Movielens Project

KST

08-Jun-2025

Contents

Acknowledgements

I would like to gratefully thank Professor Rafael A Irizzary , The Teaching & Support Staff of the Department of Biostatistics, The Harvard University and the edX platform for providing students from all over world the opportunity to learn the fundamentals of Data Science at their own pace and convenience. The availability of such facilities and the accommodating structure of the course is valuable beyond words to those who are disadvantaged for time and who otherwise cannot afford to attend classroom sessions.

I would like to thank my curious, inquisitive fellow learners whose questions, answers and comments spared me the time, the effort and the blushes.

I would also like to thank the authors, collaborators and contributors of the articles listed below, for the guidance provided by these articles in the execution of the project and the preparation of this report.

https://mirror.niser.ac.in/ctan/macros/latex/contrib/listings/listings.pdf

https://bookdown.org/yihui/rmarkdown-cookbook/

https://berkeley-scf.github.io/tutorial-parallelization/parallel-R

https://stackoverflow.com/questions/25664007/reorder-bars-in-geom-bar-ggplot2-by-value

https://stackoverflow.com/questions/46287086/how-to-center-ggplot-plot-title

https://stackoverflow.com/questions/38708529/how-to-use-str-split-in-r#38708610

https://stringr.tidyverse.org/reference/str_detect.html

https://dplyr.tidyverse.org/reference/rows.html

https://www.statology.org/ggplot2-legend-multiple-rows/

https://statisticsglobe.com/move-position-of-ggplot2-legend-in-r

https://r-graphics.org/recipe-bar-graph-labels

https://www.r-tutor.com/r-introduction/vector/numeric-index-vector

https://stackoverflow.com/questions/48499400/suppress-automatic-output-to-console-in-r

https://stackoverflow.com/questions/2501895/how-to-suppress-output?noredirect=1&lq=1

https://stackoverflow.com/questions/62946906/fitting-bartmachine-in-caret-getting-argument-of-length-zero-incorrect-number-of

https://machinelearningmastery.com/tune-machine-learning-algorithms-in-r/

https://topepo.github.io/caret/index.html

https://majkamichal.github.io/naivebayes/reference/naive_bayes.html

https://en.wikipedia.org/wiki/Linear_regression

*Note: All referenced URLs in this report were current and the content accessible at the time of writing this report.

1 Introduction

Recommendation Systems are widely used in many online retail businesses like Streaming media and E-commerce. Almost anybody who has ever visited an E-commerce site, subscribed to a Streaming media service or downloaded an Application on their Mobile phones from an "App store" has seen such Recommendation Systems in action. Their main purpose being to provide suggestions to customers about, and help them quickly find, products or services that they may be interested in.

Recommendation Systems take many forms and usually try to gain the customer's attention with auditory or visual cues & prompts like "Products you may like", "You may also be interested in" , "Users like you also bought" and many other appealing phrases. Recommendation Systems also try to help the customer make a choice by providing "Ratings" for the products and services that they recommend. These ratings are usually a quantitative representation of the product or service's popularity and could be based on a number of stars, a bar whose length indicates the rating, different shades, emojis or more unique and imaginative representations. More often than not, a numerical value accompanies other representations for quick comprehension and ease of comparison.

Some of the requirements that influence the design of Recommendation Systems are

1. A huge number of customers.
2. A huge number of products or services available for the customers to choose from.
3. Products or services delivered to a Local, National, Regional Confederation or Global customer base.
4. A large diversity in the customer types and buying behaviour.
5. A large number of customers accessing on-line content and services via Mobile devices, often with limited screen space.
6. A limited amount of time for any Recommendation to get a customer's attention.
7. A very simple and intuitive User Interface that provides customers with the most important information required to gain their interest and/or provide feedback. Any metrics used to collect and/or make a recommendation must be easily comprehensible to the audience.
8. An increased emphasis on Personal Data Protection and User Privacy which limits the amount of information that businesses can collect, retain and share about customers.
9. Availability of many algorithms and methods that are often used in conjunction with each other to build a system that is specific to the business and the customers that it serves. Any algorithms and methods used must be efficient in terms of cost, time and resources and must be fit for purpose.
10. Many competing Service Providers seeking to acquire and retain customers, with "Personalised Content" or "Personalised Recommendations" often being seen as a key differentiator.

One of the most important needs of any business is the ability to increase the volume of business from each Returning Customer. It is a well known fact of business that it is easier to get additional revenue from Returning Customers than it is from New Signings. Due to the huge number of products available, being able to predict what customers are likely to buy and offering it to them on the limited screen space and screen time available can help drive additional revenue and profits. Recommendation Systems are built to serve this need by gauging and reacting to customer choices and buying behaviour over a period of time.

Most modern Recommendation Systems are extremely complex and multi-tiered. Some references to get started with understanding them are:-

The "Recommendation Systems" chapter from the book, Mining of Massive Datasets.

The book is available on-line at :-

http://www.mmds.org/

A direct link to the relevant chapter is available at :-

http://infolab.stanford.edu/~ullman/mmds/ch9.pdf

The sections on Content-Based Recommendations, Collaborative Filtering and the Summary should be particularly useful for developing initial insight into the theory behind how modern large scale Recommendation Systems are built and work.

Please see the following short papers about how some extremely large corporations have implemented Recommendation Systems for their businesses. As one can imagine, they are a lot more comprehensive, detailed and advanced.

https://assets.amazon.science/76/9e/7eac89c14a838746e91dde0a5e9f/two-decades-of-recommender-systems-at-amazon.pdf

https://dl.acm.org/doi/pdf/10.1145/2843948

https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/45530.pdf

As part of this Project, we seek to build a small rudimentary component of the overall Recommendation System by developing some algorithms to generate predictions for "Unknown" ratings based on our analysis of "Known" ratings (or Observations) from users. We then compare our predictions against actual values to evaluate how accurate these predictions are. We explore some visualisations that help us make sense of the data and the predictions. We look at how insights that are not clearly evident can be extracted by extending or expanding the data.

We will also discuss some practical recommendations that might help the Readers save time and effort while avoiding some of the pitfalls that might appear along the way.

While all of the requirements for a Recommendation System are equally important, some have greater bearing on our current project for the choices of methods and analyses and the predictions that we make. We will see in due course that the Requirements that impact our Project the most are 4, 8 and 9.

1.1 Project Background

As stated earlier, The goal of the current Project is to build some algorithms to be used in a Movie Recommendation System to predict a user's preference for any particular movie. The gauge used to study and model this preference is the user's "rating" for that movie. As common sense would indicate, this rating is influenced by the user 's own personal choices and preferences over time. What we will try to do through the course of this Project is try to model this behaviour mathematically.

All Data Analysis is performed using the R Programming Language. Please learn more about R at :-

https://www.r-project.org/about.html

The dataset to be used as reference, is the "MovieLens 10M Dataset". It is made available freely for study and research, thanks to the courtesy of GroupLens.

More about them here:- https://www.grouplens.org

The dataset is available at https://grouplens.org/datasets/movielens/10m/

It contains 10000054 ratings for 10681 movies by 71567 users of the online Movie Recommender Service "MovieLens".

Users were selected at random for inclusion. All users selected had rated at least 20 movies. No demographic information is included. Each user is represented by an Id, and no other information is provided.

The data in the set is split between 3 files.

1. movies.dat - Each line of this file represents one movie, and has the following format:

   `MovieID::Title::Genres`

2. ratings.dat - Each line of this file represents one rating of one movie by one user, and has the following format:

   `UserID::MovieID::Rating::Timestamp .`

   The lines within this file are ordered first by UserID, then, within user, by MovieID.

3. tags.dat - Each line of this file represents one tag applied to one movie by one user, and has the following format:

   `UserID::MovieID::Tag::Timestamp`

Individual fields within each row are separated by double colons (::).

For our Project, we will only use the data from the movies.dat and ratings.dat files.

The fields in the 2 files are defined as follows:-

1. UserID - Anonymised numeric Identifier of variable length.
2. MovieID - A numeric Identifier that has variable length.
3. Rating - Made on a 5-star scale from 0 to 5 in half-star increments
4. Timestamp - Time when Rating was provided. Represented in seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970.
5. Title - Name of the movie including phonetic representations if any. Titles are synchronised with the title in the "Internet Movie Database" (IMDb) available at https://www.imdb.com. Titles are manually entered so discrepancies are possible.
6. Genres - Genres are a pipe-separated (|) list, There could zero or more Genre attached to a Movie and are selected from the following list:

Table 1: **List of Genre**

Action	Documentary	Mystery
Adventure	Drama	Romance
Animation	Fantasy	Sci-Fi
Children's	Film-Noir	Thriller
Comedy	Horror	War
Crime	Musical	Western

As per the guidance and the relevant setup scripts provided by the Project's Administrators, data from the ratings.dat and movies.dat files is combined together into a single dataset organised in the form of a matrix. Each row in the matrix represents a single rating and the values in the fields listed above are organised as columns within each row. Each row constitutes a unique Observation. There is only one rating per user per movie and duplicates are not allowed.

The resulting dataset is split or partitioned into "Training" and "Testing" sets with about 90% of the Observations allocated to the Training set and 10% of the Observations allocated to the Testing set.

The Training set forms the "Known" set of Observations and is to be used for building or "Training" our Models for prediction. The Testing set forms the "Unknown" set of Observations and is to be used for evaluation of the final Model(s) and is not to be used for "Training" or shortlisting Models. Users and movies need to be available on both the Training and Testing sets so that any prediction made can be checked against actual ratings provided. Because of the partitioning scheme used, the number of unique users and movies reduces slightly to 69878 and 10677 respectively.

These two Training and Testing sets together form the authoritative source of truth for the Project and facilitate consistency in the data to be used for analysis and evaluation of the final results. We will progressively elaborate more about them as we go along.

1.1.1 Dataset partitioning into Training and Testing sets

As stated earlier, The Project's Administrators have already provided guidance and the relevant setup scripts on how the "Movielens 10M Dataset" needs to be partitioned between Training and Testing sets. The guidance also includes the names to be used for the fields which is slightly different from what is used by MovieLens and uses lower camel case ("camelCase") instead.

Only the Observations in the Training set can be used for building or "Training" the Recommendation System. The Observations in the Testing set need to be used for verifying the RMSE between the actual ratings and our predictions.

Once the initial setup scripts are run, we are left with two datasets with "userId", "movieId,"rating", "title", "genres" and "timestamp" used as the field names.

1. "edx" - Dataset to be used for Modelling and Training our Recommendation System. It consists of about 90% of the ratings and forms the "Known" set of Observations. We have full visibility into it.
2. "final_holdout_test" - Dataset that consists of about 10% of the ratings and forms the "Unknown" set of Observations. In this dataset, our visibility will be limited to the "userId", "movieId", "title", "genres" and "timestamp" fields and we will not be allowed to access the "rating" field for anything other than the final RMSE verification.

```
############################################################
# BEGIN MOVIELENS PROJECT
############################################################

############################################################
# Reproduced AS-IS from the Course Guidelines
############################################################
# Create edx and final_holdout_test sets
############################################################

# Note: this process could take a couple of minutes

if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.
    r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-
    project.org")

library(tidyverse)
library(caret)

# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip

options(timeout = 120)

dl <- "ml-10M100K.zip"
if(!file.exists(dl))
  download.file("https://files.grouplens.org/datasets/movielens/ml-10m.zip",
      dl)

ratings_file <- "ml-10M100K/ratings.dat"
if(!file.exists(ratings_file))
  unzip(dl, ratings_file)
```

```
movies_file <- "ml-10M100K/movies.dat"
if(!file.exists(movies_file))
  unzip(dl, movies_file)

ratings <- as.data.frame(str_split(read_lines(ratings_file), fixed("::"),
    simplify = TRUE),
                         stringsAsFactors = FALSE)
colnames(ratings) <- c("userId", "movieId", "rating", "timestamp")
ratings <- ratings %>%
  mutate(userId = as.integer(userId),
         movieId = as.integer(movieId),
         rating = as.numeric(rating),
         timestamp = as.integer(timestamp))

movies <- as.data.frame(str_split(read_lines(movies_file), fixed("::"),
    simplify = TRUE),
                        stringsAsFactors = FALSE)
colnames(movies) <- c("movieId", "title", "genres")
movies <- movies %>%
  mutate(movieId = as.integer(movieId))

movielens <- left_join(ratings, movies, by = "movieId")

# Final hold-out test set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
# set.seed(1) # if using R 3.5 or earlier
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1,
    list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

# Make sure userId and movieId in final hold-out test set are also in edx set
final_holdout_test <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Add rows removed from final hold-out test set back into edx set
removed <- anti_join(temp, final_holdout_test)
edx <- rbind(edx, removed)

rm(dl, ratings, movies, test_index, temp, movielens, removed)

#############################################################
# Reproduction AS-IS from the Course Guidelines Ends
#############################################################

# Remove Variables used to hold filenames as they are not required anymore
rm(movies_file, ratings_file)
```

1.1.2 Project Success Criteria

The objective criteria for the success of the Project is the Root Mean Square Error (RMSE) which is a measure of the difference between the actual ratings provided by a user against the predictions made by our Recommendation System.

The RMSE is defined as:-

RMSE = $\sqrt{\frac{1}{N}\sum_{i,j}^{N}(y_{i,j} - \hat{y}_{i,j})^2}$

$y_{i,j}$ is used to denote the actual rating

$\hat{y}_{i,j}$ is used to denote the prediction

i identifies the user and j identifies the movie

N is the number of user/movie combinations for which we make predictions using the Recommendation System.

RMSE Evaluation is done for predictions made against the "final_holdout_test" set.

An RMSE value < 0.86490 is considered a job well done and qualifies for the best grading. Any RMSE value $>= 0.90000$ is considered a poor effort and would only qualify for the lowest grade. The Target of this project will be to achieve an RMSE < 0.86490. The lower we can get, the better.

1.2 Project Constraints

Though not imposed by the Project's Administrators as such, Some constraints are practical in nature and are imposed based on the computing resources and time likely available to students in general.

1. Computing Resources - The Code should run on any modern Laptop owned by students. These days many students tend to own Ultrabooks or Macs which are more likely to be Memory constrained than Workstation class Laptops. Memory is a bigger constraint than Processing Power for manipulating the datasets referenced in the Project.
2. Time - Any model should not take an inordinate amount of time to train particularly if the students need to use their computers for other work too.

Using Cloud Services could possibly alleviate some of these constraints. However, Cloud Services that provide the levels of Memory required are likely to be expensive and not available under any "Free Tier" or "Student" offers made to individuals.

For some Students, access to Public Cloud Services is hindered by Geographical and/or Network Connectivity Restrictions, so it is better for them if the Code can be run locally.

The Project does its best to work with any such practical constraints. The Code has been revised and rewritten to accommodate the same.

Students who have access to their University or College's Cloud Computing resources or those who can procure them from Public Cloud Providers are not bound by such constraints and can enjoy the benefits provided by Cloud Services. Readers are welcome to use the Computing Resources of their choice.

We must also note that as datasets become larger and larger in actual practice, optimal utilisation of Computing Resources and Time is vital and such frugality is not necessarily misplaced.

1.3 Accompanying Text Book & Further References

For more details about the logic used and explanation of some of the Code, Readers are referred to the relevant chapter of the Text Book for the Course available online at :-

https://rafalab.dfci.harvard.edu/dsbook-part-2/highdim/regularization.html

The Text Book also has several other sections that are very useful to understand Recommendation Systems. The entire chapter about "High dimensional data" is a suggested read.

For students interested in learning more about Data Analysis using R, Reading the Text Book and it's sister publication in their entirety is highly recommended.

Another excellent suggestion is "An Introduction to Statistical Learning - with Applications in R". The book is available at no cost thanks to the magnanimity of the Authors and the Publishers at

https://www.statlearning.com/

2 Analysis

As we begin the examination of our "edx" dataset, we can see that we have 5 fields "userId", "movieId", "timestamp", "title" and "genres" that can be used as predictors while we need to generate predictions for the "rating" field. The "rating" field is also available to us for building and training our models.

Delving deeper, we can see some obvious but important features that can guide us further in our analysis.

1. User ratings are a direct reflection of user preferences or user behaviour. They can be seen as being primary in nature. They can be extracted by combining the "userId" & "rating" fields and used to represent the behaviour of individual users or subgroups of users.
2. Movie ratings are the grouping of user preferences or user behaviour within the context of each movie. They can be extracted by combining the "movieId" & "rating" fields and used to represent individual movies or subgroups of movies. They can be seen as being secondary in nature.
3. User ratings and Movie ratings are the two most important groups in the data.
    1. The User ratings influence the Movie ratings in complex ways.
    2. They cannot be considered as being orthogonal to each other nor can they be considered as converging into each other due to the natural diversity that they both embody.
4. Algorithms that take Users, Movies and their interaction into account are likely to provide the best Predictions
5. The "genres" field is likely to code some user preferences but it cannot be used as-is and will need further processing.
6. The "timestamp" field can possibly help fine tune our predictions by providing information about rating trends over time.
7. The "genres" and "timestamp" data can be organised and analysed in the context of the User ratings and Movie ratings groups.
8. For optimisation, We can look at techniques that optimise the predictions for
    1. Groups of User ratings
    2. Groups of Movie ratings
    3. Combination of the two

*Note: In this Project, we will not consider the "title" field for prediction.

To start with, Let us look at reusing some models of the Recommendation System developed already in the Course earlier and the accompanying Text Book. We will call these models out explicitly. We will use the Results obtained to establish a baseline for further incremental improvement.

2.1 Libraries required for Analysis

We will begin by loading the Libraries required for our Analysis. Some of them would have already been loaded during the creation of the initial datasets. However, they are all listed in one place for convenience.

```
# Let us load the libraries
library(tidyverse)
library(caret)
library(ggthemes)
library(gridExtra)
library(stringr)
library(kableExtra)
```

2.2 Understanding the "edx" dataset

Let us try to understand the "edx" dataset that we are working with in more detail. As a quick reminder, the "edx" dataset consists of about 90% of the Observations in the "Movielens 10M Dataset" and into which we have full visibility.

Let us get the total number of entries in the dataset

```
## [1] The Total Number of Entries in the edx Dataset is :
## [2] 9000055
```

Let us get some insight into what the data is structured like, by listing the first 10 entries.

Table 2: First 10 entries of edx

	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
8	1	356	5	838983653	Forrest Gump (1994)	Comedy|Drama|Romance|War
9	1	362	5	838984885	Jungle Book, The (1994)	Adventure|Children|Romance
10	1	364	5	838983707	Lion King, The (1994)	Adventure|Animation|Children|Drama|Musical
11	1	370	5	838984596	Naked Gun 33 1/3: The Final Insult (1994)	Action|Comedy

We can see the data organised by userId, movieId, rating, timestamp, title and genres. We have already seen the explanation of these fields in the Project Background section.

Let us get the number of Unique Users and Movies

```
## [1] The Number of Unique Users are :  69878
```

```
## [1] The Number of Unique Movies are :  10677
```

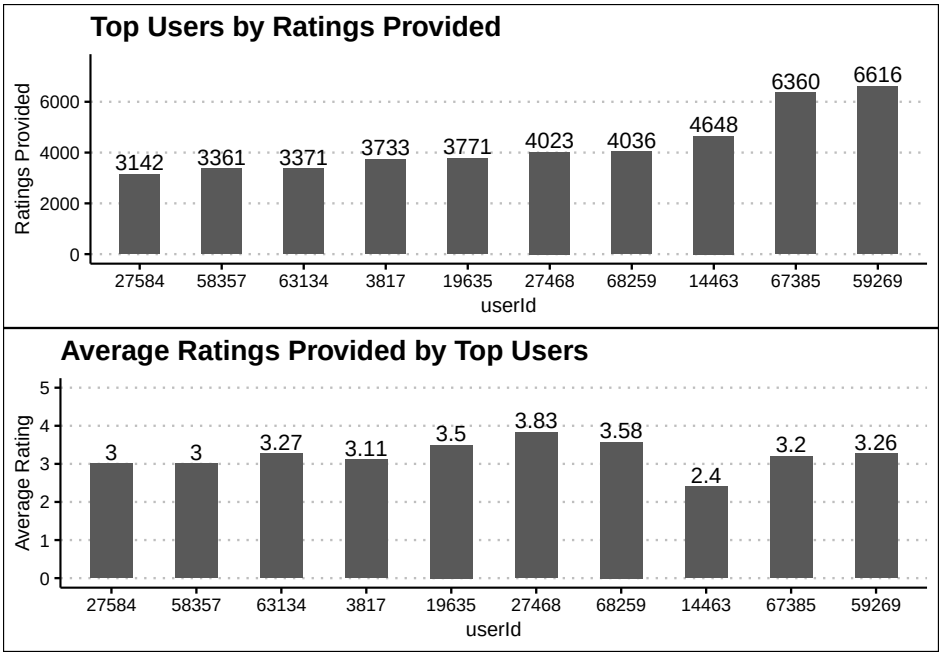
The First and Second Requirements that we had for Recommendation Systems was related to the number of Customers and Products.

1. A huge number of Consumers or Customers.
2. A huge number of Products or Services available for Customers to choose from.

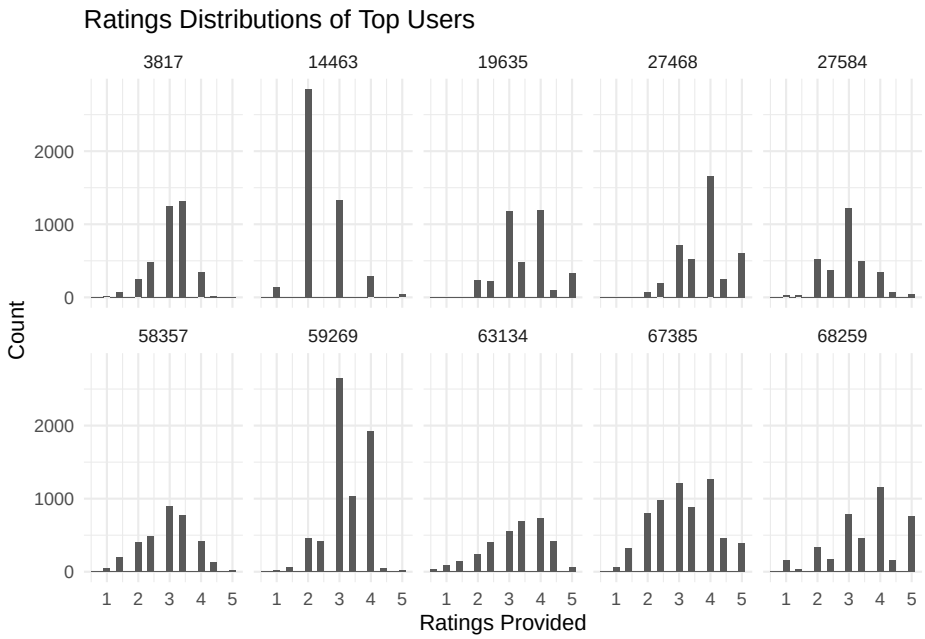
Our "edx" dataset contain about 90% of the observations from the larger "MovieLens 10M Dataset". So by extension we can see that the "MovieLens 10M Dataset" is not very large but is still quite substantial for a beginner's project and helps provide insight into the complexity of building a recommendation system for large datasets.

Let us get the Top 10 Users in terms of Ratings Provided.

We will combine it with the Average Rating they provide to understand how they rate movies.

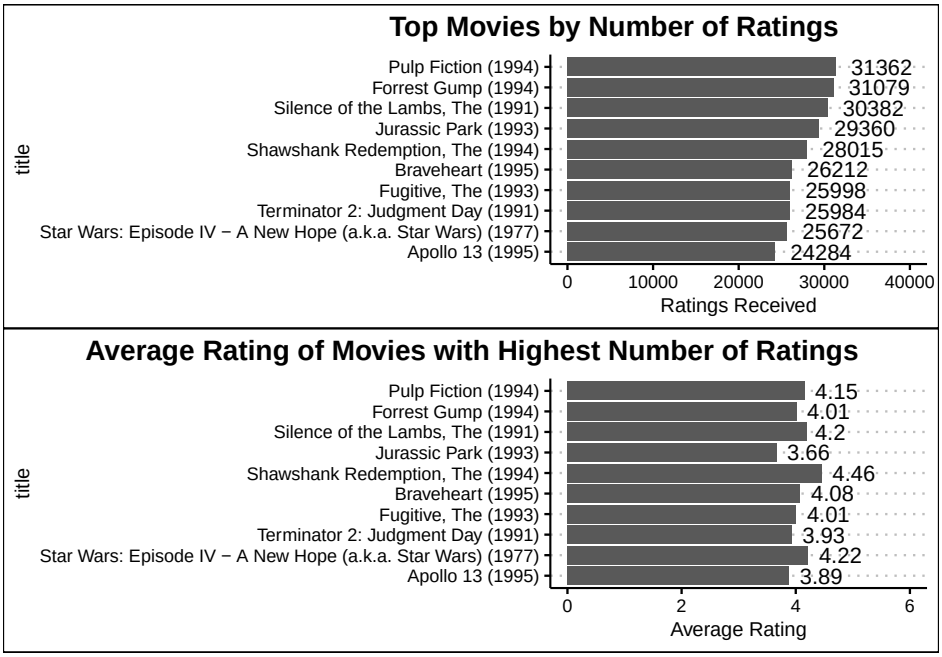


We can see a fair amount of variability in how users rate movies on an average. Let us also visualise the distribution of ratings from these users across the rating range.

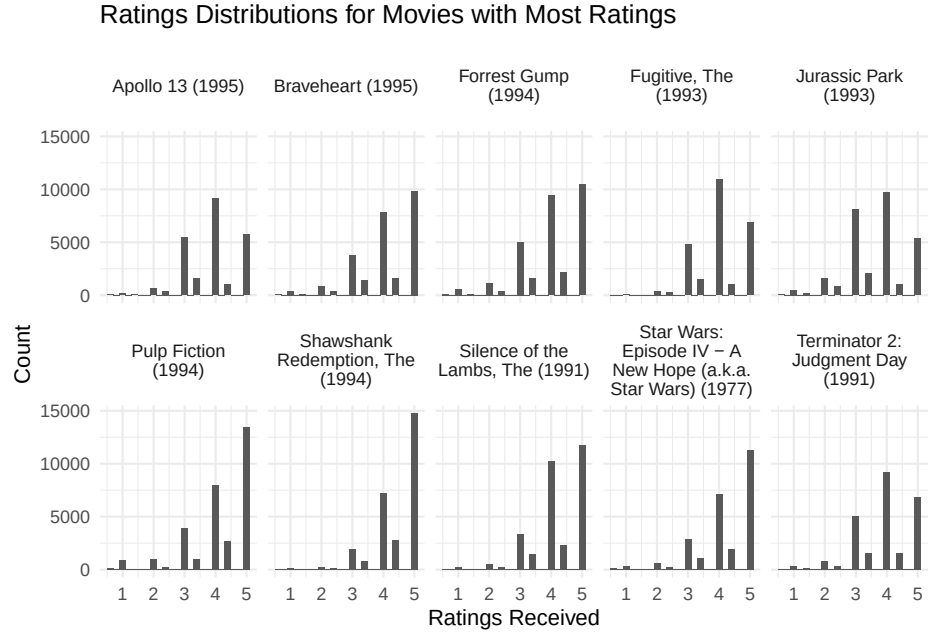


We see that the distributions vary quite a lot. We are only looking at 10 out of 69878 users. So, by extension, we can understand that the distributions of ratings vary widely between users and cannot be modelled using a single distribution as such for all users.

Let us get some idea about the Top 10 Movies in terms of Ratings Received and their Average Ratings



Let us now understand the distribution of ratings for these movies across the rating range.



Just like with users, we can see that the distributions vary quite a lot. Again, we are looking at only 10 out of 10677 movies. So, by extension, we can understand that the distributions of ratings too vary widely between movies and cannot be modelled using a single distribution as such for all movies.

At this point, we pause and take note of the fact that we are building distributions of the ratings primarily for visualisation of data and to illustrate their diversity. When it comes to making predictions, Just being able to predict the distributions accurately for each user or movie is not enough and we also need to be able to predict individual ratings within the distributions just as accurately.

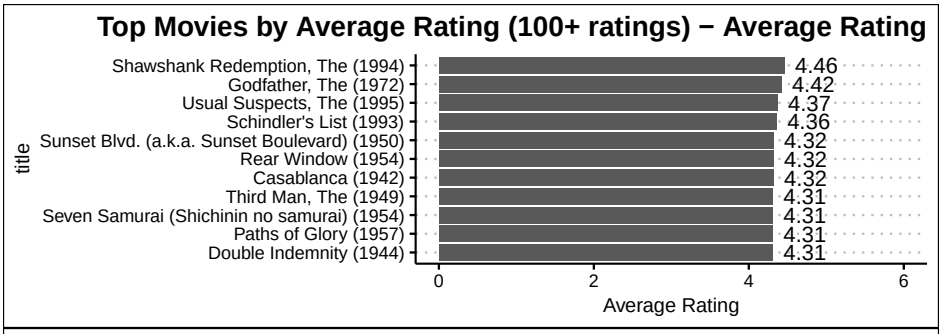
Our fourth Requirement for Recommendation Systems was

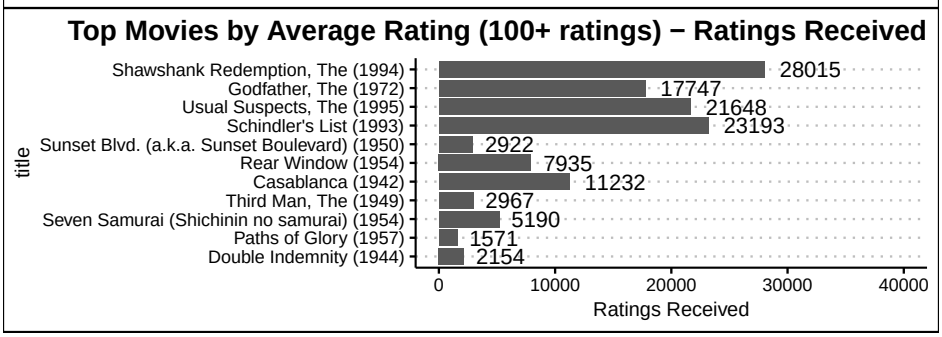
4. A large diversity in the Customer types and buying behaviour.

