Harvard Capstone Project - Movie Recommendation System

HA W.M.ALEX

2024-10-16

Contents

1 Executive Summary

The Objective of Capstone Project is to develop a **Recommendation System** that **Predict Movie ratings** using a **Machine Learning Model** based on subset of **MovieLens** datasets. This subset represents a smaller portion of much larger datasets containing several Millions of **rating**. This datasets encompasses approximately **10 Millions Movies ratings**. The Primary task is to leverage this data to **Predict Movie ratings**.

For this endeavor, this **MovieLens** datasets is partitioned into **edx** sets for developing the Algorithm , comprising approximately **9 Million** Rows and **final_holdout_test** sets for **Testing on Final Model**, consisting approximately **1 Millions** Rows. The **edx** datasets includes **69,878 Unique Users**, **10,677 Unique Movies** and **797 Combined genres**. Each **Movie** is also categorized by its **Combined genres** and **ratings** range from **0.5** and **5.0** with increment of **0.5**.

Prior to Models Building, **Data exploration Analysis (EDA)** were conducted which encompassing Data cleaning, Data pre-processing/Featuring Engineering, Data Visualization, Stratification Analysis, Distributions Analysis and Frequency Analysis. The Model employs a **Collaborative Filtering Approach**, augmented by **Regularization Method** to estimate the **Movie Effects**, **User Effects**, **Genres Effects** and **Time Effects**. These Methods are instrumental in penalizing the magnitude of the **Parameters** to avoid **Overfitting**.

The **edx** datasets is also split into **train sets** and **test sets** for training and testing the **Algorithm** using **Cross-Validation Method**. Our ultimate goal is to construct a Model that minimizes the loss, measured by our **loss function, Root Mean Squared Error (RMSE)**.

If N is the **number of User-Movie combinations**, $y_{u,i}$ is the **rating** for **Movie** i by **User** u, and $\hat{y}_{u,i}$ is our **Prediction**, the **RMSE** is defined as follow:

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

Regularized Movie+User+Time Based Model comply with an **Lowest RMSE** of **0.863759**.

The **Final Model** is defined as follow:

$$Y_{u,i} = \mu + b_i + b_u + f(bt_i) + \varepsilon_{u,i} \text{ with } f \text{ a smooth function of } bt_i$$

$$\hat{bt}_i(\lambda) = \frac{1}{\lambda + n_i}\sum_{u=1}^{n_i}(Y_{u,i} - \hat{\mu t}_i - \hat{b}_u(\lambda)) \text{ ,with } \hat{\mu t}_i = \hat{f}(x_0) = \frac{1}{N_0}\sum_{i \in A_0} Y_i \quad , |x_i - x_0| \leq 7$$

The Model's **Performance** was evaluated using the Metric **- Root Mean Squared Error**. The **reliability** and **trustworthiness** of the Model are substantiated through incorporating **Regularization** and **Cross-Validation Method** during Model development.

Optimal Machine Learning Model achieved an RMSE of 0.863759, Signifying a Remarkable Outcome. Thus, I firmly assert our confidence in adopting the ultimate Model and Algorithm to construct the Movie Recommendation System.

2 Introduction

2.1 Movie Recommendation System

Purpose of the Project is to build a **Recommendation System** based on **MovieLens** Dataset that **Predict Movie ratings** using **Machine Learning Algorithm**. The **Movie ratings** are on a **0.5-5.0** scale, where **5.0** represents the **best Movie** and **0.5** suggests it to be a **bad Movie**. **Movie Recommendation System** are more complicated Machine Learning challenges because each Outcome has a different set of **Features/Predictors**. For example, different Users Rate a different number of Movies and Rate different Movies. Consider the following scenario:

- If the average **rating** for all **Movies** is **3.7**, and **"Jurassic Park"** is better than an average **Movie**. User **"A"** might Rate it **0.5** points above the average.
- User **"B"** is a cranky User, tends to Rate **0.8** points lower than average. Thus, the estimate for **"Jurassic Park"** by User **"B"** would be calculated as **3.7-0.8+0.5=3.4**.

2.2 Overview

Many of the Movie **ratings** are influenced by **Effects** associated with either **Users** and **Movies** of their interactions. Different **Users** employ different **rating** scales, and a **User** can change their **rating** scales over **time** or **genres**. A **Movie's** popularity may also change over **time** or **genres** due to external events. For example, cranky **User** who tended to Rate an average Movie **4.0**, may now Rate such a movie **3.0** or even lower. But in some occasion, those user may Rate much higher.

To address this, we add **Time Effects** and **Genres Effects** to the baseline **features/predictors**. Including **Time Specific Effects** does not attempt to capture future changes but aims to capture transient effects that significantly influenced past User feedback. Some people may like a **Movie** and remember it as my most favorites because of it's **genres**. While others may dislike it and forget about it. Thus only those liking them will mark the **Movie** as favorites, while those disliking them will not mention them at all. This behavior is expected towards most popular Movies, which can be either remembered as very good or not to be remembered.

However,some Movies are known to be bad and people who did not like them always give them a lower scores, indicating what they do not want to watch. However, for the other part of the population, who liked those Movies, they are not going to be remembered long Time as notable. Thus, long Time after watching the Movie, only those who disliked the Movie will Rate it. Some Movies are natural selection of Movies to be Rated. some Movies are natural candidates as bad Movies, while others are natural candidates as good Movies.

Thus, **Time Effects** and **Genres Effects** can explain **Portion of Variability** of the Movie **ratings**.

3 Data Exploration Analysis (EDA)

3.1 Data Explorations

The **MovieLens** datasets comprises approximately **10 Million** Rows of data. This datasets is randomly split into two separated datasets, **edx** and **final__holdout__test**. The **edx** datasets serves as the **training sets**, while the **final__holdout__test** datasets is used for **Final Model Testing** Purpose. Both datasets contain **6 Columns/Features/Predictors**. The **edx** datasets includes approximately **9 Millions** of Rows with **69,878 Unique Users**, **10,677 Unique Movies** and **797 combined genres**. Movie **rating** in this datasets range from **0.5** and **5.0** with increments of **0.5**.

A) Dimension

Table 1: edx datasets

Rows	Columns
9000055	6

There are **9000055** Rows and **6** Columns in the **edx** datasets.

B) Column Name and Class

Table 2: edx datasets

	Class
userId	integer
movieId	integer
rating	numeric
timestamp	integer
title	character
genres	character

There are **6** Columns and Class in the **edx** datasets.

C) Number of Unique movieId, userId and genres.

Table 3: edx datsets

Description	Count
Number of Unique movieId	10677
Number of unique userId	69878
Number of unique combined genres	797

There are **10677** Unique **Movies**, **69878** Unique **Users** and **797** Unique **Combined genres** in the **edx** datasets.

List of 10 Examples - Counting the Occurrences of rating (By movieId)

```
# - generate 10 examples - counting the Occurrences of rating by movieId
# - display output using kable function

m_result <- edx %>% count(movieId,name="Occurency") %>%
                     count(Occurency, name="Count") %>%
                     arrange(desc(Count)) %>%
                     dplyr::slice(1:10)
m_result %>% knitr::kable(caption="edx datasets")
```

Table 4: edx datasets

Occurency	Count
4	154
2	152
1	126
5	120
3	119
6	107
11	107
7	96
12	93
8	91

List of 10 Examples - Counting the Occurrences of rating (By userId)

