Movielens Project Report

Puneeth Nikin Krishnan

30/04/2020

Executive Summary

The objective of this project is to develop a recommendation system based on the Movielens Dataset. To achieve this, ratings need to be predicted for a user based on past ratings. The goal here is to accurately predict ratings. To be able to gauge the performance of different models, RMSE (Root Mean Square Error) will be used. The dataset has been split into two sets, one for training(edx) and the other testing(validation) our model. The validation set will not be used, except to validate the best performing model. The training set will be further split into a train set and a test set to identify the best model. The best model will be one which has the least RMSE. The steps followed to build this model are first importing the dataset, preparing the dataset, exploring or analysing the dataset, building the models, choosing the best model and finally validating the results on the validation set.

Importing the Dataset

The code to import the dataset has been proveded by edx. link to import dataset

Running the code will give us two dataframes namely edx and validation. The edx dataframe will be used to build our model.

Analysis

```
dim(edx)
```

```
## [1] 9000055       6
```

We notice that the dataset is pretty huge with 9,000,055 rows and 6 columns

```
names(edx)
```

```
## [1] "userId"    "movieId"   "rating"    "timestamp" "title"     "genres"
```

```
edx%>%head()%>%knitr::kable()
```

	userId	movieId	rating	timestamp	title	genres
1	1	122	5	838985046	Boomerang (1992)	Comedy|Romance
2	1	185	5	838983525	Net, The (1995)	Action|Crime|Thriller
4	1	292	5	838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller
5	1	316	5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi
6	1	329	5	838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi
7	1	355	5	838984474	Flintstones, The (1994)	Children|Comedy|Fantasy

The edx dataframe comprises of 5 columns. 'userId' represents the unique user ID, 'movieId' represents the unique ID for each movie, timestamp denotes the time of the rating, title represents title and genres the combination of genres. Each row represents one rating by a user. We can notice that the title section contains the year the movie was released in. We will use regex to extract the year released from this column. The regex pattern used to extract the data is "(\d{4})\)$" . This can be evidenced by the fact that the year is at

the end of the string in column title. We use the str_match() function from stringr in tidyverse package to extract the year and convert to a date format using the as.date() and year() functions from the lubridate package.

```
# extracting the year released from the title column
regex_pattern<-"(\\d{4})\\)$"
edx<-edx%>%
  mutate(year_released=year(as.Date(str_match(title,regex_pattern)[,2],format="%Y")))
edx%>%head()%>%
  knitr::kable()%>%kable_styling(latex_options = c("striped", "scale_down"))
```

userId	movieId	rating	timestamp	title	genres	year_released
1	122	5	838985046	Boomerang (1992)	Comedy|Romance	1992
1	185	5	838983525	Net, The (1995)	Action|Crime|Thriller	1995
1	292	5	838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller	1995
1	316	5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi	1994
1	329	5	838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi	1994
1	355	5	838984474	Flintstones, The (1994)	Children|Comedy|Fantasy	1994

We can convert the timestamp column to a readable datetime object using the as_datetime() from the lubridate package

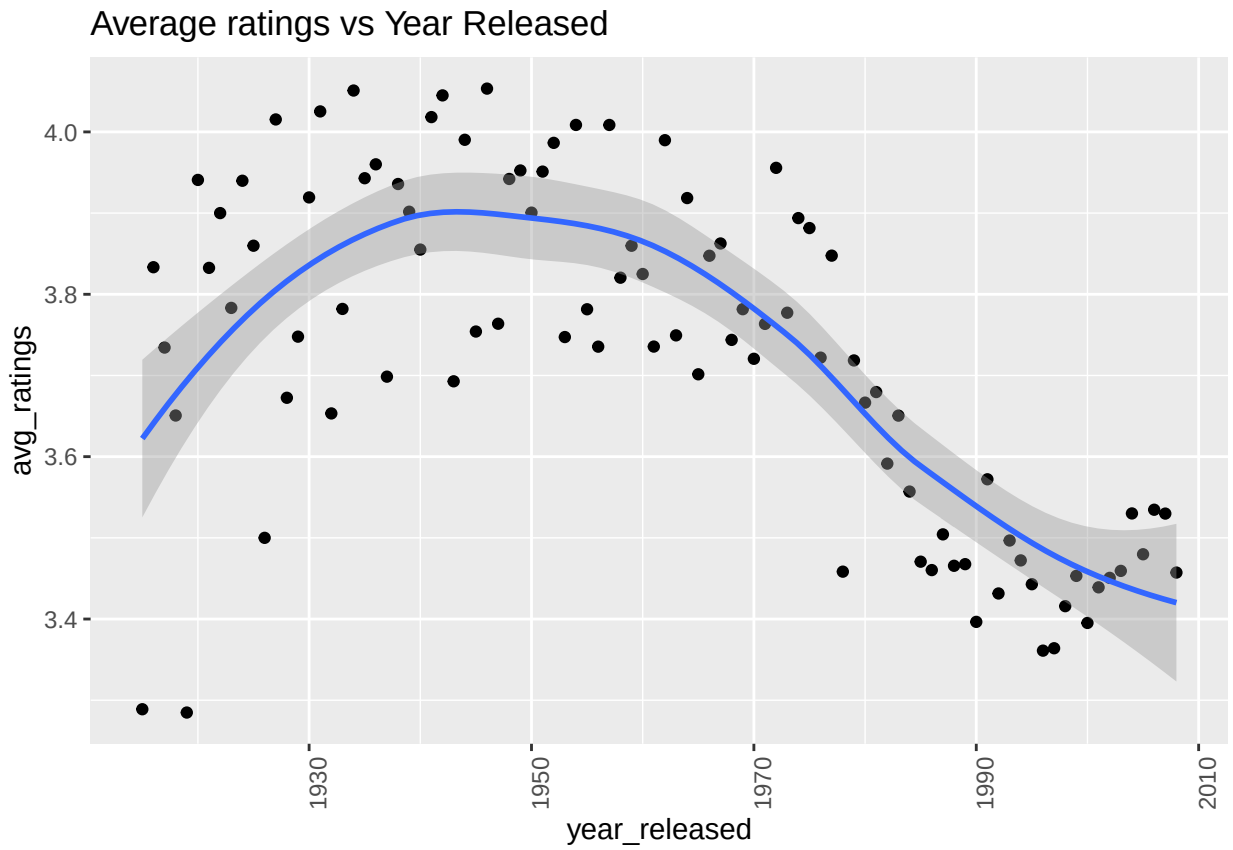
```
# converting the timestamp column to readable date
edx <- edx%>%
  mutate(datetime=as_datetime(timestamp))
edx%>%
  head()%>%
  knitr::kable()%>%kable_styling(latex_options = c("striped", "scale_down"))
```