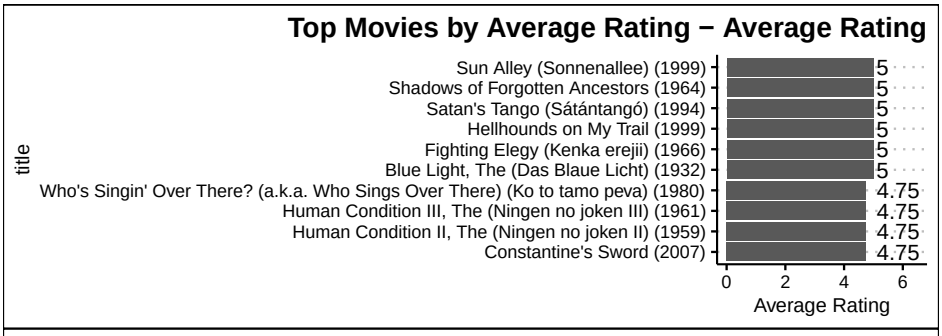
Based on our initial investigation of the "edx" dataset, we can see easily see that there is a huge diversity in users' rating behaviour and consequently the ratings received by movies.

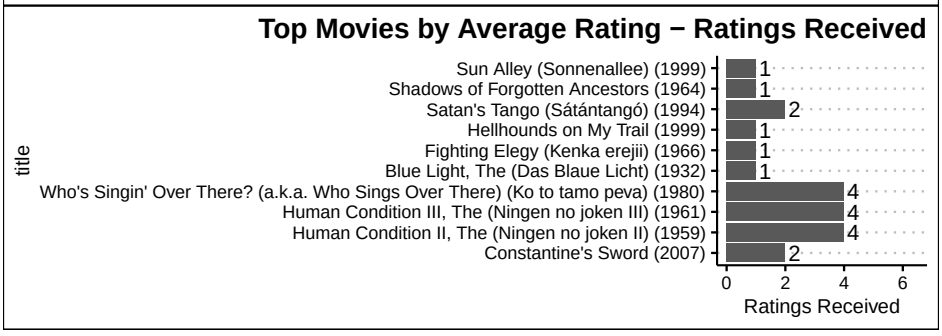
Let us now look at the Top 10 Movies by Average Rating.

We will look at 2 sets of movies. The first set only has movies which have been rated by 100 or more users. The second set imposes no such restriction.









We can see that for the second set, there are many movies which are very highly rated but by very few users. Quite a few of these movies also seem to made for National or Regional Audiences. Since we do not have the language and country of release coded in the "MovieLens 10M Dataset", we will need to perform a manual lookup on IMDb to check these movies.

- Sun Alley (Sonnenallee) (1999) - German
- Shadows of Forgotten Ancestors (1964) - Ukranian

- Satan's Tango (satantango) (1994) - Hungarian
- Hellhounds on My Trail (1999) - English Documentary & Biography
- Fighting Elegy (Kenka Erejjii) (1966) - Japanese
- Who's Singin' Over There? (a.k.a Who Sings Over There) (Ko to tamo peva) (1980) - Serbian
- Blue Light, The (Das Blaue Licht) (1932) - German
- Human Condition II, The (Ningen no joken II) (1959) - Japanese
- Human Condition III, The (Ningen no joken III) (1961) - Japanese
- Constantine's Sword (2007) - English Documentary & Biography

Though these movies may be critically acclaimed by a niche audience, It is very tough to provide these as "Top 10" Recommendations to a general audience.

Our third & eighth Requirements for Recommendation Systems were

3. Products or Services delivered to a Local, National, Regional Confederation or Global Audience.
4. An increased emphasis on Personal Data Protection and User Privacy which limits the amount of information that businesses can collect, retain and share about Customers.

We can see movies with very large Average Ratings but rated by only a few users and more suited for a Regional, National or Niche Audience.

UserID are also anonymised and we do not have the user's country, language, race, age, gender, stated preferences for movies, genre, time period of release, or any other such indicators that might be used to identify or profile a single user or groups of users to make recommendations.

While we have movies in our dataset that wide ranging and serve many audiences, choosing which recommendation to make is also hindered by the lack of such specific predictors.

Create Movie List

Let us now create a movie list with columns for "movieId", "title" and "genres", we will also add "avg_rating" and "n" to represent the Average Rating and Number of Ratings respectively. We will use this as a reference whenever we need to have a look at the list of movies in our database and their summary characteristics.

```
# Extract Unique MovieId along with title and genres
movie_map <- edx %>%
  select(movieId, title, genres) %>%
  group_by(movieId, title, genres) %>% summarise(movieId= unique(movieId), .
      groups= "keep")

# Compute Average Rating for each movie along with number of ratings
movie_rating <- edx %>% group_by(movieId) %>%
  summarise(avg_rating = mean(rating), number_of_ratings = n())

# Create Movie List by Combining above information
movie_list <- left_join(movie_map,movie_rating,by="movieId")

# remove datasets that are no longer needed
rm(movie_rating, movie_map, users_movies)
```

2.3 Create Basic Functions, Training and Validation Sets

2.3.1 Create RMSE Loss Evaluation and Clamp Functions

Let us create the functions for RMSE Evaluation and for Clamping. The Clamp function ensures that predictions for ratings stay between 0.5 and 5.0

```
# Let us build the loss function which we will use for evaluation
rmse <- function(actual_rating, predicted_rating) {
  sqrt(mean((actual_rating -predicted_rating)^2))
}
# Let us use the Clamp Function as provided in the text book to ensure that
    ratings stay within the defined lower and upper boundaries viz. 0.5 <=
    rating <= 5.0. It helps improve our RMSE quite substantially.
clamp <- function(x, min = 0.5, max = 5) pmax(pmin(x, max), min)
```

2.3.2 Create Cross Validation (CV) Set

As we build our Models , we will also need some way to verify the accuracy of their predictions and whether they are fit for purpose. We will use Cross Validation for the same.

For a more detailed explanation of the Scheme, the Reader can refer to the Text Book

https://rafalab.dfci.harvard.edu/dsbook-part-2/ml/resampling-methods.html#cross-validation

We will carve out a validation set from our "edx" dataset. We will treat this validation set the same as we would use the "final_holdout_test" and use it for only verifying the RMSE of the Models that we will build. However, an important distinction between this validation set and the "final_holdout_test" is that with our validation set, we will also use the RMSE obtained for iterative improvement or optimisation of our models. This optimisation process constitutes a part of the "Training" of our Model. By very definition, we cannot and should not use the "final_holdout_test" for any such optimisation.

We will divide our "edx" dataset into "edx_cv_train_set" and "edx_cv_test_set" for Training and Validation respectively. Our validation set will consist of 10% of the observations from the "edx" set. To keep things simple and consistent, we will use the same partitioning scheme as used in the creation of the "edx" and "final_holdout_test" datasets.

```
# Create Data Sets for Cross Validation from the "edx" Dataset. Set Seed same
    as original to maintain consistency. Create edx_cv_train_set to contain
    90% of Observations. We will use it for Training.

set.seed(1, sample.kind="Rounding")

edx_cv_test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1,
    list = FALSE)
edx_cv_train_set <- edx[-edx_cv_test_index,]

temp_cv<- edx[edx_cv_test_index,]

# Create edx_cv_test_set to contain 10% of Observations. We will use it for
    Testing/Validation
# Make sure userId and movieId in edx_cv_test_set set are also in edx_cv_train
    _set.

edx_cv_test_set<- temp_cv %>%
```

```
  semi_join(edx_cv_train_set, by = 'userId') %>%
  semi_join(edx_cv_train_set, by = 'movieId')

# Add rows removed from edx_cv_test_set back into edx_cv_train_set
removed_cv <- anti_join(temp_cv, edx_cv_test_set, by = join_by(userId, movieId
    , rating, timestamp, title, genres))
edx_cv_train_set <- rbind(edx_cv_train_set, removed_cv)

# Set userId and movieId as factors for further processing. Their numerical
    value has no significance

edx_cv_train_set <- edx_cv_train_set %>%
    mutate(userId = as.factor(userId), movieId = as.factor(movieId))

edx_cv_test_set <- edx_cv_test_set %>%
    mutate(userId = as.factor(userId), movieId = as.factor(movieId))

# Create Mean and SD for each user for future reference

user_mean_sd <- edx_cv_train_set %>%
    group_by(userId) %>% summarise(mean=mean(rating), sd = sd(rating))

# remove datasets that are no longer required.
rm(edx_cv_test_index, temp_cv, removed_cv)
```

2.4 Linear Regression Models

Linear Regression Models are often the simplest and the most easily interpretable. For those who are just starting off with Data Analysis, Linear Regression Models provides insight into how the various predictors work in isolation and when combined together and help visualise and understand the characteristics of the data in greater detail.

2.4.1 Typical Linear Regressions Models

Let us quickly remind ourselves about how a typical Linear Regression Model is built.

https://en.wikipedia.org/wiki/Linear_regression#Formulation

For any dataset $\{y_1, x_{i,1}, x_{i,2}.....x_{i,p}\}_{i=1}^{n}$ with n Observations

Linear Regression Models are built in the form

$y_i = \beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2}.....\beta_p x_{i,p} + \varepsilon_i, i = 1, 2...n$

Where

1. y_i is the Dependent, Target, Outcome or Response variable
2. $x_{i,1}, x_{i,2}, ...x_{i,p}$ are the Independent, Input, Feature or Predictor variables
3. β_0 is a constant value independent of the other predictor variables. It is usually called the Intercept.
4. $\beta_1, \beta_2, ...\beta_p$ are the "Effects" or Regression Coefficients, each Regression Coefficient is usually calculated using Least Squares Estimates (LSE) while holding the others constant.
5. ε_i is the error term
6. n is the number of observations as stated earlier

These models are very simple, intuitive and easy to understand when all the terms involved have numerically significant and continuous values.

2.4.2 Linear Regressions Models adapted to our datasets

When it comes to our "edx" and "final_holdout_test" datasets, the variables are evidently different and we cannot use the typical Linear Regression Model as-is.

In our case, each observation contains exactly one rating provided by one user for one movie. The movie has genres associated with it and the timestamp records the time at which the rating was provided. Hence, when we try to fit our variables onto a typical Linear Regression Model, we arrive at a situation where :-

1. y_i corresponds to the "rating" and takes on discrete values in the range $\{0.5 \rightarrow 5.0\}$ in increments of 0.5 .
2. $x_{i,1}$ corresponds to the "userId" with its numerical value being insignificant.
3. $x_{i,2}$ corresponds to the "movieId" with its numerical value being insignificant.
4. $x_{i,3}$ corresponds to the "genres" and it is a concatenation of zero or more character strings and has no numerical significance.
5. $x_{i,p}$ corresponds to the "timestamp" and is the only numerical and continuous value.

*Note: As a reminder, we are not using the "title" field for generating predictions in this Project.

So, how do we fit a Linear Regression Model to these datasets, We can make some assumptions and modifications to the variables.

1. Treat the rating y_i as a continuous variable for the interim. We will change it back to a discrete variable later.
2. Encode the userId as a dummy variable with $x_{i,1} = 1$ for the userId who has provided the rating, $x_{i,1} = 0$ for all other userId
3. Encode the movieId as a dummy variable with $x_{i,2} = 1$ for the movieId which has been rated, $x_{i,2} = 0$ for all other movieId
4. Encode genres as dummy variables. This is more complicated than coding the userId and movieId as each movie is associated with a variable list of zero or more genres. We cannot use a single predictor like $x_{i,3}$ to represent the whole list of genres. We will need to expand the number of predictors.
5. There is no special treatment required for the timestamp as it is already in the desired form. However, we can encode $x_{i,p}$ as a calendar year rather than use the timestamp as-is because :-
    1. Dates are more easily understood by humans as compared to the timestamp.
    2. The timestamp offers too much granularity in measurement which is not required in our case.

We will also define and build some "Effects" or Regression Coefficients for each of these predictor variables. However, before we do that let us also simplify the Expression for Linear Regression so that it is easier to understand and more suited to our datasets. To do that, let us recollect the notes from the beginning of our Analysis.

1. User specific ratings are a direct reflection of user preferences or user behaviour. These can be seen as being primary in nature.
2. Movie specific ratings are the grouping of user preferences or user behaviour within the context of each movie.
3. User ratings and Movie ratings are the two most important groups in the data.
    1. The User ratings influence the Movie ratings in complex ways.
    2. They cannot be considered as being orthogonal to each other nor can they be considered as converging into each other due to the natural diversity that they both embody.
4. The "genres" and "timestamp" data can be organised and analysed in the context of the User ratings and Movie ratings groups.

With these points in mind, We will restate the Typical Linear Regression Model as

$y_{i,j} = \mu + \alpha_i x_i + \beta_j x_j + \sum_{k=1}^{K} \beta_{i,k} x_{i,k} + \beta_{i,t} x_i + \beta_{j,t} x_j + \varepsilon_{i,j}$

where

1. $y_{i,j}$ is the rating provided by user i for movie j . We will treat it as a continuous variable for the interim. We will change it back to a discrete variable later
2. μ is a Constant analogous to β_0 . Its value does not change with any of the other predictors. We will soon see why we have chosen μ as the symbol for this Constant.
3. α_i is the "User Treatment Effect" or "User Effect" or Regression Coefficient for user i . The value is specific to each user. It is reflective of the user behaviour directly.
4. β_j is the "Movie Treatment Effect" or "Movie Effect" or Regression Coefficient for movie j. The value is specific to each movie. It is reflective of the behaviour of groups of users as applied to each movie.
5. $\beta_{i,k}$ is the "Genre User Treatment Effect" or "Genre User Effect" or Regression Coefficient for user i for applicable genre k. It is reflective of the user's rating behaviour for that particular genre. There is one value for each genre k for each user i . $x_{i,k} = 1$ if the genre k is applicable for the movie j rated in the observation, $x_{i,k} = 0$ otherwise.

6. K is the number of genre attached to the movie
7. $\beta_{i,t}$ is the "Time User Treatment Effect" or "Time User Effect" or Regression Coefficient for user i for the year of rating. There is one value for each user i for each year t. It is reflective of how the the user's rating behaviour changes with time.
8. $\beta_{j,t}$ is the "Time Movie Treatment Effect" or "Time Movie Effect" or Regression Coefficient for movie j for the year of rating. There is one value for each movie j for each year t. It is reflective of how the movie's rating changes with time.
9. $\varepsilon_{i,j}$ is the error term

$x_i = 1$ for user i and $x_i = 0$ for all other users

$x_j = 1$ for movie j and $x_j = 0$ for all other movies.

Because of the values that x_i and x_j can take for any single Observation, The Linear Regression Equation can be further simplified as :-

$y_{i,j} = \mu + \alpha_i + \beta_j + \sum_{k=1}^{K} x_{i,k}\beta_{i,k} + \beta_{i,t} + \beta_{j,t} + \varepsilon_{ij}$ **{ Initial LRE**

Let us call it the "Initial LRE" for easier reference.

Though the Initial LRE looks a bit contrived and complex at first glance, it serves to summarise and model our Predictors according to our Analysis so far. It will start to become clearer as we build some Linear Regression Models step-by-step, with each step helping us add a Predictor and better our predictions. We will review our progress against the Initial LRE as we go along.

In our Models, we will follow the notation(s) from the text book as much as possible. We will modify it only when it is not possible to express the Model unambiguously using the notation(s) from the text book.

However, when it comes to the R Code, it is not possible to follow the notation(s) exactly as we cannot have LaTeX type of formatting for variables, so we will use whatever makes sense and is easy to follow.

2.4.3 Model 1 - Base

The first Linear Regression Model is the simplest and directly from the course. It will also help the Reader understand why we chose to use μ as the symbol for the Intercept in our Linear Regression Equation.

In this model the prediction is the Arithmetic Mean of all "known" ratings with differences being treated as random errors. Known here means all of the ratings from "edx_cv_train_set" .

The Model can be written as

$Y_{i,j} = \mu + \varepsilon_{i,j}$

where

$Y_{i,j}$ is the predicted rating

μ is the Mean value of all known ratings

$\epsilon_{i,j}$ is the random error

When compared to our Inital LRE,

$y_{i,j} = \mu + \alpha_i + \beta_j + \sum_{k=1}^{K} x_{i,k}\beta_{i,k} + \beta_{i,t} + \beta_{j,t} + \varepsilon_i$

we have added our very first Regression Coefficient μ which is the Intercept.

We will test our RMSE against the ratings in "edx_cv_test_set".

We will record the RMSE for this Model as "RMSE Base"

We will also record the Mean value μ (mu) and the Standard Deviation σ (sigma) for further reference.

*Note: In the R Code we use "mu" to denote μ, "sigma" to denote σ.

```
# Compute values of Mean & Standard deviation
mu <- round(mean(edx_cv_train_set$rating), digits = 5)
sigma <- round(sd(edx_cv_train_set$rating), digits = 5)

# Compare Actual Ratings against the Mean as Prediction
rmse_base <-round(rmse(edx_cv_test_set$rating, mu),digits = 5)

# Print RMSE, Mean & Standard Deviation
print( c(" RMSE Base : ", round(rmse_base, digits = 5)), quote = FALSE,
    justify = "left")
```

```
## [1]  RMSE Base :  1.06005
```

```
print( c(" Mean Value (mu) : ", round(mu, digits = 5)), quote = FALSE, justify
     = "left")
```

```
## [1]  Mean Value (mu) :  3.51246
```

```
print( c(" Standard Deviation (sigma) : ", round(sigma,digits = 5)), quote =
    FALSE, justify = "left")
```

```
## [1]  Standard Deviation (sigma) :  1.06036
```

2.4.4 Model 2 - User Effects

The second Model is also directly from the course.

As we have seen already seen in our section "Understanding the edx dataset" not all users rate movies the same, some users tend to rate movies kindly, while some are more critical. To account for user rating behaviour, We use a Linear model with a Treatment Effect for each user.

The Model can be written as

$Y_{i,j} = \mu + \alpha_i + \varepsilon_{i,j}$

where

$Y_{i,j}$ is the predicted rating

μ is the Mean value of all known ratings

α_i is the User Treatment Effect for each user i . $\alpha_i = \frac{1}{N_j} \sum_j^{N_j} (y_{i,j} - \mu)$ where $y_{i,j}$ is the rating provided by the user i for movie j and N_j = Number of movies rated by the user,

$\epsilon_{i,j}$ is the random error

The User Effect summarises each user's rating behaviour across movies.

When compared to our Inital LRE,

$y_{i,j} = \mu + \alpha_i + \beta_j + \sum_{k=1}^{K} x_{i,k}\beta_{i,k} + \beta_{i,t} + \beta_{j,t} + \varepsilon_i$

we have added our second Regression Coefficient α_i which is the User Effect.

We will test our RMSE against the ratings in "edx_cv_test_set".

We will record the RMSE for this Model as "RMSE User Effects"

*Note: We use "alpha_i" in the R Code to denote α_i

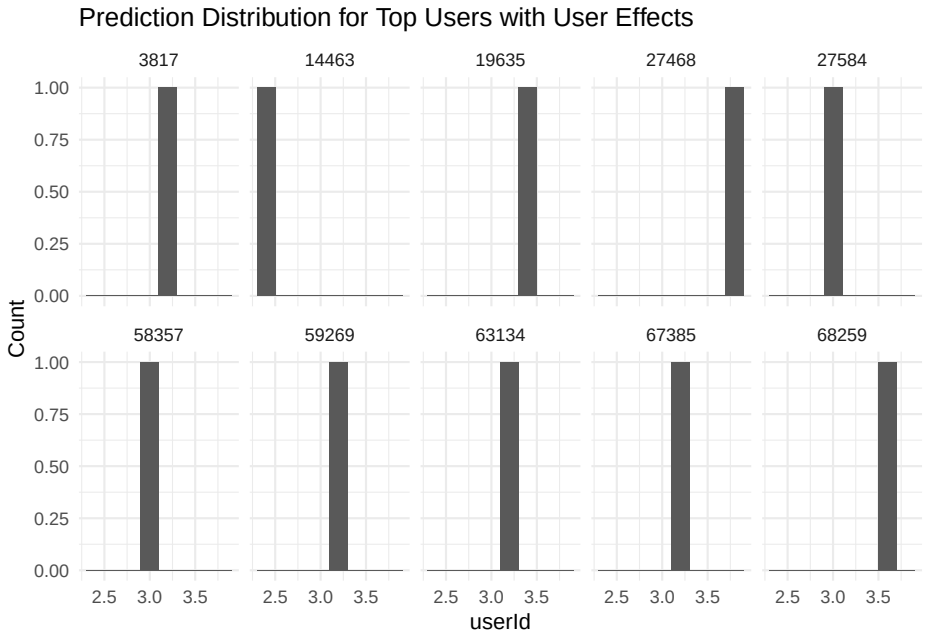
```
# The Code is more suited for Memory Constrained Machines.

# THE PIVOT WIDER METHOD FROM THE TEXT BOOK WITH "userId" AS ROWS AND "movieId
    " AS COLUMNS IS RECOMMENDED ON A STUDENT LAPTOP WITH SUFFICIENT MEMORY AND
    /OR CLOUD SERVICES PROVIDING SIMILAR AMOUNTS OF MEMORY. FOR REFERENCE THE
    CREATED MATRIX TAKES UP ALMOST 6GB OF SPACE ON ITS OWN IN R AND RSTUDIO
    USES UP ANOTHER 6GB ADDITIONAL WHEN THE MATRIX IS CREATED.

# For each User, Compute User Effects with alpha_i = mean (rating - mu) and
    Store in a new dataframe

user_effects <- edx_cv_train_set %>%
  group_by(userId) %>%
  summarize(alpha_i = mean(rating - mu))

# Prepare Predictions for edx_cv_test_set. Our Prediction for each Rating will
     be (mu + alpha_i)

predicted_ratings_user <- mu +
  edx_cv_test_set %>% left_join(user_effects, by= 'userId') %>%
  pull(alpha_i)

# Compute and record the RMSE by comparing our Predictions with the actual
    ratings provided by the user.
```

```
rmse_user <- round(rmse(edx_cv_test_set$rating, predicted_ratings_user),
    digits = 5)

# Print the RMSE as "RMSE User Effects"
print( c(" RMSE User Effects : ", rmse_user), quote = FALSE, justify = "left")
```

```
## [1]  RMSE User Effects :  0.97771
```

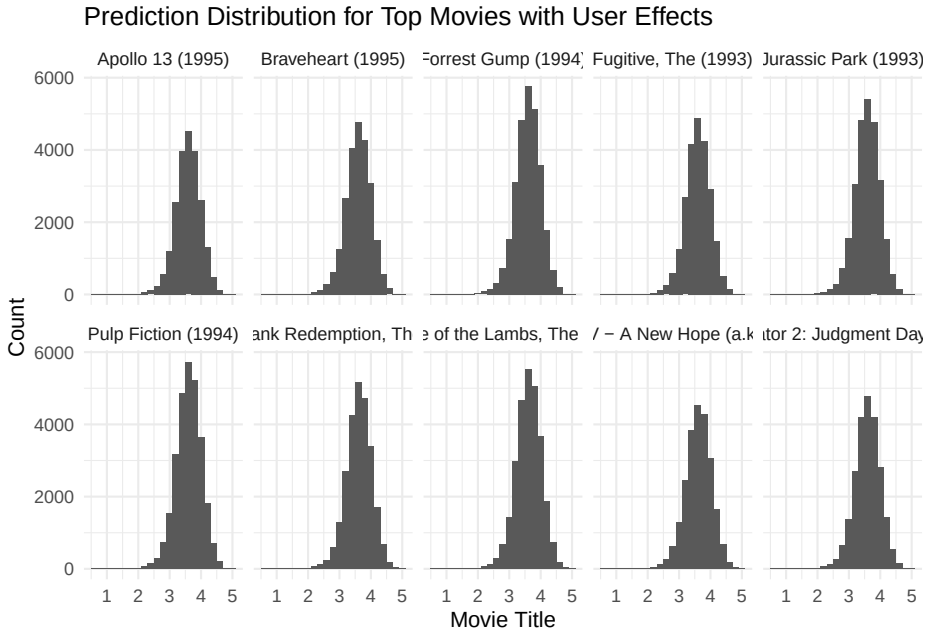
```
# Remove Data that is no longer required.
rm(predicted_ratings_user)
```

Let us see how User Effects influence our predictions for the ratings. We will choose the Top 10 users by number of ratings provided to illustrate the same.



As one would imagine, there is a Single prediction for each user according to the equation $Y_{i,j} = \mu + \alpha_i$. This is far from the desired situation.

Now, let us also see the distribution of predictions for the ratings for the Top 10 movies by ratings received.



As might be expected all the movies seem to have a normal distribution with the ratings centred around μ.

2.4.5 Model 3 - User Effects & Movie Effects

The Third Model is also directly from the Course.

Similar to User Effects, Movies too tend to have Good Ratings or otherwise, depending on their Popularity with the Audience. We can use a Linear model with a treatment effect for each Movie.

The Model can be written as

$Y_{i,j} = \mu + \alpha_i + \beta_j + \varepsilon_{i,j}$

where

$Y_{i,j}$ is the Predicted Rating

μ is the Mean Value of all known Ratings

α_i is the Treatment Effect for each User i .

β_j is the Treatment Effect for each Movie j . $\beta_j = \frac{1}{N_i} \sum_i^{N_i} (y_{i,j} - \mu - \alpha_i)$ where $y_{i,j}$ is the Rating provided by User i for the Movie j , α_i is the Treatment effect for User i and N_i = Number of Users Rating the Movie j,

$\epsilon_{i,j}$ is the Random Error

The Movie Effect summarises each Movie's Rating by different Users. We can see that the Movie Effect is conditioned by the presence of the User Effect in this model.

When compared to our Inital LRE,

$y_{i,j} = \mu + \alpha_i + \beta_j + \sum_{k=1}^{K} x_{i,k}\beta_{i,k} + \beta_{i,t} + \beta_{j,t} + \varepsilon_{ij}$

we have added our third Regression Coefficient β_j which is the Movie Effect.

We will test our RMSE against the ratings in "edx_cv_test_set".

We will record the RMSE for this Model as "RMSE User & Movie Effects"

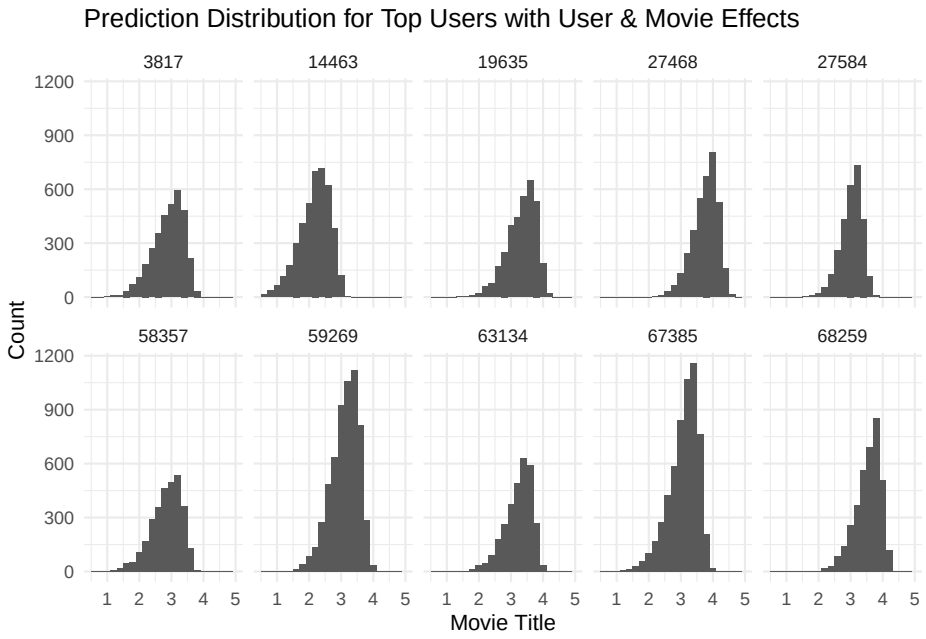
*Note: We use "beta_j" in the R Code to denote β_j

```
#  For each Movie, compute Movie Effects as beta_j = mean(rating - mu - alpha_
    i)

movie_effects <- edx_cv_train_set %>% left_join(user_effects, by = "userId")
    %>%
  group_by(movieId) %>%
  summarize(beta_j = mean(rating - mu - alpha_i))

# Prepare Predictions for edx_cv_test_set. Our Prediction for each Rating will
     be mu + alpha_i + beta_j
predicted_ratings_user_movie <- edx_cv_test_set %>%
      left_join(movie_effects, by='movieId') %>%
      left_join(user_effects, by='userId') %>%
      mutate(pred = clamp(mu + beta_j + alpha_i)) %>%
      pull(pred)

# Compute and record the RMSE by comparing our Predictions with the actual
    ratings provided by the user.
rmse_user_movie <- round(rmse(edx_cv_test_set$rating, predicted_ratings_user_
    movie), digits = 5)

# Print the RMSE as "RMSE User & Movie Effects"
print( c(" RMSE User & Movie Effects : ", rmse_user_movie), quote = FALSE,
    justify = "left")
```

```
## [1]  RMSE User & Movie Effects :  0.88082
```

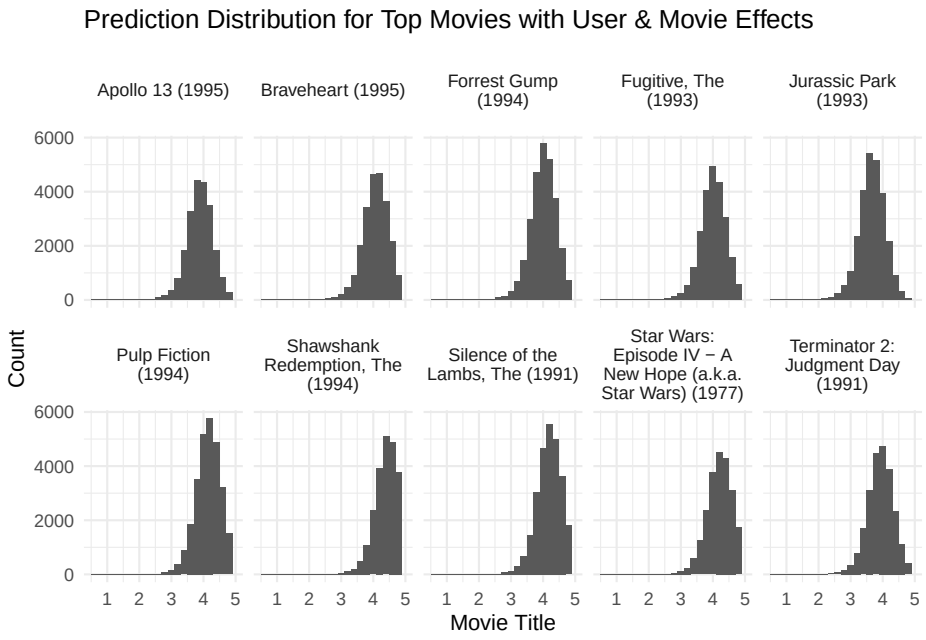
```
# Remove Data that is no longer required
rm(predicted_ratings_user_movie)
```

Let us look at the Distribution of our Predictions for Users with the Movie Effect included.