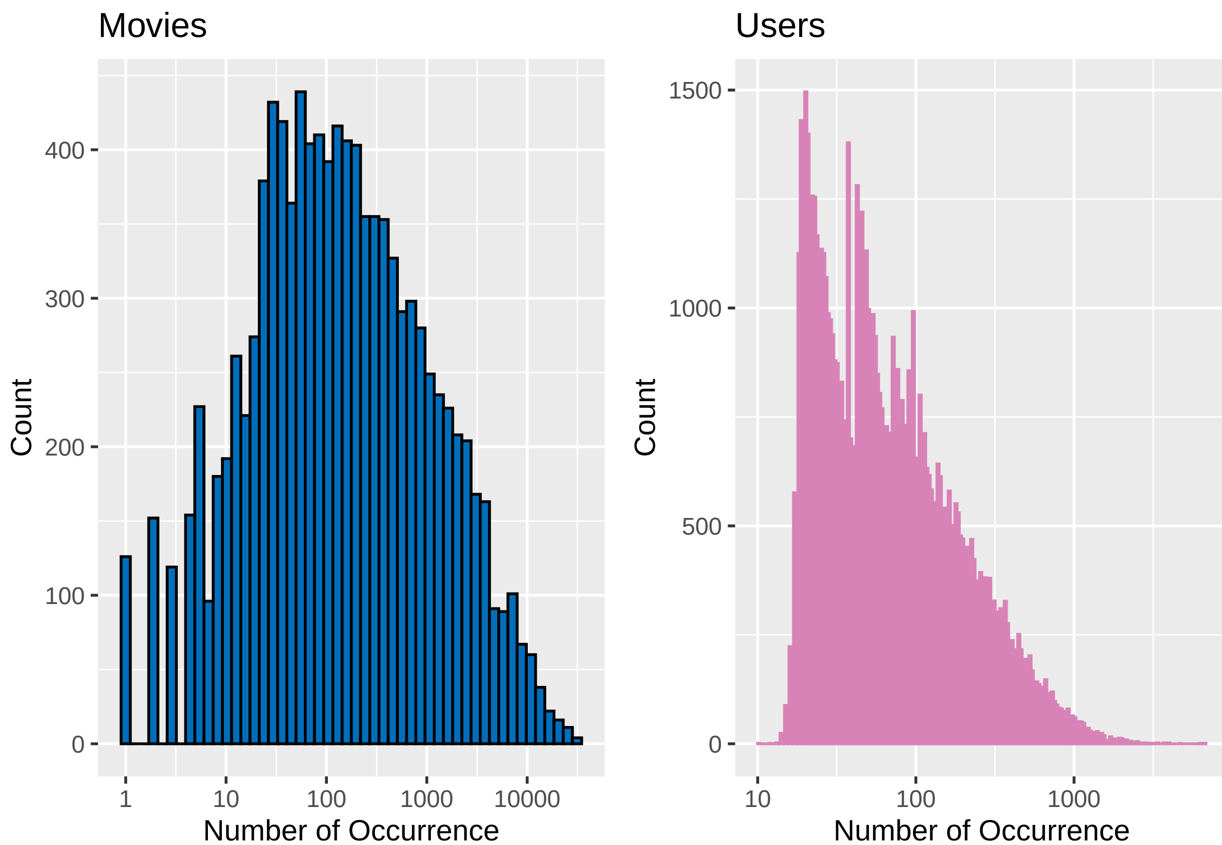
```
# - generate 10 examples - counting the occurrences of rating by userId
# - display output using kable function

u_result <- edx %>% count(userId,name="Occurency") %>%
                     count(Occurency, name="Count") %>%
                     dplyr::slice(30:40)
u_result %>% knitr::kable(caption="edx datasets")
```

Table 5: edx datasets

Occurency	Count
40	682
41	568
42	641
43	640
44	636
45	607
46	613
47	532
48	574
49	557
50	510

Plot Distributions (Number of Occurrence) - movieId and userId

```
# - generate distributions (number of occurrence) using movieId and userId
# - ggplot histogram - m1 and m2

m1 <- edx %>%
          dplyr::count(movieId) %>%
          ggplot(aes(n)) +
          geom_histogram(bins = 50, fill="#006EBB", color="black") +
          labs(y="Count",x="Number of Occurrence") +
          scale_x_log10() +
          ggtitle("Movies")

m2 <- edx %>%
          dplyr::count(userId) %>%
          ggplot(aes(n)) +
          geom_histogram(bins = 200, fill="cyan", color="#D883B7"    ) +
          labs(y="Count",x="Number of Occurrence") +
          scale_x_log10()+
          ggtitle("Users")

grid.arrange(m1, m2, ncol = 2)
```



It is evident from the distributions plotted above that some Movies are rated more than the other Movies and some Users are more active and Rate more number of Movies than the other Users do.

3.1.1 Description of Columns/Features/Predictors

- **movieId** : A Unique identification number assigned to each **Movie**.
- **title** : The **title** for each **Movie** follow by the Release Year inside the parentheses "(1996)".
- **genres** : The **Combined genres** of each **Movie**, where each **genres** are separated by a pipe "|".
- **userId** : A unique identification assigned to each **User**.
- **rating** : The **rating** given by a Unique **User** for specific **Movie**.
- **timestamp** : The **timestamp** associated with a User's **rating** for a particular **Movie**.

3.1.2 Data Cleaning

R Codes check columns with "NA" values in "edx" datasets

```
# - generate missing value (if any) in column rating of edx
# - display output using kable function

na_results  <- edx[apply(is.na(edx),1,any),]

na_results %>% select(userId,movieId,rating,timestamp,title,genres) %>%
               knitr::kable(caption="Columns with NA values in edx datasets")
```

Table 6: Columns with NA values in edx datasets

userId	movieId	rating	timestamp	title	genres

As we can see from the output above, there are **NO missing value** in **rating** Column of **edx** datasets.

R Code check "rating" column for "Zeros" value in the "edx" datasets

```
# generate and display zeros value (if any) in column rating of edx

length(which(edx$rating==0))
```

```
## [1] 0
```

There are also **No zeros value** were given as **rating** in the **edx** datasets.

List of 8 Examples - edx datasets

```
# - generate 8 examples: the original edx - rating given by one User to one movie
# - display output of edx using kable function

    edx %>%
          group_by(title) %>%
          mutate(f=length(rating)) %>% ungroup() %>%
          filter(f > 10000) %>%
          distinct(title, .keep_all=TRUE) %>%
          arrange(desc(f)) %>%
          select(movieId,title,genres,userId,rating,timestamp) %>%
          dplyr::slice(1:8) %>%
          knitr::kable(caption="edx datasets")
```

Table 7: edx datasets

movieId	title	genres	userId	rating	timestamp
296	Pulp Fiction	Comedy|Crime|Drama	10	2.0	941529864
356	Forrest Gump	Comedy|Drama|Romance|War	1	5.0	838983653
593	Silence of the Lambs, The	Crime|Horror|Thriller	7	3.0	1049764435
480	Jurassic Park	Action|Adventure|Sci-Fi|Thriller	4	5.0	844416834
318	Shawshank Redemption, The	Drama	18	4.5	1111545917
110	Braveheart	Action|Drama|War	2	5.0	868245777
457	Fugitive, The	Thriller	6	5.0	1001083175
589	Terminator 2: Judgment Day	Action|Sci-Fi	1	5.0	838983778

Each Row in “edx” represents a “rating” given by one User to one Movie.

3.1.3 Pre-Processing/Feature Engineering

The **edx** datasets contain **Columns/Features/Predictors** like **timestamp**, **genres**, **rating**. Not all **Features** may be useful for **Prediction** and some may even be detrimental. Therefore we will select the most important **Features**. We will Transform and Extract the **timestamp** Column into **Week of Date** and **Year of Date** for readable format that better represent the underlying problem to the Model. Additionally, We will compute the **Mean of rating** by **genres**, **Week of Date** and **Year of Date**. These **new Features** are important for the Predictive Model as they provide insights into the problem, potentially **improving the Model Performance**.

3.1.3.1 Column genres

The **genres** Column includes every Genre that applies to **Movie** so I will define it as **Combined genres**. Some **Movies** also fall under several **genres**. The Extraction of **Combined genres** from the **Movie** into separate Rows is **NOT Necessary**, even those each Genre is separated by a pipe "|". In reality, **Movies** often belong to multiple **genres** simultaneously. Manipulating the original **Combined genres** Column by splitting it into individual Genre per Row could lead to **Data Misrepresentation**. It might introduce data bias when the **combined genres** of a **Movie** are artificially separated.

In truth, the **combined genres** for each **Movie** provide a more **Accurate Representation of Data**.