userId	movieId	rating	timestamp	title	genres	year_released	datetime
1	122	5	838985046	Boomerang (1992)	Comedy|Romance	1992	1996-08-02 11:24:06
1	185	5	838983525	Net, The (1995)	Action|Crime|Thriller	1995	1996-08-02 10:58:45
1	292	5	838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller	1995	1996-08-02 10:57:01
1	316	5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi	1994	1996-08-02 10:56:32
1	329	5	838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi	1994	1996-08-02 10:56:32
1	355	5	838984474	Flintstones, The (1994)	Children|Comedy|Fantasy	1994	1996-08-02 11:14:34

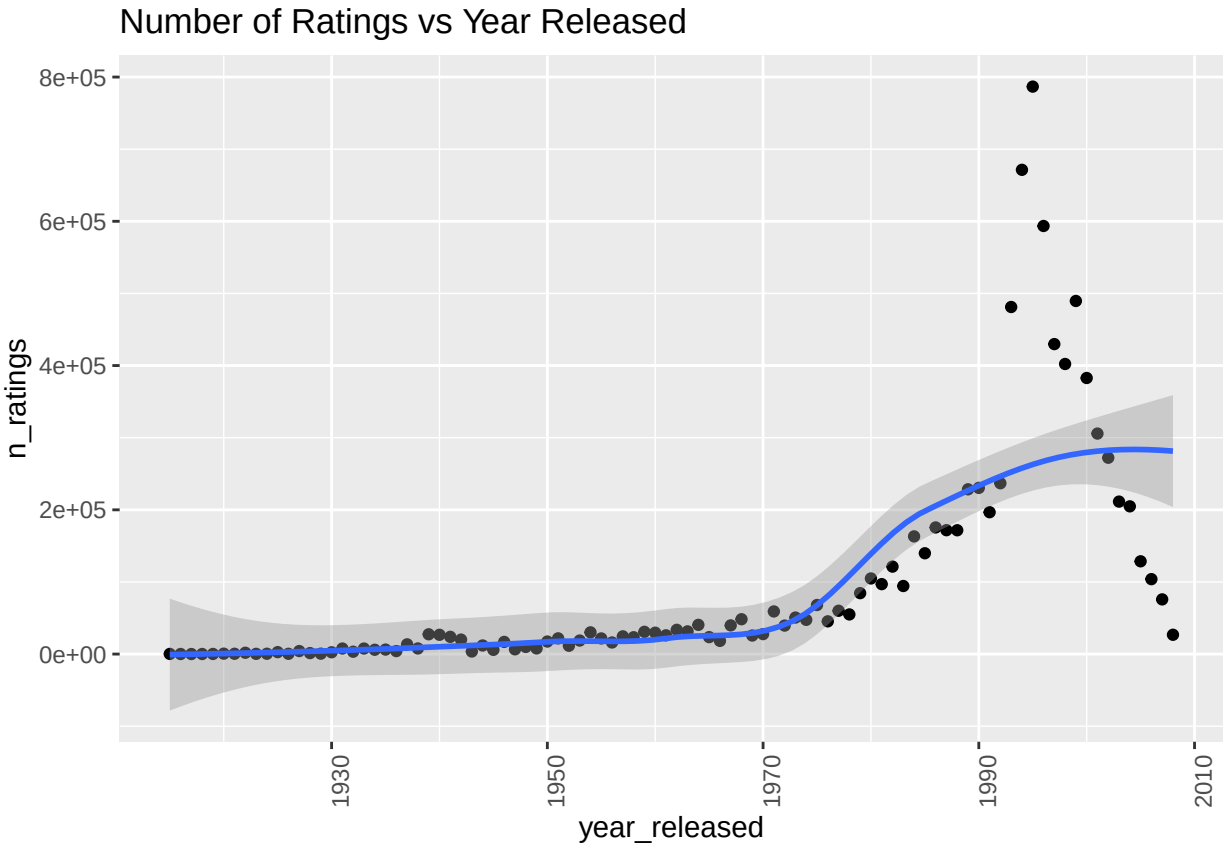
Exploring the relationship between ratings and the year the movie was released

```
#exploring relation between ratings and the year movie was released

edx%>%
  group_by(year_released)%>%
  summarise(avg_ratings=mean(rating))%>%
  ggplot(aes(year_released,avg_ratings))+
  geom_point()+
  geom_smooth(method = "loess")+
  theme(axis.text.x = element_text(angle = 90,hjust = 1))+
  ggtitle("Average ratings vs Year Released")
```



