Movielens project

Anthony S N Maina

November 2020

INTRODUCTION

For this project, I have created a movie recommendation system using the MovieLens dataset. I have used a subset of the movielens data included in the dslabs package.

This project starts from the code provided to download and generate the subset of MovieLens data to be used for analysis. The code for generating the edx and validation datasets is thus not shown here.

The data contains roughly 10 million records of movie ratings, with attributes related to the movies themselves (title, genre, year of release etc) as well as the users who rated the movies (userId, timestamp of rating etc)

The goal of the project is to generate and run a model which takes these attributes for the validation sets, generates predicted ratings, and also computes the RMSE vs the actual ratings.

This report is split into the following sections:

1) Exploratory Data Analysis
2) Generation of Machine Leaning Model
3) Results
4) Discussion

EXPLORATORY DATA ANALYSIS

We begin by simply scanning the form of the data in the edx dataset. The idea here is to get a sense of the format of the data in each column, and understand whether further manipulation is necessary to extract additionl information.

We start by initialising the necessary libraries and extracting a view of the first 6 rows of data

```
library(lubridate)
library(tidyverse)
library(caret)
library(data.table)
library(anytime)
library(stringr)
```

```
edx<-read.csv(file.path("edx.csv"), stringsAsFactors=FALSE)
edx %>% head()
```

```
##   X userId movieId rating timestamp                          title
## 1 1      1     122      5 838985046              Boomerang (1992)
## 2 2      1     185      5 838983525               Net, The (1995)
## 3 4      1     292      5 838983421               Outbreak (1995)
## 4 5      1     316      5 838983392               Stargate (1994)
## 5 6      1     329      5 838983392 Star Trek: Generations (1994)
## 6 7      1     355      5 838984474       Flintstones, The (1994)
```

```
##                          genres
## 1                Comedy|Romance
## 2         Action|Crime|Thriller
## 3  Action|Drama|Sci-Fi|Thriller
## 4       Action|Adventure|Sci-Fi
## 5 Action|Adventure|Drama|Sci-Fi
## 6       Children|Comedy|Fantasy
```

We see here that there are a few columns where some additional effort in data manipulation could yield more interesting attributes:

- Title - We notice that in each title, the release year is contained within parentheses.
- Timestamp - We notice that this is encoded as an integer, possibly containing info on the year, month, day and hour that the rating was generated
- Genres - We notice that this column contains multiple genre categories, which could potentially yield interesting results if separated and analysed separately

We carry out some operations in R to extract the necessary data - see code below

```
edx["release_year"]<-str_sub(edx$title, -5,-2) # extracts the four last characters from the "title" att
edx["datetime"]<-edx$timestamp %>% anytime() # converts the timestamp variable into a date-time format
edx["year"]<-edx$datetime %>% year() # extracts the year of the time stamp
edx["month"]<-edx$datetime %>% month() # extracts the month of the time stamp
edx["dow"]<-edx$datetime %>% wday() # extracts the day of the time stamp
edx["hour"]<-edx$datetime %>% hour() # extracts the hour of the time stamp

#extract a list of each of the individual genre names in the Genre column
gen<-list()
a_head<-unique(edx$genres)%>% strsplit('|', fixed=TRUE)
for (j in 1:length(unique(edx$genres))) {
    for (i in 1:length(a_head[[j]])) {
        gen<-append(gen,a_head[[j]][i])
    }
}

# Drop the duplicate genres
gen<- gen %>% unique()

# Generate a column for each individual genre category, and populate with "1"True" if the category is fo
for (i in 1:length(gen)) {
    chars <- edx$genres
    value <-gen[[i]][1]
    edx[value]<-str_detect(chars, value)
}

# Re-name the columns with unexpected characters : "-", "(" and ")"
names(edx)[names(edx) == 'Sci-Fi'] <- 'SciFi'
names(edx)[names(edx) == 'Film-Noir'] <- 'FilmNoir'
names(edx)[names(edx) == '(no genres listed)'] <- 'Blank'

# Convert "True" and "False" labels into 1 and 0 respectively
cols <- sapply(edx, is.logical)
edx[,cols] <- lapply(edx[,cols], as.numeric)
head(edx)
```

```
##   X userId movieId rating timestamp                          title
```

```
## 1 1      1     122      5 838985046              Boomerang (1992)
## 2 2      1     185      5 838983525               Net, The (1995)
## 3 4      1     292      5 838983421               Outbreak (1995)
## 4 5      1     316      5 838983392               Stargate (1994)
## 5 6      1     329      5 838983392 Star Trek: Generations (1994)
## 6 7      1     355      5 838984474       Flintstones, The (1994)
##                          genres release_year            datetime year month dow
## 1                Comedy|Romance         1992 1996-08-02 13:24:06 1996     8   6
## 2         Action|Crime|Thriller         1995 1996-08-02 12:58:45 1996     8   6
## 3  Action|Drama|Sci-Fi|Thriller         1995 1996-08-02 12:57:01 1996     8   6
## 4       Action|Adventure|Sci-Fi         1994 1996-08-02 12:56:32 1996     8   6
## 5 Action|Adventure|Drama|Sci-Fi         1994 1996-08-02 12:56:32 1996     8   6
## 6       Children|Comedy|Fantasy         1994 1996-08-02 13:14:34 1996     8   6
##   hour Comedy Romance Action Crime Thriller Drama SciFi Adventure Children
## 1   13      1       1      0     0        0     0     0         0        0
## 2   12      0       0      1     1        1     0     0         0        0
## 3   12      0       0      1     0        1     1     1         0        0
## 4   12      0       0      1     0        0     0     1         1        0
## 5   12      0       0      1     0        0     1     1         1        0
## 6   13      1       0      0     0        0     0     0         0        1
##   Fantasy War Animation Musical Western Mystery FilmNoir Horror Documentary
## 1       0   0         0       0       0       0        0      0           0
## 2       0   0         0       0       0       0        0      0           0
## 3       0   0         0       0       0       0        0      0           0
## 4       0   0         0       0       0       0        0      0           0
## 5       0   0         0       0       0       0        0      0           0
## 6       1   0         0       0       0       0        0      0           0
##   IMAX Blank
## 1    0     0
## 2    0     0
## 3    0     0
## 4    0     0
## 5    0     0
## 6    0     0
```

Data Visualisation

Having processed the data we are now in a position to examine the data in a little more detail.

We begin by examining the basic distribution of the ratings.

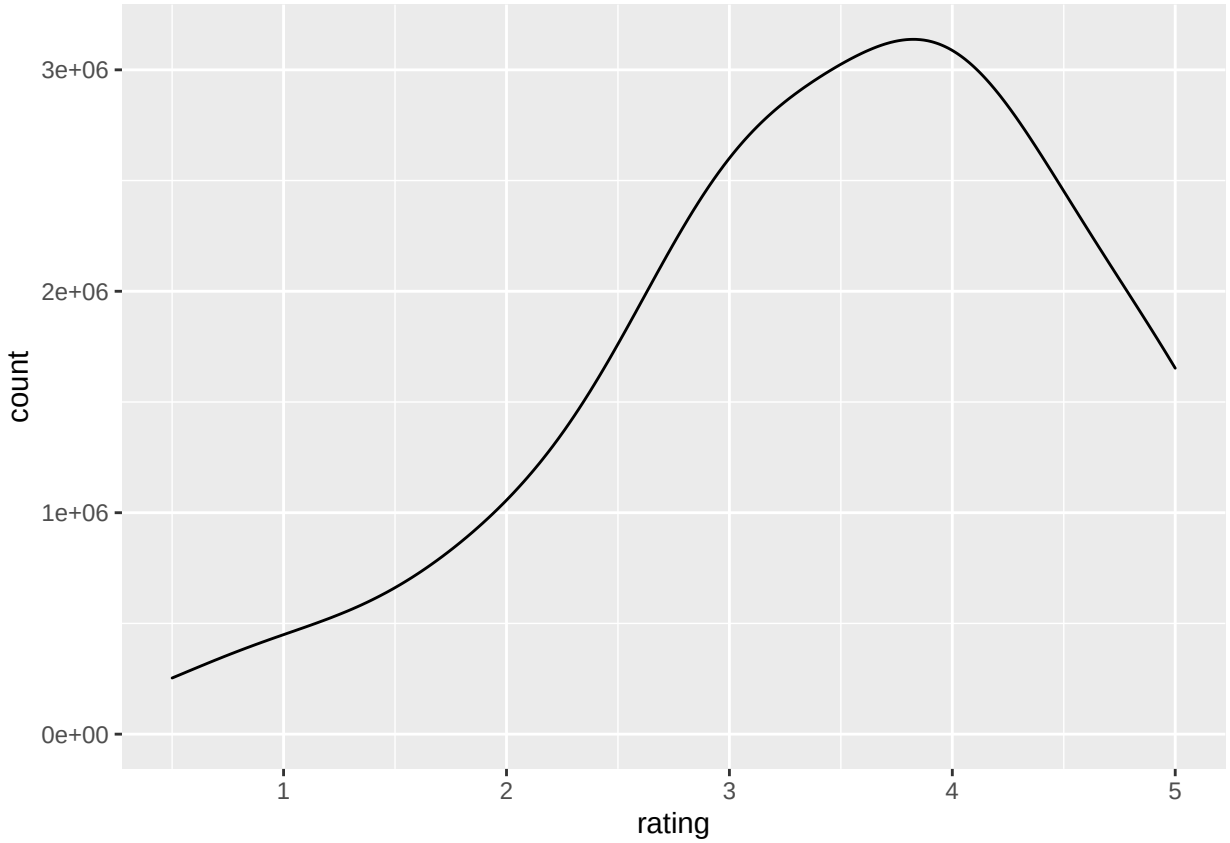
```
#mean and standard deviation
mu <- mean(edx$rating)
mu
```

```
## [1] 3.512465
```

```
s<-sd(edx$rating)
s
```

```
## [1] 1.060331
```

```
# basic histograms
p <- edx %>%
    ggplot(aes(x = rating,  y = ..count..))
p + geom_density(bw = 0.5)
```