We can see the ratings being more spread and centred around $(\mu + \alpha_i)$. The shape of the curves does not resemble a normal distribution for most of the Users.

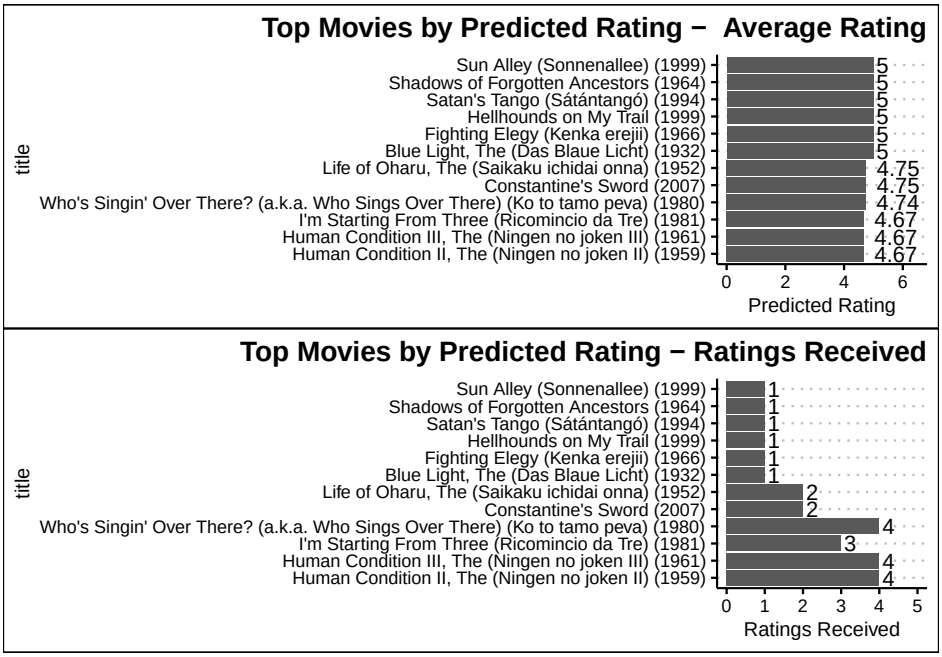
Let us look at the distribution of our predictions for movies with the Movie Effect included.



We can observe that most of the Predictions are no longer centred around μ and are now centred around the Average Rating for the movie. The predictions seem to be still following a Normal Distribution with the "clamp" function coming into play to clip predictions that exceed the rating scale.

Let us have a look at the movies with the best ratings. We briefly touched upon the topic during the Introduction to the Methods/Analysis Section. We will now delve into greater detail on this topic. Let us now look at the movies with Top Average Ratings but with no other filter applied. We will use our

predictions for the "edx_cv_train_set". We will revisit this topic often as a "Top 10" board is one of the important features of any Recommendation System.



We can see that the movies that have very high ratings are rated by very few people and also do not seem to be very popular. Though our predictions are largely aligned with the actual ratings, Any commercial system offering such recommendations for a "Top 10" set of movies would see very poor reception as it does not reflect the general sentiment of the population. Let us see if we can align our results with the popularity of movies, The preferred method for this leads us to the next section.

2.4.6 Model 4 - User Effects & Movie Effects (R)

The Fourth Model is also from the Course.

This Model uses Regularisation for Movie Effects. Here, for the Movie Effects, instead of minimizing the least squares equation, we minimize an equation that adds a penalty:

$\sum_{i,j}^{N} \left(y_{i,j} - \mu - \alpha_i - \beta_j\right)^2 + \lambda_j \sum_j \beta_j^2$

Where,

$y_{i,j}$ is the actual rating by user i for movie j

μ is the Mean value of all known ratings

α_i is the Treatment Effect for each user i

β_j is the Treatment Effect for each movie j before Regularisation

The first term is just the sum of squares and the second is a penalty that gets larger when many β_j s are large. Using calculus it can be shown that, the values of β_j that minimize this equation are:

$\hat{\beta_{j(\lambda_j)}} = \frac{1}{\lambda_j + N_i} \sum_{i}^{N_i} \left(Y_{i,j} - \mu - \alpha_i\right)$

where

$\hat{\beta_{j(\lambda_j)}}$ is the estimated value for β_j for the prediction

$Y_{i,j}$ is the predicted rating for user i for movie j

N_i is the number of ratings made for movie j (or alternately the number of users who have rated the movie j)

λ_j is a penalty applied to the Movie Effects

When our sample size N_i is very large, we obtain a stable estimate and the penalty λ_j is effectively ignored since $N_i + \lambda_j \approx N_i$. Yet when N_i is small, then the estimate $\hat{\beta}_{i(\lambda_j)}$ is shrunk towards 0. The larger the value of λ_j , the more we shrink $\hat{\beta}_{j(\lambda_j)}$. Depending on the numerical value of $\hat{\beta}_{j(\lambda_j)}$ and its sign, our predictions can either increase or decrease accordingly. We will see an example of it in action later in this section.

In more simple terms, when a movie has been rated by many users, We assume that the Mean rating of the movie is more "reliable" because of the large sample size. However, when the movie has been rated by very few users, we treat it with scepticism because of the small sample size. Our objective is to ensure that we have more "reliably" rated movies at the top. Our predictions for movies whose ratings are more "reliable" must more closely match the actual rating as compared to our predictions for movies who ratings are not very reliable. This will also help improve our RMSE overall.

We use a simple method to choose the best value of λ_j by using the computed RMSE. The best value of λ_j is the one that provides the minimum RMSE when evaluated against the Validation set which in our case is the "edx_cv_test_set".

The observant reader might remember our notes from the "Create Cross Validation (CV) Set" section that for optimisation of our models, We can use our Validation set "edx_cv_test_set" but cannot use the "final_holdout_test".

The single biggest utility of our Validation set is to check and optimise our predictions in a iterative manner and the calculation of the Regularisation factor λ_j constitutes such an optimisation process.

Now our model for the prediction becomes:

$Y_{i,j} = \mu + \alpha_i + \beta_{j(\lambda_j)} + \varepsilon_{i,j}$

Where,

$Y_{i,j}$ is the predicted rating

μ is the Mean value of all known ratings for all movies

α_i is the Treatment Effect for each user i .

$\beta_{j(\lambda_j)}$ is the Treatment Effect for each movie j after Regularisation.

$\epsilon_{i,j}$ is the random error

When compared to our Inital LRE,

$y_{i,j} = \mu + \alpha_i + \beta_j + \sum_{k=1}^{K} x_{i,k}\beta_{i,k} + \beta_{i,t} + \beta_{j,t} + \varepsilon_i$

we have not added any new Regression Coefficients but have optimised the third Regression Coefficient β_j which is the Movie Effect.

We will test our RMSE against the ratings in "edx_cv_test_set".

*Note: We shall use "beta_r" to represent $\beta_{j(\lambda_j)}$ and "lambda_j" to represent λ_j in R Code. The prints from R will contain these names accordingly.

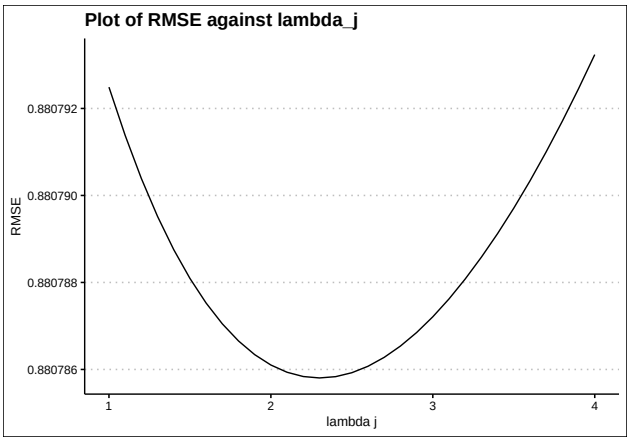
```
# Create a dataframe for computing the new values of Movie Effects using Sum
    of (rating - mu - alpha_i) and the Number of Ratings (n) for each Movie.

penalised_least_squares_movie <- edx_cv_train_set %>%
  left_join(user_effects, by = "userId") %>%
  group_by(movieId) %>%
  summarize(sums = sum(rating - mu -alpha_i), n = n())

# Set up a sequence of values for lambda_j to compute the least RMSE.

lambdas_j <- seq(1, 4, 0.1)

# Write a function to compute the RMSE and store it in a Table.

rmses_penalised_movie <- sapply(lambdas_j, function (lambda_j){

        # Compute the new value for Movie Effects. We will use beta_r (r for
            regularised) instead of          reusing beta_j to avoid confusion.
             The value is beta_r = sums/(lambda_j + n). Store the
                        values in a new dataframe.

  beta_r_df <- penalised_least_squares_movie %>% mutate("beta_r" = sums/(n +
     lambda_j))

        # Prepare Predictions for different values of lambda_j

  predicted_ratings_penalised_movie <- edx_cv_test_set%>%
    left_join(beta_r_df, by='movieId') %>%
    left_join(user_effects, by='userId') %>%
    mutate(pred = clamp(mu + beta_r + alpha_i)) %>%
    pull(pred)

        # Compare our predictions against the actual ratings in edx_cv_test_
            set and return the same          from the function

  rmse(edx_cv_test_set$rating, predicted_ratings_penalised_movie)
```

```
})

# Find and record the Minimum RMSE and the corresponding lambda_j value
min_lambda_j <- lambdas_j[which.min(rmses_penalised_movie)]
min_rmse_penalised_movie <- round(min(rmses_penalised_movie), digits = 5)

# Prepare final regularised Predictions with the Minimum computed RMSE and
    corresponding Lambda.

beta_r_df_initial <- penalised_least_squares_movie %>%
                     mutate(beta_r = sums/(n + min_lambda_j)) %>%
                     select(movieId, beta_r)
```

Let us plot the RMSE versus the λ_j values.

We will find and record the Minimum RMSE and the corresponding λ_j value

We will record the RMSE for this Model as "RMSE User & Movie Effects (R)"




```
## [1]  Min lambda j :  2.3
```

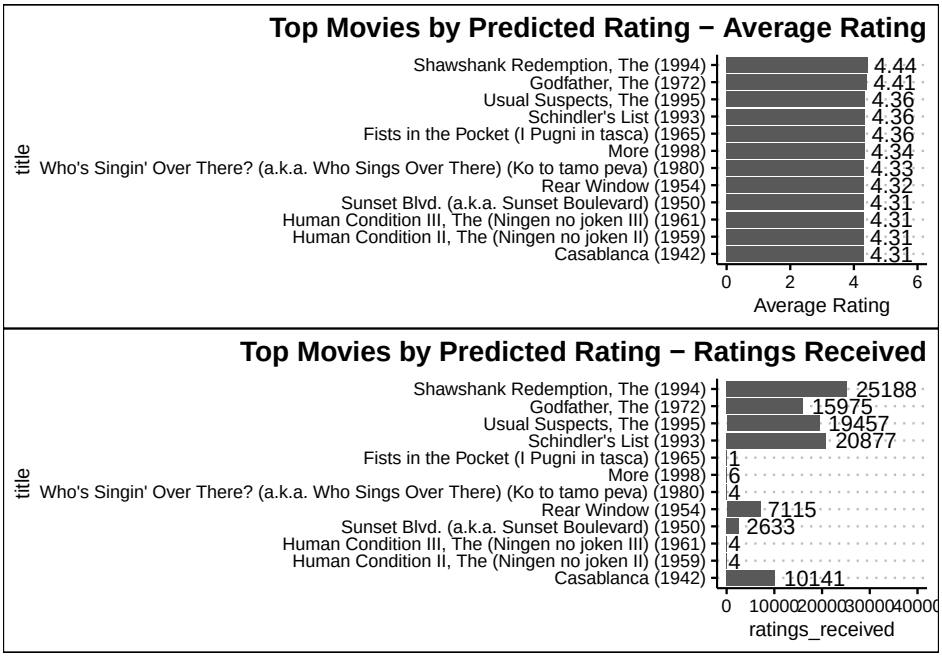
```
## [1]  Min RMSE Regularised:  0.88079
```

```
## [1]  RMSE User & Movie Effects (R):  0.88079
```

```
## [1] Improvement in RMSE with Regularisation :
## [2] 3e-05
```

Let us review our RMSE computations with Movie Effects Regularisation. The actual improvement is very small and we will need to look at other parameters for improving the RMSE substantially. This seems to be something that is characteristic of the data itself.

Let us check how regularisation has affected our Top 10 movies by Average Rating and whether we can now predict Top 10 movies by Average Rating better than before.



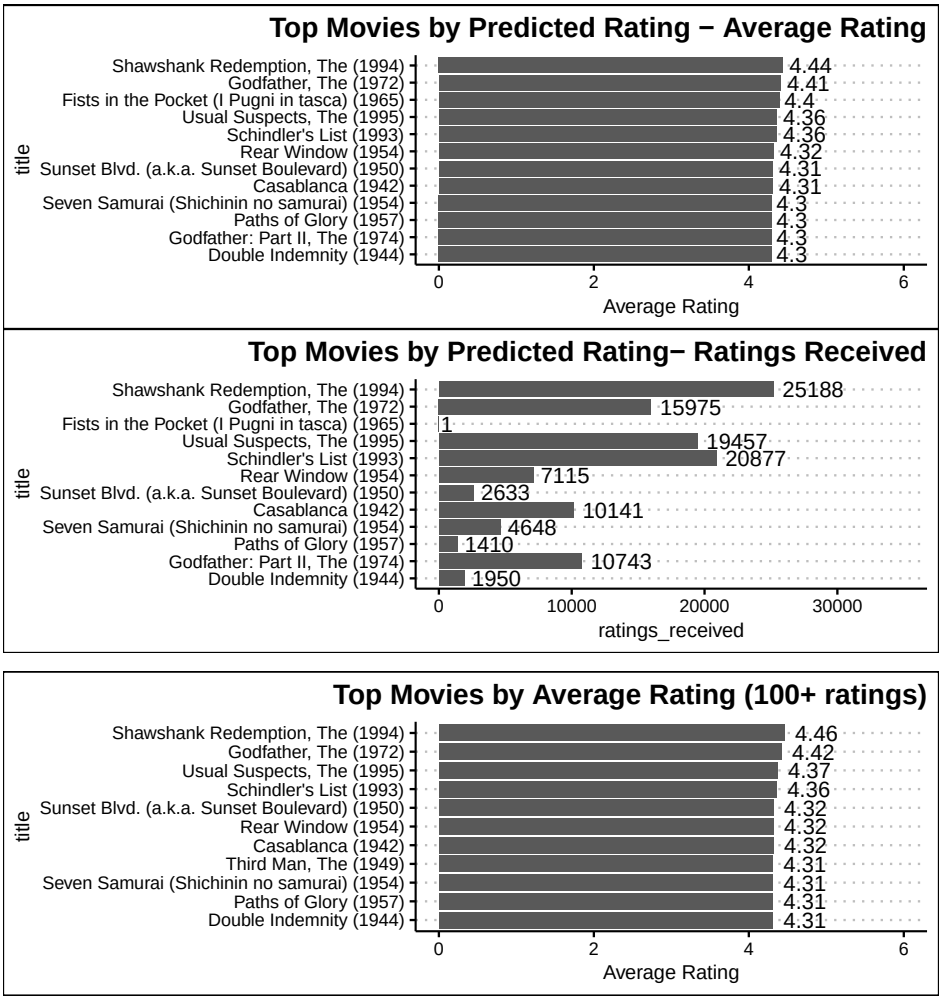
We definitely see some improvement, We can now see some really popular and well received movies in our Top 10 list but there are still a lot of entries that make little sense as they have very high average ratings while the number of ratings is still small.

The Regularisation has had the desired effect in shrinking the predicted ratings of these movies towards μ. However the value of λ_j that we have chosen optimises the overall RMSE and is not large enough to move these movies out of the "Top 10"

To see how adjustments in the value of λ_j can affect our top 10, let us now perform a small experiment and manually increment the value of λ_j by 1.

```
## [1]  Min lambda_j Regularised :  2.3
```

```
## [1]  lambda_j Manually Modified :  3.3
```



We now have a pretty good "Top 10" list and we can compare the results with "Top Movies by Average Rating (100+ ratings)". We also notice that our Predictions for the "Top 10" Movies by Predicted Rating now is much closer to the Actual Ratings accorded by Users.

While our objective in computing λ_j is to reduce the RMSE overall, we can see how minor tweaks in the values can help us present the data much better for specific features.

Let us check our RMSE for the manually incremented value of λ_j

```
## [1]  RMSE with manually incremented value of lambda j:
## [2] 0.88079
```

There is no change in the RMSE noting the fact that we are rounding it off to 5 digits after the decimal

point. Any such small change in RMSE has no practical relevance but the increment in λ_j helps in moving out entries that have very few ratings from our list of Top Rated Movies..

Table 3: RMSE Summary

Model	RMSE
1-Base	1.06005
2-User Effects	0.97771
3-User & Movie Effects	0.88082
4-User & Movie Effects (R)	0.88079

```
## [1] Improvement in RMSE with Regularisation :
## [2] 3e-05
```

To understand how our Predictions are being affected by Regularisation and the kind of side effects they can cause, Have a look at a single seemingly obscure entry with the title "Fists in the Pocket (I Pugni in tasca) (1965)" which had appeared in our list and still remains. It is a movie with a Single user rating of 4.0.

Table 4: Movie Details

movieId	title	genres	avg_rating	number_of_ratings
40833	Fists in the Pocket (I Pugni in tasca) (1965)	Drama	4	1

Table 5: User Details

	userId	movieId	title	rating
2004881	14598	40833	Fists in the Pocket (I Pugni in tasca) (1965)	4

Table 6: Mean Rating for User

userId	mean_rating
14598	4.52

Table 7: alpha_i Value for User

userId	alpha_i
14598	1.00754

Table 8: beta_j value for Movie

movieId	beta_j
40833	-0.52

Table 9: beta_r value for Movie

movieId	beta_r
40833	-0.1575758

We can see that the user with "userId" 14598 who is the only person to rate this movie, has a very high Mean rating of 4.52 and the α_i for this user is about 1.0075, The initial β_j value before Regularisation for this movie was -0.52.

The predicted rating for the movie before Movie Effects Regularisation was

$\mu + \alpha_i + \beta_j = 3.5125 + 1.0075 + (-0.52) = 4$, where $i = 14598$ & $j = 40833$

With Movie Effects Regularisation, the revised β_j or $\beta_{j(\lambda_j)}$ value is -0.1576. The revised predicted rating is

$\mu + \alpha_i + \beta_{j(\lambda_j)} = 3.5125 + 1.0075 + (-0.1576) = 4.3624$, where $i = 14598$ & $j = 40833$

Because of Regularisation, the value of β_j has shrunk, However in this case, since it has a negative value , The predicted rating of the movie is higher than its actual rating.

We can also see that λ_j is indeed working by shrinking the value of β_j given the value of N=1, however the α_i rating is now having a larger effect on the Predicted Rating.

While Movie Effects Regularisation is helping reduce the overall RMSE, it is also having some unfortunate side effects. We will explore some remediation in the next section.

2.4.7 Model 5 - User Effects (R) & Movie Effects (R) - Combined Regularisation

Just like with the Movie Effects Regularisation, We can try something similar with the User Effects to compensate for outlier user ratings that are skewing our predictions.

User Effects Regularisation is eventually likely to help us better our RMSE compared to Movie Effects Regularisation alone because there are a lot more users (69878) than movies (10667) and hence we can expect more variability and more outliers as a natural consequence.

User Effects Regularisation combined with Movie Effects Regularisation using a common Regularisation Factor is not explicitly articulated in the Text Book or on the course text or Videos, however the scheme is presented in the R Code that is available as a companion to the Machine Learning Course. It uses hierarchical regularisation that has not been articulated in the course so far.

The only modification is the addition of the clamp function to make sure that our predictions stay within the boundaries of the rating scale.

The next model follows the same.

With our new model, we will try to minimise the following equation.

$\sum_{i,j} \left(y_{i,j} - \mu - \alpha_i - \beta_j\right)^2 + \lambda_{ij}(\sum_i \alpha_i^2 + \sum \beta_j^2)$

Where,

$y_{i,j}$ is the actual rating by user i for movie j

μ is the Mean value of all known ratings

α_i is the Treatment Effect for each user i before Regularisation.

β_j is the Treatment Effect for each movie j before Regularisation.

$\epsilon_{i,j}$ is the random error

λ_{ij} is a combined penalty applied to the User Effects and Movie Effects

Since the penalty λ_{ij} is affecting two coefficients α_i and β_j at the same time, it is difficult to express the equation that minimises λ_{ij} mathematically due to mutual recursion between α_i and β_j . However it can be computed in R by iteration with increments that are small enough to understand and track changes in the resulting values.

Now our model for the Prediction becomes:

$Y_{i,j} = \mu + \alpha_{i(\lambda_{ij})} + \beta_{j(\lambda_{ij})} + \varepsilon_{i,j}$

Where,

$Y_{i,j}$ is the Predicted Rating

μ is the Mean Value of all known Ratings

$\alpha_{i(\lambda_{ij})}$ is the Treatment Effect for each user i after combined User Effects and Movie Effects Regularisation.

$\beta_{j(\lambda_{ij})}$ is the Treatment Effect for each movie j after combined User Effects and Movie Effects Regularisation.

$\epsilon_{i,j}$ is the random error

When compared to our Inital LRE,

$y_{i,j} = \mu + \alpha_i + \beta_j + \sum_{k=1}^{K} x_{i,k}\beta_{i,k} + \beta_{i,t} + \beta_{j,t} + \varepsilon_{ij}$

we have not added any new Regression Coefficients but have optimised the second Regression coefficient α_i which is the User Effects and the third Regression coefficient β_j

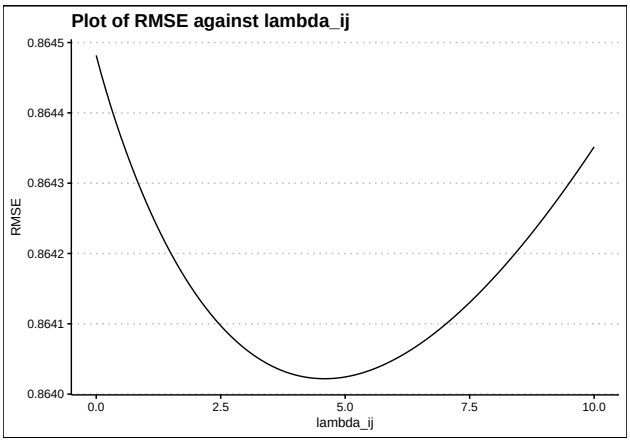
*Note: We shall use "alpha_ij" to represent $\alpha_{i(\lambda_{ij})}$ in R Code. The Prints from R will contain "alpha_ij" as the variable name for $\alpha_{i(\lambda_{ij})}$. Similarly we will use "beta_ij" as the variable name for $\beta_{j(\lambda_{ij})}$

```
# We will set up a sequence of values for lambda_ij to compute the least RMSE.
lambdas_ij <- seq(0, 10, 0.1)

# Setup a function to return the RMSE for each value of lambda_ij
rmses_penalised_user_movie  <- sapply(lambdas_ij, function(lambda_ij){
  # Create & update new dataframe to store values of beta_ij
     penalised_movie_effects <- edx_cv_train_set %>%
          group_by(movieId) %>%
          summarize(beta_ij = sum(rating - mu)/(n()+lambda_ij))

  # Create & update new dataframe to store values of alpha_ij
     penalised_user_movie_effects <- edx_cv_train_set %>%
          left_join(penalised_movie_effects, by="movieId") %>%
          group_by(userId) %>%
          summarize(alpha_ij = sum(rating - beta_ij - mu)/(n()+lambda_ij))

  # Prepare predictions for Testing/Validation set using the computed beta_ij
      and alpha_ij
     predicted_ratings_user_movie <-
          edx_cv_test_set %>%
          left_join(penalised_movie_effects, by = "movieId") %>%
          left_join(penalised_user_movie_effects, by = "userId") %>%
          mutate(pred = clamp( mu + alpha_ij + beta_ij)) %>%
          pull(pred)

  # Compute and return RMSE
     return(rmse(predicted_ratings_user_movie, edx_cv_test_set$rating))
})

# We will find and record the Minimum RMSE and the corresponding lambda_ij
    value
min_lambda_ij <- lambdas_ij[which.min(rmses_penalised_user_movie )]
min_rmse_penalised_user_movie <- round(min(rmses_penalised_user_movie ),digits
    = 5)

# Compute final values for alpha_ij and beta_ij for later reference

beta_r_df_combined <- edx_cv_train_set %>%
                      group_by(movieId) %>%
                      summarize(beta_ij = sum(rating - mu)/(n() + min_lambda_
                         ij)) %>%
                      select(movieId, beta_ij)

alpha_r_df_combined <- edx_cv_train_set %>%
                      left_join(beta_r_df_combined, by="movieId") %>%
                      group_by(userId) %>%
                      summarize(alpha_ij = sum(rating - beta_ij - mu)/(n() +
                         min_lambda_ij)) %>%
                      select(userId, alpha_ij)
```

Let us plot the RMSE versus the λ_i values.

We will find and record the Minimum RMSE and the corresponding lambda_ij.

We will record the RMSE for this Model as "RMSE User Effects (R) & Movie Effects (R)"




```
## [1] Min lambda_ij :  4.6
```

```
## [1] RMSE User Effects (R):  0.86402
```

Table 10: RMSE Summary

Model	RMSE
1-Base	1.06005
2-User Effects	0.97771
3-User & Movie Effects	0.88082
4-User & Movie Effects (R)	0.88079
5a-User Effects (R) & Movie Effects (R)	0.86402

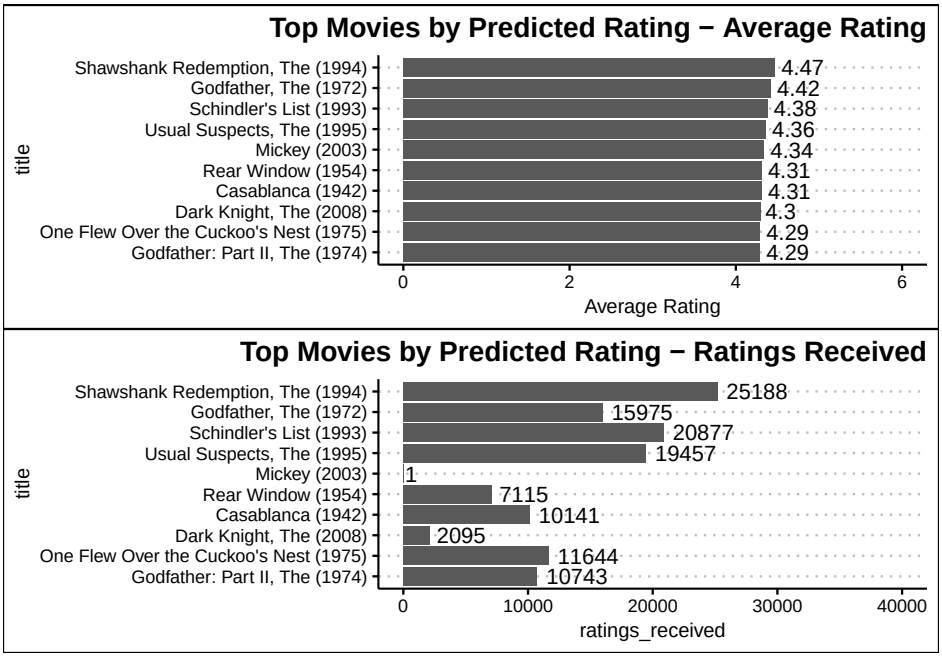
```
## [1] Improvement in RMSE with User Regularisation over User Effects :
## [2] 0.11369
```

```
## [1] Improvement in RMSE with User Regularisation over User and Movie
    Effects (R):
## [2] 0.01677
```

***Note: We will reuse the same lambda_ij value in our final results**

We see that our RMSE has indeed improved from Model 4 and by a more substantial amount.

Let us check our ratings again



So we are back to a slightly modified list with the Movie " "Fists in the Pocket (I Pugni in tasca) (1965)" disappearing from our "Top 10" list.

Let us check our Prediction for this movie now,

Table 11: User Details

	userId	movieId	title	rating
2004881	14598	40833	Fists in the Pocket (I Pugni in tasca) (1965)	4

Table 12: alpha_ij for User

userId	alpha_ij
14598	0.5642827

Table 13: beta_ij value for Movie

movieId	beta_ij
40833	0.0870607

The Predicted Value of the Rating is

$Y_{i,j} = \mu + \alpha_{i(\lambda_{ij})} + \beta_{j(\lambda_{ij})} = 3.512456 + 0.5642827 + 0.0870607 = 4.163799$ which while better than the previous values is not very close to the Actual Rating of 4.0.