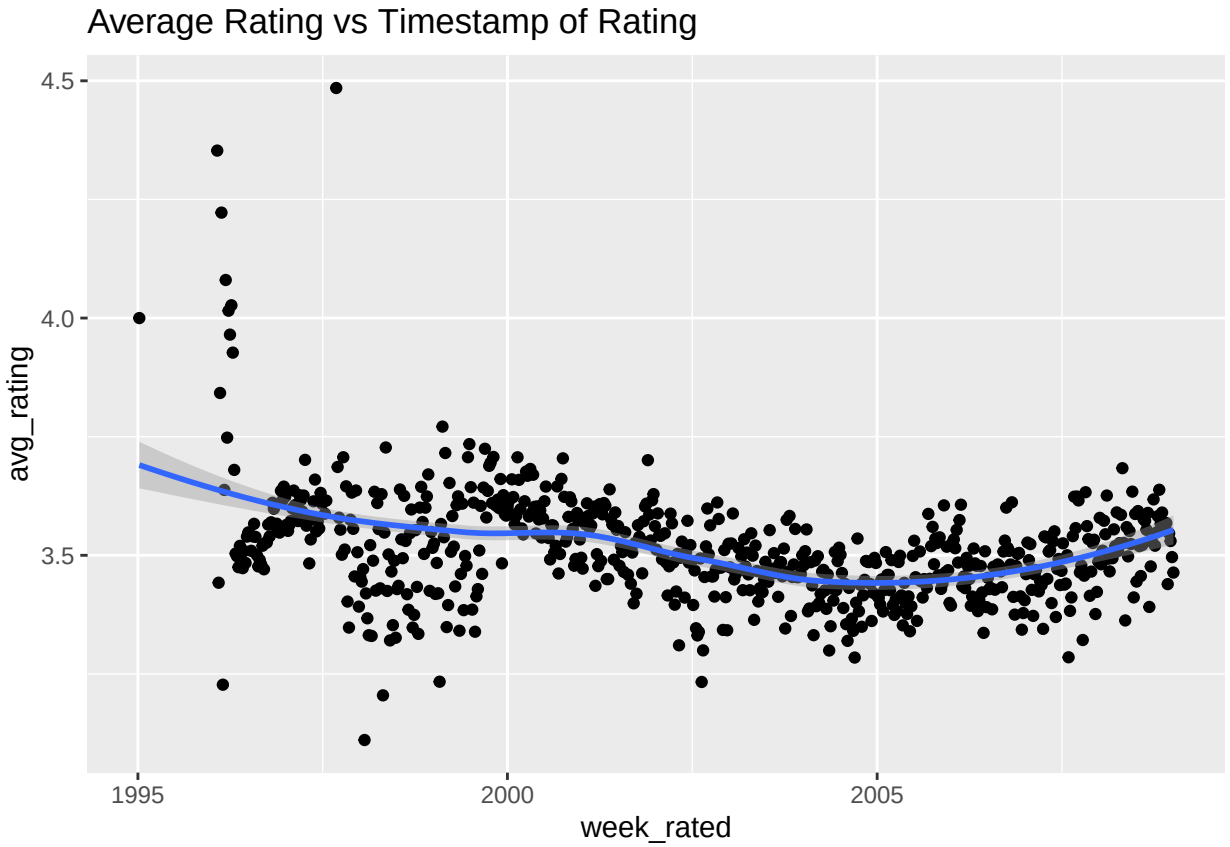

```
edx%>%
  group_by(year_released)%>%
  summarise(n_ratings=n())%>%
  ggplot(aes(year_released,n_ratings))+
  geom_point()+
  geom_smooth(method = "loess")+
  theme(axis.text.x = element_text(angle = 90,hjust = 1))+
  ggtitle("Number of Ratings vs Year Released ")
```