Following Example explain why Extraction of **Combined genres** into separate Rows is **Not Necessary**

3.1.3.1.1 Mean of rating by genres BEFORE Combined genres Separation

List Mean of rating by genres in descending order by Number of rating (count > 10000)

```
## # A tibble: 20 x 4
##    rank  genres                                        average_rating_genres  count
##    <chr> <chr>                                                         <dbl>  <int>
##  1 1     Crime|Mystery|Thriller                                         4.20  26892
##  2 2     Action|Adventure|Comedy|Fantasy|Romance                        4.20  14809
##  3 3     Adventure|Drama|Film-Noir|Sci-Fi|Thriller                      4.15  13957
##  4 4     Adventure|Drama|War                                            4.08  14137
##  5 5     Crime|Horror|Thriller                                          4.08  33757
##  6 6     Crime|Film-Noir|Mystery|Thriller                               4.06  24961
##  7 7     Comedy|Crime|Drama|Thriller                                    4.06  24341
##  8 8     Adventure|Mystery|Thriller                                     4.04  14712
##  9 9     Comedy|Drama|Romance|War                                       4.01  41762
## 10 10    Crime|Drama|Sci-Fi|Thriller                                    4.00  10730
## 11 11    Comedy|Crime|Drama                                             4.00  59071
## 12 12    Adventure|Children|Fantasy|Musical                             3.99  11784
## 13 13    Drama|War                                                      3.98 111029
## 14 14    Crime|Drama                                                    3.95 137387
## 15 15    Action|Adventure|Drama|Fantasy                                 3.94  11941
## 16 16    Action|Drama|War                                               3.92  99183
## 17 17    Action|Crime|Romance                                           3.91  16090
## 18 18    Crime|Drama|Film-Noir                                          3.91  11249
## 19 19    Adventure|Animation|Children|Comedy|Fanta~                     3.90  13063
## 20 20    Crime|Horror|Mystery|Thriller                                  3.88  27240
```

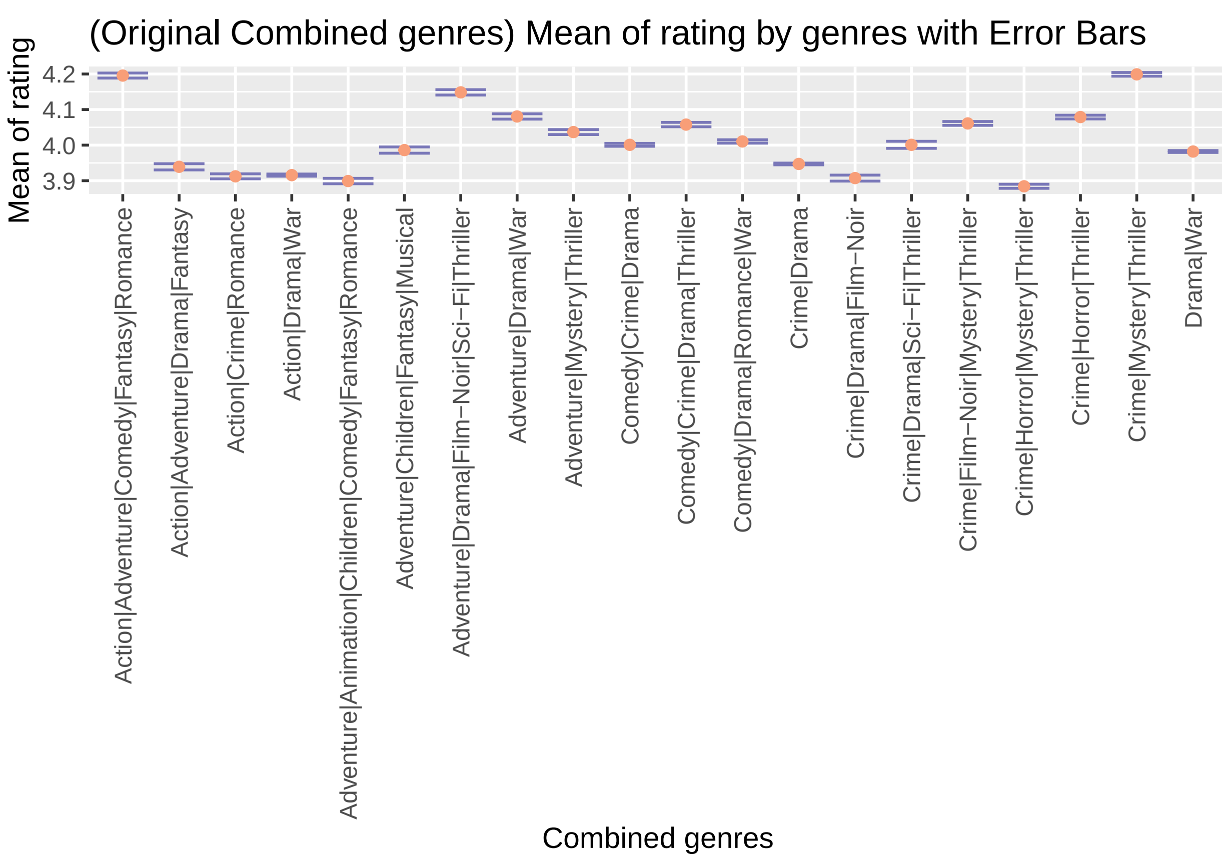
Mean of rating by genres (genres="Drama")

```
## # A tibble: 1 x 4
##   rank  genres average_rating_genres  count
##   <chr> <chr>                  <dbl>  <int>
## 1 48    Drama                   3.71 733296
```

Mean of rating by genres (genres="Comedy")

```
## # A tibble: 1 x 4
##   rank  genres average_rating_genres  count
##   <chr> <chr>                  <dbl>  <int>
## 1 131   Comedy                  3.24 700889
```