We also have some other unfortunate Side Effects as a new Movie "Mickey (2003)" with a single user rating has now surfaced among the Top 10 highly rated movies. This seems to be happening because of the fact that we are using a single Regularisation factor for both User Effects and Movie Effects.

***Note: We can get much closer to the actual prediction for outliers using a more complex regularisation scheme which is recursively hierarchical. It is computationally expensive and**

quite complex for the improvement it offers in the overall RMSE values. So it is not presented here.

From this point onwards, we will start developing on top of what has already been demonstrated in the Course and accompanying Text Book.

2.4.8 Model 6 - User Effects (R), Movie Effects (R) & Genre Effects (R)

As intuitive human behaviour, We can understand that genre have an important bearing on user choices and ratings.

Genre preferences could be analysed at various levels.

1. Overall - Specified at the level of the entire population. A superset of preferences. Not specific to any user or movie.
2. User specific - The genre preferences of individual users.
3. Movie specific - Applicability of user genre preferences to a movie.
    1. A movie may still find favour with the audience while not belonging to a popular genre or vice versa.
    2. Multiple genre can be attached to the movie with different levels of applicability for each.

As a first step, let us try to understand how the genre influences overall user rating behaviour. Since there is no direct coding of a user's genre preferences in our dataset, we will need to extract and extrapolate the same from the "genres" and "rating" fields. We will try to predict users' preference for a particular genre by looking at their ratings for various movies with different genre and then trying to infer their preferences for specific genre. For this to work we will need to identify the individual genre attached to each movie.

Since there is no way for us to know for sure about whether the rating accorded by a user to a movie is specific to some genre associated with the movie, we assume that the rating provided by the user for the movie applies to all the genre associated with that movie and we extrapolate (replicate) this rating to all genre associated with that movie. This is an important assumption and will introduce some error into our predictions. However, without this assumption, we will not be able to proceed with the rest of our Analysis.

This leads to another question, Why should we not split the rating evenly between all of the genre associated with a movie? Should we do this, splits will not be comparable between movies or users as every movie is associated with a variable number of genre ranging from 0 to 8.

Since the genre rating for every Observation is split among the genre attached to the movie equally, averaging them back together recreates the same value. However, we are interested in understanding whether we can glean some additional user preferences by splitting them. What we will seek to do is find out whether users rate any particular genre better than others on average. This will give us some insight into users' preferences for various genre.

To understand the impact of various genre on user rating behaviour. The features we will choose will be Sum, Count and Mean for each genre.

Let us define for each genre k,

- "Individual Genre Sum" $\sum_{i,j,k} y_{i,j,k}$ as the sum of extrapolated ratings for all users for all movies that belong to that particular genre k
- "Individual Genre Count" $\sum_{i,j,k}(y_{i,j,k} \neq 0)$ as the count of extrapolated ratings by all users for all movies that belong to that particular genre k
- "Individual Genre Mean" as $\mu_{g,k}$ = Individual Genre Sum/Individual Genre Count = $\sum_{i,j,k} y_{i,j,k} / \sum_{i,j,k}(y_{i,j,k} \neq 0)$
- "Overall Genre Mean" as μ_g = Sum of all Individual Genre Sums/ Sum of all Individual Genre Counts = $(\sum_k^g \sum_{i,j,k} y_{i,j,k}) / (\sum_k^g \sum_{i,j,k} y_{i,j,k} \neq 0)$
- "Individual Genre Difference from Overall Genre Mean" = $\mu_{g,k} - \mu_g$
- "Genre Significance" $(\mu_{g,k} - \mu) * \sum_{i,j,k} y_{i,j,k}$ as the Product of the Individual Genre Sum and the Individual Genre Difference from the Overall Genre Mean. This is primarily to understand how much variability the genre will contribute at the overall level. Larger positive and negative values indicate more significant contributions towards the variability.

For us to build a solution for this model, we will need to modify the "edx_cv_train_set" as we cannot use it as-is. The genre are all combined together in one field. We will start by splitting up the same. We will create a new dataset called "genre_assignment" and populate the individual genre column with a 1 if the genre is applicable for a rating or a 0 otherwise. We will also define and compute K as the number of genre attached to each movie. This will help us normalise the ratings later. We will use the new dataset created to replicate the rating in further computations.

The procedure to be followed should get clearer for the Reader as we go along. Please check accompanying R Code for more granular detail. The comments in the Code have some important observations around string operations. The Reader is advised to read through them to understand how these operations work.

To check our work, We will need to compare our predictions with our Validation set "edx_cv_test_set". The Validation set will also need to be modified to split the genre so that we can compare our predictions based on the genre associated with each movie. We will create a new dataset called "genre_assignment_test" by following the exact same procedure as we did with the "edx_cv_train_set".

The following code is more complex than what we have seen so far. The comments should help follow the code.

```
# There is probably a much better way to do this than what has been done. An
    exercise for the Reader who has more expertise in R.
# Thanks to the authors of the Articles below for providing the right guidance
# https://stackoverflow.com/questions/38708529/how-to-use-str-split-in-r
    #38708610
# https://stringr.tidyverse.org/reference/str_detect.html


# Create a list of genre, used whenever we need to address the genre
    collectively. Genre for Sci-Fi and Film-Noir have been changed to Sci_Fi
    and Film_Noir respectively. This is done as R otherwise interprets the
    Symbol for the Hyphen (-) as the Minus/Subtraction Operator

genre_list <- c( "Action",  "Adventure",    "Animation",    "Children",     "
    Comedy",   "Crime",    "Documentary",  "Drama",    "Fantasy",  "Film_Noir"
    ,    "Mystery",  "Horror",   "Thriller",     "Musical",  "Sci_Fi",   "
    Romance",  "War",  "Western" )

# String Detection and Separation is a computationally expensive, single
    threaded operation and takes a long time, we will retrieve the file from
    disk if it exists.

genre_assignment_file <- "genre_assignment.RData"

if(file.exists(genre_assignment_file)){
      load(genre_assignment_file)

      genre_assignment <- genre_assignment %>%
      mutate(userId = as.factor(userId), movieId = as.factor(movieId))

      rm(genre_assignment_file)

} else {


# Genre are all combined together in the "genres" column in a single string.
    Individual Genres within the string are separated using the pipe (|)
```

```
    symbol.

# Split genre individually for further Analysis. Use the "stringr" library and
    str_split function.

# The pipe (|) symbol happens to have its own meaning in Regular Expressions
    so must be escaped using a double backslash.

# Assign genre individually to each movie based on the Split Strings. Each
    genre is tracked individually in its own column. Assign 1 if genre matches
    , 0 if it does not. Based on the code below, the "mutate" function from
    the "Tidyverse" Package will automatically create a Column if it does not
    exist already. If the Column exists, it will just update the corresponding
     row. Please note that the Column names for Genres Sci-Fi and Film-Noir
    have been changed to Sci_Fi and Film_Noir respectively. This is done so
    that it is easier to address the columns as R otherwise interprets the
    Symbol for the Hyphen (-) as the Minus/Subtraction Operator.

# Add a genre "count" column for counting & tracking the number of genre
    attached to each movie, this will help us normalise the genre treatment
    effects values later.


genre_assignment <-  edx_cv_train_set %>%
  mutate("individual_genre" = str_split(genres, "\\|")) %>%
  mutate (Action  =  ifelse (str_detect ( individual_genre, "Action"), 1 , 0 )
      )   %>%
  mutate (Adventure  =  ifelse (str_detect ( individual_genre, "Adventure") ,
     1 , 0 ) )  %>%
  mutate (Animation  =  ifelse (str_detect ( individual_genre, "Animation") ,
     1 , 0 ) )  %>%
  mutate (Children  =  ifelse (str_detect ( individual_genre, "Children") , 1
     , 0 ) )  %>%
  mutate (Comedy  =  ifelse (str_detect ( individual_genre, "Comedy") , 1 , 0
     ) )  %>%
  mutate (Crime  =  ifelse (str_detect ( individual_genre, "Crime") , 1 , 0 )
     )  %>%
  mutate (Documentary  =  ifelse (str_detect ( individual_genre, "Documentary"
     ) , 1 , 0 ) )  %>%
  mutate (Drama  =  ifelse (str_detect ( individual_genre, "Drama") , 1 , 0 )
     )  %>%
  mutate (Fantasy  =  ifelse (str_detect ( individual_genre, "Fantasy") , 1 ,
     0 ) )  %>%
  mutate (Film_Noir  =  ifelse (str_detect ( individual_genre, "Film-Noir") ,
     1 , 0 ) )  %>%
  mutate (Mystery  =  ifelse (str_detect ( individual_genre, "Mystery") , 1 ,
     0 ) )  %>%
  mutate (Horror  =  ifelse (str_detect ( individual_genre, "Horror") , 1 , 0
     ) )  %>%
  mutate (Thriller  =  ifelse (str_detect ( individual_genre, "Thriller") , 1
     , 0 ) )  %>%
  mutate (Musical  =  ifelse (str_detect ( individual_genre, "Musical") , 1 ,
     0 ) )  %>%
  mutate (Sci_Fi  =  ifelse (str_detect ( individual_genre, "Sci-Fi") , 1 , 0
```

```
    ) )  %>%
  mutate (Romance  =  ifelse (str_detect ( individual_genre, "Romance") , 1 ,
     0 ) )  %>%
  mutate (War  =  ifelse (str_detect ( individual_genre, "War") , 1 , 0 ) )
     %>%
  mutate (Western  =  ifelse (str_detect ( individual_genre, "Western") , 1 ,
     0 ) ) %>%
  mutate("genre_count" = Action + Adventure + Animation + Children + Comedy +
     Crime + Documentary + Drama + Fantasy + Film_Noir + Mystery + Horror +
     Thriller + Musical + Sci_Fi + Romance + War + Western )

# Save file in R's native Format as it offers excellent compression. CSV is
    almost 4 times larger and the number of rows are more than most
    spreadsheet programs can safely handle, so saving it in CSV format is of
    limited utility.

save(genre_assignment, file = "genre_assignment.RData")

}
rm(genre_assignment_test_file) # Remove variable used to hold file name

# Let us modify our edx_cv_test_set beforehand to test our Predictions.

# We will repeat the Operations for our edx_cv_test_set. We will import it
    from the Disk if the file exists as it a computationally expensive
    operation and takes some time.

genre_assignment_test_file <- "genre_assignment_test.RData"

if(file.exists(genre_assignment_test_file)){
      load(genre_assignment_test_file)
      genre_assignment_test <- genre_assignment_test %>%
      mutate(userId = as.factor(userId), movieId = as.factor(movieId))
    rm(genre_assignment_test_file)
} else {

# Split consolidated genre Field into individual genre

# Create Genre Column and Populate the individual Genre Column with a 1 if the
     Genre is applicable for the movie or a 0 otherwise.

# Add a Count column to track number of genre attached to the movie

genre_assignment_test <- edx_cv_test_set %>%
  mutate("individual_genre" = str_split(genres, "\\|")) %>%
  mutate (Action  =  ifelse (str_detect ( individual_genre, "Action"), 1 , 0 )
      )   %>%
  mutate (Adventure  =  ifelse (str_detect ( individual_genre, "Adventure") ,
     1 , 0 ) )  %>%
  mutate (Animation  =  ifelse (str_detect ( individual_genre, "Animation") ,
     1 , 0 ) )  %>%
  mutate (Children  =  ifelse (str_detect ( individual_genre, "Children") , 1
     , 0 ) )  %>%
  mutate (Comedy  =  ifelse (str_detect ( individual_genre, "Comedy") , 1 , 0
```

```
    ) )  %>%
  mutate (Crime  =  ifelse (str_detect ( individual_genre, "Crime") , 1 , 0 )
    )  %>%
  mutate (Documentary  =  ifelse (str_detect ( individual_genre, "Documentary"
    ) , 1 , 0 ) )  %>%
  mutate (Drama  =  ifelse (str_detect ( individual_genre, "Drama") , 1 , 0 )
    )  %>%
  mutate (Fantasy  =  ifelse (str_detect ( individual_genre, "Fantasy") , 1 ,
    0 ) )  %>%
  mutate (Film_Noir  =  ifelse (str_detect ( individual_genre, "Film-Noir") ,
    1 , 0 ) )  %>%
  mutate (Mystery  =  ifelse (str_detect ( individual_genre, "Mystery") , 1 ,
    0 ) )  %>%
  mutate (Horror  =  ifelse (str_detect ( individual_genre, "Horror") , 1 , 0
    ) )  %>%
  mutate (Thriller  =  ifelse (str_detect ( individual_genre, "Thriller") , 1
    , 0 ) )  %>%
  mutate (Musical  =  ifelse (str_detect ( individual_genre, "Musical") , 1 ,
    0 ) )  %>%
  mutate (Sci_Fi  =  ifelse (str_detect ( individual_genre, "Sci-Fi") , 1 , 0
    ) )  %>%
  mutate (Romance  =  ifelse (str_detect ( individual_genre, "Romance") , 1 ,
    0 ) )  %>%
  mutate (War  =  ifelse (str_detect ( individual_genre, "War") , 1 , 0 ) )
    %>%
  mutate (Western  =  ifelse (str_detect ( individual_genre, "Western") , 1 ,
    0 ) ) %>%
  mutate("genre_count" = Action + Adventure + Animation + Children + Comedy +
    Crime + Documentary + Drama + Fantasy + Film_Noir + Mystery + Horror +
    Thriller + Musical + Sci_Fi + Romance + War + Western )

# Save file to disk
save(genre_assignment_test, file="genre_assignment_test.RData")
}


rm(genre_assignment_file, genre_assignment_test_file)  # Remove variables used
    to hold file name
```

Let us now, try to compute the Overall Genre Mean μ_g and the Individual Genre Mean $\mu_{g,k}$ for each genre and build some tables to visualise how they vary.

```
# Create a new dataframe that will be populated with the Genre Specific
   Ratings going forward. We will use the dataframe created in the earlier
   section for identifying the Genre associated with each movie. The new
   dataframe only retains numerical values for ease of operations, speed up
   computation and reduce memory usage by retaining only necessary fields.

genre_assignment_matrix <- genre_assignment %>%
            select (userId, movieId, rating, all_of(genre_list), genre_count,
               timestamp)

# Replicate the User rating for each movie to the individual Genre Columns as
   applicable. Use a temporary dataframe to store the results
```

```
temp <- genre_assignment_matrix %>% ungroup %>% select(rating, all_of(genre_
    list)) %>%
            mutate (  Action = ifelse ( Action == 1 , rating , 0 )  )  %>%
            mutate (  Adventure = ifelse ( Adventure == 1 , rating , 0 )  )
                %>%
            mutate (  Animation = ifelse ( Animation == 1 , rating , 0 )  )
                %>%
            mutate (  Children = ifelse ( Children == 1 , rating , 0 )  )  %>%
            mutate (  Comedy = ifelse ( Comedy == 1 , rating , 0 )  )  %>%
            mutate (  Crime = ifelse ( Crime == 1 , rating , 0 )  )  %>%
            mutate (  Documentary = ifelse ( Documentary == 1 , rating , 0 )
                )  %>%
            mutate (  Drama = ifelse ( Drama == 1 , rating , 0 )  )  %>%
            mutate (  Fantasy = ifelse ( Fantasy == 1 , rating , 0 )  )  %>%
            mutate (  Film_Noir = ifelse ( Film_Noir == 1 , rating , 0 )  )
                %>%
            mutate (  Mystery = ifelse ( Mystery == 1 , rating , 0 )  )  %>%
            mutate (  Horror = ifelse ( Horror == 1 , rating , 0 )  )  %>%
            mutate (  Thriller = ifelse ( Thriller == 1 , rating , 0 )  )  %>%
            mutate (  Musical = ifelse ( Musical == 1 , rating , 0 )  )  %>%
            mutate (  Sci_Fi = ifelse ( Sci_Fi == 1 , rating , 0 )  )  %>%
            mutate (  Romance = ifelse ( Romance == 1 , rating , 0 )  )  %>%
            mutate (  War = ifelse ( War == 1 , rating , 0 )  )  %>%
            mutate (  Western = ifelse ( Western == 1 , rating , 0 )) %>%
            select ( all_of(genre_list))

# Compute the Genre Mean by manually calculating the Individual Genre Sums and
     dividing them by the number of ratings. Using "colMeans" will provide
    erroneous results as it will also consider rows where no rating has been
    assigned. So this computation needs to be done manually.

individual_genre_sums <- colSums(temp) # compute sum of ratings for each
    column
individual_genre_sums <- round(individual_genre_sums, digits = 0) # Round Sum
    to remove decimal values
individual_genre_count <- colSums(temp != 0) # for each column count rows
    where a rating is available
# compute & print Overall Genre Mean & Overall Mean for comparison

mu_g <- round(sum(individual_genre_sums)/sum(individual_genre_count),digits =
    5)

print(c("Overall Genre Mean (mu_g): ", round(mu_g, digits = 5)), quote = FALSE
    , justify = "Left")
```

```
## [1] Overall Genre Mean (mu_g):  3.52691
```

```
print(c("Overall Mean (mu): ", round(mu, digits = 5)), quote = FALSE, justify
    = "Left")
```

```
## [1] Overall Mean (mu):  3.51246
```

```
print(c("Difference (mu_g - mu): ", round(mu_g -mu, digits = 5)), quote =
    FALSE, justify = "Left")
```

```
## [1] Difference (mu_g - mu):  0.01445
```

```
# compute Individual Genre Mean and Difference from Overall Genre Mean and
    Store in a common Table,

genre_summary <- data.frame(individual_genre_sums,individual_genre_count)
genre_summary <- genre_summary %>% mutate(individual_genre_mean = (individual_
    genre_sums/individual_genre_count), difference_from_genre_mean = (
    individual_genre_mean - mu_g), genre_significance = round(difference_from_
    genre_mean*individual_genre_count, digits = 0))



# Remove Temp Dataset from Memory
rm(temp)

# Remove Data that is no longer required
rm(individual_genre_count, individual_genre_sums)
```

Table 14: Genre Summary

Genre Name	Genre Sum	Genre Count	Genre Mean	Difference from Overall Genre Mean	Genre Significance
Action	7883838	2304395	3.42122	-0.10569	-243556
Adventure	6001488	1717992	3.49332	-0.03359	-57715
Animation	1513181	420349	3.59982	0.07291	30648
Children	2270196	664146	3.41822	-0.10869	-72187
Comedy	10954984	3187285	3.43709	-0.08982	-286283
Crime	4379592	1194740	3.66573	0.13882	165852
Documentary	316410	83664	3.78191	0.25500	21335
Drama	12925520	3518986	3.67308	0.14617	514373
Fantasy	2917592	833130	3.50196	-0.02495	-20783
Film_Noir	427999	106704	4.01109	0.48418	51664
Mystery	1882060	511788	3.67742	0.15051	77030
Horror	2034728	622183	3.27030	-0.25661	-159655
Thriller	7342984	2093191	3.50803	-0.01888	-39512
Musical	1388243	389569	3.56354	0.03663	14268
Sci_Fi	4099648	1207309	3.39569	-0.13122	-158422
Romance	5479066	1541727	3.55385	0.02694	41534
War	1739682	460203	3.78025	0.25334	116587
Western	605899	170406	3.55562	0.02871	4892

Table 15: Genre Summary Sorted by Count

Genre Name	Count
Drama	3518986
Comedy	3187285

Genre Name	Count
Action	2304395
Thriller	2093191
Adventure	1717992
Romance	1541727
Sci_Fi	1207309
Crime	1194740
Fantasy	833130
Children	664146
Horror	622183
Mystery	511788
War	460203
Animation	420349
Musical	389569
Western	170406
Film_Noir	106704
Documentary	83664

Table 16: Genre Summary Sorted by Difference from the Genre Mean

Genre Name	Difference from Overall Genre Mean
Film_Noir	0.48418
Documentary	0.25500
War	0.25334
Mystery	0.15051
Drama	0.14617
Crime	0.13882
Animation	0.07291
Musical	0.03663
Western	0.02871
Romance	0.02694
Thriller	-0.01888
Fantasy	-0.02495
Adventure	-0.03359
Comedy	-0.08982
Action	-0.10569
Children	-0.10869
Sci_Fi	-0.13122
Horror	-0.25661

Table 17: Genre Significance - Product of Genre Count & Difference from Genre Mean

Genre Name	Genre Significance
Drama	514373
Crime	165852
War	116587
Mystery	77030
Film_Noir	51664

Genre Name	Genre Significance
Romance	41534
Animation	30648
Documentary	21335
Musical	14268
Western	4892
Fantasy	-20783
Thriller	-39512
Adventure	-57715
Children	-72187
Sci_Fi	-158422
Horror	-159655
Action	-243556
Comedy	-286283

Looking at the Tables, we can get a quick view of the various genre in terms of movies made and Averages of ratings provided by users for the same.

Drama and Comedy seem to be among movies made the most, as expected Documentary and Film-Noir are more niche.

We can see that the Overall Genre Mean μ_g is close in value to μ which serves as a rough indicator that our computations are on track.

We can also see that Genre like Film-Noir, Documentary, War, Mystery, Drama and Crime are more highly rated as $\mu_{g,k} > \mu_g$ for these Genre . Similarly, Genre like Action, Children, Sci-Fi and Horror are rated lower as $\mu_{g,k} < \mu_g$.

In terms of Genre Significance, We can see that Drama has the largest Positive Variability and that Comedy and Action have among the largest Negative Variability.

Now that we have some idea of the Popularity of various Genre. Let us check if we can use this information to better our Predictions.

We will now build our next model that takes User Specific Ratings for different genre into account.

If we define $g_{i,k}$ as the genre applicable for user $i's$ rating of genre k, and there is a list K of values of genre attached to the Movie

The Model for the prediction can be defined as

$Y_{i,j} = \mu + \alpha_{i(\lambda_{ij})} + \beta_{j(\lambda_{ij})} + \frac{1}{K}\sum_{k=1}^{K} x_{i,k}\beta_{i,k} + \varepsilon_{i,j}$

where $x_{i,k} = 1$ if $g_{i,k}$ is applicable for the Movie, $x_{i,k} = 0$ otherwise

Symbols have their usual meaning

$Y_{i,j}$ is the predicted rating

μ is the Mean value of all known ratings

$\alpha_{i(\lambda_{ij})}$ is the Regularised Treatment Effect for each user i .

$\beta_{j(\lambda_{ij})}$ is the Regularised Treatment Effect for each movie j .

K is the number of genre attached to the movie, If no genre are attached, then $K = 1$

$\beta_{i,k}$ is the Genre User Treatment Effect for genre k for user i

$\epsilon_{i,j}$ is the random error

Let us illustrate with an example so that it is a bit clearer, Let us choose the entry "Toy Story (1995)" which is the first entry in our Movie List. It has multiple genre attached and is also rated by many users.

```
knitr::kable(movie_list %>% filter(title == "Toy Story (1995)"), "simple",
    align = "cllcc", caption = "Genre attached to Movie - Toy Story (1995) ")
```

Table 18: Genre attached to Movie - Toy Story (1995)

movieId	title	genres	avg_rating	number_of_ratings
1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	3.927638	23790

As can be seen, the movie has 5 genre attached to it viz. Adventure, Animation, Children, Comedy & Fantasy.

In our Example,

$K = 5$

$x_{j,k} = 1$ for each genre k in the genre list = [Adventure, Animation, Children, Comedy & Fantasy]

$x_{j,k} = 0$ otherwise.

Genre User Effects are computed in a manner similar to the User Effects. For each user i, we will compute a Genre User Effect for a particular genre k based on observed ratings . Let us call it $\beta_{i,k}$ We will then apply this value to each movie rating j that we need to predict.

We define the Genre User Effect as

$\beta_{i,k} = (\frac{1}{N}_{j,k} \sum_{j,k}^{N_{j,k}} y_{i,j,k}) - (\frac{1}{N}_{j} \sum_{j}^{N_j} y_{i,j})$

Where

$\beta_{i,k}$ - Genre User Effect for user i for genre k

$N_{j,k}$ - Number of movies rated by the user i for genre k

$y_{i,j,k}$ - Extrapolated rating for user i for movie j and genre k

$\frac{1}{N}_{j,k} \sum_{j,k}^{N_{j,k}} y_{i,j,k}$ is the average rating accorded by user i to all movies having genre k in them

N_j and $y_{i,j}$ have their usual meaning,

N_j - Number of movies rated by user i

$y_{i,j}$ - Rating provided by user i for movie j

$\frac{1}{N}_{j} \sum_{j}^{N_j} y_{i,j}$ is just the average rating accorded by the user i to all movies

When compared to our Inital LRE,

$y_{i,j} = \mu + \alpha_i + \beta_j + \sum_{k=1}^{K} x_{i,k}\beta_{i,k} + \beta_{i,t} + \beta_{j,t} + \varepsilon_{ij}$

we have added our fourth term for Regression Coefficients $\sum_{k=1}^{K} x_{i,k}\beta_{i,k}$. We use the plural because there is one Regression Coefficient for each genre. We have chosen to normalise the Regression Coefficient for each genre by dividing it by number of genre attached to the movie because we extrapolated (replicated) the "rating" field to all genre associated with that movie