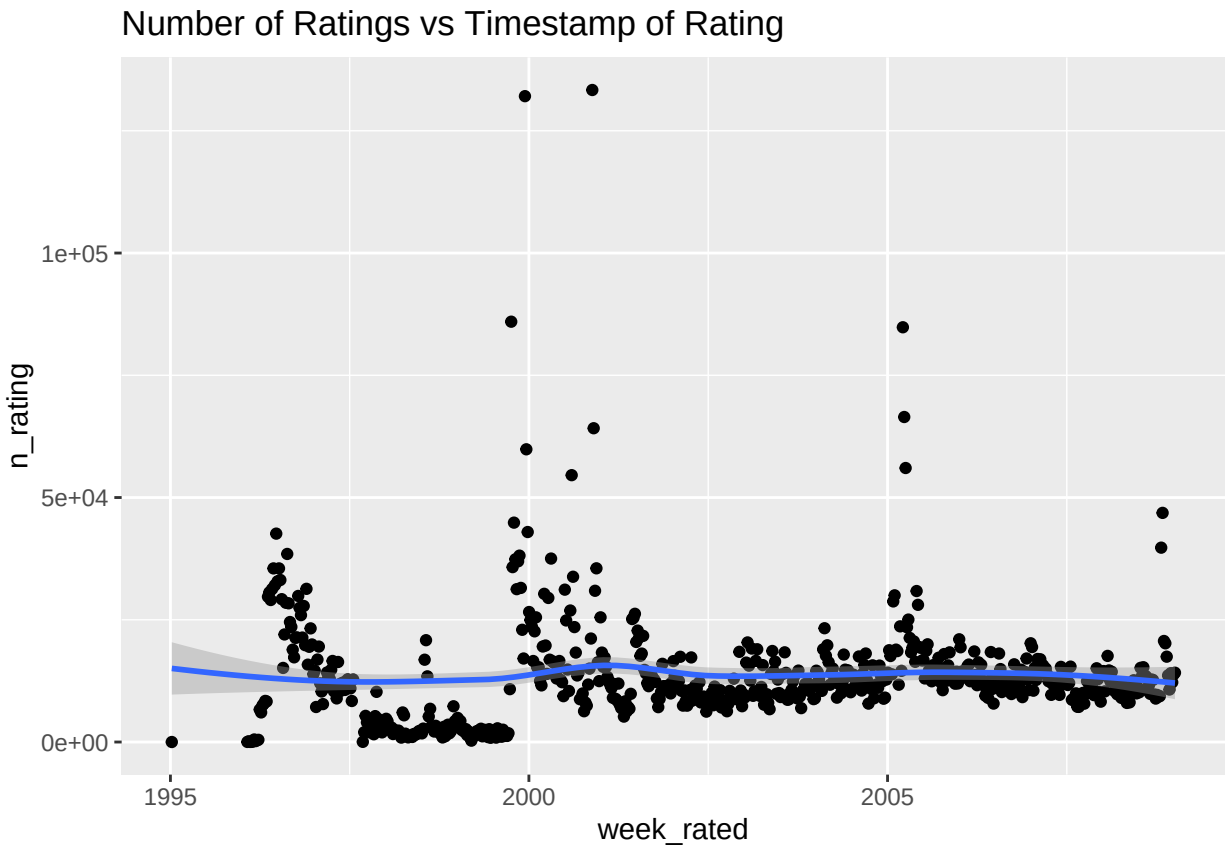
The first plot shows that there is a clear dip in ratings for movies that were released after the late 1980s. There could be a relationship between the year the movie was released and the ratings as there are two clear clusters. The second plot shows that post 1993 the number of ratings for movies shot up while also showing a rapid declining pattern post that.

Exploring the relationship between ratings and when the movie was rated

```
edx%>%
  mutate(week_rated=round_date(datetime,unit = 'week'))%>%
  group_by(week_rated)%>%
  summarise(avg_rating=mean(rating))%>%
  ggplot(aes(week_rated,avg_rating))+
  geom_point()+
  geom_smooth(method="loess")+
  ggtitle("Average Rating vs Timestamp of Rating")
```




```
edx%>%
  mutate(week_rated=round_date(datetime,unit = 'week'))%>%
  group_by(week_rated)%>%
  summarise(n_rating=n())%>%
  ggplot(aes(week_rated,n_rating))+
  geom_point()+
  geom_smooth(method="loess")+
  ggtitle("Number of Ratings vs Timestamp of Rating")
```



The relationship does not seem to be strong between ratings and the timestamp as a predictor . We can safely ignore the timestamp of rating for predicting the ratings.

Exploring the relationship between ratings and different genres.

```
unique(unlist(str_split(as.vector(unique(edx$genres)),"\\|")))
```

```
##  [1] "Comedy"             "Romance"            "Action"
##  [4] "Crime"              "Thriller"           "Drama"
##  [7] "Sci-Fi"             "Adventure"          "Children"
## [10] "Fantasy"            "War"                "Animation"
## [13] "Musical"            "Western"            "Mystery"
## [16] "Film-Noir"          "Horror"             "Documentary"
## [19] "IMAX"               "(no genres listed)"
```

```
length(unique(edx$genres))
```

```
## [1] 797
```

There are 20 unique genres and 797 unique combination of these genres(including "(no genres listed)").

```
edx%>%group_by(genres)%>%
  summarise(avg_ratings=mean(rating))%>%
  top_n(10,avg_ratings) %>%
  knitr::kable(caption = "Top 10 genres by rating")
```

```
edx%>%group_by(genres)%>%
  summarise(avg_ratings=mean(rating),n_ratings=n())%>%
  top_n(10,n_ratings)%>%
```

Table 1: Top 10 genres by rating

genres	avg_ratings
Action|Adventure|Comedy|Fantasy|Romance	4.195557
Action|Crime|Drama|IMAX	4.297068
Animation|Children|Comedy|Crime	4.275429
Animation|IMAX|Sci-Fi	4.714286
Crime|Film-Noir|Mystery	4.216803
Crime|Film-Noir|Thriller	4.210157
Crime|Mystery|Thriller	4.198981
Drama|Film-Noir|Romance	4.304115
Film-Noir|Mystery	4.239479
Film-Noir|Romance|Thriller	4.216470