The ratings have a negatively skewed distribution with a mean of 3.51 and a standard deviation of 1.06. Note here that standard deviation is nothing more than the RMSE vs the mean of the distribution.

We will examine the data by grouping the attributes in 2 main super-groups:

1) Attributes related to the Movie ID (genre, release year)
2) Attributes related to the User ID (timestamp)

We should also note already that **the attributes here are only groupings of either userID related or movie ID related characteristics**

Movie ID related attributes

```
# VISUALISE NUMBER OF RATINGS PER MOVIE
# How many movies?
unique(edx$movieId) %>% length()
```

Number of ratings per movie

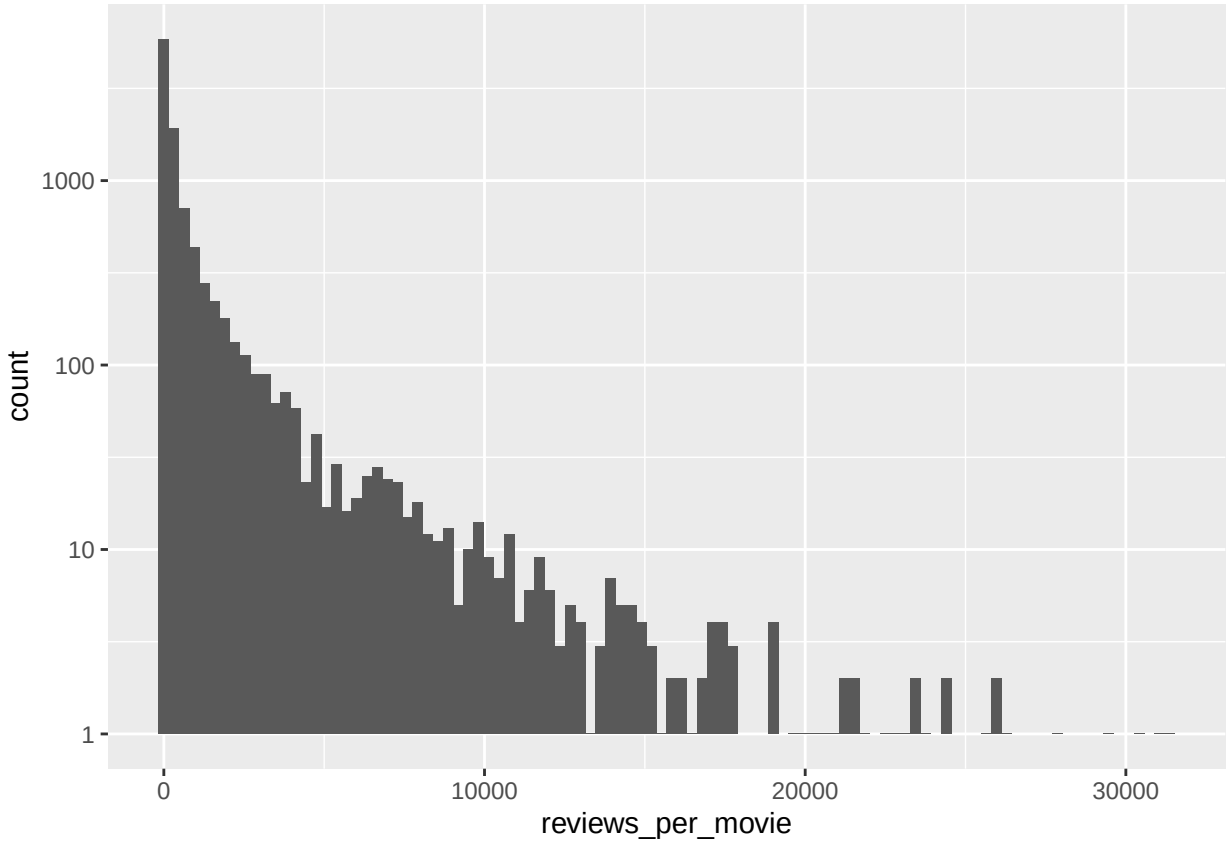
```
## [1] 10677
```

```
#Distribution of ratings by movie?

edx %>%
group_by(movieId) %>%
summarize(reviews_per_movie=n()) %>%
ggplot(aes(x = reviews_per_movie)) +
```

```
geom_histogram(bins=100) +
scale_y_log10()
```

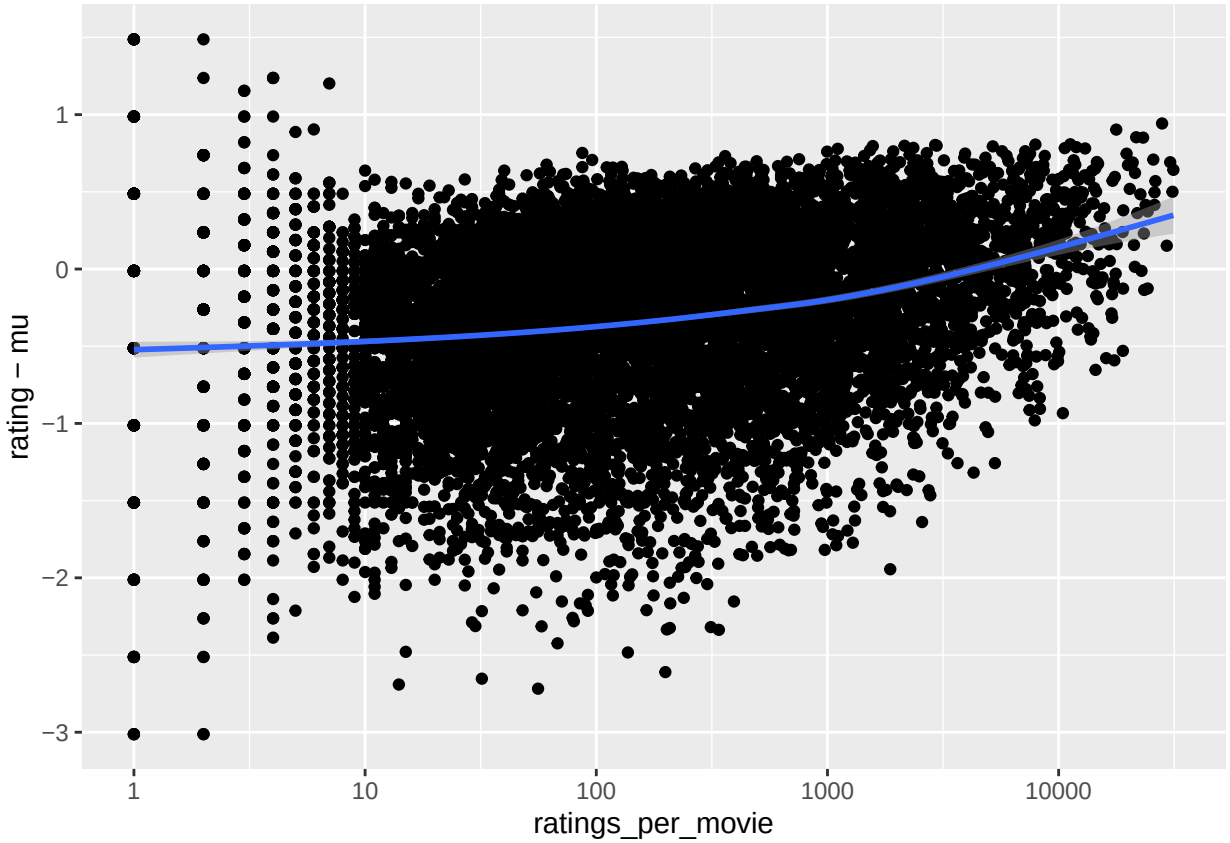
```
## Warning: Transformation introduced infinite values in continuous y-axis
```