3.1.3.1.2 Plot Mean of rating by genres with error bars BEFORE genres Separated

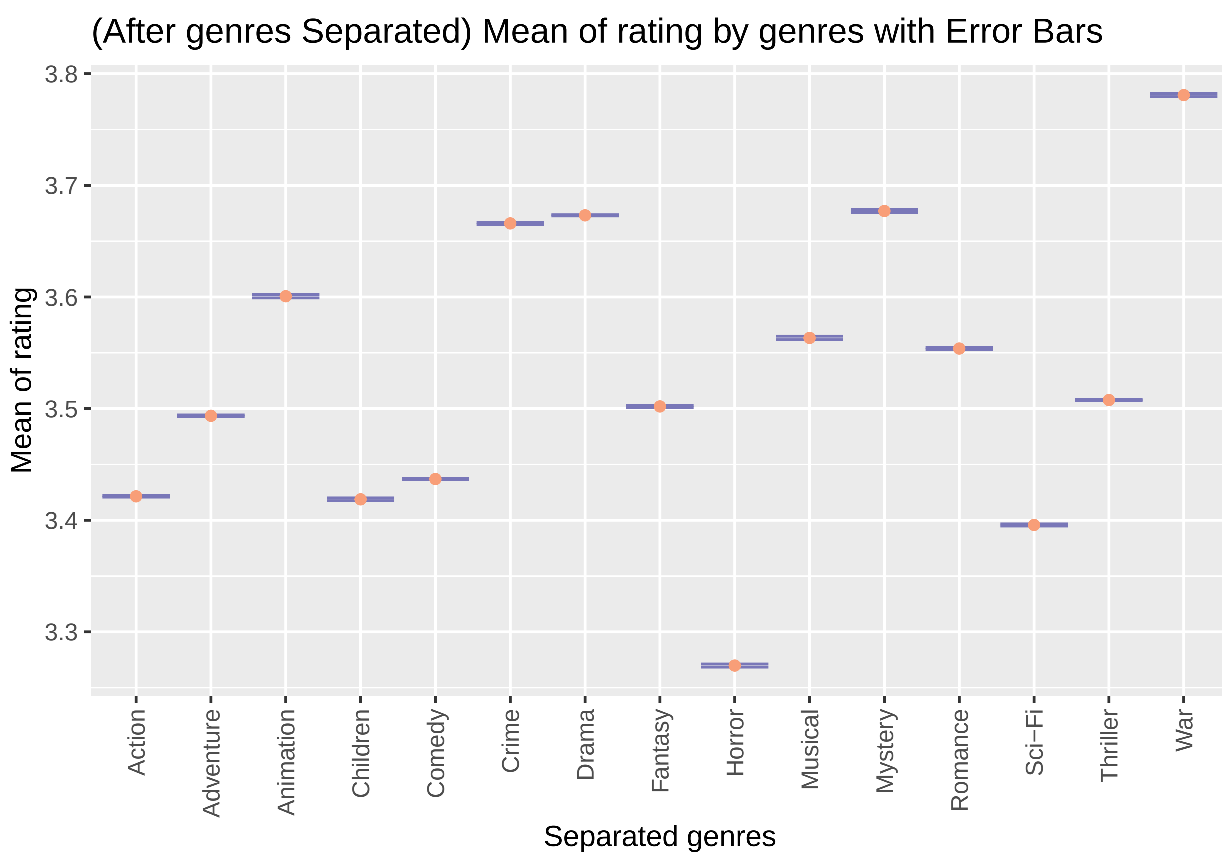


3.1.3.1.3 Mean of rating by genres AFTER Combined genres Separated

List of 15 Examples - Mean of rating by genres in descending order. (count > 20000)

```
## # A tibble: 15 x 4
##    rank  genres    average_rating_genres   count
##    <chr> <chr>                     <dbl>   <int>
##  1 1     War                        3.78  511147
##  2 2     Mystery                    3.68  568332
##  3 3     Drama                      3.67 3910127
##  4 4     Crime                      3.67 1327715
##  5 5     Animation                  3.60  467168
##  6 6     Musical                    3.56  433080
##  7 7     Romance                    3.55 1712100
##  8 8     Thriller                   3.51 2325899
##  9 9     Fantasy                    3.50  925637
## 10 10    Adventure                  3.49 1908892
## 11 11    Comedy                     3.44 3540930
## 12 12    Action                     3.42 2560545
## 13 13    Children                   3.42  737994
## 14 14    Sci-Fi                     3.40 1341183
## 15 15    Horror                     3.27  691485
```

3.1.3.1.4 Plot Mean of rating by genres with Error Bars AFTER genres Separated



List of rank, genres, Mean of rating by genres & Number of rating of "Comedy" and "Drama"

Table 8: BEFORE Combined genres Separated

rank	genres	average_rating_genres	count
48	Drama	3.71236	733296
131	Comedy	3.23786	700889

Table 9: After Combined genres Separated

rank	genres	average_rating_genres	count
3	Drama	3.67313	3910127
11	Comedy	3.43691	3540930

As illustrated by above Example, Data Misrepresentation is observed in Column of "rank", "Mean of rating" by "genres" and "Number of rating" after the Separation of "Combined genres" in edx datasets.

3.1.3.2 Column timestamp

Create **new Features** based on **timestamp** to capture insights related to **Time Effects**.

- **Week of the Date** (d_w): Transform & Extract the **Week of the Date** in **timestamp** column.
- **Month of the Date** (d_m): Transform & Extract the **Month of the Date** in **timestamp** column.
- **Year of the Date** (d_y): Transform & Extract the **Year of the Date** in **timestamp** column.

These **new Features** will enhance our Models ability to capture **Time Effects patterns**.

R codes generate new Columns (d_w, d_m, d_y)

```
# data Wrangling: generate new columns (d_w, d_m, d_y) from column timestamp

   edx <- edx %>% mutate(d_w=format(round_date(as_datetime(timestamp),"week"),"%Y-%m-%d"))

   edx <- edx %>% mutate(d_m=format(round_date(as_datetime(timestamp),"month"),"%Y-%m-%d"))

   edx <- edx %>% mutate(d_y=format(round_date(as_datetime(timestamp),"year"),"%Y-%m-%d"))
```

3.1.3.3 Column rating

Create **new Features/Predictors** can provide more detailed insights into **Mean of rating** by various dimensions such as **Week of Date**, **Week of Year**, **genres**. Additionally, I will also capture the total **Number of rating** by **movieId**.

The following steps outline the Feature Engineering process:

- Compute **Mean of rating** by **movieId** and generate a new Column **m_r**.
- Compute **Mean of rating** by **movieId**, **Week of the Date** and generate a new Column **m_rw**.
- Compute **Mean of rating** by **movieId**, **Year of the Date** and generate a new Column **m_ry**.
- Compute **Mean of rating** by **userId**, **genres** and generate a new Column **m_rg**.
- Compute Total **Number of rating** by **movieId** and generate a new Column **tot_nr**.

new Features enable us to capture more information & improve the **robustness** of **Data Analysis**.

R codes generate new Columns (m_r, m_rw, m_ry, m_rg, tot_nr)

```
# data Wrangling: generate new columns (m_r, m_rw, m_ry, m_rg, tot_nr) from column rating

edx <- edx %>% group_by(movieId) %>% mutate(m_r = mean(rating, na.rm=TRUE))

edx <- edx %>% group_by(movieId,d_w) %>% mutate(m_rw= mean(rating, na.rm=TRUE))

edx <- edx %>% group_by(movieId,d_y) %>% mutate(m_ry= mean(rating, na.rm=TRUE))

edx <- edx %>% group_by(userId,genres) %>% mutate(m_rg=mean(rating, na.rm=TRUE))

edx <- edx %>% group_by(movieId) %>% mutate(tot_nr=n()) %>% ungroup()
```

3.1.3.4 Column title

To facilitate easier interpretation and explanation, we will also create the **new Features** that capture more information than original **Features**.

- Extract the **Release Year** of **Movie** from **title** column and generate a **new** column **release**.

This **new Feature** will enhance our datasets by providing a clear and easily interpretable attribute related to the **Movie's Release Year**.

R codes generate new column "release" from column "title"

```
# data wrangling: generate new column release from column title

edx <- edx %>% mutate(release = str_extract(title,"\\d{4}")) %>%
               mutate(title = str_replace(title,"\\s*\\(\\d{4}\\)",""))
```

3.1.4 New Features/Predictors

List 10 Examples encompassing new Features (d_w)

```
## # A tibble: 11 x 5
##    userId movieId title                                     d_w        rating
##     <int>   <int> <chr>                                     <chr>       <dbl>
##  1  39748     592 Batman                                    1996-03-03      4
##  2  40233     150 Apollo 13                                 1996-03-03      5
##  3  35139       1 Toy Story                                 1996-01-28      4
##  4  20095     780 Independence Day (a.k.a. ID4)             1996-06-30      5
##  5  49668     590 Dances with Wolves                        1996-03-31      5
##  6  39748     527 Schindler's List                          1996-03-03      5
##  7  34955     380 True Lies                                 1996-03-03      4
##  8  15767    1210 Star Wars: Episode VI - Return of the Jedi 1996-10-27      4
##  9  35139      32 12 Monkeys (Twelve Monkeys)               1996-01-28      5
## 10  36008      50 Usual Suspects, The                       1996-02-04      5
## 11  41500     608 Fargo                                     1996-03-17      5
```

List 8 Examples encompassing new Features (release, m_r, m_rw, m_ry, m_rg)

```
## # A tibble: 8 x 8
##   userId release title                      rating   m_r  m_rw  m_ry  m_rg
##    <int> <chr>   <chr>                       <dbl> <dbl> <dbl> <dbl> <dbl>
## 1  34955 1994    Pulp Fiction                    5  4.15 5      4.04   3.5
## 2  35435 1994    Forrest Gump                    4  4.01 4.17   4.13   4
## 3  34955 1991    Silence of the Lambs, The       5  4.20 5      4.31   5
## 4  37153 1993    Jurassic Park                   1  3.66 3.14   3.90   1
## 5  40233 1994    Shawshank Redemption, The       3  4.46 3      4.49   4
## 6  36881 1995    Braveheart                      4  4.08 4      4.32   4
## 7  35435 1993    Fugitive, The                   5  4.01 4.71   4.16   5
## 8  34955 1991    Terminator 2: Judgment Day      5  3.93 3      4.06   3.5
```