Let us now build the Genre User Effect for each user for each genre, which is likely to help us predict their genre preferences.

```
# For each user compute their average rating for every genre. We then compute
    the difference between a User's Average Rating across all genres against
    the genre under analysis. If for some reason, the user has not rated even
    a single movie belonging to a particular genre, then ensure that genre
    specific ratings are not considered for such cases. The "ifelse"
    statements in the denominators are necessary to avoid divide by 0 problems
    .

# Code can be built using a single "summarise" statement. However, it is
    deliberately broken into each genre to avoid too much complexity and to
    keep it modular and clear. It also helps avoid lookup errors due to typos
    as each genre has its own statement and cannot lookup values from another
    genre erroneously.

# Use "b" rather than "beta" as prefix to genre names to keep Variable names
    more concise and easier to work with

genre_action <- genre_assignment_matrix %>% group_by(userId) %>% summarise ("
    mean_rating" = mean(rating)  , "count_action" = sum(Action == 1), "sum_
    action" = sum((ifelse(Action==0,0,rating)))  , "mean_action" = sum_action/
    (ifelse(count_action==0,1,count_action)), "b_action"  = ifelse(sum_action
    ==0,0,(mean_action - mean_rating ))) %>% select(userId, mean_action, b_
    action)

genre_adventure <- genre_assignment_matrix %>% group_by(userId) %>% summarise
    ("mean_rating" = mean(rating)  , "count_adventure" = sum(Adventure == 1),
    "sum_adventure" = sum((ifelse(Adventure==0,0,rating)))  , "mean_adventure"
     = sum_adventure/(ifelse(count_adventure==0,1,count_adventure)), "b_
    adventure"  = ifelse(sum_adventure==0,0,(mean_adventure - mean_rating )))
    %>% select(userId, mean_adventure, b_adventure)

genre_animation <- genre_assignment_matrix %>% group_by(userId) %>% summarise
    ("mean_rating" = mean(rating)  , "count_animation" = sum(Animation == 1),
    "sum_animation" = sum((ifelse(Animation==0,0,rating)))  , "mean_animation"
     = sum_animation/(ifelse(count_animation==0,1,count_animation)), "b_
    animation"  = ifelse(sum_animation==0,0,(mean_animation - mean_rating  )))
     %>% select(userId, mean_animation, b_animation)

genre_children <- genre_assignment_matrix %>% group_by(userId) %>% summarise (
    "mean_rating" = mean(rating)  , "count_children" = sum(Children == 1), "
    sum_children" = sum((ifelse(Children==0,0,rating)))  , "mean_children" =
    sum_children/(ifelse(count_children==0,1,count_children)), "b_children"  =
     ifelse(sum_children==0,0,(mean_children - mean_rating ))) %>% select(
    userId, mean_children, b_children)

genre_comedy <- genre_assignment_matrix %>% group_by(userId) %>% summarise ("
    mean_rating" = mean(rating)  , "count_comedy" = sum(Comedy == 1), "sum_
    comedy" = sum((ifelse(Comedy==0,0,rating)))  , "mean_comedy" = sum_comedy/
    (ifelse(count_comedy==0,1,count_comedy)), "b_comedy"  = ifelse(sum_comedy
    ==0,0,(mean_comedy - mean_rating ))) %>% select(userId, mean_comedy, b_
    comedy)

genre_crime <- genre_assignment_matrix %>% group_by(userId) %>% summarise ("
```

```
    mean_rating" = mean(rating)  , "count_crime" = sum(Crime == 1), "sum_crime
    " = sum((ifelse(Crime==0,0,rating)))  , "mean_crime" = sum_crime/(ifelse(
    count_crime==0,1,count_crime)), "b_crime"  = ifelse(sum_crime==0,0,(mean_
    crime - mean_rating  ))) %>% select(userId, mean_crime, b_crime)

genre_documentary <- genre_assignment_matrix %>% group_by(userId) %>%
    summarise ("mean_rating" = mean(rating) , "count_documentary" = sum(
    Documentary == 1), "sum_documentary" = sum((ifelse(Documentary==0,0,rating
    )))  , "mean_documentary" = sum_documentary/(ifelse(count_documentary
    ==0,1,count_documentary)), "b_documentary"  = ifelse(sum_documentary
    ==0,0,(mean_documentary - mean_rating  ))) %>% select(userId, mean_
    documentary, b_documentary)

genre_drama <- genre_assignment_matrix %>% group_by(userId) %>% summarise ("
    mean_rating" = mean(rating), "count_drama" = sum(Drama == 1), "sum_drama"
    = sum((ifelse(Drama==0,0,rating)))  , "mean_drama" = sum_drama/(ifelse(
    count_drama==0,1,count_drama)), "b_drama"  = ifelse(sum_drama==0,0,(mean_
    drama - mean_rating ))) %>% select(userId, mean_drama, b_drama)

genre_fantasy <- genre_assignment_matrix %>% group_by(userId) %>% summarise ("
    mean_rating" = mean(rating)  , "count_fantasy" = sum(Fantasy == 1), "sum_
    fantasy" = sum((ifelse(Fantasy==0,0,rating)))  , "mean_fantasy" = sum_
    fantasy/(ifelse(count_fantasy==0,1,count_fantasy)), "b_fantasy"  = ifelse(
    sum_fantasy==0,0,(mean_fantasy - mean_rating ))) %>% select(userId, mean_
    fantasy, b_fantasy)

genre_film_noir <- genre_assignment_matrix %>% group_by(userId) %>% summarise
    ("mean_rating" = mean(rating)  , "count_film_noir" = sum(Film_Noir == 1),
    "sum_film_noir" = sum((ifelse(Film_Noir==0,0,rating)))  , "mean_film_noir"
     = sum_film_noir/(ifelse(count_film_noir==0,1,count_film_noir)), "b_film_
    noir"  = ifelse(sum_film_noir==0,0,(mean_film_noir - mean_rating ))) %>%
    select(userId, mean_film_noir, b_film_noir)

genre_mystery <- genre_assignment_matrix %>% group_by(userId) %>% summarise ("
    mean_rating" = mean(rating) , "count_mystery" = sum(Mystery == 1), "sum_
    mystery" = sum((ifelse(Mystery==0,0,rating)))  , "mean_mystery" = sum_
    mystery/(ifelse(count_mystery==0,1,count_mystery)), "b_mystery"  = ifelse(
    sum_mystery==0,0,(mean_mystery - mean_rating  ))) %>% select(userId, mean_
    mystery, b_mystery)

genre_horror <- genre_assignment_matrix %>% group_by(userId) %>% summarise ("
    mean_rating" = mean(rating) , "count_horror" = sum(Horror == 1), "sum_
    horror" = sum((ifelse(Horror==0,0,rating)))  , "mean_horror" = sum_horror/
    (ifelse(count_horror==0,1,count_horror)), "b_horror"  = ifelse(sum_horror
    ==0,0,(mean_horror - mean_rating ))) %>% select(userId, mean_horror, b_
    horror)

genre_thriller <- genre_assignment_matrix %>% group_by(userId) %>% summarise (
    "mean_rating" = mean(rating)  , "count_thriller" = sum(Thriller == 1), "
    sum_thriller" = sum((ifelse(Thriller==0,0,rating)))  , "mean_thriller" =
    sum_thriller/(ifelse(count_thriller==0,1,count_thriller)), "b_thriller"  =
     ifelse(sum_thriller==0,0,(mean_thriller - mean_rating ))) %>% select(
    userId, mean_thriller, b_thriller)
```

```
genre_musical <- genre_assignment_matrix %>% group_by(userId) %>% summarise ("
    mean_rating" = mean(rating)  , "count_musical" = sum(Musical == 1), "sum_
    musical" = sum((ifelse(Musical==0,0,rating)))  , "mean_musical" = sum_
    musical/(ifelse(count_musical==0,1,count_musical)), "b_musical"  = ifelse(
    sum_musical==0,0,(mean_musical - mean_rating ))) %>% select(userId, mean_
    musical, b_musical)

genre_sci_fi <- genre_assignment_matrix %>% group_by(userId) %>% summarise ("
    mean_rating" = mean(rating)  , "count_sci_fi" = sum(Sci_Fi == 1), "sum_sci
    _fi" = sum((ifelse(Sci_Fi==0,0,rating)))  , "mean_sci_fi" = sum_sci_fi/(
    ifelse(count_sci_fi==0,1,count_sci_fi)), "b_sci_fi"  = ifelse(sum_sci_fi
    ==0,0,(mean_sci_fi - mean_rating   ))) %>% select(userId, mean_sci_fi, b_
    sci_fi)

genre_romance <- genre_assignment_matrix %>% group_by(userId) %>% summarise ("
    mean_rating" = mean(rating)  , "count_romance" = sum(Romance == 1), "sum_
    romance" = sum((ifelse(Romance==0,0,rating)))  , "mean_romance" = sum_
    romance/(ifelse(count_romance==0,1,count_romance)), "b_romance"  = ifelse(
    sum_romance==0,0,(mean_romance - mean_rating  ))) %>% select(userId, mean_
    romance, b_romance)

genre_war <- genre_assignment_matrix %>% group_by(userId) %>% summarise ("mean
    _rating" = mean(rating) , "count_war" = sum(War == 1), "sum_war" = sum((
    ifelse(War==0,0,rating)))  , "mean_war" = sum_war/(ifelse(count_war==0,1,
    count_war)), "b_war"  = ifelse(sum_war==0,0,(mean_war - mean_rating )))
    %>% select(userId, mean_war, b_war)

genre_western <- genre_assignment_matrix %>% group_by(userId) %>% summarise ("
    mean_rating" = mean(rating)  , "count_western" = sum(Western == 1), "sum_
    western" = sum((ifelse(Western==0,0,rating)))  , "mean_western" = sum_
    western/(ifelse(count_western==0,1,count_western)), "b_western"  = ifelse(
    sum_western==0,0,(mean_western - mean_rating   ))) %>% select(userId, mean
    _western, b_western)

# Create a Common dataframe for all genre coefficients to reduce repetitions
    and errors later. Order is slightly different because it is as per Genre
    Significance. Should not make any difference to the end result.

genre_user_summary <- genre_drama %>%
  left_join(genre_comedy, by='userId') %>%
  left_join(genre_action, by='userId') %>%
  left_join(genre_thriller, by='userId') %>%
  left_join(genre_adventure, by='userId') %>%
  left_join(genre_romance, by='userId') %>%
  left_join(genre_sci_fi, by='userId') %>%
  left_join(genre_crime, by='userId') %>%
  left_join(genre_fantasy, by='userId') %>%
  left_join(genre_children, by='userId') %>%
  left_join(genre_horror, by='userId') %>%
  left_join(genre_mystery, by='userId') %>%
  left_join(genre_war, by='userId') %>%
  left_join(genre_animation, by='userId') %>%
  left_join(genre_musical, by='userId') %>%
  left_join(genre_western, by='userId') %>%
```

```
  left_join(genre_film_noir, by='userId') %>%
  left_join(genre_documentary, by='userId')

# Extract all genre coefficients values for all Users
genre_effects_user <- genre_user_summary %>% select(userId, b_action, b_
    adventure, b_animation, b_children, b_comedy, b_crime, b_documentary, b_
    drama, b_fantasy, b_film_noir, b_horror, b_musical, b_mystery, b_romance,
    b_sci_fi, b_thriller, b_war, b_western)

# Extract all genre_mean values for all Users. We will use them later
genre_user_mean <- genre_user_summary %>% select(userId, mean_action, mean_
    adventure, mean_animation, mean_children, mean_comedy, mean_crime, mean_
    documentary, mean_drama, mean_fantasy, mean_film_noir, mean_horror, mean_
    musical, mean_mystery, mean_romance, mean_sci_fi, mean_thriller, mean_war,
     mean_western)

# Rmeove Individual Genre dataframes to save memory and avoid confusion
rm(genre_action,genre_adventure,genre_animation, genre_children, genre_comedy,
     genre_crime, genre_documentary, genre_drama, genre_fantasy, genre_film_
    noir, genre_horror, genre_musical, genre_mystery, genre_romance, genre_sci
    _fi, genre_thriller, genre_war, genre_western)
```

*Note: Since the combination of user, genre and movie is unique, $\beta_{i,k}$ needs to be computed every time for the validation set.

We will record the RMSE for this Model as "RMSE User Effects (R), Movie Effects (R) & Genre Effects User"

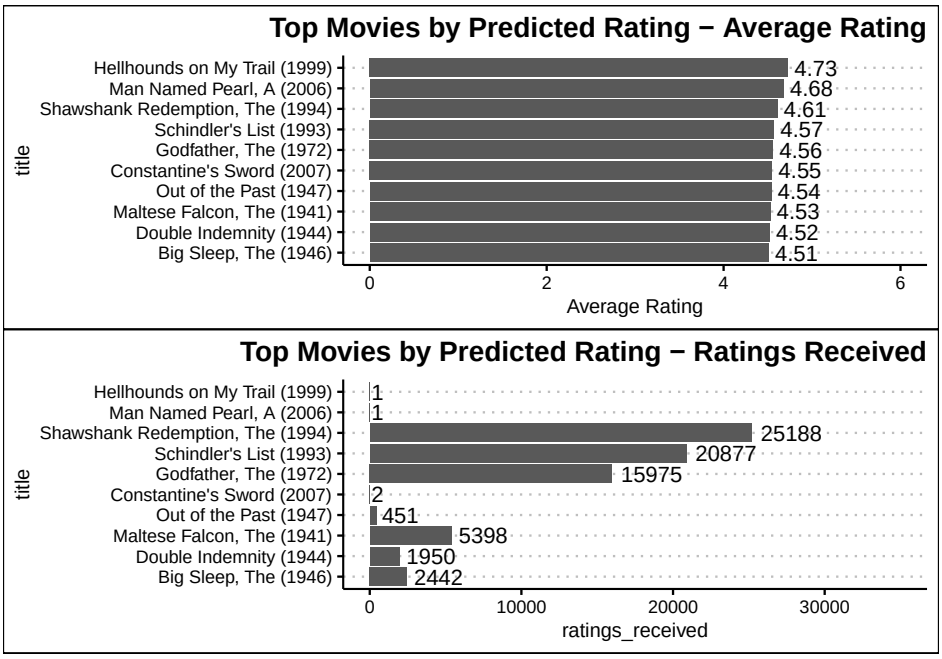
```
# Compute the Predicted Ratings taking into account the Regularised User
    Effects, Regularised Movie Effects, Genre Effects Overall and Genre User
    Effects

# The beta_ig values are computed based on the combination of User and Movie
    and the entries are unique, hence entries in the genre_assignment_matrix
    will not match with any entries in the edx_cv_test_set or genre_assigment_
    test set, hence beta_ig will need to be computed everytime for the genre_
    assigment_test set for cross validation.

predicted_ratings_genre_user <- genre_assignment_test %>%
  left_join(alpha_r_df_combined, by='userId') %>%
  left_join(beta_r_df_combined, by='movieId') %>%
  left_join(genre_effects_user, by = 'userId') %>%
  mutate(beta_ig = (b_drama*Drama + b_comedy*Comedy + b_action*Action + b_
      thriller*Thriller + b_adventure*Adventure + b_romance*Romance + b_sci_fi
      *Sci_Fi + b_crime*Crime + b_fantasy*Fantasy + b_children*Children + b_
      horror*Horror + b_mystery*Mystery + b_war*War + b_animation*Animation +
      b_musical*Musical + b_western*Western + b_film_noir*Film_Noir + b_
      documentary*Documentary)/(ifelse(genre_count==0,1,  genre_count  ))) %>%
       mutate("pred" = clamp(mu + alpha_ij +  beta_ij + beta_ig  ))  %>%   pull
          (pred)

# Record and Store RMSE with Genre User Effects included
rmse_genre_effects_user <- round(rmse(edx_cv_test_set$rating, predicted_
    ratings_genre_user),digits = 5)
```

```
# Remove Data that is no longer required
rm(predicted_ratings_genre_user)



# Print RMSE with Genre User Effects Included
print( c(" RMSE User, Movie & Genre Effects User: ", rmse_genre_effects_user),
     quote = FALSE, justify = "left")
```

```
## [1]  RMSE User, Movie & Genre Effects User:
## [2] 0.85505
```

We have some more improvement in our RMSE. We can see that the genre actually do code user preferences which we can extract. Let us now visit our "Top 10" List again to see how the Genre User Effects have in any way modified the list.

*Note: Once we incorporate the Genre User Effects, our Table sizes grow quite large and we must be careful not to try to manipulate all of the entries together. The Reader is cautioned to choose only the first few Entries after Sorting for generating the Plots.



As expected, the results seem to follow what we have already seen for User Effects and Movie Effects. The Addition of Genre Effects pushes up Movies which have a Higher Genre Rating compared to the Mean.

Let us now check if Regularisation can help us improve our predictions and our RMSE by shrinking outliers. We will use a different Regularisation Factor for the Genre User Effect. Since our definition of the Genre User Effect is fairly complex, we cannot use the Regularisation Schemes that we have used so far.

To quickly recap, our Genre User Effect is defined as

$\beta_{i,k} = (\frac{1}{N}_{j,k} \sum_{j,k}^{N_{j,k}} y_{i,j,k}) - (\frac{1}{N}_{j} \sum_{j}^{N_j} y_{i,j})$

And there is a Genre User Effect $\beta_{i,k}$ for each genre k for each user i. Depending on the list of genre attached to the movie j, this Effect varies even within each user's ratings. Trying to build a Regularisation Factor for each genre separately will lead to too much complexity.

Let us use a slightly different model for Regularisation.

Since our model of prediction with the Genre User Effect included is

$Y_{i,j} = \mu + \alpha_{i(\lambda_{ij})} + \beta_{j(\lambda_{ij})} + \frac{1}{K}\sum_{k=1}^{K} x_{i,k}\beta_{i,k} + \varepsilon_{i,j}$

We have a candidate in K which can be targeted for Regularisation. It is likely to provide us with some intermediate but acceptable results. So we will perform Regularisation against the value of K attached to each rating. This should be a lot simpler to compute.

If we define $\beta_{i,g} = \frac{1}{K}\sum_{k=1}^{K} x_{i,k}\beta_{i,k}$

Our aim for Regularisation will be to compute λ_{ig} such that

$\hat{\beta_{i,g(\lambda_{ig})}} = \frac{1}{\lambda_{ig}+K}\sum_{k=1}^{K} x_{i,k}\beta_{i,k}$

minimises the equation

$\sum_{i,j}\left(y_{i,j} - \mu - \alpha_{i(\lambda_{ij})} - \beta_{j(\lambda_{ij})} - \beta_{i,g(\lambda_{ig})}\right)^2$

It is easier to compute in R using iteration for different values of λ_{ig} than it is to define it Mathematically. Please have a look at the code and accompanying comments for explanation of how it can be done.

When compared to our Inital LRE,

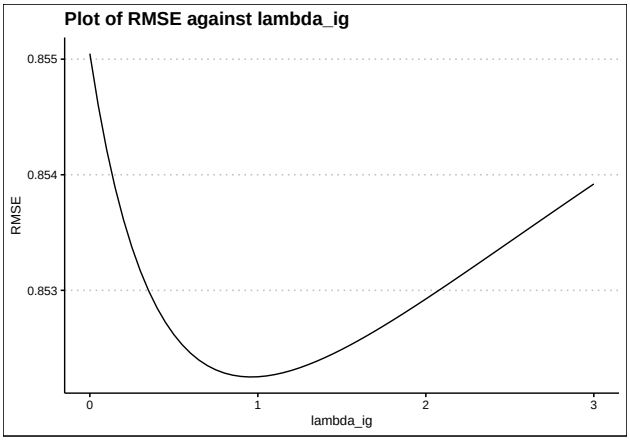
$y_{i,j} = \mu + \alpha_i + \beta_j + \sum_{k=1}^{K} x_{i,k}\beta_{i,k} + \beta_{i,t} + \beta_{j,t} + \varepsilon_{ij}$

we have optimised our fourth term of Regression Coefficients $\sum_{k=1}^{K} x_{i,k}\beta_{i,k}$. Again we use the plural because there is one Regression Coefficient for each genre.

*Note: We will use "lambda_ig" to represent λ_{ig} in the R Code.

```
# Let use first compute the Regularisation for Genre User Effects.

# Set up a sequence of Lambdas. Sequence Values chosen to avoid unnecessary
    computations. If required, Reader can check how the RMSE varies by
    changing the Sequence and Increments

lambdas_ig <- seq(0, 3, 0.05)

# Set up a function to calculate the RMSE for each value of lambdas_ig

rmses_genre_function_user <- function(lambda_ig) {

# Check our Predictions for each value of lambdas_ig
  predicted_ratings_genre_user <- mutate(genre_assignment_test, userId = as.
      factor(userId), movieId = as.factor(movieId)) %>%
  left_join(alpha_r_df_combined, by='userId') %>%
  left_join(beta_r_df_combined, by='movieId') %>%
  left_join(genre_effects_user, by = 'userId') %>%
    mutate(beta_ig = (b_drama*Drama + b_comedy*Comedy + b_action*Action + b_
        thriller*Thriller + b_adventure*Adventure + b_romance*Romance + b_sci_
        fi*Sci_Fi + b_crime*Crime + b_fantasy*Fantasy + b_children*Children +
        b_horror*Horror + b_mystery*Mystery + b_war*War + b_animation*
        Animation + b_musical*Musical + b_western*Western + b_film_noir*Film_
        Noir + b_documentary*Documentary)/(ifelse(genre_count==0,1,genre_count
        ) + lambda_ig )) %>%
    mutate("pred" = clamp(mu + alpha_ij +  beta_ij + beta_ig )) %>%
        pull(pred)

# return the RMSE value for the current value of lambdas_ig
```

```
  rmse_genre_regularised <- rmse(edx_cv_test_set$rating, predicted_ratings_
      genre_user)

}

rmses_genre_user <- sapply(lambdas_ig, rmses_genre_function_user)

# Let us record the Min RMSE and the Corresponding value of lambda_ig
min_lambda_ig <- lambdas_ig[which.min(rmses_genre_user)]
min_rmse_genre_user <- round(min(rmses_genre_user),digits = 5)

# Create Dataframe to compute and store Regularised Values of beta_ig - Valid
    only for edx_cv_train_set, genre_assignment, genre_assignment_matrix for
    the reason cited earlier.

# The beta_ig values are computed based on the combination of User and Movie
    and the entries are unique, hence entries in the genre_assignment_matrix
    will not match with any entries in the edx_cv_test_set or genre_assigment_
    test set, hence beta_ig will need to be computed everytime for the genre_
    assigment_test set for cross validation.

beta_ig_df_final <- genre_assignment_matrix %>%
    left_join(genre_effects_user, by='userId') %>%
    mutate(beta_ig =  (b_drama*Drama + b_comedy*Comedy + b_action*Action + b_
        thriller*Thriller + b_adventure*Adventure + b_romance*Romance + b_sci_
        fi*Sci_Fi + b_crime*Crime + b_fantasy*Fantasy + b_children*Children +
        b_horror*Horror + b_mystery*Mystery + b_war*War + b_animation*
        Animation + b_musical*Musical + b_western*Western + b_film_noir*Film_
        Noir + b_documentary*Documentary)/((ifelse(genre_count==0,1,genre_
        count))+ min_lambda_ig )) %>%
    ungroup() %>%
    select(userId,movieId,beta_ig)
```




```
## [1]  Min lambda gi:  0.95
```

```
## [1]  Min RMSE User, Movie & Genre User Effects - Regularised:
## [2] 0.85225
```

```
## [1]  Improvement in RMSE with Genre User Effects:
## [2] 0.01177
```

Table 19: RMSE Summary

Model	RMSE
1-Base	1.06005
2-User Effects	0.97771
3-User & Movie Effects	0.88082
4-User & Movie Effects (R)	0.88079
5-User Effects (R) & Movie Effects (R)	0.86402
6-User Effects (R), Movie Effects (R) & Genre Effects User (R)	0.85225

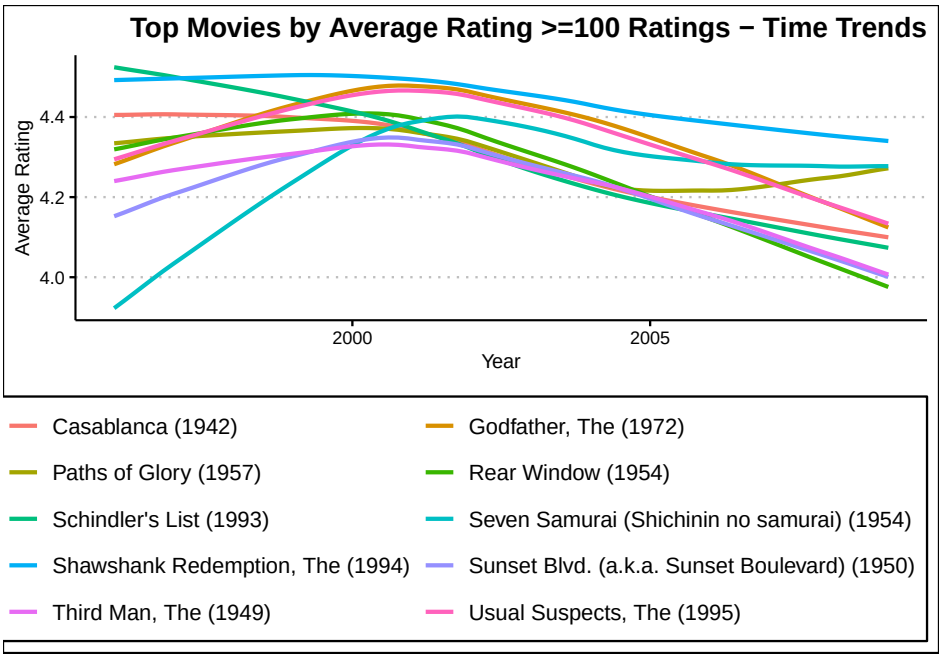
We have some improvement in the RMSE but not by a lot. Though the definition of Genre User Effects would indicate a strong predictor, the actual results do not match such an expectation. Our improvement in RMSE is slight but not very substantial.

Let us continue with improving our RMSE by adding more effects.

2.4.9 Model 7 - User Effects (R), Movie Effects (R), Genre Effects (R) & Time Effects (R)

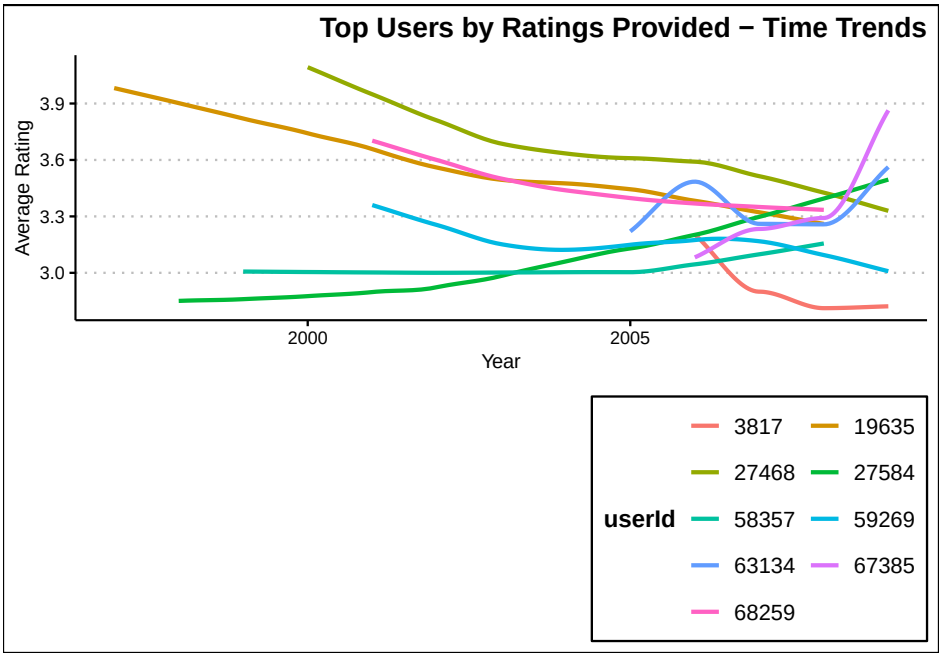
While we have seen some fairly reasonable improvements in the RMSE, by incorporating the Genre Effects and Regularisation of Genre Effects, Let us also explore how we can use the "timestamp" field to improve our predictions. Knowing that user preferences change over time, let us check if there are insights provided by the "timestamp" field can help us better our predictions. To avoid too much granularity, we will use Yearly Averages for Ratings. We will smoothen the plots using LOESS (Locally Estimated Scatterplot Smoothing) with Degree 1 (Linear Smoothing) to get a feel for the overall trend.

Let us look at our Traditional Top 10 list (>=100 Ratings) and how their ratings change over time.



We can see that Ratings seem to change over time, but there is no set pattern to their increase or decrease but most of them seem to first increase in popularity and then slowly wane over time.

Let us check if our User Preferences also change over time. We will use the same settings as we did for the Movies.



User Ratings too seem to change quite a bit with Time. Let us incorporate Time Effects in our predictions and check if our RMSE Improves with their inclusion. We will average the ratings for each User for a period of a Year and compare it with their Mean Rating.

We will extend our Prediction Model to include Time Effects as under

$$Y_{i,j} = \mu + \alpha_{i(\lambda_{ij})} + \beta_{j(\lambda_{ij})} + \beta_{i,g(\lambda_{ig})} + \beta_{i,t} + \beta_{j,t} + \varepsilon_{i,j}$$

Where symbols have their usual meaning.

$Y_{i,j}$ is the Predicted Rating

μ is the Mean Value of all known Ratings for all Movies

$\alpha_{i(\lambda_{ij})}$ is the Regularised Treatment Effect for each user i .

$\beta_{j(\lambda_{ij})}$ is the Regularised Treatment Effect for each Movie j .

$\beta_{i,g(\lambda_{ig})}$ is the Regularised Treatment Effect for User i for all applicable Genre k.

*Note: $\beta_{i,g(\lambda_{ig})}$ replaces $\frac{1}{K}\sum_{k=1}^{K} x_{i,k}\beta_{i,k}$ which was used before regularisation.

$\epsilon_{i,j}$ is the Random Error

We will define

$$\beta_{j,t} = \frac{1}{\sqrt(N)}\sum_{j}^{N}(\mu_{n.j} - \mu_j)$$

Where $\mu_{n.j}$ is the Average Rating for Movie j for year n, μ_j is the Average Rating for the Movie overall and N is the number of Years for which Ratings are available.

Similarly,

$$\beta_{i,t} = \frac{1}{\sqrt(N)}\sum_{i}^{N}(\mu_{n.i} - \mu_i)$$

Where $\mu_{n.i}$ is the Average Rating by User i for year n, μ_i is the Average Rating by the User overall and N is the number of Years for which Ratings are Provided by the User.

When compared to our Inital LRE,

$$y_{i,j} = \mu + \alpha_i + \beta_j + \sum_{k=1}^{K} x_{i,k}\beta_{i,k} + \beta_{i,t} + \beta_{j,t} + \varepsilon_i$$

we have added our fifth and sixth Regression Coefficients $\beta_{i,t}$ and $\beta_{j,t}$. So we are now finished with our entire list of predictors as per our Initial LRE.

We will compute the Time Effects together as we now have a pretty good idea of how the various Predictors, Coefficients and Regularisation are put together.

```
# Compute Time Effects for the edx_cv_train_set for Movies

# Extract Year from the Time Stamp, Record as date_year_j. Group by Year and
    Compute Mean Yearly Rating for each movie.
time_assignment_movie_edx_cv_train_set <- edx_cv_train_set %>%
  mutate(date = as_datetime(timestamp)) %>%
  mutate(date_year_j = year(date)) %>%  select(movieId,rating, date_year_j)
     %>%
  group_by(movieId, date_year_j) %>%
  summarise(mean_yearly_rating = mean(rating, na.rm = TRUE), .groups = "drop")

# Group and Select by movieId and Year to Extract Number of Years for which
    Ratings are available. Record as n_years_j

time_assignment_n_years_movie_edx_cv_train_set <- time_assignment_movie_edx_cv
    _train_set %>%
  group_by(movieId) %>%
  select(movieId, date_year_j) %>%
  summarise(movieId, date_year_j, n_years_j = n(), .groups = "drop")

# Compute Average Rating for each movie for edx_cv_train_set
movie_list_edx_cv_train_set <- edx_cv_train_set %>% group_by(movieId) %>%
  summarise(avg_rating = mean(rating))

# Compute b_time_m as difference between Mean Yearly Rating and Average Rating
    . Combine Tables to consolidate movieId, date_year_j, beta_jt and n_years_
    j

time_effects_movie_edx_cv_train_set <- movie_list_edx_cv_train_set %>%
  ungroup %>% mutate(movieId = as.factor(movieId)) %>%
  select(movieId, avg_rating) %>%
  left_join(time_assignment_movie_edx_cv_train_set, by = "movieId") %>%
  mutate("beta_jt" = (mean_yearly_rating - avg_rating)) %>%
  left_join(time_assignment_n_years_movie_edx_cv_train_set, by = join_by(
     movieId,date_year_j)) %>%
  select(movieId, date_year_j, beta_jt, n_years_j)

# Compute Time Effects for the edx_cv_train_set for Users

# Extract Year from the Time Stamp, Record as date_year_u. Group by Year and
    Compute Mean Yearly Rating for each User.

time_assignment_user_edx_cv_train_set <- edx_cv_train_set %>%
  mutate(userId = as.factor(userId)) %>%
  mutate(date = as_datetime(timestamp)) %>%
  mutate(date_year_i = year(date)) %>%  select(userId,rating,date_year_i) %>%
  group_by(userId, date_year_i) %>%
  summarise(mean_yearly_rating = mean(rating, na.rm = TRUE), .groups = "drop")
```

```
# Group and Select by userId and Year to Extract Number of Years for which
    Ratings are available. Record as n_years_i
time_assignment_n_years_user_edx_cv_train_set <- time_assignment_user_edx_cv_
    train_set %>%
  group_by(userId) %>%
  select(userId, date_year_i) %>%
  summarise(userId, date_year_i, n_years_i = n(), .groups = "drop")

# Compute beta_it as difference between Mean Yearly Rating and Average Rating.
     Combine Tables to consolidate userId, date_year_i, beta_it and n_years_i

time_effects_user_edx_cv_train_set <- edx_cv_train_set %>% mutate(userId = as.
    factor(userId)) %>%
  group_by(userId) %>% summarise(avg_rating = mean(rating)) %>%
  select(userId, avg_rating) %>%
  left_join(time_assignment_user_edx_cv_train_set, by = "userId") %>%
  mutate("beta_it" = (mean_yearly_rating - avg_rating)) %>%
  left_join(time_assignment_n_years_user_edx_cv_train_set, by = join_by(userId
      ,date_year_i)) %>%
  select(userId, date_year_i, beta_it, n_years_i)


# Let us prepare the test set for input of time effects. We will reuse the
    existing Dataset to avoid rework

# For Some very strange reason, R Generates NaN when beta_t is divided by n_
    years i.e. beta_t/n_years= NaN. This is a bit surprising because n_years
    cannot be 0 or -ve. The smallest value of n_years is 1. Offsetting n_years
     even by some very miniscule value causes the NaN to disappear.

  predicted_ratings_time <- genre_assignment_test %>%
    mutate(date = as_datetime(timestamp)) %>%
    mutate(date_year_j = year(date)) %>%
    mutate(date_year_i = year(date)) %>%
  left_join(alpha_r_df_combined, by='userId') %>%
  left_join(beta_r_df_combined, by='movieId') %>%
  left_join(genre_effects_user, by = 'userId') %>%
    mutate(beta_ig =  (b_drama*Drama + b_comedy*Comedy + b_action*Action + b_
        thriller*Thriller + b_adventure*Adventure + b_romance*Romance + b_sci_
        fi*Sci_Fi + b_crime*Crime + b_fantasy*Fantasy + b_children*Children +
        b_horror*Horror + b_mystery*Mystery + b_war*War + b_animation*
        Animation + b_musical*Musical + b_western*Western + b_film_noir*Film_
        Noir + b_documentary*Documentary)/((ifelse(genre_count==0,1,genre_
        count)) + min_lambda_ig  )) %>%
    left_join(time_effects_movie_edx_cv_train_set, by= join_by (movieId, date_
        year_j)) %>%
    left_join(time_effects_user_edx_cv_train_set, by= join_by (userId, date_
        year_i)) %>%
    mutate_all(list(~replace_na(., 0))) %>%
        mutate("pred" = clamp(mu + alpha_ij +  beta_ij + beta_ig + beta_jt/
            (0.0001 + sqrt(n_years_j)) + beta_it/(0.0001 + sqrt(n_years_i)) ))
             %>%
        select(userId, movieId, pred)
```

```
  rmse_time <- round(rmse(edx_cv_test_set$rating, predicted_ratings_time$pred)
     ,digits = 5)

rm(time_assignment_movie_edx_cv_train_set, time_assignment_n_years_movie_edx_
   cv_train_set, time_assignment_user_edx_cv_train_set, time_assignment_n_
   years_user_edx_cv_train_set)

print( c(" RMSE Time Effects: ", rmse_time), quote = FALSE, justify = "left")
```

```
## [1]  RMSE Time Effects:  0.84825
```

We have some marginal improvement in the RMSE with the incorporation of Time Effects.