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



We see here that a significant number of movies have 5000 reviews or less, while a few have more than 20k reviews. Lets have a look first at how the more reviewed movies perform vs the average.

```
edx %>% group_by(movieId) %>%
  summarize(ratings_per_movie=n(), rating = mean(rating)) %>%
  ggplot(aes(x = ratings_per_movie, y = rating-mu)) +
  geom_point() +
  geom_smooth() +
  scale_x_log10()
```



We see here that movies with more reviews generally received higher ratings; with movies exceeding 10000 reviews receving scores higher than the mean. So we are saying a small number of very popular movies is responsible for dragging the overall score up.

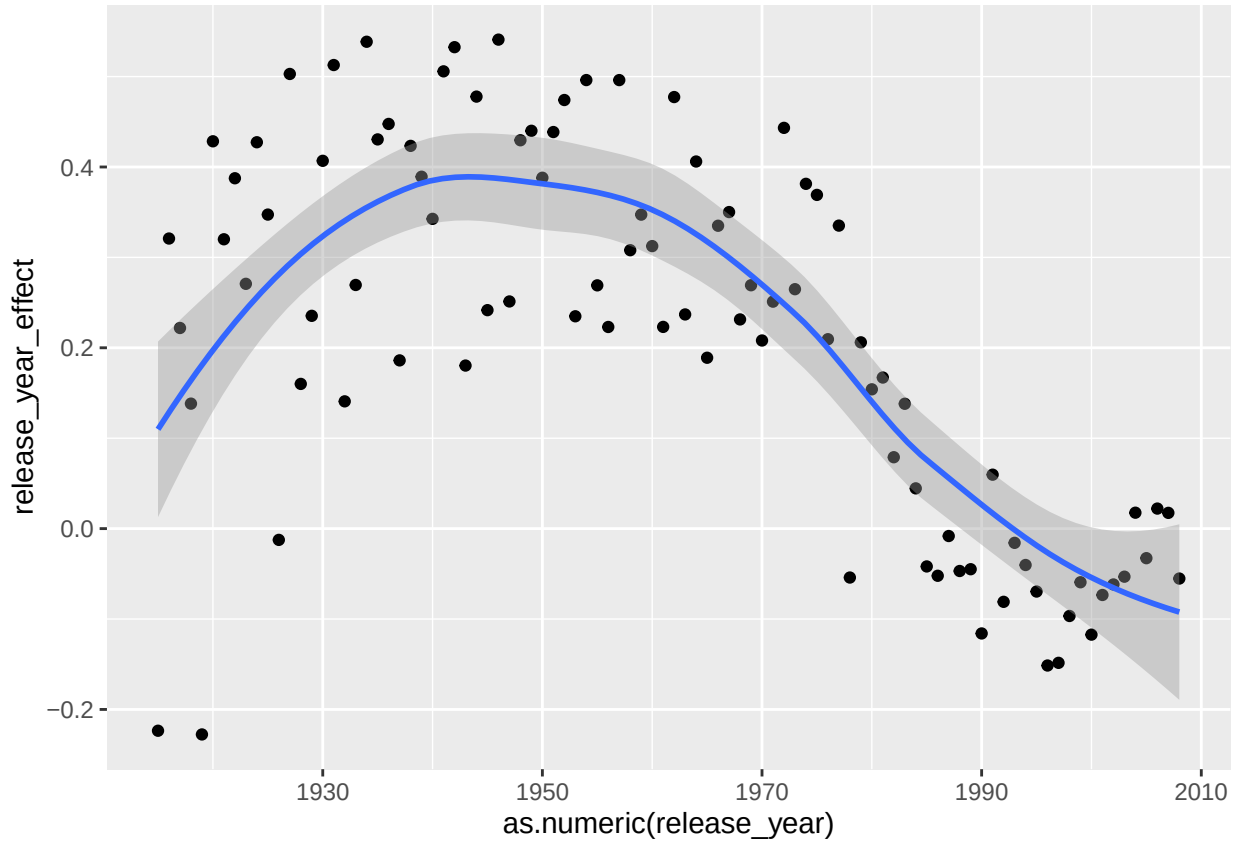
```
gen<-c("Comedy", "Romance", "Action", "Crime", "Thriller", "Drama", "SciFi","Adventure", "Children", "Fa

gen_sum <- data.frame("Genre"=c(), "Rating"=c(), "Number"=c())

for (i in gen){
  gen_sum<-rbind(gen_sum, data.frame("genre" = i, "genre_effect" = edx %>% filter (eval(as.symbol(i))==
}
gen_sum
```

```
##          genre genre_effect       n
## 1       Comedy  -0.07555710 3540930
## 2      Romance   0.04134824 1712100
## 3       Action  -0.09106058 2560545
## 4        Crime   0.15346009 1327715
## 5     Thriller  -0.00478946 2325899
## 6        Drama   0.16066549 3910127
## 7        SciFi  -0.11672204 1341183
## 8    Adventure  -0.01892130 1908892
## 9     Children  -0.09374974  737994
## 10     Fantasy  -0.01051897  925637
## 11         War   0.26834736  511147
## 12   Animation   0.08817846  467168
## 13     Musical   0.05083950  433080
```

```
## 14     Western   0.04345262  189394
## 15     Mystery   0.16453609  568332
## 16    FilmNoir   0.49915947  118541
## 17      Horror  -0.24265024  691485
## 18 Documentary   0.27102179   93066
## 19        IMAX   0.25522823    8181
## 20       Blank   0.13039194       7
```

Year of release We next examine the variation in ratings vs year of release.

```
# VISUALISE TREND OVER TIME
edx %>%
group_by(release_year) %>%
summarize (N=n(), release_year_effect = mean(rating)-mu, st_dev=sd(rating), median = median(rating), Te
ggplot(aes(as.numeric(release_year), release_year_effect)) +
geom_point() +
geom_smooth()
```



We see here that the variation is significant, with movies released earlier generally greatly out-performing movies released more recently. Hipster much?

User ID related attributes

Number of ratings per user We start by looking at the number of ratings per user, and how these relate to the ratings given.

