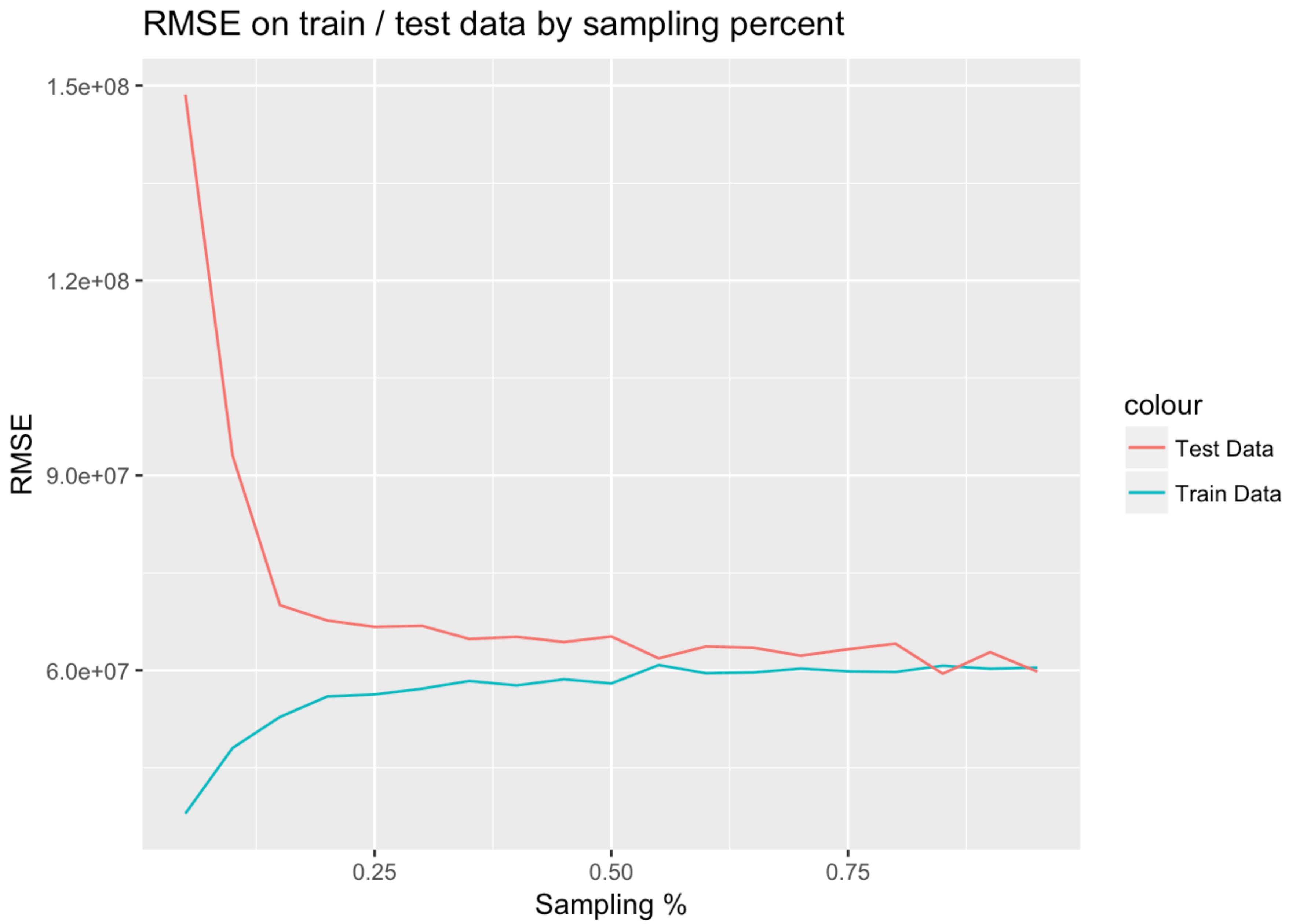
```
##                         Estimate Std. Error t value Pr(>|t|)    
## (Intercept)           -1.716e+09  8.017e+08  -2.140 0.032439 *  
## Year                   1.229e+05  4.328e+06   0.028 0.977354    
## Runtime               -2.283e+05  1.109e+05  -2.058 0.039665 *  
## imdbRating            -2.556e+07  2.848e+06  -8.973  < 2e-16 ***
## imdbVotes              1.967e+02  4.119e+01   4.775 1.91e-06 ***
## tomatoMeter            9.029e+05  2.598e+05   3.475 0.000521 ***
## tomatoRating          -8.229e+06  4.715e+06  -1.745 0.081097 .  
## tomatoReviews          4.043e+05  7.347e+04   5.502 4.18e-08 ***
## tomatoFresh           -3.191e+05  1.031e+05  -3.094 0.001997 ** 
## tomatoUserMeter       -1.975e+05  2.456e+05  -0.804 0.421394    
## tomatoUserRating       6.160e+07  9.404e+06   6.550 7.13e-11 ***
## tomatoUserReviews      3.188e+00  4.364e-01   7.305 3.84e-13 ***
## Budget                 7.269e-01  1.350e-01   5.385 8.02e-08 ***
## Date                   2.610e+06  3.693e+06   0.707 0.479775    
## Released_year         -1.929e+06  4.508e+06  -0.428 0.668762    
## Budgetbw50_100         4.008e+06  6.445e+06   0.622 0.534020    
## imdbVotesless300k      3.053e+07  1.044e+07   2.923 0.003499 ** 
## imdbVotessqrt          7.827e+06  1.931e+06   4.054 5.21e-05 ***
## imdbVotessqrtmore350          NA         NA      NA       NA    
## horror                 1.410e+07  5.305e+06   2.658 0.007907 ** 
## sci                   -2.577e+07  6.134e+06  -4.202 2.76e-05 ***
## adventure             -4.097e+06  4.704e+06  -0.871 0.383960    
## comedy                 2.791e+06  3.864e+06   0.722 0.470205    
## family                 2.698e+07  5.791e+06   4.659 3.36e-06 ***
## crime                 -1.123e+07  3.876e+06  -2.897 0.003800 ** 
## music                 -7.226e+06  6.914e+06  -1.045 0.296121    
## drama                 -8.752e+05  3.798e+06  -0.230 0.817780    
## mystery               -3.420e+06  