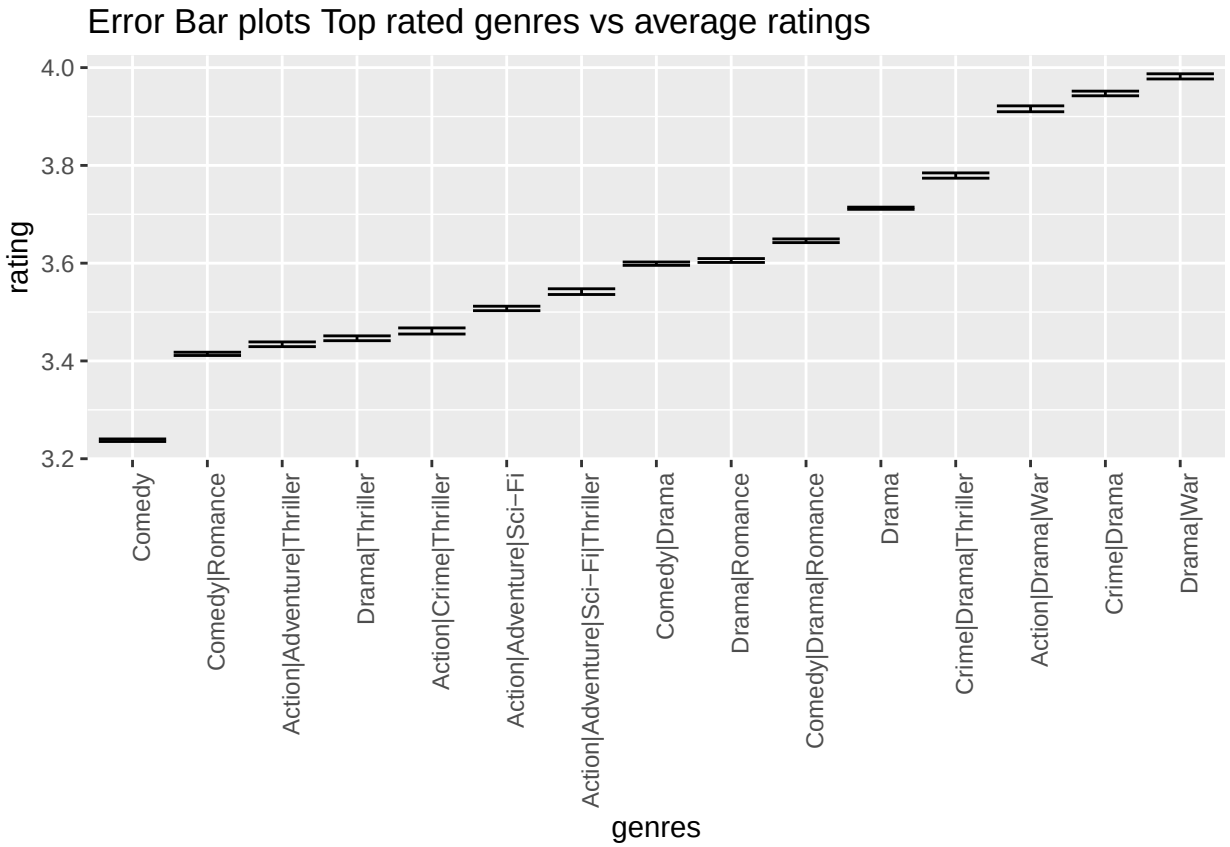
Table 2: Top 10 genres by number of ratings

genres	avg_ratings	n_ratings
Action|Adventure|Sci-Fi	3.507407	219938
Action|Adventure|Thriller	3.434101	149091
Comedy	3.237858	700889
Comedy|Drama	3.598961	323637
Comedy|Drama|Romance	3.645824	261425
Comedy|Romance	3.414486	365468
Crime|Drama	3.947135	137387
Drama	3.712364	733296
Drama|Romance	3.605471	259355
Drama|Thriller	3.446345	145373

```
  knitr::kable(caption = "Top 10 genres by number of ratings")
```

This shows that the most rated movies are not necessarily the best rated.

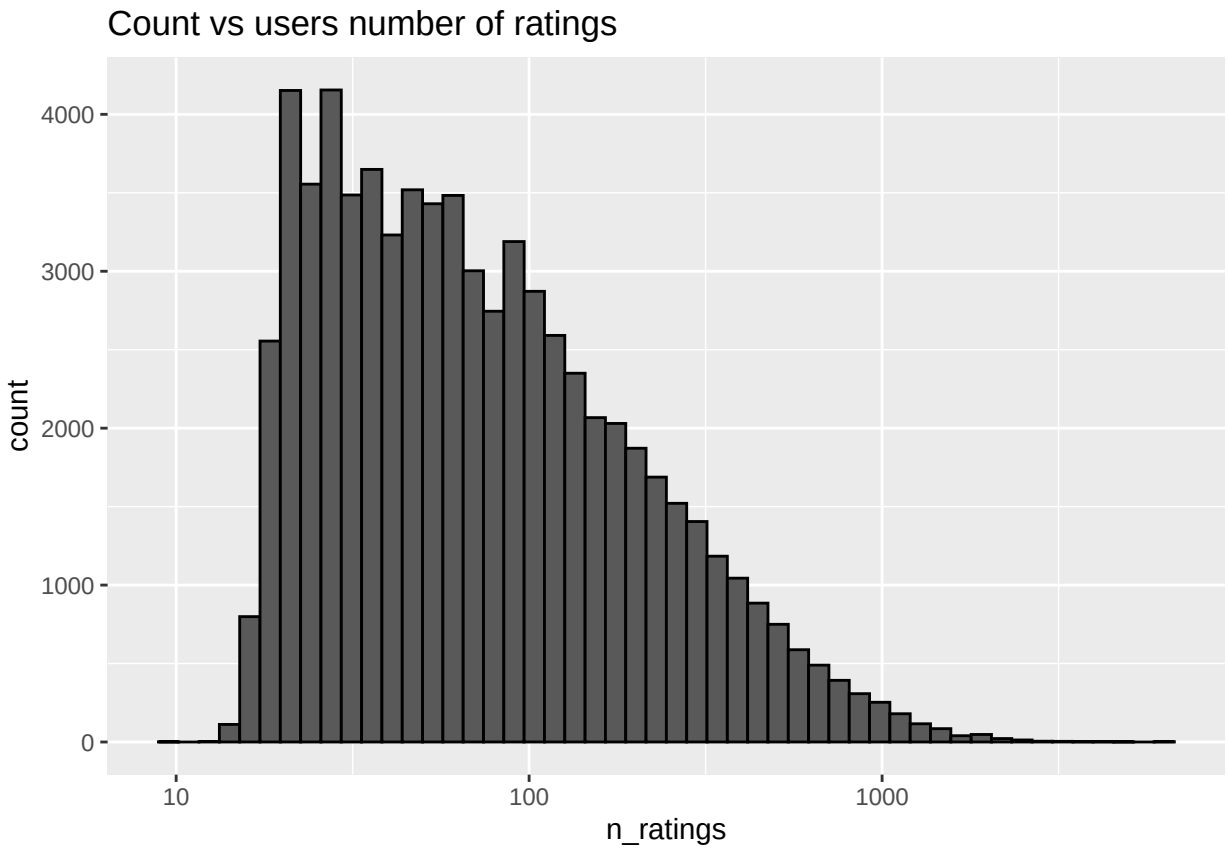
```
edx%>%group_by(genres)%>%
  summarise(avg_rating=mean(rating),
            se_rating=sd(rating)/sqrt(n()),
            n_ratings=n())%>%
  top_n(15,n_ratings)%>%mutate(genres=reorder(genres,avg_rating))%>%
  ggplot(aes(x=genres,
             y=avg_rating,
             ymin=avg_rating-(qnorm(0.975)*se_rating),
             ymax=avg_rating+(qnorm(0.975)*se_rating)))+
  geom_errorbar()+
  theme(axis.text.x = element_text(angle = 90, hjust = 1))+
  ggtitle("Error Bar plots Top rated genres vs average ratings")+ylab('rating')
```