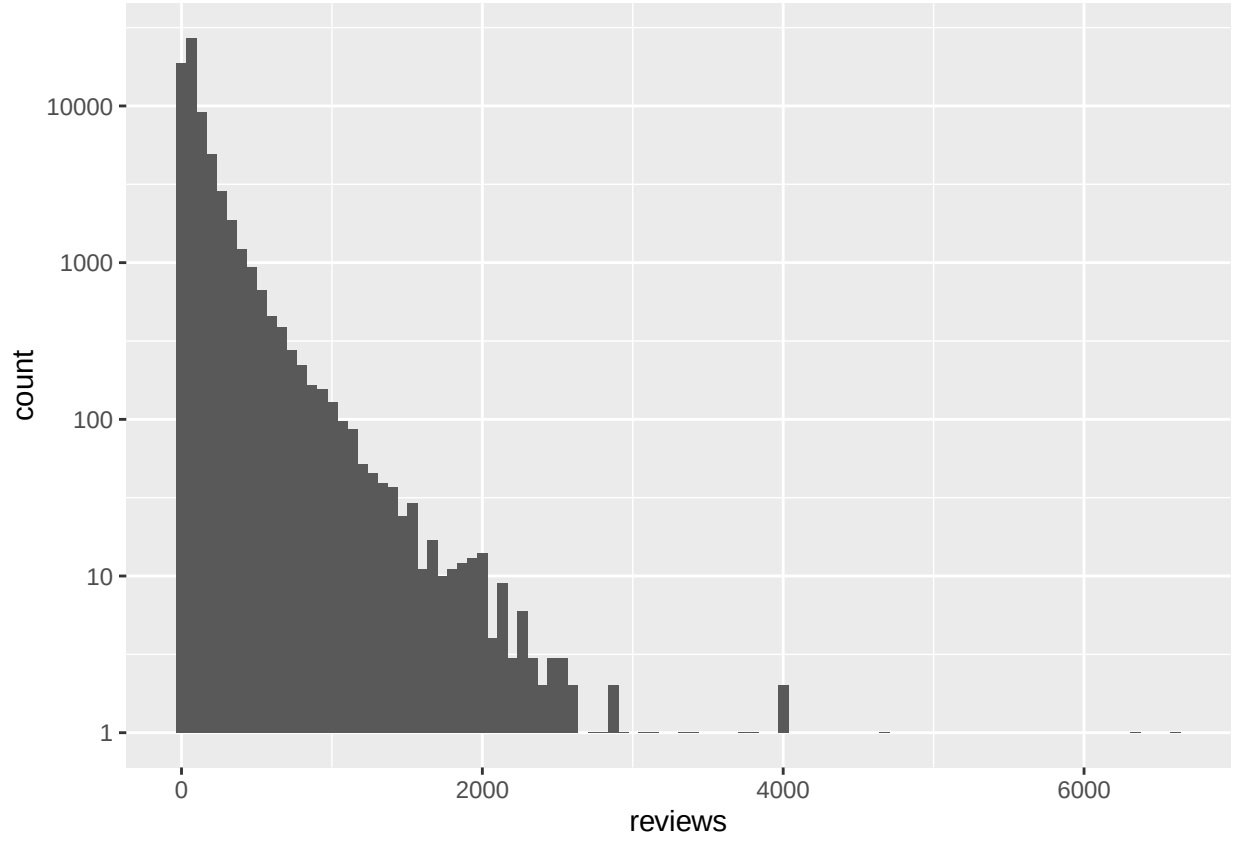
```
# VISUALISE NUMBER OF RATINGS PER USER
# How many users?

unique(edx$userId) %>% length()
```

```
## [1] 69878
```

```
# Distribution of ratings by user

edx %>%
group_by(userId) %>%
summarize(reviews=n()) %>%
ggplot(aes(x = reviews)) +
geom_histogram(bins=100) +
scale_y_log10()
```

```
## Warning: Transformation introduced infinite values in continuous y-axis
```

```
## Warning: Removed 46 rows containing missing values (geom_bar).
```

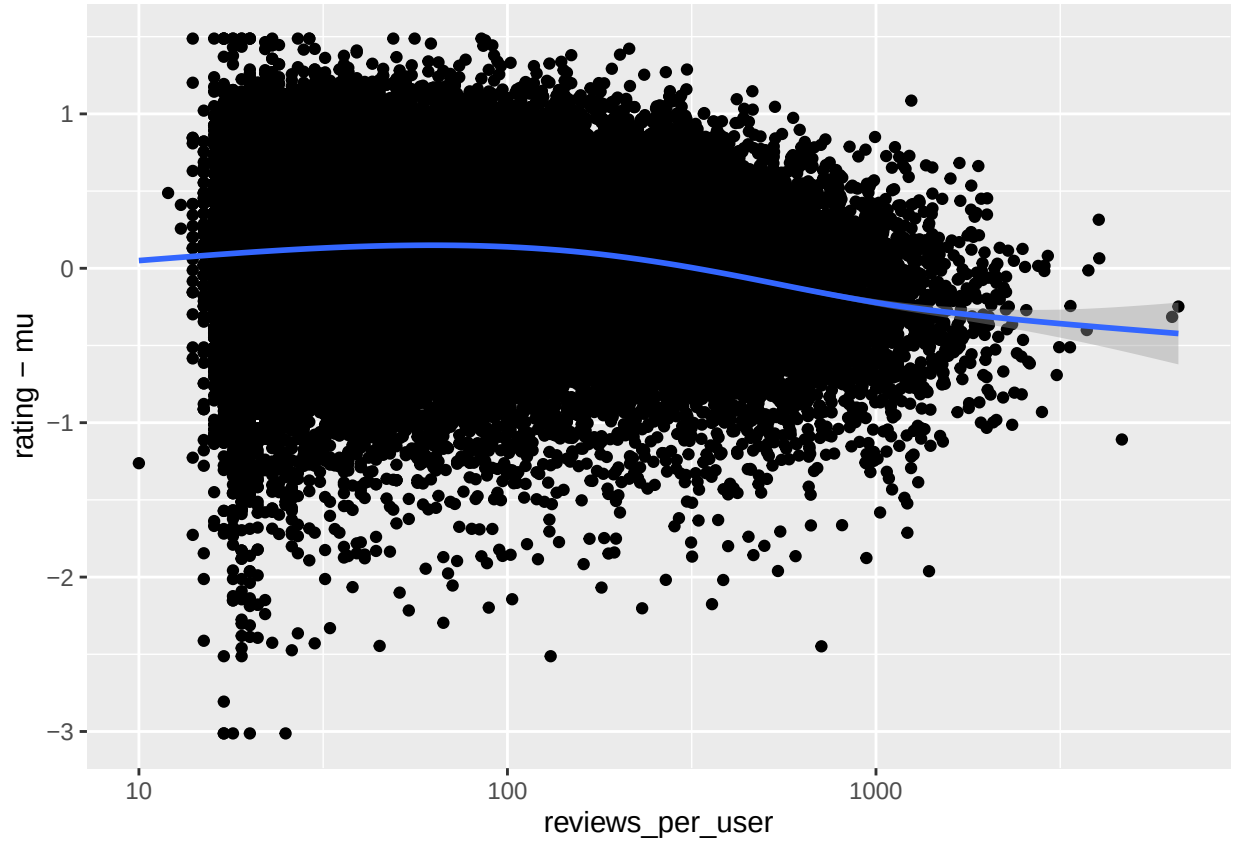


We see hee that the majority of users have given fewer than 1000 reviews. But there is a significant tail of "super-users" who have given more than 1000, as well as a decent chunk who have given very few reviews.

Lets see how this translates into the variation in reviews given.

```
edx %>% group_by(userId) %>%
  summarize(reviews_per_user=n(), rating = mean(rating)) %>%
  ggplot(aes(x = reviews_per_user, y = rating-mu)) +
  geom_point() +
```