4.981e+06  -0.687 0.492362    
## thriller              -6.133e+06  4.299e+06  -1.426 0.153889    
## romance                9.360e+05  3.845e+06   0.243 0.807692    
## sport                 -1.132e+07  7.525e+06  -1.505 0.132563    
## fantasy               -1.206e+07  5.665e+06  -2.128 0.033427 *  
## action                -9.848e+06  4.521e+06  -2.178 0.029498 *  
## biography              9.163e+05  5.688e+06   0.161 0.872040    
## documentary            1.272e+07  9.205e+06   1.382 0.167017    
## history               -3.471e+06  8.381e+06  -0.414 0.678846    
## animation              6.700e+07  9.095e+06   7.367 2.45e-13 ***
## musical                2.527e+07  1.490e+07   1.696 0.090059 .  
## western                4.346e+06  1.686e+07   0.258 0.796623    
## war                   -2.098e+07  9.562e+06  -2.194 0.028312 *  
## short                  5.832e+07  4.783e+07   1.219 0.222811    
## news                  -6.180e+06  6.239e+07  -0.099 0.921108    
## Released_month        -6.421e+05  5.052e+05  -1.271 0.203807    
## winter                 1.311e+07  4.790e+06   2.737 0.006255 ** 
## summer                 4.496e+06  3.873e+06   1.161 0.245849    
## is_english            -2.038e+07  1.033e+07  -1.973 0.048604 *  
## is_fresh              -1.258e+07  5.474e+06  -2.298 0.021640 *  
## is_certified          -1.921e+07  6.834e+06  -2.810 0.004994 ** 
## Director_top          -4.748e+06  8.491e+06  -0.559 0.576116    
```

```
## Actors_top           -9.371e+06  4.289e+06  -2.185 0.028998 *
## action:summer         2.091e+07  8.716e+06   2.399 0.016516 *
## animation:summer      1.319e+08  1.428e+07   9.236  < 2e-16 ***
## imdbVotes:Budget      6.204e-06  4.796e-07  12.935  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 61550000 on 2217 degrees of freedom
##   (415 observations deleted due to missingness)
## Multiple R-squared:   0.68,  Adjusted R-squared:  0.6726
## F-statistic: 92.37 on 51 and 2217 DF,  p-value: < 2.2e-16
```