We will record the RMSE for this Model as "RMSE User Effects (R), Movie Effects (R), Genre Effects User (R) + Movie (R) & Time Effects"

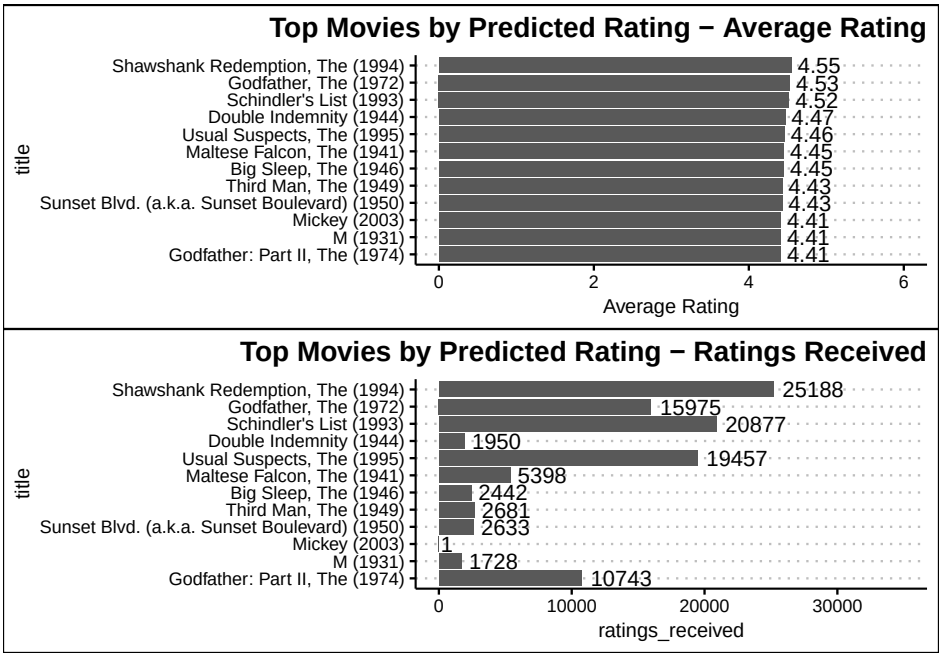
```
## [1]  RMSE User Effects (R), Movie Effects (R), Genre Effects (All) & Time
   Effects
## [2] 0.84825
```

```
## [1]  Improvement in RMSE with Time Effects
## [2] 0.004
```

Table 20: RMSE Summary

Model	RMSE
1-Base	1.06005
2-User Effects	0.97771
3-User & Movie Effects	0.88082
4-User & Movie Effects (R)	0.88079
5-User Effects (R) & Movie Effects (R)	0.86402
6-User Effects (R), Movie Effects (R) & Genre Effects User(R)	0.85225
7-User Effects (R), Movie Effects (R), Genre Effects User (R) & Time Effects	0.84825

Time to check our "Top 10" again,



Our Top 10 Ratings are now looking much better, we have 7 movies from our original list of movies by Highest Average Rating with >=100 Ratings. The movies with very small number or ratings have been replaced with movies with greater numbers of ratings. Time Effects have really helped us with our "Top 10" list.

2.4.10 Model 8 - Predictions Rounded to Discrete Values

After adding all the Effects, Let us check our distributions. Before we compile the graphs, let us recollect from our Methods & Analysis chapter on why we chose to model the Predictions as continuous variables.

1. Linear Regression Models are best suited for Continuous Variables
2. We cannot use a Typical Linear Regression model as-is for our datasets because the Variables are not suited for the same.

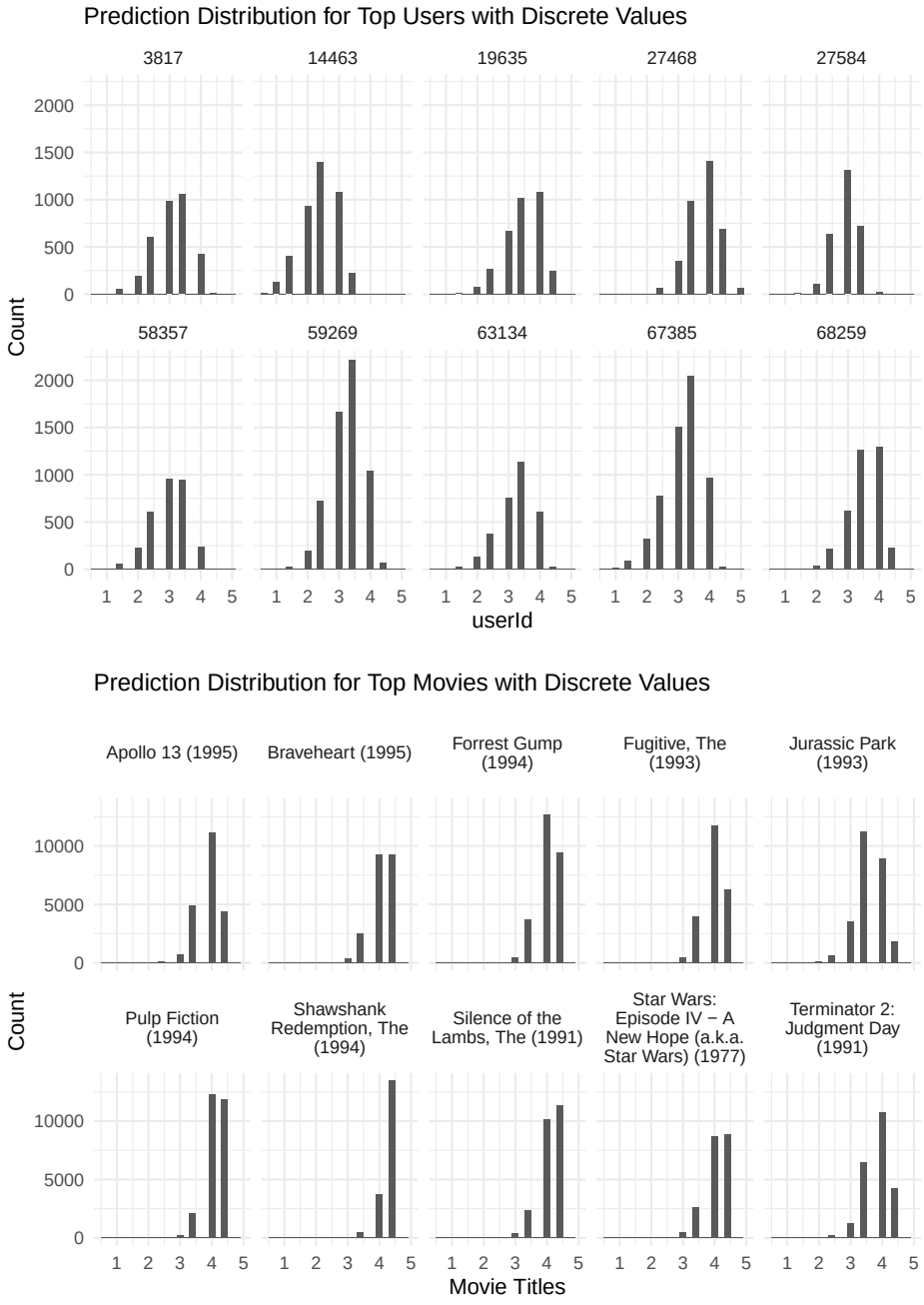
However, For the Movielens dataset, As per the rating scale, predictions need to be discrete values between 0 and 5.0 in increments of 0.5 .

So for the final set of graphs,

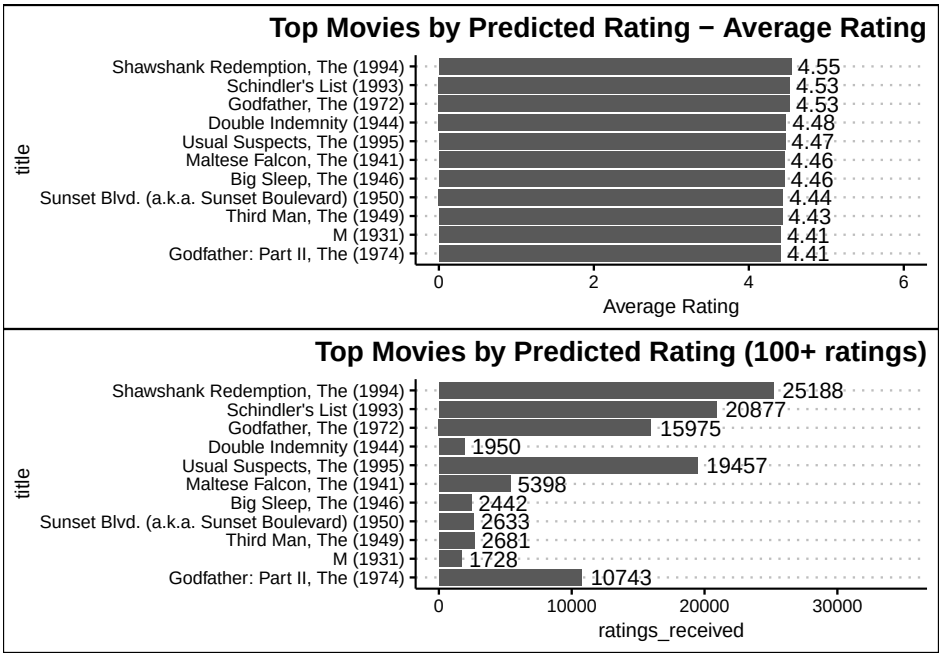
1. We will only retain operations that provide improvements in RMSE commiserate with the computing efforts required and
2. Convert our predictions to discrete values and see how our predictions match up to the actual ratings.

The expectation is that our RMSE should get a bit worse though our graphs should be more representative of the ratings as defined on the rating scale.

```
# Check Distributions with User, Movie, Genre & Time Effects Included and
   Predictions rounded to Discrete Values for Top 10 Users

genre_assignment %>%
    mutate(date = as_datetime(timestamp)) %>%
    mutate(date_year_j = year(date)) %>%
    mutate(date_year_i = year(date)) %>%
    left_join(alpha_r_df_combined, by='userId') %>%
    left_join(beta_r_df_combined, by='movieId') %>%
    left_join(genre_effects_user,  by='userId') %>%
    left_join(time_effects_movie_edx_cv_train_set, by= join_by (movieId, date_
       year_j)) %>%
    left_join(time_effects_user_edx_cv_train_set, by= join_by (userId, date_
       year_i)) %>%
    mutate_all(list(~replace_na(., 0))) %>%
    mutate(beta_ig =  (b_drama*Drama + b_comedy*Comedy + b_action*Action + b_
       thriller*Thriller + b_adventure*Adventure + b_romance*Romance + b_sci_
       fi*Sci_Fi + b_crime*Crime + b_fantasy*Fantasy + b_children*Children +
       b_horror*Horror + b_mystery*Mystery + b_war*War + b_animation*
       Animation + b_musical*Musical + b_western*Western + b_film_noir*Film_
       Noir + b_documentary*Documentary)/((ifelse(genre_count==0,1,genre_
       count))+ min_lambda_ig )) %>%
    mutate("pred" = clamp(mu + alpha_ij +  beta_ij + beta_ig +  beta_jt/
       (0.0001 + sqrt(n_years_j)) + beta_it/(0.0001 + sqrt(n_years_i))  ))
       %>%
    mutate(pred = signif(pred/5, digits = 1)) %>%
    mutate(pred = (pred*5)) %>%
  filter (userId %in% top_10_users$userId) %>%
  ggplot(aes( x= pred)) + geom_histogram(binwidth = 0.2) +
  facet_wrap( ~ userId,  nrow =2, ncol =5 ) +
  labs(x= "userId", y = "Count", title = "Prediction Distribution for Top
     Users with Discrete Values" ) + theme_minimal() + theme(plot.title =
     element_text(hjust = 0,size = 12))
```



Let us build our Final Dashboard for the "Top 10" for the Linear Regression Models. Based on the Movie List from Model 7, we will apply a filter for ratings >= 100.



We have some success but not absolute, 7 out of the 11 movies in the original list have made the cut. We have 4 new movies. Of these movies "Godfather: Part II, The (1974)" is familiar to general English speaking audiences. It is an acceptable choice. The rest will need further research on IMDB.

Our tenth requirement for Recommendation Systems was

10. Many competing Service Providers seeking to acquire and retain Customers, with "Personalised Content" or "Personalised Recommendations" often being seen as a key differentiator.

Through the course of our models, We have built a "Top 10" list or dashboard that is suitable for a general audience. It incorporates the Average Rating and the Number of Ratings and should be good for a non-personalised system.

We have not built a "Recommended for you" or "You may also like" kind of dashboard with personalised recommendations. We have some limitations in terms of grouping users together as we have only the "movieId" and "genres" fields to help us in grouping similar users together. We do not have the requisite demographic information in our dataset that will help us record and present recommendations as per users' personal preferences.

In the absence of demographic data, one possible scheme is to find and record the highest ranking genre for each user and try to find movies that contain those genre in them. This would possibly be extremely rudimentary. However, the single biggest obstacle in choosing this scheme or any other is the ability to validate if our recommendations make sense to the user.

Let us check our RMSE now,

```
## [1]  RMSE with Predictions Rounded to Discrete Values - Significant
   Predictors:
## [2] 0.86033
```

As expected, our RMSE has now degraded quite substantially due to the Rounding and Conversion to Discrete Values. However, thanks to our efforts so far, we have achieved a target RMSE lower than 0.86490 even with Discrete Values for the Rating and hence our efforts can be seen as being successful with the Validation Set.

However, we will still need to verify if our RMSE will be better than 0.86490 for our final test.

We can go a bit further and remove the Time Effects too completely and still manage to stay within our Target RMSE, but we will be very close to our Target.

Table 21: RMSE Summary

Model	RMSE
1-Base	1.06005
2-User Effects	0.97771
3-User & Movie Effects	0.88082
4-User & Movie Effects (R)	0.88079
5-User Effects (R) & Movie Effects(R)	0.86402
6-User Effects (R), Movie Effects (R) & Genre Effects User (R)	0.85225
7-User Effects (R), Movie Effects (R), Genre Effects User (R) & Time Effects	0.84825
8- RMSE with Discrete Values for Rating	0.86033

We will now compute the Residuals left after our Predictions using all the effects so far. We will have a look at them to understand where the errors stem from and what kind of remediation we can possibly explore.

```
# Let us calcuate the Residuals as the Difference between the Actual Ratings
    and Our Predictions for the edx_cv_train_set, We will also extract the
    userId, movieId and all the Co-efficients.

predicted_ratings_final_errors_file <- "predicted_ratings_final_errors.RData"

if(file.exists(predicted_ratings_final_errors_file)){
     load(predicted_ratings_final_errors_file)
      rm(predicted_ratings_final_errors_file)
} else {

  predicted_ratings_final_errors <- genre_assignment %>%
    mutate(date = as_datetime(timestamp)) %>%
    mutate(date_year_i = year(date)) %>%
    mutate(date_year_j = year(date)) %>%
    left_join(alpha_r_df_combined, by='userId') %>%
    left_join(beta_r_df_combined, by='movieId') %>%
    left_join(genre_effects_user, by="userId") %>%
    left_join(time_effects_movie_edx_cv_train_set, by= join_by (movieId, date_
       year_j)) %>%
    left_join(time_effects_user_edx_cv_train_set, by= join_by (userId, date_
       year_i)) %>%
    mutate_all(list(~replace_na(., 0))) %>%
    mutate(beta_ig =  (b_drama*Drama + b_comedy*Comedy + b_action*Action + b_
       thriller*Thriller + b_adventure*Adventure + b_romance*Romance + b_sci_
       fi*Sci_Fi + b_crime*Crime + b_fantasy*Fantasy + b_children*Children +
       b_horror*Horror + b_mystery*Mystery + b_war*War + b_animation*
       Animation + b_musical*Musical + b_western*Western + b_film_noir*Film_
       Noir + b_documentary*Documentary)/((ifelse(genre_count==0,1,genre_
       count))+ min_lambda_ig )) %>%
      mutate("pred" = clamp(mu + alpha_ij +  beta_ij + beta_ig + beta_jt/
         (0.0001 + sqrt(n_years_j)) + beta_it/(0.0001 + sqrt(n_years_i)) ))
         %>%
    mutate(pred = signif(pred/5, digits = 1)) %>%
```

```
    mutate(pred = (pred*5)) %>%
    mutate(errors = rating - pred) %>%
    select("userId", "movieId", "rating", "alpha_ij", "beta_ij", 'beta_ig', "
       beta_it", "beta_jt","pred", "errors")

save(predicted_ratings_final_errors, file = "predicted_ratings_final_errors.
   RData")
}

rm(predicted_ratings_final_errors_file)
```

2.4.11 Error Analysis for Linear Modelling

Let us look at the residuals left after application of User, Movie, Genre, Time Effects & Regularisation in our Predictions.

Let us get some idea of which entries have the largest errors.

Table 22: RMSE Error Summary for Different Ratings

RATINGS	RMSE	COUNT
0.5	2.0167925	76851
1.0	1.7508547	311052
1.5	1.3102743	95845
2.0	1.1123570	640598
2.5	0.7361284	299481
3.0	0.6134511	1909018
3.5	0.4816910	712190
4.0	0.5492204	2329863
4.5	0.7393711	474061
5.0	1.0153121	1251106

Our RMSE for Predictions is extremely poor for Ratings in the range 0.5 to 1.5.

Our RMSE for Predictions is also quite poor for Rating = 2.0 and Rating = 5.0

This is expected as the main basis of our ratings happens to be the Mean Value μ which is ≈ 3.51 for our current Dataset. However, we also have a number of Ratings that are accurate, so our Prediction Algorithm is working in some cases and not in some.

Let us check where the bulk of this error is coming from.

2.4.11.1 Ratings = 0.5

Let us look at the Largest Errors for the set where Rating = 0.5, Let us analyse it by userId & movieId to check if we can spot some trends.

Table 23: Top 10 Error Entries for Rating = 0.5

userId	movieId	rating	alpha_ij	beta_ij	beta_ig	beta_it	beta_jt	pred	errors
35290	5952	0.5	0.82932	0.60435	-0.12326	-0.01949	-0.09383	5.0	-4.5
35290	7153	0.5	0.82932	0.64286	-0.12326	-0.01949	-0.06923	5.0	-4.5
35870	912	0.5	0.91722	0.80682	-0.13296	0.00133	-0.21551	5.0	-4.5
35870	923	0.5	0.91722	0.67892	-0.08664	0.00133	-0.21987	5.0	-4.5
45308	2858	0.5	0.52224	0.67489	0.14745	0.00000	-0.11483	5.0	-4.5
49082	778	0.5	0.91492	0.49405	-0.08460	0.00000	-0.05590	5.0	-4.5
49082	7361	0.5	0.91492	0.60568	-0.17788	0.00000	-0.04055	5.0	-4.5
56204	1148	0.5	0.73557	0.76589	-0.08630	-0.00156	-0.19892	5.0	-4.5
63613	4226	0.5	0.57364	0.70333	-0.00288	0.00000	-0.09748	5.0	-4.5
52348	4816	0.5	1.17747	-0.28477	0.03489	0.04736	-0.00346	4.5	-4.0

Table 24: List of Movies in Top 10 Errors for Rating = 0.5

movieId	title	genres	avg_rating
778	Trainspotting (1996)	Comedy|Crime|Drama	4.007960
912	Casablanca (1942)	Drama|Romance	4.320424
923	Citizen Kane (1941)	Drama|Mystery	4.187267
1148	Wallace & Gromit: The Wrong Trousers (1993)	Animation|Children|Comedy|Crime	4.275429
2858	American Beauty (1999)	Drama	4.185664
4226	Memento (2000)	Crime|Drama|Mystery|Thriller	4.216862
4816	Zoolander (2001)	Comedy	3.230357
5952	Lord of the Rings: The Two Towers, The (2002)	Action|Adventure|Fantasy	4.119400
7153	Lord of the Rings: The Return of the King, The (2003)	Action|Adventure|Fantasy	4.155224
7361	Eternal Sunshine of the Spotless Mind (2004)	Comedy|Drama|Romance|Sci-Fi	4.124569

Table 25: Mean and SD for Users in the Top 10 Errors for Rating = 0.5

userId	mean	sd
35290	4.344890	0.8867313
35870	4.232954	0.8748840
45308	4.348837	1.0031788
49082	4.808823	0.7245487
52348	4.459135	0.7937030
56204	4.377083	0.7307472
63613	4.204724	0.7620834

Table 26: RMSE - Top 10 Errors for Rating = 0.5

User	Movie
0.86504	0.80927

We can see that the Largest Errors are for cases, where the User has a high Regularised $\alpha_{(\lambda_{ij})}$ value, the Movie has a high Regularised $\beta_{(\lambda_{ij})}$ value and the Algorithm would predict a high Rating for the User & Movie. The Genre Summaries are pulling the results back in Some cases with a negative value but only slightly and are not large enough to bridge the gap between the Prediction and the Actual Rating.

We can treat these Rating as anomalous, or we could investigate a bit further to understand User Behaviour in terms of both the Mean Rating as well as the Standard Deviation (SD). On further investigation, we can also see that our overall RMSE for these Users is 0.86504 which means our Predictions over all of their Ratings is only slightly worse than the overall RMSE. The RMSE for Movies is 0.80927 which is also better than our Overall RMSE.

All users have a large Mean Rating Value and the $\alpha_{(\lambda_{ij})}$ Values for these Users is also quite high.

We can understand that our Prediction Algorithm will not be able to Predict such entries accurately. We can Predict a Rating of 0.5 only when

$Y_{i,j} = \mu + \alpha_{i(\lambda_{ij})} + \beta_{j(\lambda_{ij})} + \beta_{i,g(\lambda_{ig})} + \beta_{i,t} + \beta_{j,t} + \varepsilon_{i,j} \approx 0.5$

which can happen only when

$\alpha_{i(\lambda_{ij})} + \beta_{j(\lambda_{ij})} + \beta_{i,g(\lambda_{ig})} + \beta_{i,t} + \beta_{j,t} \approx (0.5 - \mu) \approx -3.01$

Let us have a look at the cases where our Algorithm is able to Predict Rating = 0.5 accurately

Table 27: Top 10 Accurate Predictions for Rating = 0.5

userId	movieId	rating	alpha_ij	beta_ij	beta_ig	beta_it	beta_jt	pred	errors
1187	1760	0.5	-0.00623	-1.76042	-1.15854	0.00000	-0.07835	0.5	0
1456	5672	0.5	-0.14830	-2.25515	-1.49985	0.00000	0.52704	0.5	0
1828	5739	0.5	0.06420	-2.13801	-1.32959	0.00000	-0.13973	0.5	0
6023	1760	0.5	-0.25998	-1.76042	-1.09768	0.00000	-0.14633	0.5	0
6354	783	0.5	-1.00489	-0.26087	-1.61951	0.00000	-0.34715	0.5	0
8662	1495	0.5	0.11328	-2.10807	-0.71248	0.00000	-0.36638	0.5	0
8662	5672	0.5	0.11328	-2.25515	-0.87662	0.00000	-0.01585	0.5	0
9211	1976	0.5	-0.47881	-1.42302	-1.35897	0.00000	0.03385	0.5	0
9211	1986	0.5	-0.47881	-1.41528	-1.35897	0.00000	-0.03175	0.5	0
13060	8859	0.5	-0.40440	-2.52104	-0.08643	-0.01018	-0.01391	0.5	0

Table 28: List of Movies in Top 10 Accurate Predictions for Rating = 0.5

movieId	title	genres	avg_rating
783	Hunchback of Notre Dame, The (1996)	Animation|Children|Musical	3.2445339
1495	Turbo: A Power Rangers Movie (1997)	Action|Adventure|Children	1.3591371
1760	Spice World (1997)	Comedy|Musical	1.7399068
1976	Friday the 13th Part 3: 3D (1982)	Horror	2.0892608
1986	Halloween 5: The Revenge of Michael Myers (1989)	Horror	2.1021053
5672	Pokemon 4 Ever (a.k.a. Pokémon 4: The Movie) (2002)	Adventure|Animation|Children|Fantasy	1.1782178
5739	Faces of Death 6 (1996)	Documentary|Horror	1.2531646
8859	SuperBabies: Baby Geniuses 2 (2004)	Comedy	0.7946429

Table 29: Mean and SD for Users in the Top 10 Accurate Predictions for Rating = 0.5

userId	mean	sd
1187	3.680000	1.0774119
1456	3.522727	1.1070905
1828	3.648649	0.8964813
6023	3.477778	0.9997474
6354	2.632353	1.3388644
8662	3.458333	1.3745882
9211	3.150000	1.2808268
13060	3.246667	0.8790863

Table 30: RMSE - Top 10 Accurate Predictions for Rating = 0.5

User	Movie
0.79132	0.86277

Our RMSE Values for these Users are 0.79132 and Movies at 0.86277 when compared to the case of the Top 10 Errors for Rating = 0.5 where the RMSE were0.86504 and 0.80927 respectively.

As expected we can Predict Rating = 0.5 accurately when

$\alpha_{i(\lambda_{ij})} + \beta_{j(\lambda_{ij})} + \beta_{i,g(\lambda_{ig})} + \beta_{i,t} + \beta_{j,t} \approx (0.5 - \mu) \approx -3.01$

which means large negative values for $\alpha_{i(\lambda_{ij})}$ and/or $\beta_{j(\lambda_(ij)}$ and/or $\beta_{i,g(\lambda_{ig})}$ which is possible when

1. User consistently provides low ratings leading to large negative values for $\alpha_{(\lambda_{ij})}$
2. Movie is consistently rated low by most users leading to large negative values for $\beta_{(\lambda_{ij})}$
3. $\alpha_{(\lambda_{ij})}$, $\beta_{(\lambda_{ij})}$, $\beta_{i,g(\lambda_{ig})}$, $\beta_{i,t}$, $\beta_{j,t}$ when aggregated together generate a large enough negative value

2.4.11.2 Ratings = 1.0 Let us look at the Largest Errors for the set where Rating = 1.0, Again, Let us group it by userId and movieId to check if we can spot some trends.

Table 31: Top 10 Error Entries for Rating = 1.0

userId	movieId	rating	alpha_ij	beta_ij	beta_ig	beta_it	beta_jt	pred	errors
644	608	1	1.12561	0.62370	-0.19559	0	0.00243	5	-4
1716	58	1	1.37434	0.49640	-0.18973	0	0.04815	5	-4
7087	4846	1	1.14314	0.22350	-0.07363	0	-0.10345	5	-4
9728	56367	1	1.30322	0.49576	-0.26649	0	0.00147	5	-4
16246	48394	1	0.70845	0.55988	0.00715	0	-0.01710	5	-4
28122	48394	1	0.74683	0.55988	-0.04720	0	-0.01710	5	-4
29796	7153	1	0.98292	0.64286	0.02559	0	0.14382	5	-4
29977	232	1	1.00339	0.56133	0.07219	0	-0.00026	5	-4
30645	2858	1	0.83853	0.67489	-0.04572	0	0.02539	5	-4
33561	2858	1	0.58638	0.67489	0.07578	0	0.15246	5	-4

Table 32: List of Movies in Top 10 Errors for Rating = 1.0

movieId	title	genres	avg_rating
58	Postman, The (Postino, Il) (1994)	Comedy|Drama|Romance	4.014687
232	Eat Drink Man Woman (Yin shi nan nu) (1994)	Comedy|Drama|Romance	4.067126
608	Fargo (1996)	Comedy|Crime|Drama|Thriller	4.134821
2858	American Beauty (1999)	Drama	4.185664
4846	Iron Monkey (Siunin Wong Fei-hung tsi titmalau) (1993)	Action|Comedy	3.725989
7153	Lord of the Rings: The Return of the King, The (2003)	Action|Adventure|Fantasy	4.155224
48394	Pan's Labyrinth (El Laberinto del fauno) (2006)	Drama|Fantasy|Thriller	4.068488

movieId	title	genres	avg_rating
56367	Juno (2007)	Drama|Romance	4.007136

Table 33: Mean and SD for Users in the Top 10 Errors for Rating = 1.0

userId	mean	sd
644	4.863636	0.6678988
1716	4.653846	0.8374709
7087	4.647059	0.7700267
9728	4.339506	0.6653343
16246	4.167665	0.8151627
28122	4.329741	0.6869998
29796	3.995798	0.8374087
29977	4.382353	0.9308865
30645	4.573034	0.7817178
33561	4.311688	0.9768335

Table 34: RMSE - Top 10 Errors for Rating = 1.0

User	Movie
0.84982	0.8416

Again the Behaviour is somewhat the same as for Ratings = 0.5. For Ratings with the largest errors, the overall RMSE for Users & Movies is better than the RMSE of our overall Prediction system. Also the SD for most of the Users is well below the SD for the Training data set.