3.2 Data Visualization and Analysis

3.2.1 Frequency Analysis

3.2.1.1 Frequency of rating By title

List of 20 Movies which is most frequently Rated by User with frequency > 10000

```
# generate 20 movies: most frequently rated by User with frequency > 10000

c_1 <- edx %>%
            group_by(title) %>%
            mutate(frequency=length(rating)) %>% ungroup() %>%
            filter(frequency > 10000) %>%
            distinct(title, .keep_all=TRUE) %>%
            arrange(desc(frequency)) %>%
            mutate(rank=rownames(.)) %>%
            select(rank,title,frequency) %>%
            dplyr::slice(1:20)

# display c_1 using kable function

c_1 %>% knitr::kable(caption="Frequency of rating By title")
```

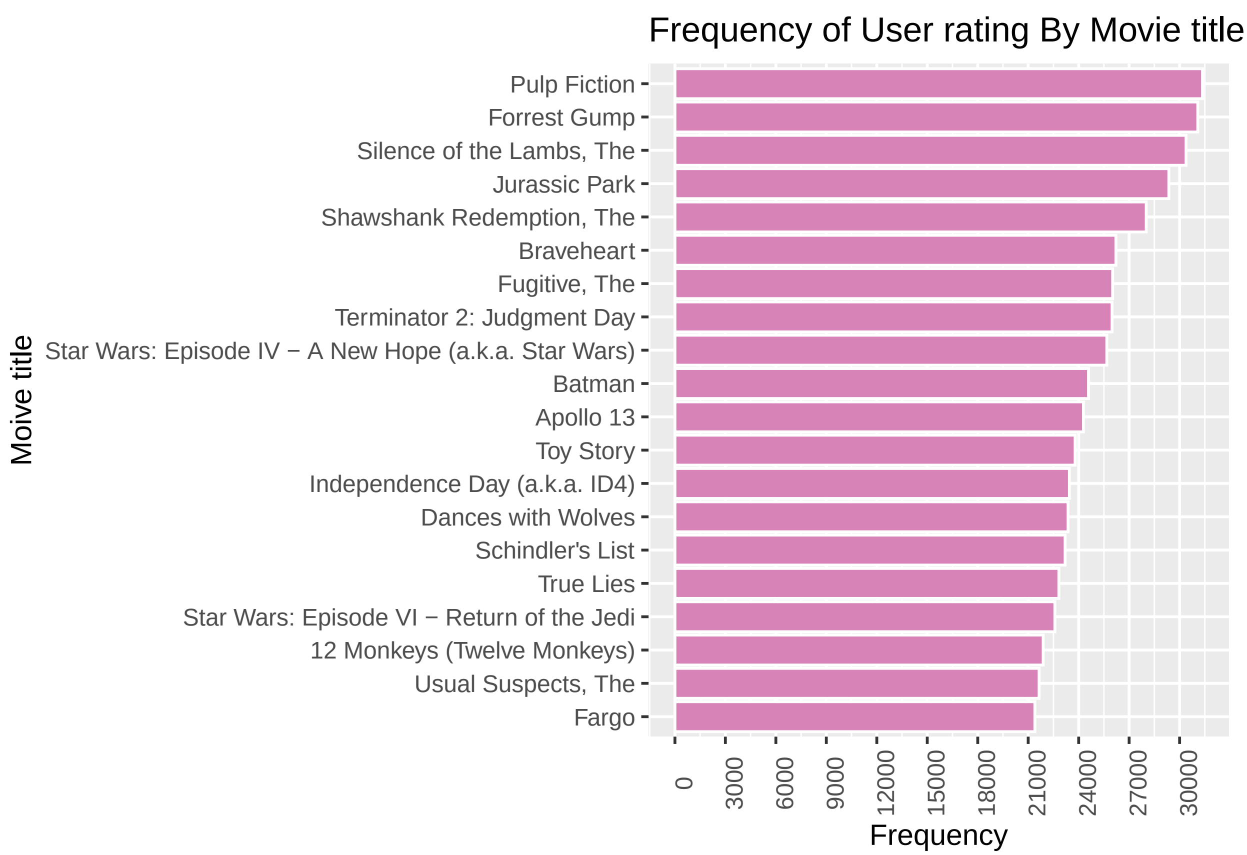
Table 10: Frequency of rating By title

rank	title	frequency
1	Pulp Fiction	31362
2	Forrest Gump	31079
3	Silence of the Lambs, The	30382
4	Jurassic Park	29360
5	Shawshank Redemption, The	28015
6	Braveheart	26212
7	Fugitive, The	26020
8	Terminator 2: Judgment Day	25984
9	Star Wars: Episode IV - A New Hope (a.k.a. Star Wars)	25672
10	Batman	24585
11	Apollo 13	24284
12	Toy Story	23790
13	Independence Day (a.k.a. ID4)	23449
14	Dances with Wolves	23367
15	Schindler's List	23193
16	True Lies	22823
17	Star Wars: Episode VI - Return of the Jedi	22584
18	12 Monkeys (Twelve Monkeys)	21891
19	Usual Suspects, The	21648
20	Fargo	21395

The Movie "Pulp Fiction" has the greatest Number of rating and is most frequently Rated by User.

3.2.1.2 Plot - Frequency of rating By title

```
# ggplot frequency of rating By title

c_1 %>% mutate(title=reorder(title,frequency)) %>%
        ggplot(aes(title,frequency)) +
        geom_bar(stat="identity",fill="#D883B7",color="white")  +
        labs(y="Frequency",x="Moive title",title="Frequency of User rating By Movie title") +
        theme(axis.text.x=element_text(angle=90)) +
        scale_y_continuous(breaks=seq(0,33000,by=3000)) +
        coord_flip()
```