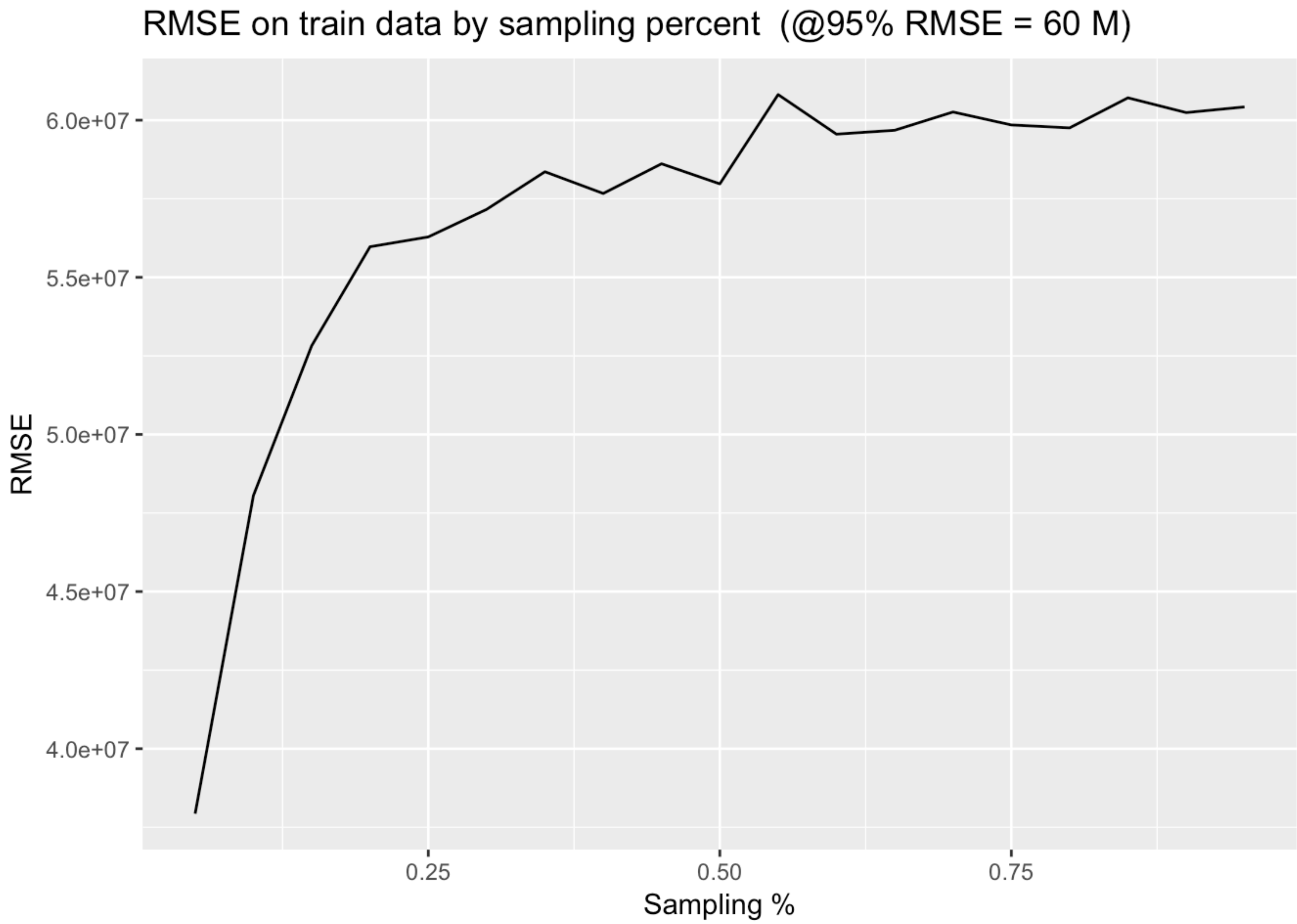
```
df.rmse.avg <- aggregate(df.rmse,list(df.rmse$random),data=df.rmse,FUN="mean")
print(df.rmse.avg)
```

```
##    Group.1 random rmse.train rmse.test
## 1     0.05   0.05   37931543 148615660
## 2     0.10   0.10   48057017  93036118
## 3     0.15   0.15   52825533  70020754
## 4     0.20   0.20   55972524  67667654
## 5     0.25   0.25   56286127  66686132
## 6     0.30   0.30   57162840  66843268
## 7     0.35   0.35   58359449  64826752
## 8     0.40   0.40   57668433  65157872
## 9     0.45   0.45   58613108  64352124
## 10    0.50   0.50   57973809  65203955
## 11    0.55   0.55   60815119  61852085
## 12    0.60   0.60   59555275  63666569
## 13    0.65   0.65   59679349  63471574
## 14    0.70   0.70   60260846  62258464
## 15    0.75   0.75   59849080  63241466
## 16    0.80   0.80   59755579  64096570
## 17    0.85   0.85   60712181  59483687
## 18    0.90   0.90   60241648  62790912
## 19    0.95   0.95   60422840  59802181
```