As can be seen in this plot it is clear that there is a definite distinction between the genres. From this analysis we can conclude that genres have a relationship with ratings.

Exploring the relationship between ratings and user

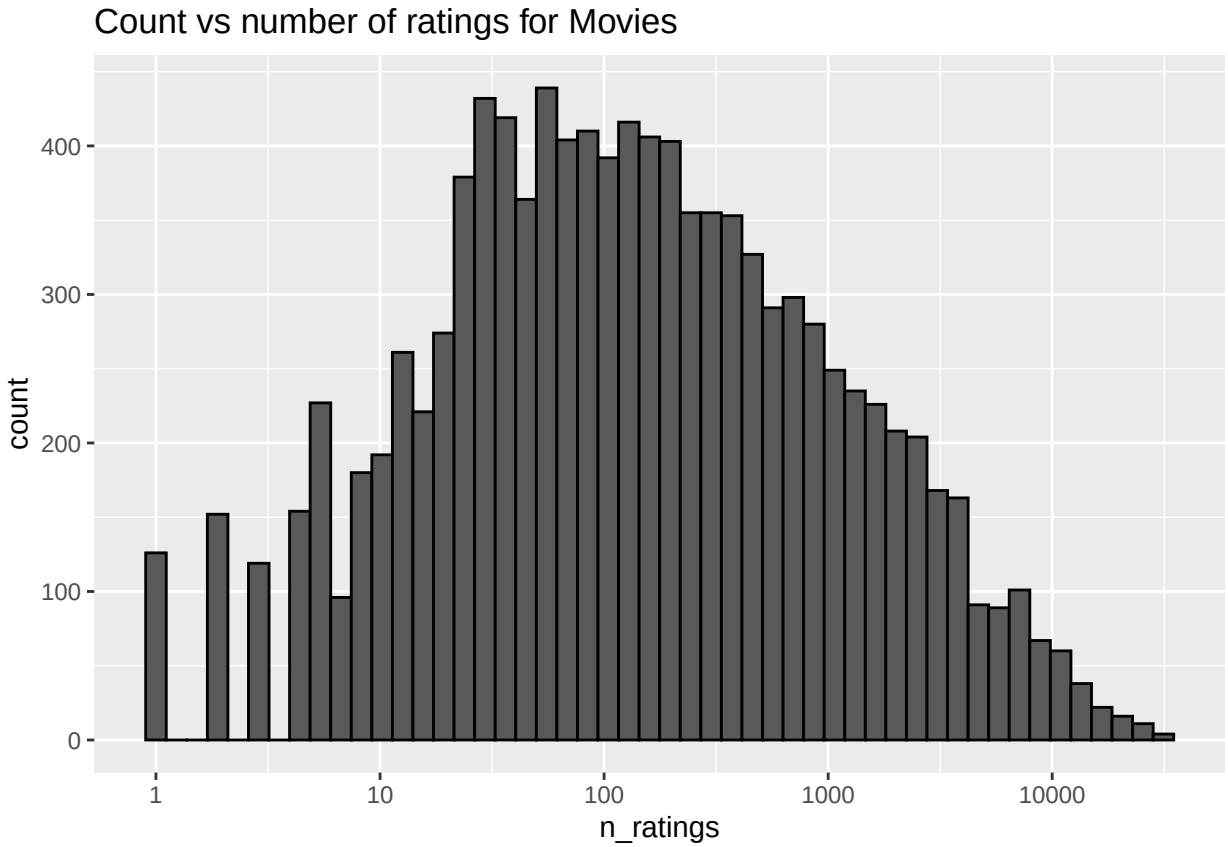
```
edx%>%
  group_by(userId)%>%
  summarise(n_ratings=n())%>%
  ggplot(aes(x=n_ratings))+
  geom_histogram(bins=50,color='black')+
  scale_x_continuous(trans = 'log10')+
  ggtitle("Count vs users number of ratings")
```



This plot shows that the number of users who rate over a 1000 movies are few in number, whereas the average user rates about 25 to 100 movies.