3.2.1.3 Frequency of rating By genres

List of 20 genres which is most frequently Rated by User with frequency > 10000

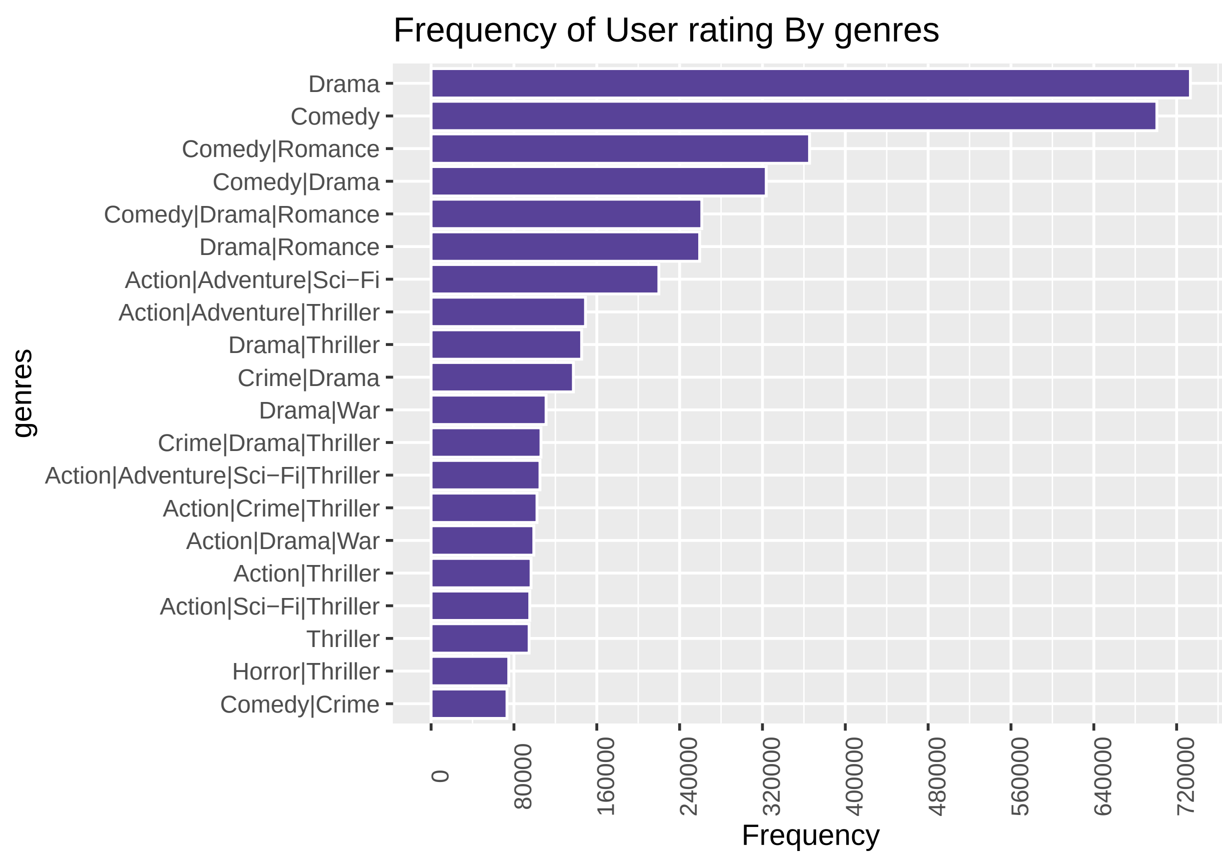
```r
# generate 20 genres: most frequently rated by User with frequency > 10000

c_2 <- edx %>%
           group_by(genres) %>%
           mutate(frequency=length(rating)) %>% ungroup() %>%
           filter(frequency > 10000) %>%
           distinct(genres, .keep_all=TRUE) %>%
           arrange(desc(frequency)) %>%
           mutate(rank=rownames(.)) %>%
           select(rank,genres,frequency) %>%
           dplyr::slice(1:20)

# display c_2

print_df <- function(title,df)
              {
                 cat(title, "\n\n")
                 cat(capture.output(print(n=50,df)), sep="\n")
              }
print_df("List of 20 genres - Frequency of rating By genres ",c_2)
```

```
## List of 20 genres - Frequency of rating By genres
##
## # A tibble: 20 x 3
##    rank  genres                            frequency
##    <chr> <chr>                                 <int>
##  1 1     Drama                                733296
##  2 2     Comedy                               700889
##  3 3     Comedy|Romance                       365468
##  4 4     Comedy|Drama                         323637
##  5 5     Comedy|Drama|Romance                 261425
##  6 6     Drama|Romance                        259355
##  7 7     Action|Adventure|Sci-Fi              219938
##  8 8     Action|Adventure|Thriller            149091
##  9 9     Drama|Thriller                       145373
## 10 10    Crime|Drama                          137387
## 11 11    Drama|War                            111029
## 12 12    Crime|Drama|Thriller                 106101
## 13 13    Action|Adventure|Sci-Fi|Thriller     105144
## 14 14    Action|Crime|Thriller                102259
## 15 15    Action|Drama|War                      99183
## 16 16    Action|Thriller                       96535
## 17 17    Action|Sci-Fi|Thriller                95280
## 18 18    Thriller                              94662
## 19 19    Horror|Thriller                       75000
## 20 20    Comedy|Crime                          73286
```

The genres "Drama" has the greatest Number of rating and is most frequently Rated by User.

3.2.1.4 Plot - Frequency of rating By genres

```
# ggplot frequency of rating By genres

c_2 %>% mutate(genres=reorder(genres,frequency)) %>%
        ggplot(aes(genres,frequency)) +
        geom_bar(stat="identity",fill="#584298",color="white")  +
        labs(y="Frequency",x="genres",title="Frequency of User rating By genres") +
        theme(axis.text.x=element_text(angle=90)) +
        scale_y_continuous(breaks=seq(0,800000,by=80000)) +
        coord_flip()
```