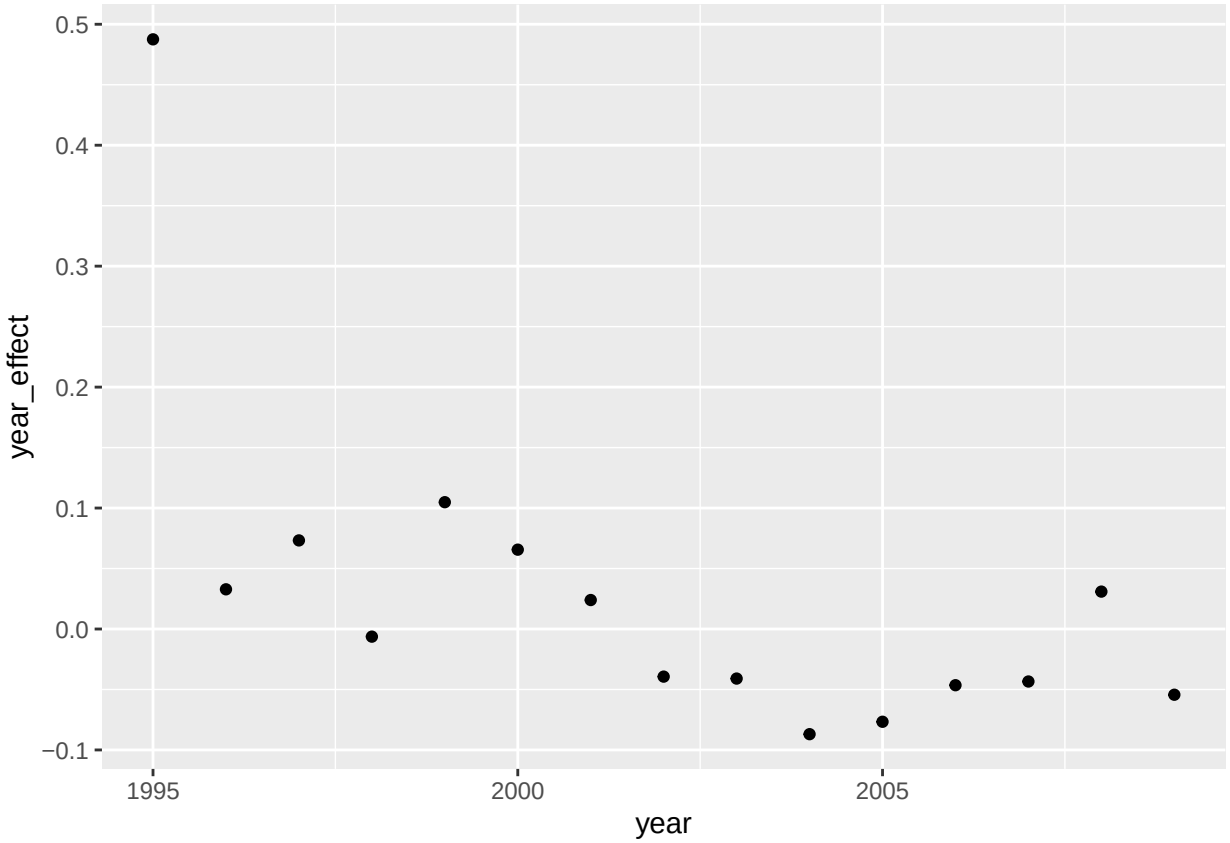
```
  geom_smooth() +
  scale_x_log10()
```



Here we see that somewhat naturally, reviewers who give more reviews are generally harsher critics.

Timestamp We begin by looking at the effect of the year the review was given.

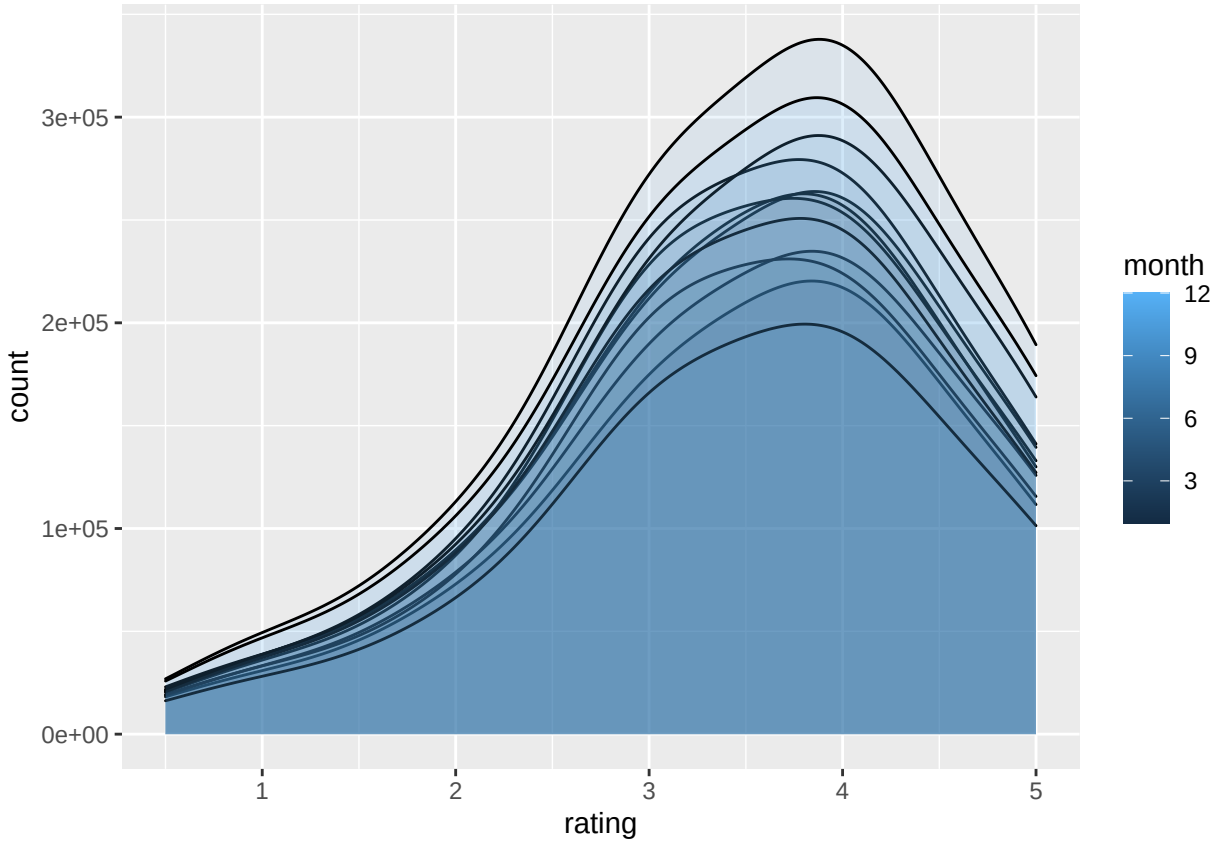
```
# VISUALISE TREND OVER TIME
edx %>%
group_by(year) %>%
summarize (N=n(), year_effect = mean(rating)-mu,st_dev=sd(rating), median = median(rating), Tenth = quan
ggplot(aes(year, year_effect)) +
geom_point()
```



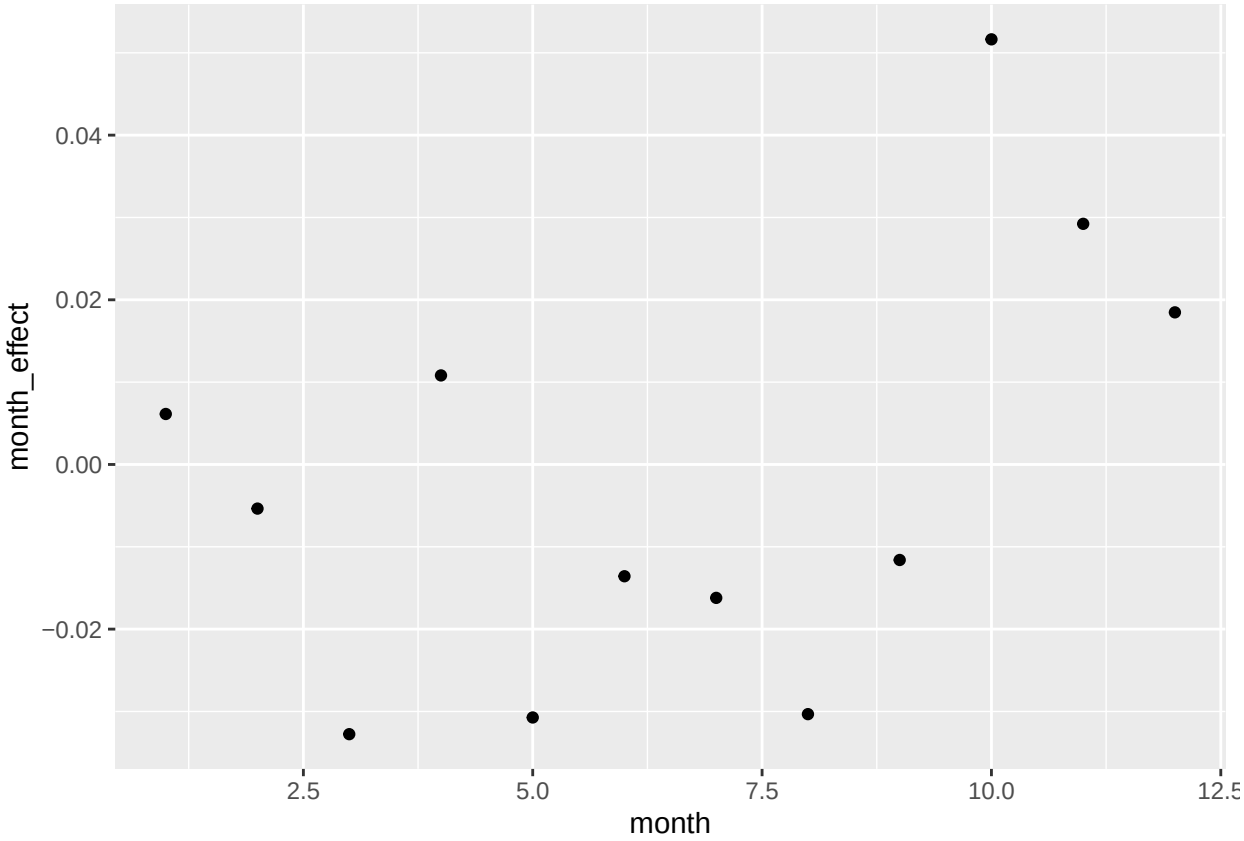
Here we see that reviews given more recently are generally lower vs the mean. So audiences are getting more critical as time goes on.