Let us have a look at the Accurate Predictions for Rating = 1.0

Table 35: Top 10 Accurate Predictions for Rating = 1.0

userId	movieId	rating	alpha_ij	beta_ij	beta_ig	beta_it	beta_jt	pred	errors
733	6587	1	-0.28281	-2.27767	-0.13625	0.0000	-0.13690	1	0
2904	6587	1	0.12775	-2.27767	0.02841	-0.2797	-0.05862	1	0
3859	397	1	0.05261	-1.45161	-1.02564	0.0000	-0.15909	1	0
4260	1381	1	0.06373	-1.57020	-0.84680	0.0000	0.00234	1	0
5315	1495	1	0.14265	-2.10807	-0.29755	0.0000	-0.36638	1	0
7055	1665	1	-0.00274	-0.74483	-0.01469	-2.6650	0.02333	1	0
7098	2461	1	0.04111	-1.50235	-0.88320	0.0000	-0.15002	1	0
7171	3593	1	0.00396	-1.93559	-0.63196	0.0000	0.04686	1	0
7733	3593	1	0.08329	-1.93559	-0.55269	0.0000	0.13103	1	0
7842	1760	1	0.16779	-1.76042	-0.97175	0.0000	-0.07835	1	0

Table 36: List of Movies in Top 10 Accurate Predictions for Rating = 1.0

movieId	title	genres	avg_rating
397	Fear, The (1995)	Horror	1.970000
1381	Grease 2 (1982)	Comedy|Musical|Romance	1.943462
1495	Turbo: A Power Rangers Movie (1997)	Action|Adventure|Children	1.359137
1665	Bean (1997)	Comedy	2.747180
1760	Spice World (1997)	Comedy|Musical	1.739907
2461	Leatherface: Texas Chainsaw Massacre III (1990)	Comedy|Horror|Thriller	1.955814
3593	Battlefield Earth (2000)	Action|Sci-Fi	1.568218
6587	Gigli (2003)	Action|Crime|Drama	1.193291

Table 37: Mean and SD for Users in the Top 10 Accurate Predictions for Rating = 1.0

userId	mean	sd
733	3.159091	1.1790469
2904	3.636842	0.7295647
3859	3.500000	0.8846517
4260	3.848039	0.8289001
5315	3.522727	0.9083242
7055	3.665000	0.7786773
7098	3.954546	1.1742180
7171	3.732143	0.9998376
7733	3.565217	1.3923847
7842	3.333333	1.0165300

Table 38: RMSE - Top 10 Accurate Predictions for Rating = 1.0

User	Movie
0.71415	0.94514

Here, we can see that the RMSE for Users is actually better than the RMSE of our overall system and the RMSE for Movies is a lot worse than our overall system. In most cases, $\alpha_{(\lambda_{ij})}$, $\beta_{(\lambda_{ij})}$, $\beta_{i,g(\lambda_{ig})}$ are helping with balancing values to bring the rating closer to 1.0. We even have cases where $\beta_{i,t}$ and $\beta_{j,t}$ are helping balance our Predictions.

2.4.11.3 Ratings = 5.0 Let us look at the Largest Errors for the set where Rating = 5.0, Again, Let us group it by userId and movieId to check if we can spot some trends.

Table 39: Top 10 Error Entries for Rating = 5.0

userId	movieId	rating	alpha_ij	beta_ij	beta_ig	beta_it	beta_jt	pred	errors
21267	6482	5	-1.14291	-1.79881	-0.18875	0	0.01019	0.5	4.5
26175	2362	5	-1.55055	-0.81371	0.01026	0	0.07813	1.0	4.0

userId	movieId	rating	alpha_ij	beta_ij	beta_ig	beta_it	beta_jt	pred	errors
69898	99	5	-1.87764	-0.47982	0.87179	0	-0.27674	2.0	3.0
69898	193	5	-1.87764	-1.25006	-0.06061	0	0.17833	0.5	4.5
69898	467	5	-1.87764	-0.10178	0.05128	0	0.39055	1.5	3.5
69898	639	5	-1.87764	-0.57326	-0.00616	0	0.31498	1.0	4.0
69898	737	5	-1.87764	-1.34157	0.05650	0	0.43954	0.5	4.5
69898	762	5	-1.87764	-1.05279	0.04520	0	0.47912	1.0	4.0
4043	76	5	-1.95102	-0.51225	0.03945	0	0.11498	1.0	4.0
4043	94	5	-1.95102	-0.00246	0.03707	0	-0.11554	1.5	3.5

Table 40: List of Movies in Top 10 Errors for Rating = 5.0

movieId	title	genres	avg_rating
76	Screamers (1995)	Action|Sci-Fi|Thriller	2.997591
94	Beautiful Girls (1996)	Comedy|Drama|Romance	3.511293
99	Heidi Fleiss: Hollywood Madam (1995)	Documentary	3.077197
193	Showgirls (1995)	Drama	2.255200
467	Live Nude Girls (1995)	Comedy	3.428571
639	Girl 6 (1996)	Comedy|Drama	2.948476
737	Barb Wire (1996)	Action|Sci-Fi	2.169672
762	Striptease (1996)	Comedy|Crime	2.456996
2362	Glen or Glenda (1953)	Drama	2.674208
6482	Dumb and Dumberer: When Harry Met Lloyd (2003)	Comedy	1.701835

Table 41: Mean and SD for Users in the Top 10 Errors for Rating = 5.0

userId	mean	sd
4043	1.266355	0.9289748
21267	1.805556	1.4766704
26175	1.880000	1.0132456
69898	1.300000	1.0602125

Table 42: RMSE - Top 10 Errors for Rating = 5.0

User	Movie
1.11837	0.97874

As we have already seen so far, the Ratings are accurate when the Rating Provided follows the general pattern for the User and the Movie and they have errors when the Rating Provided is an outlier. Our RMSE for these Users is quite poor at about 1.11837 and also quite poor for Movies at 0.97874 when compared to the Overall RMSE. In almost all cases, the $\alpha_{(\lambda_{ij})}$ and $\beta_{(\lambda_{ij})}$ with large negative values are causing the Predictions to go askew.

Let us have a look at the Accurate Predictions for Rating = 5.0

Table 43: Top 10 Accurate Predictions for Rating = 5.0

userId	movieId	rating	alpha_ij	beta_ij	beta_ig	beta_it	beta_jt	pred	errors
172	110	5	0.13750	0.56936	0.54528	0	0.17836	5	0
445	858	5	0.13879	0.90393	0.38849	0	-0.10126	5	0
1096	858	5	0.04804	0.90393	0.49020	0	-0.07312	5	0
1142	593	5	-0.31793	0.68993	0.92579	0	0.08207	5	0
1600	913	5	0.24994	0.73321	0.46660	0	-0.23715	5	0
1802	1252	5	-0.21067	0.72151	0.89394	0	-0.16201	5	0
1901	527	5	-0.37293	0.85144	0.71905	0	0.15609	5	0
1901	593	5	-0.37293	0.68993	1.13924	0	0.08207	5	0
2289	527	5	-0.23419	0.85144	0.67797	0	0.15609	5	0
2395	1289	5	-0.15442	0.39349	0.99865	0	0.10195	5	0

Table 44: List of Movies in Top 10 Accurate Predictions for Rating = 5.0

movieId	title	genres	avg_rating
110	Braveheart (1995)	Action|Drama|War	4.081852
527	Schindler's List (1993)	Drama|War	4.363493
593	Silence of the Lambs, The (1991)	Crime|Horror|Thriller	4.204101
858	Godfather, The (1972)	Crime|Drama	4.415366
913	Maltese Falcon, The (1941)	Film-Noir|Mystery	4.246136
1252	Chinatown (1974)	Crime|Film-Noir|Mystery|Thriller	4.239862
1289	Koyaanisqatsi (a.k.a. Koyaanisqatsi: Life Out of Balance) (1983)	Documentary	3.917873

Table 45: Mean and SD for Users in the Top 10 Accurate Predictions for Rating = 5.0

userId	mean	sd
172	3.615385	0.6504436
445	3.805556	0.6889653
1096	3.735294	0.6873329
1142	3.058823	1.1440383
1600	3.911765	0.5372178
1802	3.300000	1.0316422
1901	3.166667	1.0431852
2289	3.333333	0.6859943
2395	3.052632	1.2235506

Table 46: RMSE - Top 10 Accurate Predictions for Rating = 5.0

User	Movie
0.63867	0.74389

Again, Not to repeat what we have already said so far, the Ratings are accurate when the Rating follows the general pattern for the User and/or the Movie and they have errors when the Rating is an extreme outlier.

These are to be expected based on our current model of Prediction. However, we can look at Models other than Linear Regression for Users whose RMSE is extremely poor. Our Current Prediction Model is not flexible enough to accommodate the ratings of these Users.

2.4.12 Linear Regression Models - Summary

Let us review whatever we have done so far

1. We have explained and demonstrated the various Linear Regression Models for Predictions.
2. We now have an understanding for the data and its characteristics.
3. We can see how different groups of data interact and influence each other.
4. We can also see what kind of data entries surface when we apply different Effects and how they complement or negate each other.
5. We have also seen how Regularisation can help in bettering the overall RMSE and at the same time we also had a look at some of the side-effects that it can cause.
6. We have also seen how to glean information that is not explicitly mentioned by extrapolating from various fields.
7. The RMSE has continually improved after application of each effect and our Predictions are getting closer to the actual values. At the same time, application of different Effects also has some bearing on our Data Analysis and Visualisation, Without understanding how these effects work, we could draw erroneous conclusions from our Data. Methodical Analysis can help us make the right choice.
8. We have also seen that by converting our Predictions to Discrete Values using Rounding, our RMSE actually degrades as Linear Models are more suited for Continuous Variables.

We have large errors for what are essentially Ratings that are outliers and do not follow the normal Rating pattern for the User and/or the Movie. These can be due to a variety of reasons. These probably reflect the personal choices of Users, maybe a few are anomalous.

We will need a different set of Algorithms to Predict such Entries.

Maybe CART Algorithms can do better than Linear Modelling. Also Matrix Factorisation for Users where we see a large difference between the Prediction and the Rating can possibly help capture latent User Preferences. This is something for future study.

3 Results

So far, all our Analysis has been confined to the Cross Validation Sets that we carved out of the "edx" set and we are yet to touch our "final_holdout_test" data. We will now try to see how well our methods work when applied to the "final_holdout_test".

Since our Linear models evolve sequentially by adding and improving upon the coefficients, we will follow the evolution as in our Analysis. This will help us keep our results organised and more presentable and also help the evaluators understand the computation steps better.

However, to respect and adhere to the spirit of the reporting guidelines:-

1. We will not compute models which do not add to the evolution of the final model.
2. For models that contribute to the evolution of the final model, we will not generate predictions or compute the RMSE.

We will present the RMSE only for the final model.

3.1 Linear Regression Models

Model 1 - Base Model

Prediction as the Mean of Ratings

```
# Compute mu which is the base for the rest of the models
mu_edx <- mean(edx$rating)
```

Model 2 - User Effects

Prediction as the Mean of ratings & User Effects

Not implemented as it is not required for Final Model evolution

Model 3 - User & Movie Effects

Prediction as the Mean of ratings, User Effects & Movie Effects

Not implemented as it is not required for Final Model evolution

Model 4 - User Effects & Movie Effects (R)

Predictions based on the Mean of ratings, User Effects & Regularisation for Movie Effects.

Not implemented as it is not required for Final Model evolution

Model 5 - User Effects (R) & Movie Effects (R)

Predictions based on the Mean of ratings, Combined Regularisation for Movie Effects & User Effects using a common regularisation factor.

The common regularisation factor "lambda_ij" computed during the Analysis is used here.

```
# Compute Penalised values for Movie Effects and User Effects with
   Regularisation factor lambda_ij
# Use lambda_ij computed during Analysis

     beta_r_df_combined_edx <- edx %>%
          mutate("userId" = as.factor(userId),"movieId" = as.factor(movieId))
              %>%
          group_by(movieId) %>%
```

```
          summarize(beta_ij = sum(rating - mu_edx)/(n()+ min_lambda_ij))
     alpha_r_df_combined_edx <- edx %>%
          mutate("userId" = as.factor(userId),"movieId" = as.factor(movieId))
              %>%
          left_join(beta_r_df_combined_edx, by="movieId") %>%
          group_by(userId) %>%
          summarize(alpha_ij = sum(rating - beta_ij - mu_edx)/(n()+ min_lambda
              _ij))
```

Model 6 - User Effects (R), Movie Effects (R) & Genre Effects (R)

Predictions based on Regularised User Effects, Regularised Movie Effects & Regularised Genre User Effects .

For us to build a solution for this model, we will need to perform the same steps as during the Analysis. We will start by splitting up the contents of the genre field into individual genre .

We will create a new dataset called "genre_assignment_edx" and populate the individual genre column with a 1 if the genre is applicable for a rating or a 0 otherwise. We will also define and compute K as the number of genre attached to each movie. This will help us normalise the ratings later. We will use the new dataset created to replicate the rating in further computations.

We will need to compare our predictions with our Test set "final_holdout_test" . The Test set will also need to be modified to split the genre so that we can compare our predictions based on the genre associated with each movie. We will create a new dataset called "genre_assignment_test_edx" by following the exact same procedure as we did with the "edx" set.

```
# Extract Genre, Load from Disk if already available to avoid repeated
    computation. Saves time, effort and CPU cycles.
genre_assignment_file_edx <- "genre_assignment_edx.RData"

if(file.exists(genre_assignment_file_edx)){
      load(genre_assignment_file_edx)

      genre_assignment_edx <- genre_assignment_edx %>%
      mutate(userId = as.factor(userId), movieId = as.factor(movieId))

      rm(genre_assignment_file_edx)

} else {

# Genre are all combined together in the "genres" column in a single string.
    Individual Genres within the string are separated using the pipe (|)
    symbol.

# Split genre individually for further Analysis. Use the "stringr" library and
     str_split function.

# The pipe (|) symbol happens to have its own meaning in Regular Expressions
    so must be escaped using a double backslash.

# Assign genre individually to each movie based on the Split Strings. Each
    genre is tracked individually in its own column. Assign 1 if genre matches
    , 0 if it does not. Based on the code below, the "mutate" function from
    the "Tidyverse" Package will automatically create a Column if it does not
    exist already. If the Column exists, it will just update the corresponding
```

```
    row. Please note that the Column names for Genres Sci-Fi and Film-Noir
    have been changed to Sci_Fi and Film_Noir respectively. This is done so
    that it is easier to address the columns as R otherwise interprets the
    Symbol for the Hyphen (-) as the Minus/Subtraction Operator.

# Add a genre "count" column for counting & tracking the number of genre
    attached to each movie, this will help us normalise the genre treatment
    effects values later.

genre_assignment_edx <- edx %>% mutate("individual_genre" = str_split(genres,
    "\\|")) %>%
        mutate (Action = ifelse (str_detect ( individual_genre, "Action"), 1 ,
             0 ) ) %>%
        mutate (Adventure = ifelse (str_detect ( individual_genre, "Adventure"
            ) , 1 , 0 ) ) %>%
        mutate (Animation = ifelse (str_detect ( individual_genre, "Animation"
            ) , 1 , 0 ) ) %>%
        mutate (Children = ifelse (str_detect ( individual_genre, "Children")
            , 1 , 0 ) ) %>%
        mutate (Comedy = ifelse (str_detect ( individual_genre, "Comedy") , 1
            , 0 ) ) %>%
        mutate (Crime = ifelse (str_detect ( individual_genre, "Crime") , 1 ,
            0 ) ) %>%
        mutate (Documentary = ifelse (str_detect ( individual_genre, "
            Documentary") , 1 , 0 ) ) %>%
        mutate (Drama = ifelse (str_detect ( individual_genre, "Drama") , 1 ,
            0 ) ) %>%
        mutate (Fantasy = ifelse (str_detect ( individual_genre, "Fantasy") ,
            1 , 0 ) ) %>%
        mutate (Film_Noir = ifelse (str_detect ( individual_genre, "Film-Noir"
            ) , 1 , 0 ) ) %>%
        mutate (Mystery = ifelse (str_detect ( individual_genre, "Mystery") ,
            1 , 0 ) ) %>%
        mutate (Horror = ifelse (str_detect ( individual_genre, "Horror") , 1
            , 0 ) ) %>%
        mutate (Thriller = ifelse (str_detect ( individual_genre, "Thriller")
            , 1 , 0 ) ) %>%
        mutate (Musical = ifelse (str_detect ( individual_genre, "Musical") ,
            1 , 0 ) ) %>%
        mutate (Sci_Fi = ifelse (str_detect ( individual_genre, "Sci-Fi") , 1
            , 0 ) ) %>%
        mutate (Romance = ifelse (str_detect ( individual_genre, "Romance") ,
            1 , 0 ) ) %>%
        mutate (War = ifelse (str_detect ( individual_genre, "War") , 1 , 0 )
            ) %>%
        mutate (Western = ifelse (str_detect ( individual_genre, "Western") ,
            1 , 0 ) ) %>%
        mutate("genre_count" = Action + Adventure + Animation + Children +
            Comedy + Crime + Documentary + Drama + Fantasy + Film_Noir +
            Mystery + Horror + Thriller + Musical + Sci_Fi + Romance + War +
            Western ) %>%
        mutate("userId" = as.factor(userId), "movieId" = as.factor(movieId))

# Save file to disk for quick retrieval later if required
```

```
save(genre_assignment_edx, file = "genre_assignment_edx.RData")
}


# Create Genre Assignment Matrix. contains only numerical values for ease of
    processing

genre_assignment_matrix_edx <- genre_assignment_edx %>%
  group_by(userId) %>%
  select (userId, movieId, rating, all_of(genre_list), genre_count)

# Use a temp file to collect and store extrapolated ratings for the genre
temp <- genre_assignment_matrix_edx %>%
        ungroup %>% select(rating, all_of(genre_list)) %>%
        mutate ( Action = ifelse ( Action == 1 , rating , 0 ) ) %>%
        mutate ( Adventure = ifelse ( Adventure == 1 , rating , 0 ) ) %>%
        mutate ( Animation = ifelse ( Animation == 1 , rating , 0 ) ) %>%
        mutate ( Children = ifelse ( Children == 1 , rating , 0 ) ) %>%
        mutate ( Comedy = ifelse ( Comedy == 1 , rating , 0 ) ) %>%
        mutate ( Crime = ifelse ( Crime == 1 , rating , 0 ) ) %>%
        mutate ( Documentary = ifelse ( Documentary == 1 , rating , 0 ) ) %>%
        mutate ( Drama = ifelse ( Drama == 1 , rating , 0 ) ) %>%
        mutate ( Fantasy = ifelse ( Fantasy == 1 , rating , 0 ) ) %>%
        mutate ( Film_Noir = ifelse ( Film_Noir == 1 , rating , 0 ) ) %>%
        mutate ( Mystery = ifelse ( Mystery == 1 , rating , 0 ) ) %>%
        mutate ( Horror = ifelse ( Horror == 1 , rating , 0 ) ) %>%
        mutate ( Thriller = ifelse ( Thriller == 1 , rating , 0 ) ) %>%
        mutate ( Musical = ifelse ( Musical == 1 , rating , 0 ) ) %>%
        mutate ( Sci_Fi = ifelse ( Sci_Fi == 1 , rating , 0 ) ) %>%
        mutate ( Romance = ifelse ( Romance == 1 , rating , 0 ) ) %>%
        mutate ( War = ifelse ( War == 1 , rating , 0 ) ) %>%
        mutate ( Western = ifelse ( Western == 1 , rating , 0 )) %>%
        select(  all_of(genre_list))

# compute individual sums

individual_genre_sums_edx <- colSums(temp)

# compute individual counts

individual_genre_count_edx <- colSums(temp != 0)

# Compute Overall Genre Mean

mu_g_edx <- sum(individual_genre_sums_edx)/sum(individual_genre_count_edx)


# compute Individual Genre Mean and Difference from Overall Genre Mean and
    Store in a common Table,

genre_summary_edx <- data.frame(individual_genre_sums_edx,individual_genre_
    count_edx)
```

```
genre_summary_edx <- genre_summary_edx %>%
      mutate(individual_genre_mean_edx = (individual_genre_sums_edx/individual
         _genre_count_edx), difference_from_genre_mean = (individual_genre_
         mean_edx - mu_g_edx))

# remove Temp file
rm(temp)

# To compare our Predictions with the Final Holdout Test Set Data. Repeat
    operations performed on 'edx' dataset for 'final_holdout_test' dataset.
    Load Data Set from Disk if already available to save on time and computing
     efforts.

genre_assignment_test_file_final <- "genre_assignment_test_final.RData"

if(file.exists(genre_assignment_test_file_final)){
      load(genre_assignment_test_file_final)
      genre_assignment_test_file_final <- genre_assignment_test_final %>%
      mutate(userId = as.factor(userId), movieId = as.factor(movieId))
    rm(genre_assignment_test_file_final)
} else {

genre_assignment_test_final <- final_holdout_test %>%
        mutate("individual_genre" = str_split(genres, "\\|"))%>%
        mutate (Action = ifelse (str_detect ( individual_genre, "Action") , 1
            , 0 ) ) %>%
        mutate (Adventure = ifelse (str_detect ( individual_genre, "Adventure"
            ) , 1 , 0 ) ) %>%
        mutate (Animation = ifelse (str_detect ( individual_genre, "Animation"
            ) , 1 , 0 ) ) %>%
        mutate (Children = ifelse (str_detect ( individual_genre, "Children")
            , 1 , 0 ) ) %>%
        mutate (Comedy = ifelse (str_detect ( individual_genre, "Comedy") , 1
            , 0 ) ) %>%
        mutate (Crime = ifelse (str_detect ( individual_genre, "Crime") , 1 ,
            0 ) ) %>%
        mutate (Documentary = ifelse (str_detect ( individual_genre, "
            Documentary") , 1 , 0 ) ) %>%
        mutate (Drama = ifelse (str_detect ( individual_genre, "Drama") , 1 ,
            0 ) ) %>%
        mutate (Fantasy = ifelse (str_detect ( individual_genre, "Fantasy") ,
            1 , 0 ) ) %>%
        mutate (Film_Noir = ifelse (str_detect ( individual_genre, "Film-Noir"
            ) , 1 , 0 ) ) %>%
        mutate (Mystery = ifelse (str_detect ( individual_genre, "Mystery") ,
            1 , 0 ) ) %>%
        mutate (Horror = ifelse (str_detect ( individual_genre, "Horror") , 1
            , 0 ) ) %>%
        mutate (Thriller = ifelse (str_detect ( individual_genre, "Thriller")
            , 1 , 0 ) ) %>%
        mutate (Musical = ifelse (str_detect ( individual_genre, "Musical") ,
            1 , 0 ) ) %>%
        mutate (Sci_Fi = ifelse (str_detect ( individual_genre, "Sci-Fi") , 1
            , 0 ) ) %>%
```

```
        mutate (Romance = ifelse (str_detect ( individual_genre, "Romance") ,
            1 , 0 ) ) %>%
        mutate (War = ifelse (str_detect ( individual_genre, "War") , 1 , 0 )
            ) %>%
        mutate (Western = ifelse (str_detect ( individual_genre, "Western") ,
            1 , 0 ) ) %>%
        mutate("genre_count" = Action + Adventure + Animation + Children +
            Comedy + Crime + Documentary + Drama + Fantasy + Film_Noir +
            Mystery + Horror + Thriller + Musical + Sci_Fi + Romance + War +
            Western )

# Save file to disk for quick retrieval later if required
save(genre_assignment_test_final, file="genre_assignment_test_final.RData")
}

# Remove variables used to hold filenames
rm(genre_assignment_file_edx, genre_assignment_test_file_final)
```