Exploring the relationship between ratings and movie

```
edx%>%
  group_by(movieId)%>%
  summarise(n_ratings=n())%>%
  ggplot(aes(x=n_ratings))+
  geom_histogram(bins=50,color='black')+
  scale_x_continuous(trans = 'log10')+
  ggtitle("Count vs number of ratings for Movies")
```



This plot shows that some movies are rated more often than others.

Inference from Analysis

1) Movies that were released before 1990s have a higher rating than movies that were released later.
2) Some genres are more popular than others while some have a higher rating.
3) Some Users tend to be very active in rating while others do not. This could also mean that some users tend to watch a lot of movies.

4)Some movies are popular than others and tend to have many more ratings.

The goal of the model is to incorporate these findings and to accurately predict ratings.

Building the Model

RMSE

Here we define a function to measure the performance of the model.

```
RMSE <-  function(predicted_ratings,actual_ratings){
  sqrt(mean((predicted_ratings-actual_ratings)^2))
}
```

A tibble to store all our results

```
results<-tibble()
```

Split edx to train_set and test_set

The caret package provides createDataPartition(function). Using this function we divide the edx dataset to a train set and a test set. To replicate the split the seed has been set to 1. Lastly to ensure that the test_set has the same set of movies and users the semi_join() function from the tidyverse package is used.

```
#split edx to train and test set
set.seed(1, sample.kind="Rounding")
# if using R 3.5 or earlier, use `set.seed(1)` instead
test_index<-createDataPartition(edx$rating,times=1,p=0.5,list=FALSE)
test_set<-edx[test_index,]
train_set<-edx[-test_index,]
#ensuring the test set has same userId and movieId
test_set <- test_set %>% semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")
```

simplest model

```
mu<-mean(train_set$rating)
result_simple_model<-RMSE(mu,test_set$rating)
results<-data.frame(model='Mean Only',result=result_simple_model)
```

We start the process by building the simplest possible model where we assume that all movies are rated equally and use the result as the base for our upcoming models.

Movie Effect b_i

We have seen that different movies have different level of popularity and are rated differently. Here we attempt to incorporate this effect into the model.

```
avg_ratings_movie<- train_set%>%
  group_by(movieId)%>%
  summarise(b_i=mean(rating)-mu)
predictions_movie_effect <- mu +
  test_set%>%
  left_join(avg_ratings_movie,by='movieId')%>%
  pull(b_i)
result_movie_effect<-RMSE(predictions_movie_effect,test_set$rating)
results<-rbind(results,
               data.frame(model='Movie Effect',
                          result=result_movie_effect))
```

Movie Effect + User Effect b_u

Similar to movies, users also have different biases. Consequently some users tend to rate movies higher while some rate the same movies lower. Here, we incorporate this bias into the model.

```
avg_ratings_user  <- train_set%>%
  left_join(avg_ratings_movie,by='movieId')%>%
  group_by(userId)%>%
  summarise(b_u= mean(rating-mu-b_i))
predictions_movie_user_effect <- test_set%>%
  left_join(avg_ratings_movie,by='movieId')%>%
  left_join(avg_ratings_user,by='userId')%>%
  mutate(pred=mu+b_i+b_u)%>%
  pull(pred)
```

```
result_movie_user_effect<-RMSE(predictions_movie_user_effect,test_set$rating)
results<-rbind(results,
               data.frame(model='Movie+User Effect',
                          result=result_movie_user_effect))
```

Movie Effect + User Effect +Genre Effect b_g

Our analysis also shows that some genres tend to be more popular than others. Here we incorporate this bias into the model.

```
avg_ratings_genre<-train_set %>%
  left_join(avg_ratings_movie,by='movieId')%>%
  left_join(avg_ratings_user,by='userId')%>%
  group_by(genres)%>%summarise(b_g=mean(rating-mu-b_i-b_u))
predictions_movie_user_genre_effect<- test_set%>%
  left_join(avg_ratings_movie,by='movieId')%>%
  left_join(avg_ratings_user,by='userId')%>%
  left_join(avg_ratings_genre,by='genres')%>%
  mutate(pred=mu+b_i+b_u+b_g)%>%pull(pred)
result_movie_user_genre_effect<-RMSE(predictions_movie_user_genre_effect,
                                     test_set$rating)
results <- rbind(results,
                 data.frame(model='Movie+User+Genre Effect',
                            result=result_movie_user_genre_effect))
```

Movie Effect + User Effect +Genre Effect + year realeased effect b_g

Finally there was a clear distinction in ratings when it came to the year when the movies were released, especially in case of movies that were released before the 1990s. This snippet of code incorporates this bias in the model.

```
avg_ratings_year<-train_set%>%
  left_join(avg_ratings_movie,by='movieId')%>%
  left_join(avg_ratings_user,by='userId')%>%
  left_join(avg_ratings_genre,by='genres')%>%
  group_by(year_released)%>%
  summarise(b_t=mean(rating-mu-b_i-b_u-b_g))
predictions_movie_user_genre_year_effect<-test_set%>%
  left_join(avg_ratings_movie,by='movieId')%>%
  left_join(avg_ratings_user,by='userId')%>%
  left_join(avg_ratings_genre,by='genres')%>%
  left_join(avg_ratings_year,by='year_released')%>%
  mutate(pred=mu+b_i+b_u+b_g+b_t)%>%pull(pred)
result_movie_user_genre_year_effect<-RMSE(predictions_movie_user_genre_year_effect,
                                          test_set$rating)
results<-rbind(results,
               data.frame(model='Movie+User+genre+year_released Effect',
                          result=result_movie_user_genre_year_effect))
```