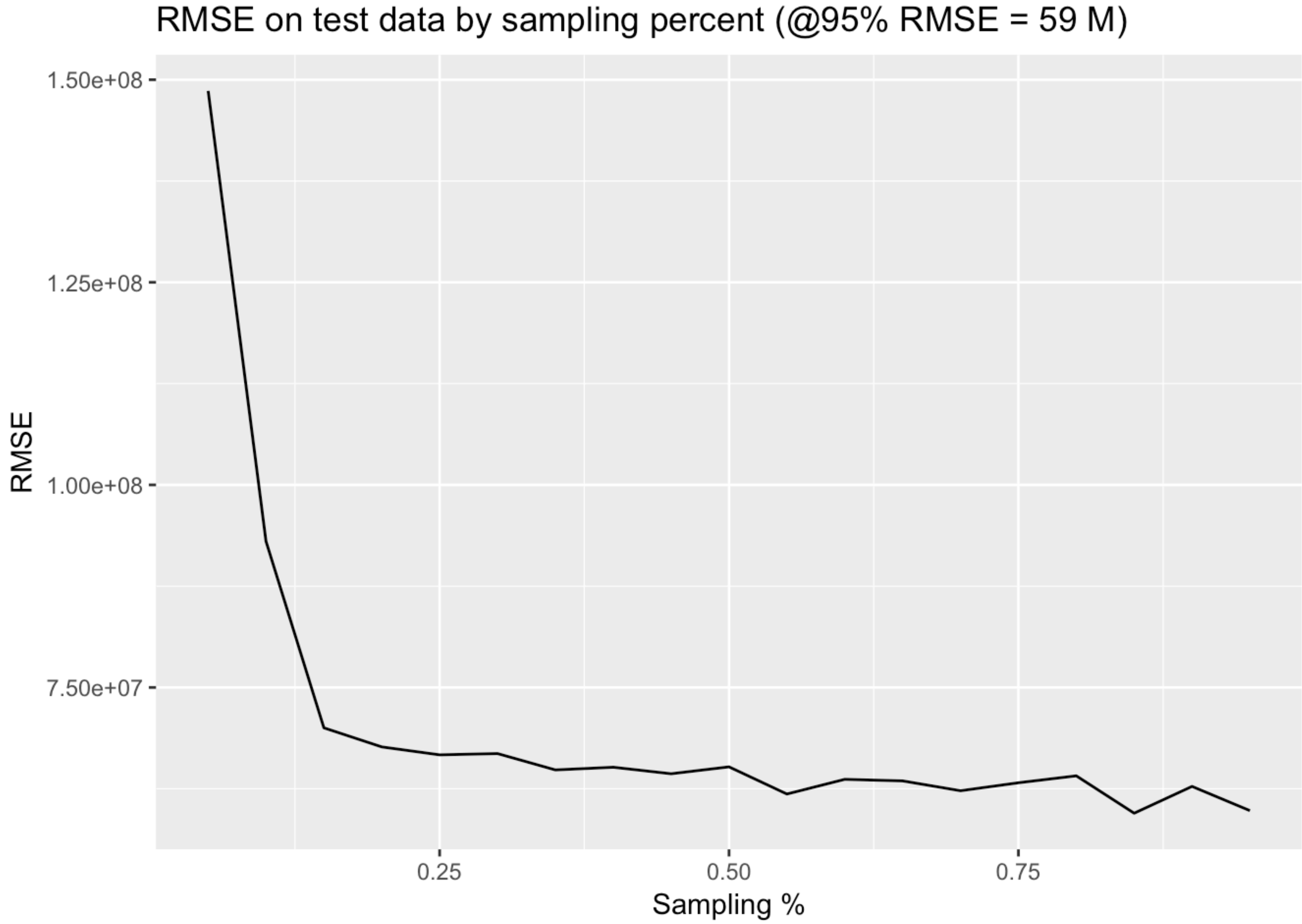
```
ggplot(df.rmse.avg, aes(x= df.rmse.avg$random))+geom_line(aes(y = df.rmse.avg$rmse.tr
ain, colour = 'Train Data')) + geom_line(aes(y = df.rmse.avg$rmse.test, colour = 'Tes
t Data')) +labs(title="RMSE on train / test data by sampling percent ", x="Sampling %
",y= "RMSE")
```




```
qplot(x= df.rmse.avg$random, y=df.rmse.avg$rmse.train, geom = "line")+ labs(title=pas
te("RMSE on train data by sampling percent ", "(@95% RMSE =", as.integer(df.rmse.avg$
rmse.train[19]/ 1e6) , "M)", sep=" "), x="Sampling %",y= "RMSE")
```




```
qplot(x= df.rmse.avg$random, y=df.rmse.avg$rmse.test, geom = "line")+labs(title=paste
("RMSE on test data by sampling percent", "(@95% RMSE =", as.integer(df.rmse.avg$rmse
.test[19]/ 1e6) , "M)", sep=" "), x="Sampling %",y= "RMSE")
```



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

A: I built on the the q4 model by creating interaction between some features.

Given what I know movies and how in the winter and summer they have “summer action hit” or winter animation movies, analyzing the interaction of genre and winter and summer, I thought would lead to lower RMSE models.

Therefore, I created the following combinations in addition to the other features I created in previous questions:

summer*action winter*animation summer*animation winter*drama summer*drama imdbVotes*Budget

However summer&action, summer&animation, and imdbVotes&Budget proved the best metrics.

Ultimately, the creating the interaction variables achieved incremental improvement of about 5-10M. (Q4 64M to Q5 60M)