Looking at the month:

```
p <- edx %>%
    ggplot(aes(rating, y = ..count.., group = month, fill = month))
p + geom_density(alpha = 0.1, bw = 0.5)
```



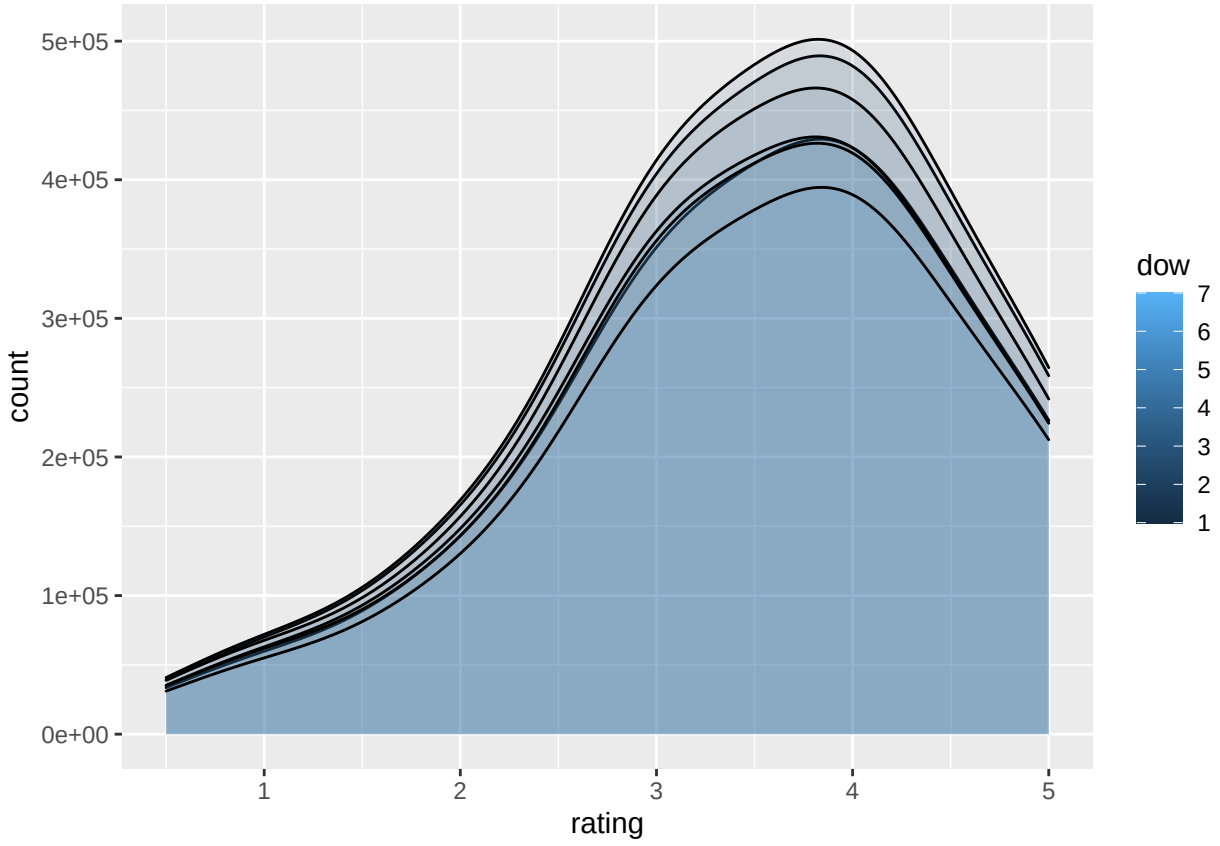

```
edx %>%
group_by(month) %>%
summarize (N=n(), month_effect = mean(rating)-mu,st_dev=sd(rating), median = median(rating), Tenth = qua
ggplot(aes(month, month_effect)) +
geom_point()
```



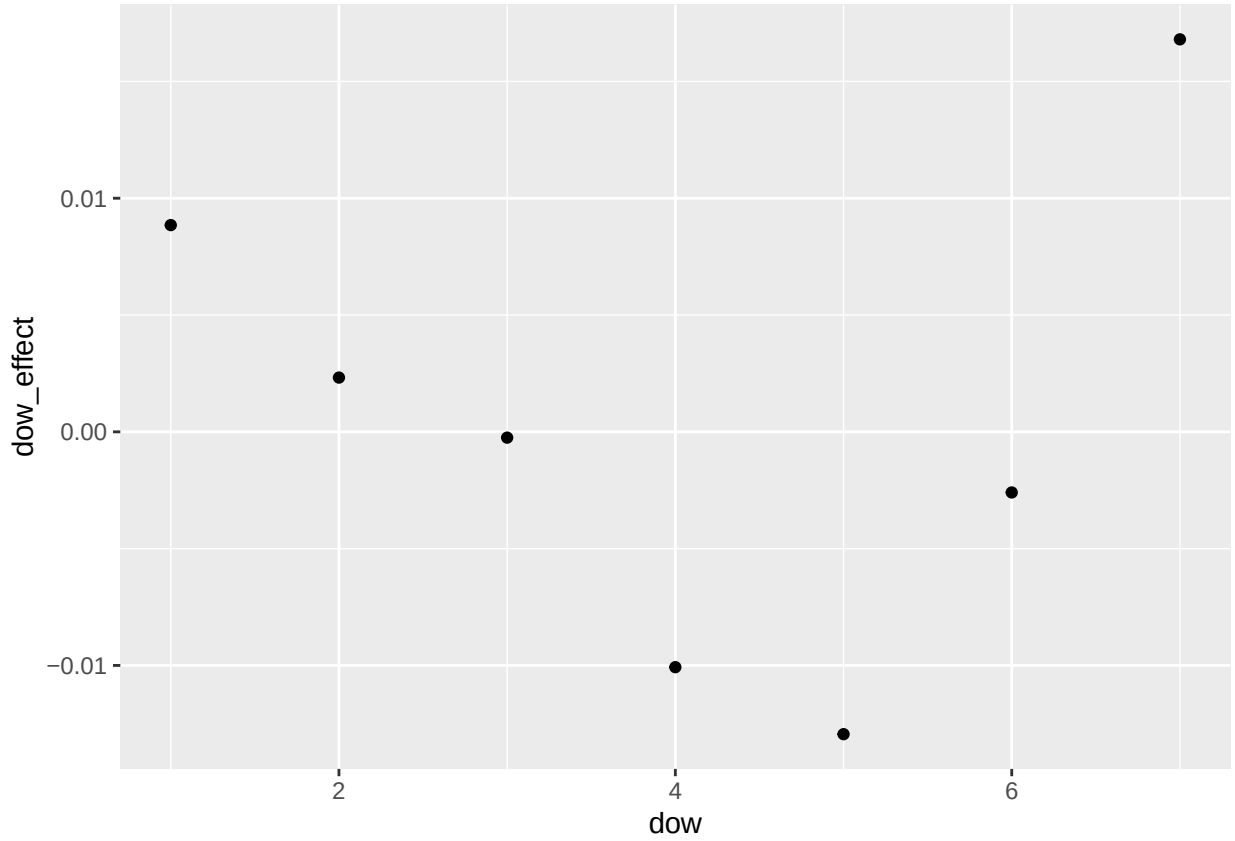
We see very small deviations from the mean, certainly not enough to swing from one rating to the next given the minimum step is 0.5.