Regularisation

```
results%>%knitr::kable()
```

model	result
Mean Only	1.0600247
Movie Effect	0.9440085
Movie+User Effect	0.8697242
Movie+User+Genre Effect	0.8693637
Movie+User+genre+year_released Effect	0.8691679

As can be seen from the results above, for each additional effect the RMSE has come down. To improve upon this result, we regularise the dataset. By regularisation we will penalise large estimates for movies with few ratings.

```
lambdas <- seq(0, 10, 0.5)
regularisation_function <- function(lambda){
  mu <- mean(train_set$rating)
  avg_ratings_movie<-train_set%>%
    group_by(movieId)%>%
    summarise(b_i=sum(rating-mu)/(n()+lambda))
  avg_ratings_user<-train_set%>%
    left_join(avg_ratings_movie,by='movieId')%>%
    group_by(userId)%>%summarise(b_u= sum(rating-mu-b_i)/(n()+lambda))
  avg_ratings_genre<-train_set %>%
    left_join(avg_ratings_movie,by='movieId')%>%
    left_join(avg_ratings_user,by='userId')%>%
    group_by(genres)%>%summarise(b_g=sum(rating-mu-b_i-b_u)/(n()+lambda))
  avg_ratings_year<-train_set%>%left_join(avg_ratings_movie,by='movieId')%>%
    left_join(avg_ratings_user,by='userId')%>%
    left_join(avg_ratings_genre,by='genres')%>%
    group_by(year_released)%>%
    summarise(b_t=sum(rating-mu-b_i-b_u-b_g)/(n()+lambda))
  predictions<-test_set%>%
    left_join(avg_ratings_movie,by='movieId')%>%
    left_join(avg_ratings_user,by='userId')%>%
    left_join(avg_ratings_genre,by='genres')%>%
    left_join(avg_ratings_year,by='year_released')%>%
    mutate(pred=mu+b_i+b_u+b_g+b_t)%>%pull(pred)
  RMSE(predictions,test_set$rating)
}
rmses_regularised <- map_dbl(lambdas, regularisation_function)
```

Choosing Lambda (tuning parameter)

```
best_lambda<-lambdas[which.min(rmses_regularised)]
results<- rbind(results,
                data.frame(model='Movie+User+genre+year_released Effect with Regularisation',
                           result=min(rmses_regularised)))
```

Training on the edx dataset

```
results%>% knitr::kable()
```

model	result
Mean Only	1.0600247
Movie Effect	0.9440085
Movie+User Effect	0.8697242
Movie+User+Genre Effect	0.8693637
Movie+User+genre+year_released Effect	0.8691679
Movie+User+genre+year_released Effect with Regularisation	0.8676160

Based on this we choose 'Movie+User+genre+year_released Effect with Regularisation' as the best model and train on the edx dataset.

```
mu<-mean(edx$rating)
avg_ratings_movie<-edx%>%
  group_by(movieId)%>%
  summarise(b_i=sum(rating-mu)/(n()+best_lambda))
avg_ratings_user<-edx%>%
  left_join(avg_ratings_movie,by='movieId')%>%
  group_by(userId)%>%
  summarise(b_u= sum(rating-mu-b_i)/(n()+best_lambda))
avg_ratings_genre<-edx %>%
  left_join(avg_ratings_movie,by='movieId')%>%
  left_join(avg_ratings_user,by='userId')%>%
  group_by(genres)%>%
  summarise(b_g=sum(rating-mu-b_i-b_u)/(n()+best_lambda))
avg_ratings_year<-edx%>%
  left_join(avg_ratings_movie,by='movieId')%>%
  left_join(avg_ratings_user,by='userId')%>%
  left_join(avg_ratings_genre,by='genres')%>%
  group_by(year_released)%>%
  summarise(b_t=sum(rating-mu-b_i-b_u-b_g)/(n()+best_lambda))
```

Result

```
validation<-validation%>%
  mutate(year_released=year(as.Date(str_match(title,regex_pattern)[,2],format="%Y")))
predictions<-validation%>%
  left_join(avg_ratings_movie,by='movieId')%>%
  left_join(avg_ratings_user,by='userId')%>%
  left_join(avg_ratings_genre,by='genres')%>%
  left_join(avg_ratings_year,by='year_released')%>%
  mutate(pred=mu+b_i+b_u+b_g+b_t)%>%pull(pred)
final_result<-RMSE(predictions,validation$rating)
```

```
print(c("RMSE_final <- ",final_result))
```

```
## [1] "RMSE_final <- "    "0.864292940221519"
```

The best model has resulted in a significant improvement to our RMSE and our final RMSE is 0.86429.

Conclusion

In conclusion, the results signify that the model can be improved by incorporating features that are significant to predicting ratings. The more number of significant features or predictors incorporated in the model the better the prediction. This also entails that the problem is complex and there can be more predictors that can be incorporated. For example, user demographics such as age, sex and geography can play an import role in the prediction process and could be included to predict the ratings more accurately.Regularisation played

a significant role in optimizing the RMSE with respect to the baseline model. Another important point is that when deciding on the best model a smaller subset of the edx data was used and the RMSE was higher. This shows that the larger the dataset for training the better the predictions.