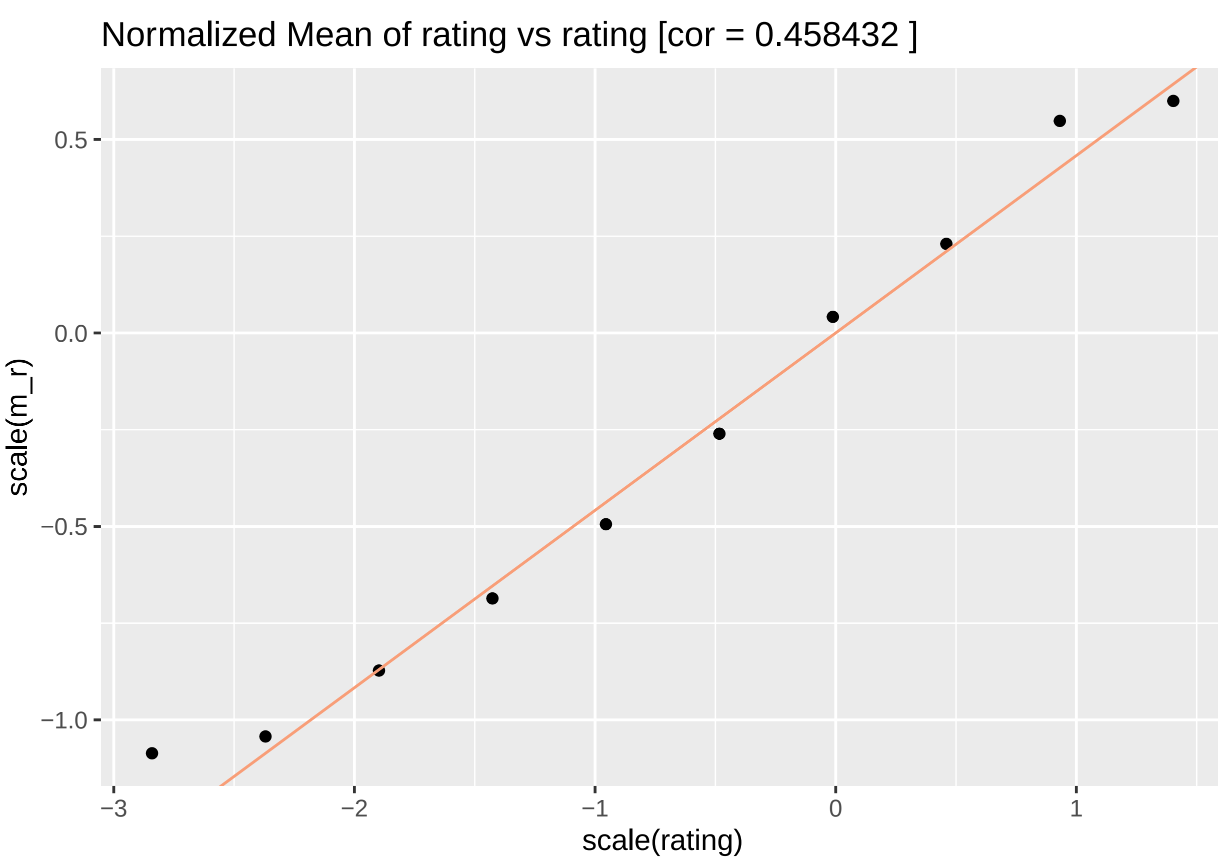


3.2.2 Stratification Analysis

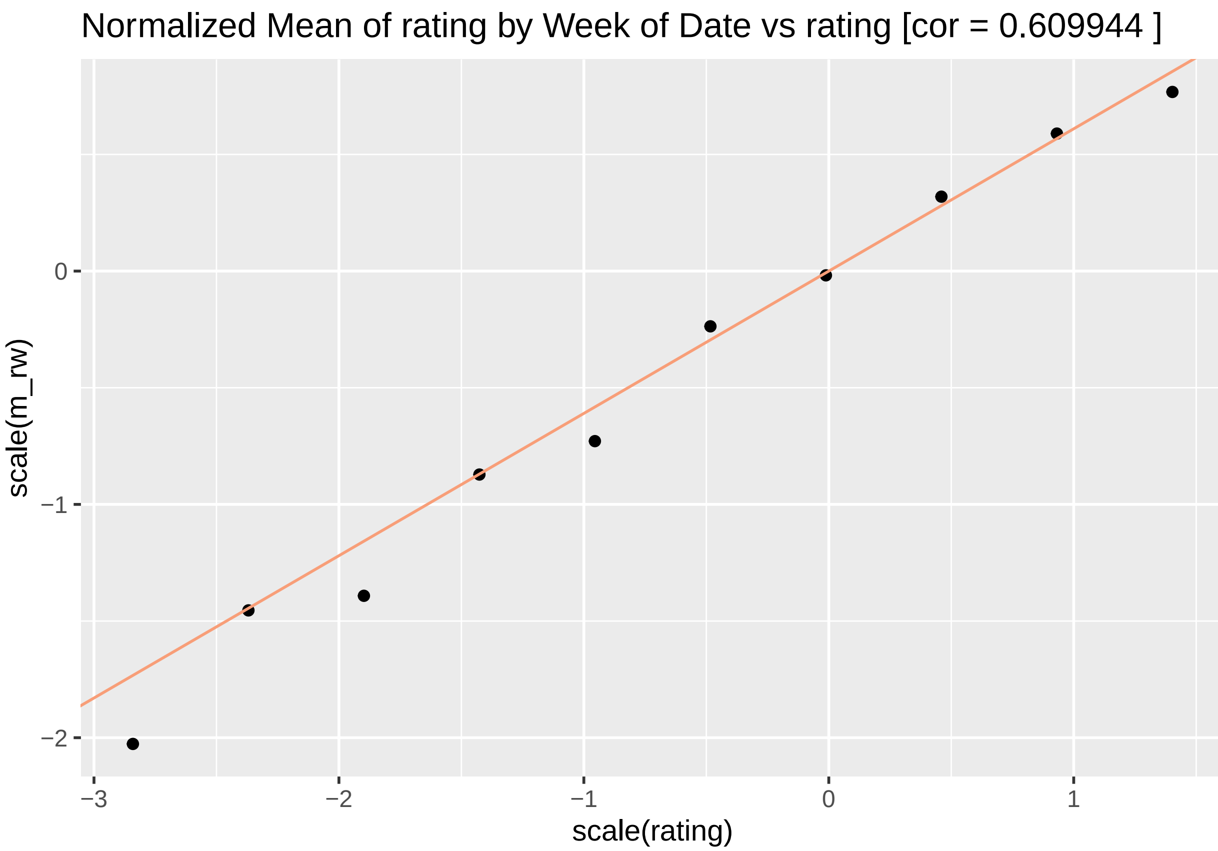
Correlation is only meaningful in a particular context. **Stratification** is used to identifying the **Correlation** is meaningful as a summary statistic for **Prediction**. I stratify a **rating** into groups and compute the **Mean** summaries in each group (**Mean of rating**, **Mean of rating** by **Week of Date**, **Mean of rating** by **Year of Date**, **Mean of rating** by **genres**). After normalizing the **rating** and **Mean of rating**, the **Mean of rating** will be equal to **0** and the **Standard Deviation** will be equal to **1**. Therefore, I will set **intercept** to **0** and **slope** to **Correlation coefficient** (ρ).

3.2.2.1 Plot - Correlation of Normalized Mean of rating and rating

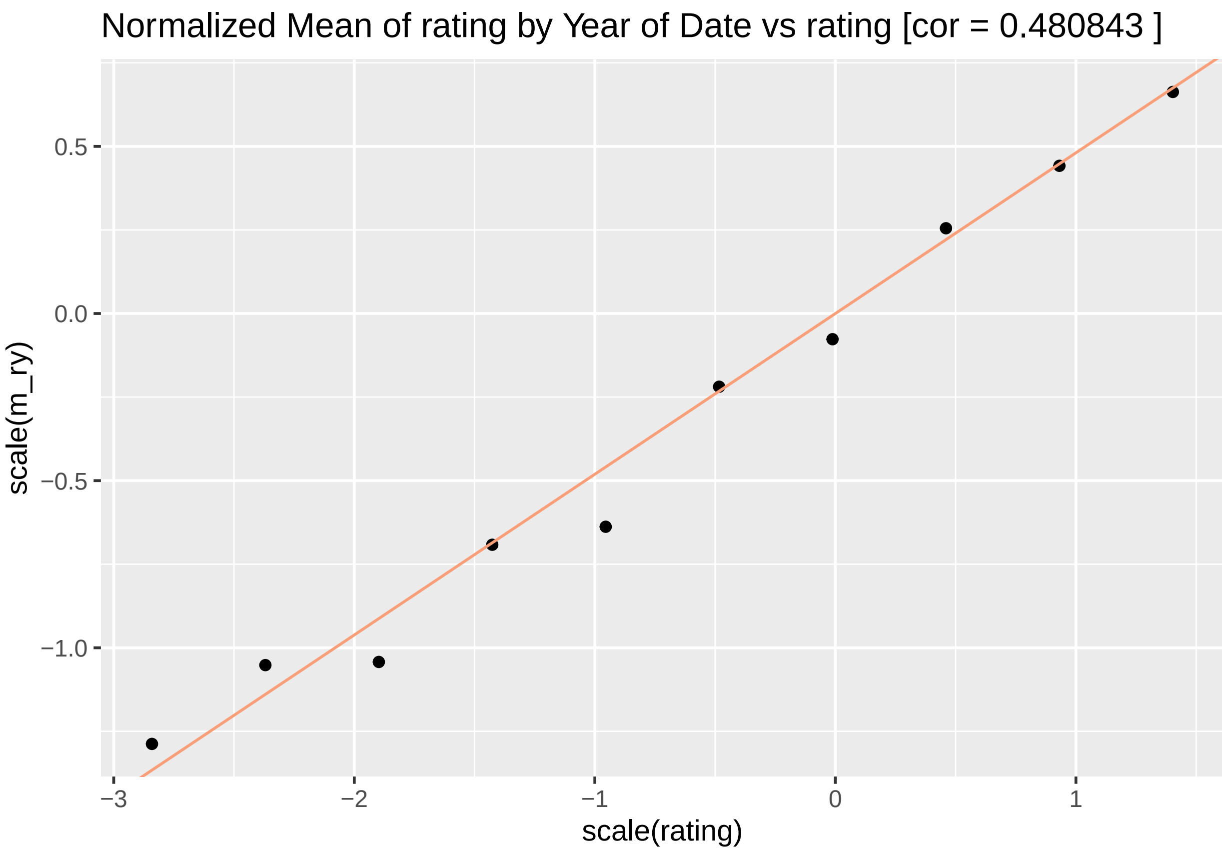
```
# compute correlation coefficient (r) of rating and m_r with 6 decimals place
r <- edx %>%
         summarize(r=cor(rating,m_r)) %>%
         pull(r)
r <- round(r,6)

# ggplot correlation of Normalized Mean of rating and rating
edx %>% mutate(rating=scale(rating),m_r=scale(m_r)) %>%
        group_by(rating) %>%
        summarize(m_r=mean(m_r)) %>%
        ggplot(aes(rating,m_r)) + geom_point()+
        geom_abline(intercept=0, slope=r,color="#F89E78")+
        labs(x="scale(rating)",y="scale(m_r)",
        title=paste("Normalized Mean of rating vs rating [cor =",r,"]"))
```