The story is similar for day of week (dow), and hour of day (hour)

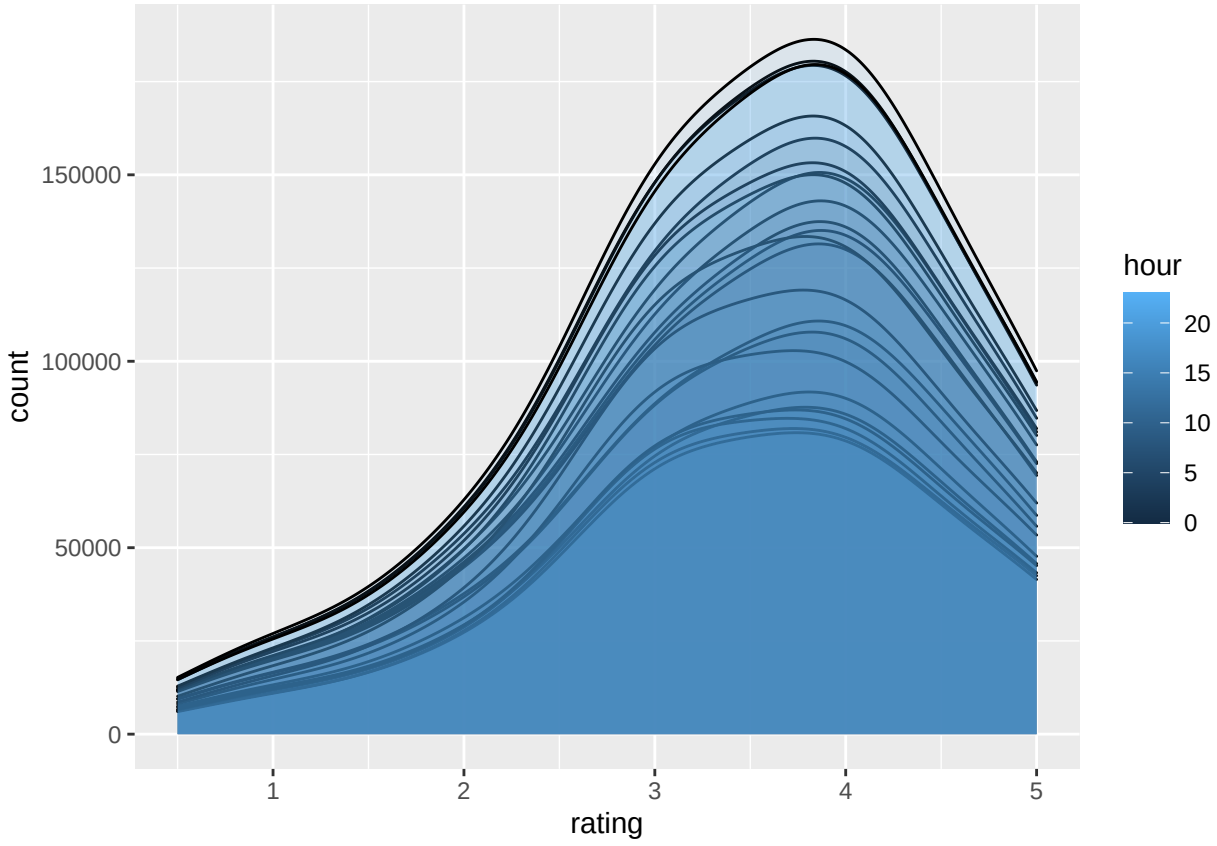
```
p <- edx %>%
    ggplot(aes(rating, y = ..count.., group = dow, fill = dow))
p + geom_density(alpha = 0.1, bw = 0.5)
```



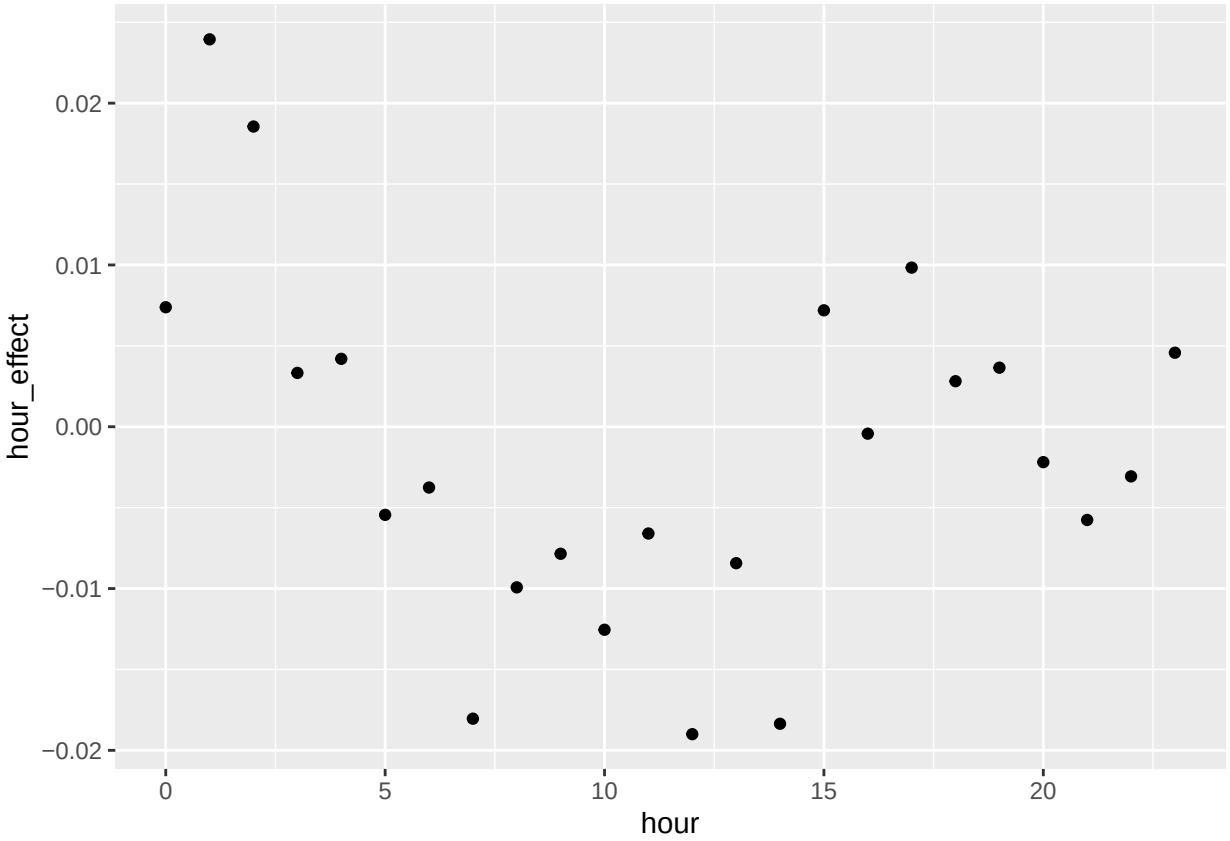

```
edx %>%
group_by(dow) %>%
summarize (N=n(), dow_effect = mean(rating)-mu,st_dev=sd(rating), median = median(rating), Tenth = quan
ggplot(aes(dow, dow_effect)) +
geom_point()
```




```
p <- edx %>%
    ggplot(aes(rating, y = ..count.., group = hour, fill = hour))
p + geom_density(alpha = 0.1, bw = 0.5)
```




```
edx %>%
group_by(hour) %>%
summarize (N=n(), hour_effect = mean(rating)-mu,st_dev=sd(rating), median = median(rating), Tenth = quar
ggplot(aes(hour, hour_effect)) +
geom_point()
```



We now have done enough analysis to train our model.

Train model on subset of data

We will train our model based on linear regression, using movieID and userID as our variables. We will carry out the following steps:

1) Split the edx dataset into training and test groups, in order to select and tune our model
2) Apply the tuned model to the validation dataset, and compute the resulting RMSE.