Here we add the User Genre Effects.

```
# Compute Genre User Effects for each Genre

genre_action_edx <- genre_assignment_matrix_edx %>% group_by(userId) %>%
    summarise ("mean_rating" = mean(rating)  , "count_action" = sum(Action ==
    1), "sum_action" = sum((ifelse(Action==0,0,rating)))  , "mean_action" =
    sum_action/(ifelse(count_action==0,1,count_action)), "b_action"  = ifelse(
    sum_action==0,0,(mean_action - mean_rating ))) %>% select(userId, mean_
    action, b_action)

genre_adventure_edx <- genre_assignment_matrix_edx %>% group_by(userId) %>%
    summarise ("mean_rating" = mean(rating)  , "count_adventure" = sum(
    Adventure == 1), "sum_adventure" = sum((ifelse(Adventure==0,0,rating)))  ,
     "mean_adventure" = sum_adventure/(ifelse(count_adventure==0,1,count_
    adventure)), "b_adventure"  = ifelse(sum_adventure==0,0,(mean_adventure -
    mean_rating ))) %>% select(userId, mean_adventure, b_adventure)

genre_animation_edx <- genre_assignment_matrix_edx %>% group_by(userId) %>%
    summarise ("mean_rating" = mean(rating)  , "count_animation" = sum(
    Animation == 1), "sum_animation" = sum((ifelse(Animation==0,0,rating)))  ,
     "mean_animation" = sum_animation/(ifelse(count_animation==0,1,count_
    animation)), "b_animation"  = ifelse(sum_animation==0,0,(mean_animation -
    mean_rating  ))) %>% select(userId, mean_animation, b_animation)

genre_children_edx <- genre_assignment_matrix_edx %>% group_by(userId) %>%
    summarise ("mean_rating" = mean(rating)  , "count_children" = sum(Children
     == 1), "sum_children" = sum((ifelse(Children==0,0,rating)))  , "mean_
    children" = sum_children/(ifelse(count_children==0,1,count_children)), "b_
    children"  = ifelse(sum_children==0,0,(mean_children - mean_rating ))) %>%
     select(userId, mean_children, b_children)

genre_comedy_edx <- genre_assignment_matrix_edx %>% group_by(userId) %>%
    summarise ("mean_rating" = mean(rating)  , "count_comedy" = sum(Comedy ==
    1), "sum_comedy" = sum((ifelse(Comedy==0,0,rating)))  , "mean_comedy" =
    sum_comedy/(ifelse(count_comedy==0,1,count_comedy)), "b_comedy"  = ifelse(
```

```
    sum_comedy==0,0,(mean_comedy - mean_rating ))) %>% select(userId, mean_
    comedy, b_comedy)

genre_crime_edx <- genre_assignment_matrix_edx %>% group_by(userId) %>%
    summarise ("mean_rating" = mean(rating)  , "count_crime" = sum(Crime == 1)
    , "sum_crime" = sum((ifelse(Crime==0,0,rating)))  , "mean_crime" = sum_
    crime/(ifelse(count_crime==0,1,count_crime)), "b_crime"  = ifelse(sum_
    crime==0,0,(mean_crime - mean_rating  ))) %>% select(userId, mean_crime, b
    _crime)

genre_documentary_edx <- genre_assignment_matrix_edx %>% group_by(userId) %>%
    summarise ("mean_rating" = mean(rating) , "count_documentary" = sum(
    Documentary == 1), "sum_documentary" = sum((ifelse(Documentary==0,0,rating
    )))  , "mean_documentary" = sum_documentary/(ifelse(count_documentary
    ==0,1,count_documentary)), "b_documentary"  = ifelse(sum_documentary
    ==0,0,(mean_documentary - mean_rating  ))) %>% select(userId, mean_
    documentary, b_documentary)

genre_drama_edx <- genre_assignment_matrix_edx %>% group_by(userId) %>%
    summarise ("mean_rating" = mean(rating), "count_drama" = sum(Drama == 1),
    "sum_drama" = sum((ifelse(Drama==0,0,rating)))  , "mean_drama" = sum_drama
    /(ifelse(count_drama==0,1,count_drama)), "b_drama"  = ifelse(sum_drama
    ==0,0,(mean_drama - mean_rating ))) %>% select(userId, mean_drama, b_drama
    )

genre_fantasy_edx <- genre_assignment_matrix_edx %>% group_by(userId) %>%
    summarise ("mean_rating" = mean(rating)  , "count_fantasy" = sum(Fantasy
    == 1), "sum_fantasy" = sum((ifelse(Fantasy==0,0,rating)))  , "mean_fantasy
    " = sum_fantasy/(ifelse(count_fantasy==0,1,count_fantasy)), "b_fantasy"  =
     ifelse(sum_fantasy==0,0,(mean_fantasy - mean_rating ))) %>% select(userId
    , mean_fantasy, b_fantasy)

genre_film_noir_edx <- genre_assignment_matrix_edx %>% group_by(userId) %>%
    summarise ("mean_rating" = mean(rating)  , "count_film_noir" = sum(Film_
    Noir == 1), "sum_film_noir" = sum((ifelse(Film_Noir==0,0,rating)))  , "
    mean_film_noir" = sum_film_noir/(ifelse(count_film_noir==0,1,count_film_
    noir)), "b_film_noir"  = ifelse(sum_film_noir==0,0,(mean_film_noir - mean_
    rating ))) %>% select(userId, mean_film_noir, b_film_noir)

genre_mystery_edx <- genre_assignment_matrix_edx %>% group_by(userId) %>%
    summarise ("mean_rating" = mean(rating) , "count_mystery" = sum(Mystery ==
     1), "sum_mystery" = sum((ifelse(Mystery==0,0,rating)))  , "mean_mystery"
    = sum_mystery/(ifelse(count_mystery==0,1,count_mystery)), "b_mystery"  =
    ifelse(sum_mystery==0,0,(mean_mystery - mean_rating  ))) %>% select(userId
    , mean_mystery, b_mystery)

genre_horror_edx <- genre_assignment_matrix_edx %>% group_by(userId) %>%
    summarise ("mean_rating" = mean(rating) , "count_horror" = sum(Horror ==
    1), "sum_horror" = sum((ifelse(Horror==0,0,rating)))  , "mean_horror" =
    sum_horror/(ifelse(count_horror==0,1,count_horror)), "b_horror"  = ifelse(
    sum_horror==0,0,(mean_horror - mean_rating ))) %>% select(userId, mean_
    horror, b_horror)

genre_thriller_edx <- genre_assignment_matrix_edx %>% group_by(userId) %>%
```

```
    summarise ("mean_rating" = mean(rating)  , "count_thriller" = sum(Thriller
     == 1), "sum_thriller" = sum((ifelse(Thriller==0,0,rating)))  , "mean_
    thriller" = sum_thriller/(ifelse(count_thriller==0,1,count_thriller)), "b_
    thriller"  = ifelse(sum_thriller==0,0,(mean_thriller - mean_rating ))) %>%
     select(userId, mean_thriller, b_thriller)

genre_musical_edx <- genre_assignment_matrix_edx %>% group_by(userId) %>%
    summarise ("mean_rating" = mean(rating)  , "count_musical" = sum(Musical
    == 1), "sum_musical" = sum((ifelse(Musical==0,0,rating)))  , "mean_musical
    " = sum_musical/(ifelse(count_musical==0,1,count_musical)), "b_musical"  =
     ifelse(sum_musical==0,0,(mean_musical - mean_rating ))) %>% select(userId
    , mean_musical, b_musical)

genre_sci_fi_edx <- genre_assignment_matrix_edx %>% group_by(userId) %>%
    summarise ("mean_rating" = mean(rating)  , "count_sci_fi" = sum(Sci_Fi ==
    1), "sum_sci_fi" = sum((ifelse(Sci_Fi==0,0,rating)))  , "mean_sci_fi" =
    sum_sci_fi/(ifelse(count_sci_fi==0,1,count_sci_fi)), "b_sci_fi"  = ifelse(
    sum_sci_fi==0,0,(mean_sci_fi - mean_rating   ))) %>% select(userId, mean_
    sci_fi, b_sci_fi)

genre_romance_edx <- genre_assignment_matrix_edx %>% group_by(userId) %>%
    summarise ("mean_rating" = mean(rating)  , "count_romance" = sum(Romance
    == 1), "sum_romance" = sum((ifelse(Romance==0,0,rating)))  , "mean_romance
    " = sum_romance/(ifelse(count_romance==0,1,count_romance)), "b_romance"  =
     ifelse(sum_romance==0,0,(mean_romance - mean_rating  ))) %>% select(
    userId, mean_romance, b_romance)

genre_war_edx <- genre_assignment_matrix_edx %>% group_by(userId) %>%
    summarise ("mean_rating" = mean(rating) , "count_war" = sum(War == 1), "
    sum_war" = sum((ifelse(War==0,0,rating)))  , "mean_war" = sum_war/(ifelse(
    count_war==0,1,count_war)), "b_war"  = ifelse(sum_war==0,0,(mean_war -
    mean_rating ))) %>% select(userId, mean_war, b_war)

genre_western_edx <- genre_assignment_matrix_edx %>% group_by(userId) %>%
    summarise ("mean_rating" = mean(rating)  , "count_western" = sum(Western
    == 1), "sum_western" = sum((ifelse(Western==0,0,rating)))  , "mean_western
    " = sum_western/(ifelse(count_western==0,1,count_western)), "b_western"  =
     ifelse(sum_western==0,0,(mean_western - mean_rating   ))) %>% select(
    userId, mean_western, b_western)

# Create consolidated Genre User Effects dataframe

genre_user_summary_edx <- genre_drama_edx %>%
  left_join(genre_comedy_edx, by='userId') %>%
  left_join(genre_action_edx, by='userId') %>%
  left_join(genre_thriller_edx, by='userId') %>%
  left_join(genre_adventure_edx, by='userId') %>%
  left_join(genre_romance_edx, by='userId') %>%
  left_join(genre_sci_fi_edx, by='userId') %>%
  left_join(genre_crime_edx, by='userId') %>%
  left_join(genre_fantasy_edx, by='userId') %>%
  left_join(genre_children_edx, by='userId') %>%
  left_join(genre_horror_edx, by='userId') %>%
  left_join(genre_mystery_edx, by='userId') %>%
```

```
  left_join(genre_war_edx, by='userId') %>%
  left_join(genre_animation_edx, by='userId') %>%
  left_join(genre_musical_edx, by='userId') %>%
  left_join(genre_western_edx, by='userId') %>%
  left_join(genre_film_noir_edx, by='userId') %>%
  left_join(genre_documentary_edx, by='userId')

# Extract all Genre User Effects values for all Users
genre_effects_user_edx <- genre_user_summary_edx %>% select(userId, b_action,
    b_adventure, b_animation, b_children, b_comedy, b_crime, b_documentary, b_
    drama, b_fantasy, b_film_noir, b_horror, b_musical, b_mystery, b_romance,
    b_sci_fi, b_thriller, b_war, b_western)


# Rmeove Individual Genre Tables to Save Memory and avoid Confusion
rm(genre_action_edx,genre_adventure_edx,genre_animation_edx, genre_children_
    edx, genre_comedy_edx, genre_crime_edx, genre_documentary_edx, genre_drama
    _edx, genre_fantasy_edx, genre_film_noir_edx, genre_horror_edx, genre_
    musical_edx, genre_mystery_edx, genre_romance_edx, genre_sci_fi_edx, genre
    _thriller_edx, genre_war_edx, genre_western_edx)
```

Model 7 - User Effects (R), Movie Effects (R), Genre Effects (R) & Time Effects

The Model can be Written as

$Y_{i,j} = \mu + \alpha_{i(\lambda_{ij})} + \beta_{j(\lambda_{ij})} + \frac{1}{K}\sum_{k=1}^{K} x_{i,k}\beta_{i,k} + \beta_{i,t} + \beta_{j,t} + \varepsilon_{i,j}$

Where,

$Y_{i,j}$ is the Predicted Rating

μ is the Mean Value of all known Ratings for all Movies

$\alpha_{i(\lambda_{ij})}$ is the Regularised Treatment Effect for each user i .

$\beta_{j(\lambda_{ij})}$ is the Regularised Treatment Effect for each Movie j .

K is the number of genre attached to the movie, If no genre are attached, then $K = 1$

$\beta_{i,k}$ is the Genre User Treatment Effect for genre k for user i

$\beta_{i,t}$ is the Time User Treatment Effect for User i based on the Year of Rating

$\beta_{j,t}$ is the Time Movie Treatment Effect for Movie j based on the Year of Rating

$\epsilon_{i,j}$ is the Random Error

```
# Compute Time Effects for Movies for edx

time_assignment_movie_edx <- edx %>% mutate(movieId = as.factor(movieId)) %>%
  mutate(date = as_datetime(timestamp)) %>%
  mutate(date_year_j = year(date)) %>%  select(movieId,rating, date_year_j)
     %>%
  group_by(movieId, date_year_j) %>%
  summarise(mean_yearly_rating = mean(rating, na.rm = TRUE), .groups = "drop")

time_assignment_n_years_movie_edx <- time_assignment_movie_edx %>%
  group_by(movieId) %>%
  select(movieId, date_year_j) %>%
  summarise(movieId, date_year_j, n_years_j = n(), .groups = "drop")
```

```
# Extract Unique MovieId
movie_map_edx <- edx %>%
  select(movieId) %>%
  group_by(movieId) %>% summarise(movieId= unique(movieId), .groups= "drop")

# Compute Average Rating for each movie along with number of ratings for edx
movie_rating_edx <- edx %>% group_by(movieId) %>%
  summarise(avg_rating = mean(rating), number_of_ratings = n())

# Combine above information into existing Movie Map
movie_list_edx <- left_join(movie_map_edx, movie_rating_edx,by="movieId")

# remove datasets that are no longer needed
rm(movie_rating_edx, movie_map_edx)

time_effects_movie_edx <- movie_list_edx %>%
  ungroup %>% mutate(movieId = as.factor(movieId)) %>%
  select(movieId, avg_rating) %>%
  left_join(time_assignment_movie_edx, by = "movieId") %>%
  mutate("beta_jt" = (mean_yearly_rating - avg_rating)) %>%
  left_join(time_assignment_n_years_movie_edx, by = join_by(movieId,date_year_
      j)) %>%
  select(movieId, date_year_j, beta_jt, n_years_j)

# Compute Time Effect for Users

time_assignment_user_edx <- edx %>%
  mutate(userId = as.factor(userId)) %>%
  mutate(date = as_datetime(timestamp)) %>%
  mutate(date_year_i = year(date)) %>%  select(userId,rating,date_year_i) %>%
  group_by(userId, date_year_i) %>%
  summarise(mean_yearly_rating = mean(rating, na.rm=TRUE), .groups = "drop")

time_assignment_n_years_user_edx <- time_assignment_user_edx %>%
  group_by(userId) %>%
  select(userId, date_year_i) %>%
  summarise(userId, date_year_i, n_years_i = n(), .groups = "drop")

time_effects_user_edx <- edx %>% mutate(userId = as.factor(userId)) %>%
  group_by(userId) %>% summarise(avg_rating = mean(rating)) %>%
  select(userId, avg_rating) %>%
  left_join(time_assignment_user_edx, by = "userId") %>%
  mutate("beta_it" = (mean_yearly_rating - avg_rating)) %>%
  left_join(time_assignment_n_years_user_edx, by = join_by(userId,date_year_i)
      ) %>%
  select(userId, date_year_i, beta_it, n_years_i)

# Let us prepare the test set for input of time effects. We will reuse the
    existing dataset to avoid rework

genre_assignment_test_final <- genre_assignment_test_final %>%
  mutate(date = as_datetime(timestamp)) %>%
  mutate(date_year_i = year(date)) %>%
```

```
  mutate(date_year_j = year(date))

rm(time_assignment_edx, time_assignment_n_years_edx)
```

Model 8 - Predictions Rounded to Discrete Values

Here we round the Predictions from Model 7 to the closest discrete values on the rating scale.

```
## [1]  RMSE with Predictions Rounded to Discrete Values:
## [2] 0.86037
```

3.1.0.1 Linear Models - Best RMSE

```
## [1]  The Best RMSE is:  0.86037
```

We have managed to achieve our initial Goal of achieving an RMSE < 0.86490 with the ratings set to discrete values. Though not the numerically lowest, It is the RMSE that makes the most sense keeping in mind the context of our requirements.

```
############################################################
# END MOVIELENS PROJECT
############################################################
```

4 Conclusion

In this final chapter, we will review what we have done on our Project.

For any project, success is highly dependent on its ability to meet the stated business needs in full or at the least meet the most critical business needs.

Requirements for Recommendation Systems

Let us review how we have addressed the Requirements for Recommendation Systems that have the greatest bearing on our Project.

4. A large diversity in the Customer types and buying behaviour.

This is something that we have repeatedly emphasised using our rating trends and distributions for users and movies alike. There is a huge diversity in users' rating behaviour and consequently the ratings received by movies. These make our Models more challenging to design as there is not one Model that can be applied for all or even a large majority of users. Whatever Models we build must take this diversity into account.

8. An increased emphasis on Personal Data Protection and User Privacy which limits the amount of information that businesses can collect, retain and share about Customers.

We have seen how to work with Anonymised data, where we do not have the ability to understand the User's country, language, race, age, gender or stated preferences for movies, genre, cast, time period of release, or any other such indicators that might be used to identify or profile a single user or groups of users. We have built an algorithm that makes predictions using the bare minimum data.

9. Availability of many Mathematical algorithms and methods that are often used in conjunction with each other to build a system that is specific to the business and the Customers that it serves. Any algorithms and methods used must be efficient in terms of cost, time and resources and be fit for purpose.

We have painstakingly built our Linear Regression Models so that we can get insight into the data. We have expanded our data to increase the accuracy of our predictions. We have optimised our models to ensure that they provide enough gains for the effort spent. We have done everything with frugality in mind.

For notes related to Random Forest, KNN and Naive Bayes, please read through Appendix B.

Data Analysis

We have demonstrated some very important concepts in Data Analysis.

We first had a look at using Linear Regression Models in Recommendation Systems. Particularly, How to

1. Tune Predictions based on different Effects like User Effects, Movie Effects, Genre Effects and Time Effects
2. Use Regularisation to help reduce the Effects of outliers in skewing results.
3. Extract information about different Effects, even when they are not explicitly mentioned.
4. Optimise and remove effects when they do not provide sufficient gains for the efforts spent.

We have also seen, How

1. The application of different Effects causes different data entries to emerge or surface to the top in our Predictions.
2. These Effects interact with each other to complement or negate each other.

3. Considering them together or in isolation can influence our end results.
4. The application of Regularisation can increase or decrease the accuracy of our Predictions. We have also seen where it can help and where it cannot.
5. Predictors may initially appear provide good improvements but may not match up to our expectations when applied to the data.
6. To validate our assumptions with Data & Analysis.
7. Linear Regression Models can help us visualise and understand Data very well but have their own limitations for Solving Classification Problems.

Project Constraints

This particular Project has been primarily focussed on achieving the target RMSE using the simplest possible techniques and computing hardware that is commonly available. Though not the most ideal of goals, it should help motivate and encourage those who are just starting off with such Projects. It should also help them get a feel for the data and its characteristics.

The Code should work on any Student Laptop as under

1. The Linear Models should work with about 8GB of Free Memory for R
2. The Linear Models along with the Plots should work with about 16GB of Free Memory for R.
3. Any lesser and it is likely that RStudio will terminate and restart R.

Try to run garbage collection often or after each major section to ensure that you do not run out of memory.

If you plan to use Multi-Threading, please make sure you have enough Memory to spare for Multi-Threading to work.

Future Development

The Project is definitely not the best in terms of efficient code. I am a Novice in R and have just learnt the language whenever I could find a bit of time to study & practice, it is definitely possible to clean up the code and rewrite it to be more elegant and efficient.

The Project has also not touched upon several important topics that are of interest in Recommendation Systems like Matrix Factorisation.

Models which can identify and group similar users together based on their choice of movies and genres can help make our predictions faster and more accurate. These are some areas for further development. One important constraint though is that the "Training" of these models require much better computing resources than provided by Laptop Workstations. Cloud Computing Services and Desktop Workstations can be extremely powerful and provide much superior computing resources.

We can also look at applying the Techniques we have discussed to other datasets with similar Predictors to understand their performance.

Areas of development are aplenty and I hope that the Readers see this Report as useful utilisation of their time, will learn from it and improve upon what has been done so far.

I will end the report with a note of thanks to the Readers and Evaluators for their time and patience and everyone who has helped, directly and indirectly, in the execution of the Project and the creation of this Report.

Appendix A - Alternate Models

I have explored several alternate models for this Project and provide some notes here about them. These are based on my own experience and are meant to be informational. Readers can choose to skip this entirely.

The code in this appendix is only provided as reference if someone wishes to pick it up and refine it further. It is not part of the R file or the PDF report and is only available in the Rmd file. Upstream datasets should be available before running the code.

4.0.1 Random Forest in Classification Mode

The First Alternate Model is Random Forest in Classification Mode. Based on the experience gained with running the Algorithms, we must take some precautions to ensure that we can carry out our Analysis with the computing power available at our disposal.

Loading the whole dataset consumes large amounts of memory and takes a very long time to finish. To just get a feel for the predictions that can be made by Random Forest

1. We can use only 11.2% of the "edx" dataset viz. about 1008008 Observations for evaluating our Algorithm. We will use 90% viz. 907852 Observations for Training and 10% viz. 100156 Observations for Testing our Model. This Should cover about 10% of our larger "MovieLens 10M Dataset".
2. We can use the most significant genre in terms of Variability viz. Drama, Crime, War, Action, Comedy and Horror
3. Not use Caret for any of the CV or Bootstrapping features that it provides and instead use the native implementation of the Algorithm.

We also modify our dataset as under

1. Set "userId" and "movieId" as characters rather than factors. The Random Forest Implementation can only handle a limited set of factors. The numeric value of the "userId" and "movieId" is insignificant and not very relevant for purposes of computation.
2. Configure the Timestamp field to only include the Year that the rating was assigned to prevent unnecessarily loading the Algorithm.
3. Set the "rating" field as a factor as we wish to run Random Forest in Classification Mode.

For anybody trying to run the code on their own computers, it is strongly advised that at least 48GB of Free Memory be available and that they start with a fresh instance of R by clearing out, any remaining datasets from the previous chapters from memory.

Loading the entire dataset on a Laptop Workstation is not possible. It can be loaded on a Desktop Workstation or Cloud Server provided sufficient memory is available or can be provisioned.

4.0.2 K Nearest Neighbours (KNN)

Though at first glance KNN seems to be a very good fit for the dataset, we soon discover some practical difficulties once we start delving deeper into the details.

As we have noted several times earlier, a huge bulk of the information about the predictions is contained in the "userId" and "movieId" fields. The numerical values of these fields are not relevant as they are just used as character strings to identify the User and the Movie and are essentially categorical variables.

Setting the "userId" and "movieId" as Characters or Factors causes KNN to ignore these fields during computation. Converting the "userId" and "movieId" to numerically significant values using dummy variables

causes the number of dimensions to become unmanageably large and is not practical for the whole dataset. As a reminder we have about 69878 Unique Users and 10667 Unique Movies in the "edx" dataset.

The implementation of KNN using the "class" or "FNN" libraries does not allow for specification of the distance type and uses Euclidean distances by default. Euclidean distances are more suited for Regression than they are for Classification.

The "caret" package while highly regarded for ease of use and the utility it provides, also needs plenty of memory and causes memory exhaustion for even small datasets when used with the defaults. A lot of tuning is required to get "caret" to work without memory exhaustion.

Computation using the "knn" function from the "class" library uses a single thread and takes a lot of time, Parallel Computation using the "parallel" or "future" libraries for different values of K can help speed up things to a certain extent.

With KNN, as the optimal "K" value is highly dependent on the dataset, it is extremely erroneous to use a smaller subset of the data to obtain an optimal "K" value and extrapolate it to a larger dataset. Such K values are much smaller for the smaller subset of the dataset as compared to the complete Set. So we cannot obtain an optimal value of "K" on a smaller subset and extend it to the larger dataset to save on computing time.

When KNN is used without taking these constraints into consideration, it can generate Predictions that are wildly off target and have very poor RMSE after running for several hours or sometimes days leading to plenty of wasted effort and computing time.

4.0.3 Naive Bayes

Like KNN, Naive Bayes sadly seems to flatter to deceive. We would expect that Naive Bayes should give us very good results as it is very well suited for Classification problems. We can use the "naive_bayes" function from the "naivebayes" library or the "naiveBayes" function from the "e1071" library.

Ideally, Naive Bayes should not need as much computing resources as Random Forest and KNN.

For Naive Bayes to give us the right results, we will need to do some pre-work and convert all our Predictors to formats that are suitable for Classification. We need to configure the Training and Test Data set as under

1. Configure the "userId" and "movieId" predictors as characters as their numerical values have no significance. Setting them up as factors causes errors that abort the function as Naive Bayes expects that the same levels be available on both the Training and Testing Sets. Setting them up as numerics makes no sense.
2. Convert "rating" to factor as it's numerical value is not a continuous variable. If we do not convert it as a factor and retain it as a numeric, Naive Bayes will interpret it as a Continuous Variable and fit a Model accordingly. The "rating" Variable has 10 levels viz [0.5,1.0,1.5,2.0,2.5,3.0,3.5,4.0,4.5,5.0]
3. Add Genre Predictors. Please see code for more details
4. Convert the date year to factor. Same logic as for "rating" and "genre" predictors applies.
5. Model the prediction for Classification.

With the base set of features, the predictions from Naive Bayes are extremely poor even for the Training set with an RMSE above 1.7 and an prediction accuracy of about 0.27 which is terrible. The Predictions for the Testing set are just about the same which seems to indicate random choice.

Adding the Genre predictors does seem to help better the RMSE. However, it is still too large. The confusion matrix indicates that Naive Bayes is making far too many predictions for ratings "3" and "4". Replacing the "genre user mean" with "genre user effects" causes the RMSE to degrade a lot and the ratings get spread all over.

Appendix B - Additional Notes & Cautions

Here we record some information that is not related to the Project directly but may help in the conduct of this Project or others. It is meant to save Readers some time and effort and avoid some of the pitfalls that might appear along the way. Most of the content is based on practical observation and must be seen as such rather than being construed as expert opinion.

RStudio Integrated Development Environment (IDE)

Using the RStudio IDE is highly recommended for anybody who is just starting off with R. The IDE provides a lot of features which make Code Development, Verification, Modification, Testing and Debugging easy.

RStudio can also help in the creation, editing and management of R Scripts and R Markdown documents which can greatly simplify the aforementioned tasks. R Markdown documents are extremely helpful to interweave narratives with the Code in a common document.

RStudio also allows for the easy management of Libraries, Environments and Projects. It can also help view and manage chunks of code and variables in code more intuitively than is possible using a simple command line interface.

More details are available at :- https://posit.co/products/open-source/rstudio/

R Installation & Maintenance

R can be installed in different ways depending on the Reader's familiarity with various Operating Systems and Installation Techniques. For more details, please have a look at :-

https://cran.r-project.org/doc/FAQ/R-FAQ.html#How-can-R-be-installed_003f

Installing R using the Operating System's (OS) Native Package Management Systems is a lot more convenient. They provide support for automatically checking dependencies and conflicts and greatly ease Integration with RStudio. They also facilitate easy upgrades and roll-backs. They should work for most readers. The landing page below provides links to different OS and Distributions :-

https://cran.r-project.org/bin/

Installing R on any Unix like OS or Linux using the traditional CMM ("configure", "make", "make install") method provides greater flexibility in configuration and allows customisation of the Installation directories, Math libraries, Dynamic linking and generating PDF reference manuals etc. However, this requires manual upkeep and is something that should be done only after understanding how the R installation integrates with the rest of the system. Small mistakes can cause R to break, R Integration with RStudio to break or in the worst case, when used with Superuser privileges, cause the Operating System to break.

So any method that you choose has some advantages and disadvantages. Choose a method that works for you. If you are unsure, it is strongly recommended to stick with the versions provided by your OS' Package Management Systems.

As a note of caution, it is not a good idea to update R midway through your Project as some Libraries may be unavailable for the newly updated version of R. R is an extremely fast evolving project while Libraries are sometimes slower to catch up.

Garbage Collection

The Garbage Collection and Memory Release in R is not Synchronous and can sometimes cause large chunks of Memory to stay locked up until the Garbage Collector runs again. Invoking the Garbage Collection function gc() explicitly after deletion of of large datasets, finishing large matrix operations, finishing computations like Random Forest or deleting temporary variables used to hold large plots etc. can help free up memory for subsequent operations in R or for other Applications running on the Computer. This is particularly helpful if the Code is being developed or run on a Memory Constrained Computer.

Increased Memory Utilisation when using the caret package

When using the highly regarded and feature rich "caret" (Classification and Regression Training) package for Analysis, one must be aware that the defaults in caret are not optimised for Memory usage and tuning is required to achieve the same. With the defaults, caret consumes huge amounts of memory when working with large datasets. It can easily exhaust a Laptop Workstation's memory when Multifold Cross Validation or Bootstrapping are used. Similarly, multiple values of Algorithmic Hyper-parameters, can lead to high memory utilisation or memory exhaustion. Some examples of such Hyper-parameters are "mtry" values in Random Forest or "k" values in KNN.

R Markdown Conversion to PDF

R Markdown Conversion to PDF is not something very trivial. Though everything may look clean and tidy on the RMarkdown document, during conversion to PDF, we can often see Tables getting clipped, Plots getting cropped, squished or overlaid on top of each other, Titles and Text in Plots getting misaligned, Console Text output not aligning with the margins and several such issues.

When code needs to be interwoven along with the Text, the challenges increase manifold. However, not to despair. Help is available.

https://bookdown.org/yihui/rmarkdown-cookbook/

https://mirror.niser.ac.in/ctan/macros/latex/contrib/listings/listings.pdf

Some awareness of LaTeX and manual adjustments are in order to ensure that the rendered PDF documents are legible and not too misaligned.

Prefer Vector notation to formula notation where possible

Where possible, It is better to specify the Predictors & the Dependent Variables using the vector notation rather than use the formula notation as the computation is much faster and also consumes lower memory.

Running Time & Computing Resources for Various Algorithms

For beginners, it is often very difficult to understand how much Time & Computing Resources R requires to run various Algorithms based on the size & complexity of their datasets and the Libraries used to run these Algorithms.

Most of the Algorithms in R should finish successfully or throw up an error within a reasonable amount of time, However should there be a bug in the Code or an error or inefficiency in the specification of Predictors, these Algorithms can sometimes run for days before one realises that they are caught in an loop or will take far too much time to finish.

Similarly if an Algorithm seems to crash repeatedly due to lack of Computing Resources, it could be a natural occurrence due to the size & complexity of the dataset or due to some of the reasons cited earlier.

If any Algorithm seems to be doing these, it is better to research articles written around these topics or lookup questions posted on forums, rather than wait or repeatedly retry with random changes. Here are some examples.

https://stats.stackexchange.com/questions/497491/running-time-rf

https://stackoverflow.com/questions/37190135/how-can-i-speed-up-the-training-of-my-random-forest

https://github.com/szilard/benchm-ml

You can also check with someone who is an expert at the topic. There are several Internet forums where one can request expert help.

https://stats.stackexchange.com/?tags=r

https://stackoverflow.com/questions/tagged/R

Please follow the guidelines issued by these forums for requesting help.

Parallelisation, Math Libraries & Swap Files/Partitions

Parallelisation or Multi-Threading can be beneficial in speeding up operations when the Workstation has multiple CPU Cores. Since R without any tweaking is Single Threaded, some operations performed sequentially can take a very long time.

Parallelisation can help to a large extent if

1. Multiple datasets can be manipulated independently.
2. Subsets of data in an individual dataset can be manipulated independently.
3. The functions used for manipulation support Parallelisation.

You can turn on Parallelisation in R using the Parallel or Future Libraries.

Read the excellent Article on how to do it here:-

https://berkeley-scf.github.io/tutorial-parallelization/parallel-R

However, Multi Threading also leads to increased memory consumption as multiple independent instances of R are started to work on the data.

https://berkeley-scf.github.io/tutorial-dask-future/R-future#6-avoiding-copies-when-doing-multi-process-parallelization-on-a-single-node

For a more comprehensive list of the various Parallelisation Schemes possible in R, The Reader can refer to the landing page of the R High Performance Computing Group

https://cran.r-project.org/web/views/HighPerformanceComputing.html

Many modern Multi Core CPU provide Dynamic Frequency Scaling to higher Core speeds when a single core or a few cores are Active. However, in most Multi Core CPU, particularly those on Laptop Workstations, Having all cores active for longer periods of time can lead to "throttling" or reduction in speed of the individual cores. Seldom do we see the cumulative performance increase by as many times as the number of parallel threads.

Any tweaking must be done after adequate observation of all parameters and due diligence.

R includes default BLAS (Basic Linear Algebra Subprograms) and LAPACK (Linear Algebra Package) Libraries with the basic installation which are extremely capable and quite fast. They should be good enough for most work.

However, depending on the Processors used and the workloads, using different BLAS can help speed up operations further.

OpenBLAS is an Open-Source implementation of the BLAS and LAPACK API with optimisation for several processor architectures. Several Linux Distributions provide OpenBLAS through their Standard Package Management methods.

The Processor Manufacturers also provide their own BLAS and LAPACK Packages.

For Intel Processors, the landing page has links to the procedure to be followed. You can find it here :-

https://www.intel.com/content/www/us/en/developer/tools/oneapi/onemkl.html

Similarly For AMD Processors, the landing page can be found here :-

https://www.amd.com/en/developer/aocl.html

*Note: Make sure you read through the License Agreements and choose the right Licensing.

Please see notes on how to install libraries for Intel MKL and OpenBLAS here :-

http://cran.r-project.org/doc/manuals/r-release/R-admin.html#Shared-BLAS

*Note: As previously cautioned in the section on R Installation, It is strongly advised that you make changes only after you understand how the libraries link and work together, otherwise you could easily end

up breaking your R Installation, Integration with RStudio or in the worst case break your Operating System. If you are unsure, it is very strongly recommended to stick with the defaults provided with R.

If you are using SSD, It is strongly recommended that you turn off any Swap partitions or Swap files, resident on the SSD. If there is a lot of swap activity when manipulating large data sets and the activity extends over a large period of time, it can decrease the life of your SSD.

Matrix Algebra & High Dimensional-Data Analysis

The Basic knowledge of Matrix Algebra will help in understanding Data Representation and Processing using Matrices. Such Representations are compact and often much easier and faster to process in R, thanks to the fact that R has been built to speed up Matrix operations. They are also used in Recommendation Systems for Factor Analysis and Dimension Reduction. It is suggested that anybody who is interested in knowing more about Recommendation Systems get some basic knowledge of Matrix Algebra. The following course available on the edX platform should help you get started.

https://www.edx.org/learn/linear-algebra/harvard-university-introduction-to-linear-models-and-matrix-algebra

Similarly the following course should help you understand and work with data that is highly dimensional.

https://www.edx.org/learn/data-analysis/harvard-university-high-dimensional-data-analysis