Building the model

First we split the edx dataset into train and test sets

```
edx<-edx[,c("rating","title", "movieId", "userId")]

set.seed(7)

test_index <- createDataPartition(edx$rating, times = 1, p = 0.2, list = FALSE)

train_set <- edx %>% slice(-test_index)

test_set <- edx %>% slice(test_index)
```

Prediction based on mean This is the simplest possible algorithm, as it entails assuming the mean rating from the training dataset is applied uniformly to the test dataset. It produces an answer which is predictably not far off the standard deviation of the overall distribution.

```
mu_train<-mean(train_set$rating)

# RMSE based on avg alone
s<-sqrt(mean((test_set$rating-mu_train)^2))
s
```

```
## [1] 1.060641
```

Prediction based on userId and movie Id. We begin by calculating the "movie effect", which is simply the mean rating applied per movie less the mean rating.

```
bi <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu_train))
head(bi)
```

```
## # A tibble: 6 x 2
##   movieId    b_i
##     <int>  <dbl>
## 1       1  0.417
## 2       2 -0.303
## 3       3 -0.360
## 4       4 -0.631
## 5       5 -0.435
## 6       6  0.301
```

We then calculate the incremental user effect, which is the mean rating per user, less the mean rating for the dataset as a whole, less the movie effect.

```
bu <- train_set %>%
  left_join(bi, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu_train - b_i))
head(bu)
```

```
## # A tibble: 6 x 2
##   userId    b_u
##    <int>  <dbl>
## 1      1  1.69
## 2      2 -0.257
## 3      3  0.259
## 4      4  0.737
## 5      5  0.128
## 6      6  0.329
```

We are now ready to apply these two effects to the testing dataset, and compute the resulting RMSE.

```
pred_bi_bu <- test_set %>%
  left_join(bi, by = "movieId") %>%
  left_join(bu, by = "userId")
pred_bi_bu[is.na(pred_bi_bu)]<-0
pred_bi_bu<-pred_bi_bu %>% mutate (pred_bi_bu = mu_train+b_i+b_u)
sqrt(mean((pred_bi_bu$pred_bi_bu- test_set$rating)^2))
```

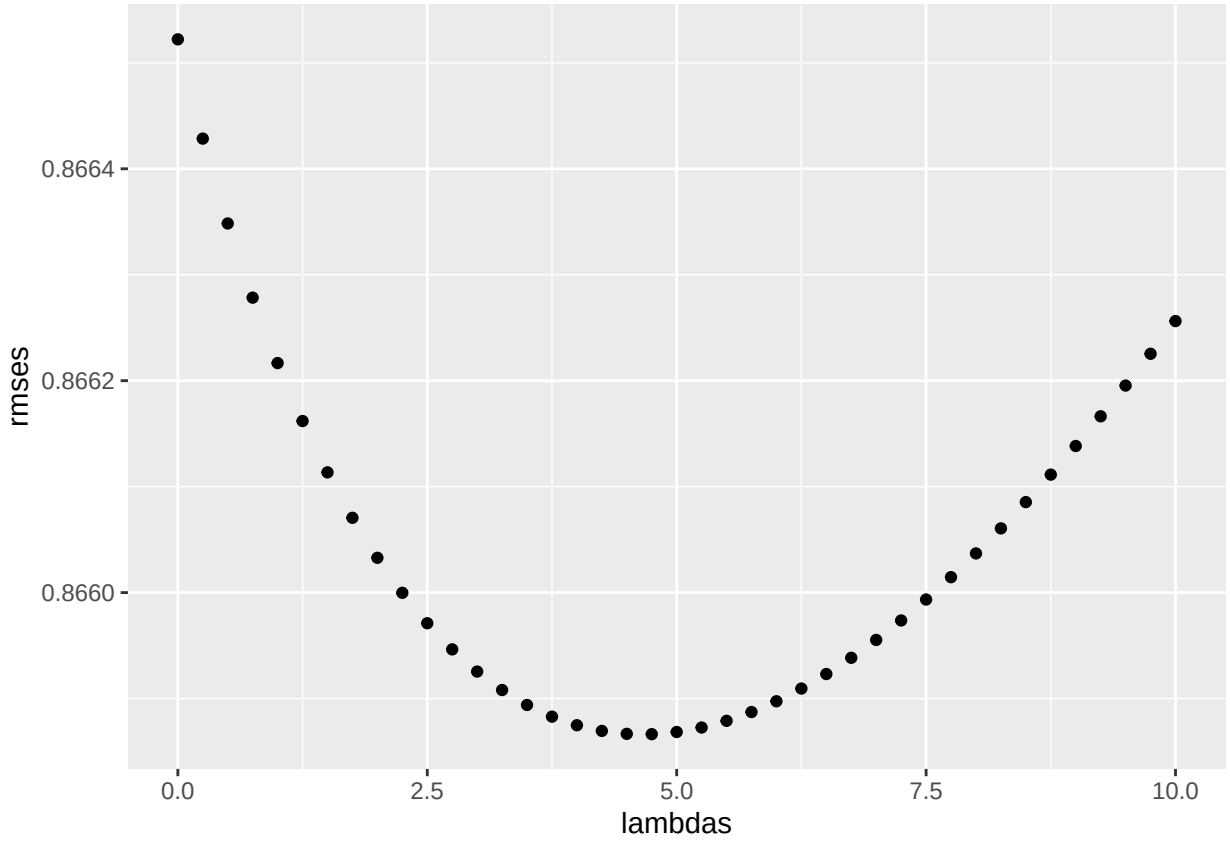
```
## [1] 0.8664154
```

```
pred_bi_bu %>% head()
```

```
##   rating                                          title movieId userId
## 1      5                                Net, The (1995)     185      1
## 2      5         Snow White and the Seven Dwarfs (1937)     594      1
## 3      3                          River Wild, The (1994)     376      2
## 4      3                                  Eraser (1996)     786      2
## 5      3                            Mars Attacks! (1996)    1391      2
## 6      5 Burnt by the Sun (Utomlyonnye solntsem) (1994)     213      3
##          b_i        b_u pred_bi_bu
## 1 -0.3841712  1.6857033   4.814018
## 2  0.1083606  1.6857033   5.306550
## 3 -0.2350319 -0.2569775   3.020476
## 4 -0.3436435 -0.2569775   2.911865
## 5 -0.5605135 -0.2569775   2.694995
## 6  0.5582400  0.2591958   4.329922
```

We know that linear regression-based algorithms are sensitive to outliers (ie otherwise isolated datapoints with extreme RMSE values); so we introduce the lambda term to penalise for this; and compute the RMSE iteratively in order to select the lambda which will minimise RMSE on our test dataset.

```
# lambda is a tuning parameter to penalise outliers
lambdas <- seq(0, 10, 0.25)

# For each lambda,find b_i & b_u, and then compute RMSE. NOTE: THIS TAKES A WHILE TO RUN
rmses <- sapply(lambdas, function(l){

  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu_train)/(n()+l))

  b_u <- train_set %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu_train)/(n()+l))

  predicted_ratings <- test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu_train + b_i + b_u)

  predicted_ratings[is.na(predicted_ratings)] <-0

  predicted_ratings <- predicted_ratings$pred

  return(RMSE(test_set$rating,predicted_ratings))
})

# Plot rmses vs lambdas to select the optimal lambda
qplot(lambdas, rmses)
```



We can see that this lambda falls on around 4.75. We now have a model to use on the Validation dataset.

RESULTS

Below our model is applied to the Validation dataset to obtain a final RMSE

```
validation<-fread("validation.csv")
```

```
l <- lambdas[which.min(rmses)]
b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu_train)/(n()+l))

b_u <- train_set %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu_train)/(n()+l))

predicted_ratings <- validation%>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu_train + b_i + b_u)

predicted_ratings[is.na(predicted_ratings)] <-0

predicted_ratings <- predicted_ratings$pred
```

```
RMSE(predicted_ratings, validation$rating)
```

```
## [1] 0.8657261
```

We have developed and tuned a ratings prediction model which uses userId and movieId as inputs, and achieves an RMSE of 0.8657 (vs a worst case of 1.06.)

DISCUSSION

We note that this model would breaks down for either a new movie or a new user.

We also note that although additional attributes were present related to the movies or users themselves, these would not reasonably be expected to give a better prediction than userId or movieID, since they are nothing more than aggregations of multiple movies and/or users. (For example, since a given user rates a given movie once, any grouping based on timestamp just an aggregation of ratings from number of userId's. Likewise for any grouping based on genre is nothing more than an aggregation of ratings from a a number of movieId's.)

This concludes this piece of work.