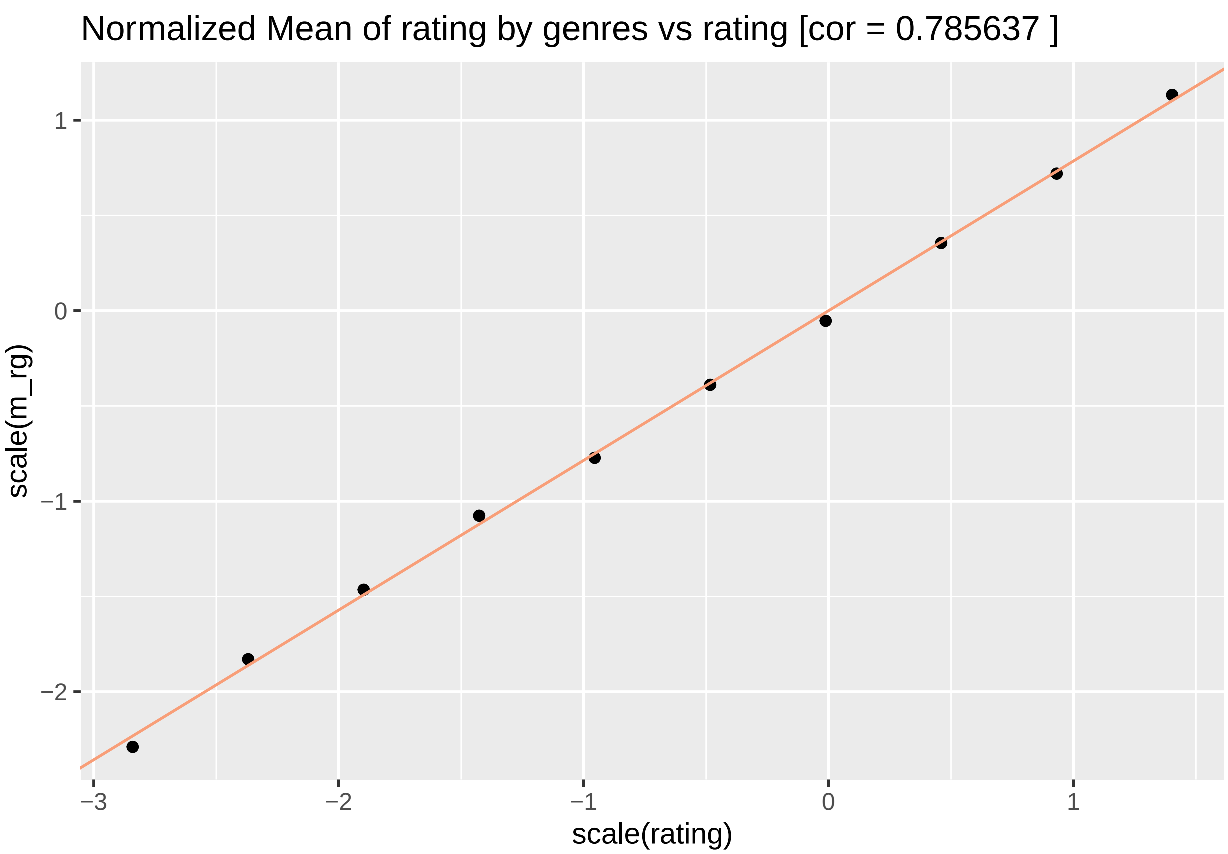
3.2.2.2 Plot - Correlation of Normalized Mean by Week of Date and rating



3.2.2.3 Plot - Correlation of Normalized Mean by Year of Date and rating



3.2.2.4 Plot - Correlation of Normalized Mean of rating by genres and rating



3.2.2.5 Correlation table - new Features (m_r, m_rw, m_ry, m_rg) of "edx" dataset

```
# generate correlation coefficient table (m_r, m_rw, m_ry, m_rg) of edx

avg_r_all <- edx %>%
                select(rating,m_r,m_rw,m_ry,m_rg)
cor_r_all <- cor(na.omit(avg_r_all[, unlist(lapply(avg_r_all, is.numeric))]))

# display correlation coefficient table using kable function

cor_r_all %>% knitr::kable(caption="Correlation - new Features of edx dataset")
```

Table 11: Correlation - new Features of edx dataset

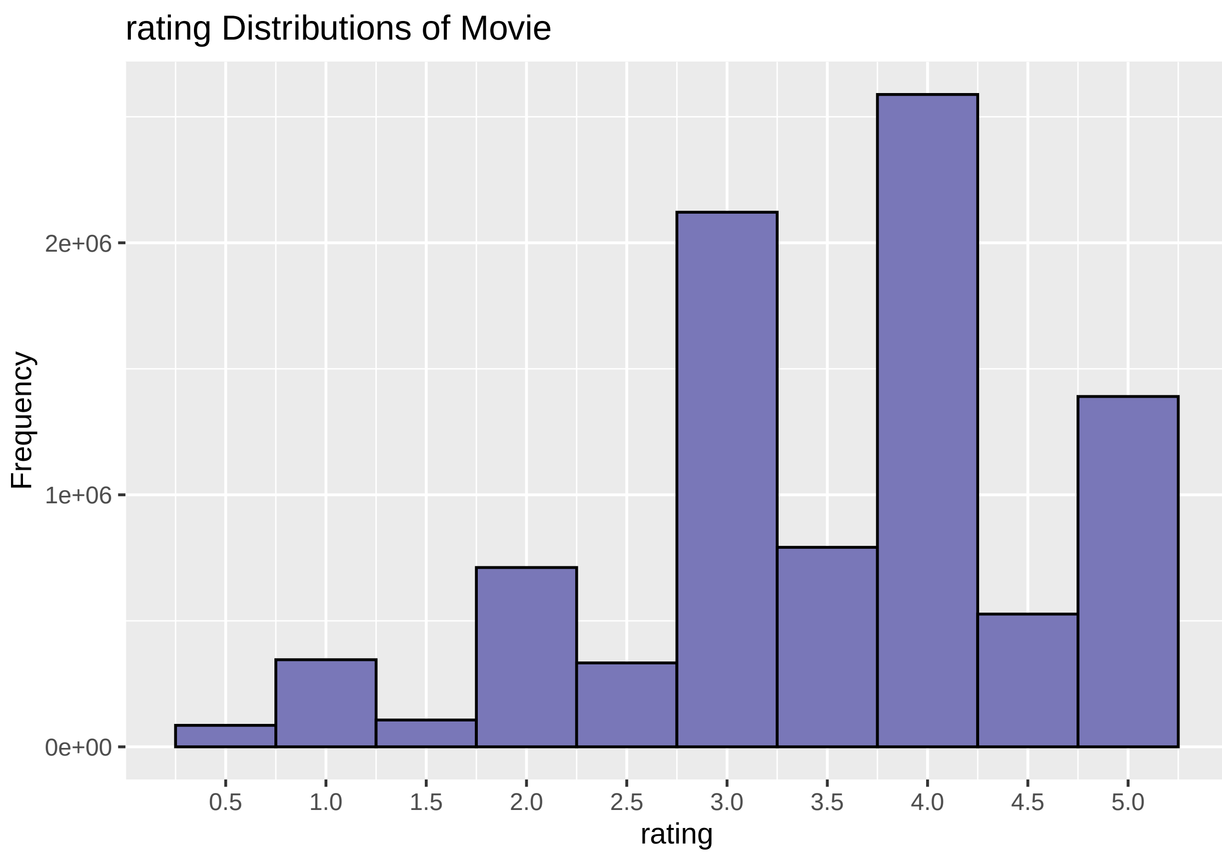
	rating	m_r	m_rw	m_ry	m_rg
rating	1.000000	0.458432	0.609944	0.480843	0.785637
m_r	0.458432	1.000000	0.751597	0.953393	0.368341
m_rw	0.609944	0.751597	1.000000	0.788227	0.463979
m_ry	0.480843	0.953393	0.788227	1.000000	0.385547
m_rg	0.785637	0.368341	0.463979	0.385547	1.000000

Data Visualization reveals that **Mean** of each group appear to follow a **linear relationship**. **Mean of rating** By **genres** seems to have more predictive power than **Mean of rating**, **Mean of rating** By **Week of Date** and **Mean of rating** By **Year of Date**. The **rating** to **Mean of rating** By **Week of Date** and **Mean of rating** By **genres** variability are quite large and this implies that they should explain a **lot of variability**.

Hence, I will add **Mean of rating** By **genres** as a **Parameter** for **Genres Effects** in Model building. Adding extra **Features/Predictors** can improve **Root Mean Squared Error(RMSE)**, but may not when the added **Features** that are highly correlated with other **Features**. From **Correlation** table, I also observe that the **Correlation** between **rating** and **Mean of rating** By **Week of Date** is second greatest. Therefore, I will also add **Mean of rating** By **Week of Date** as **Parameter** for **Time Effects** in Model building.

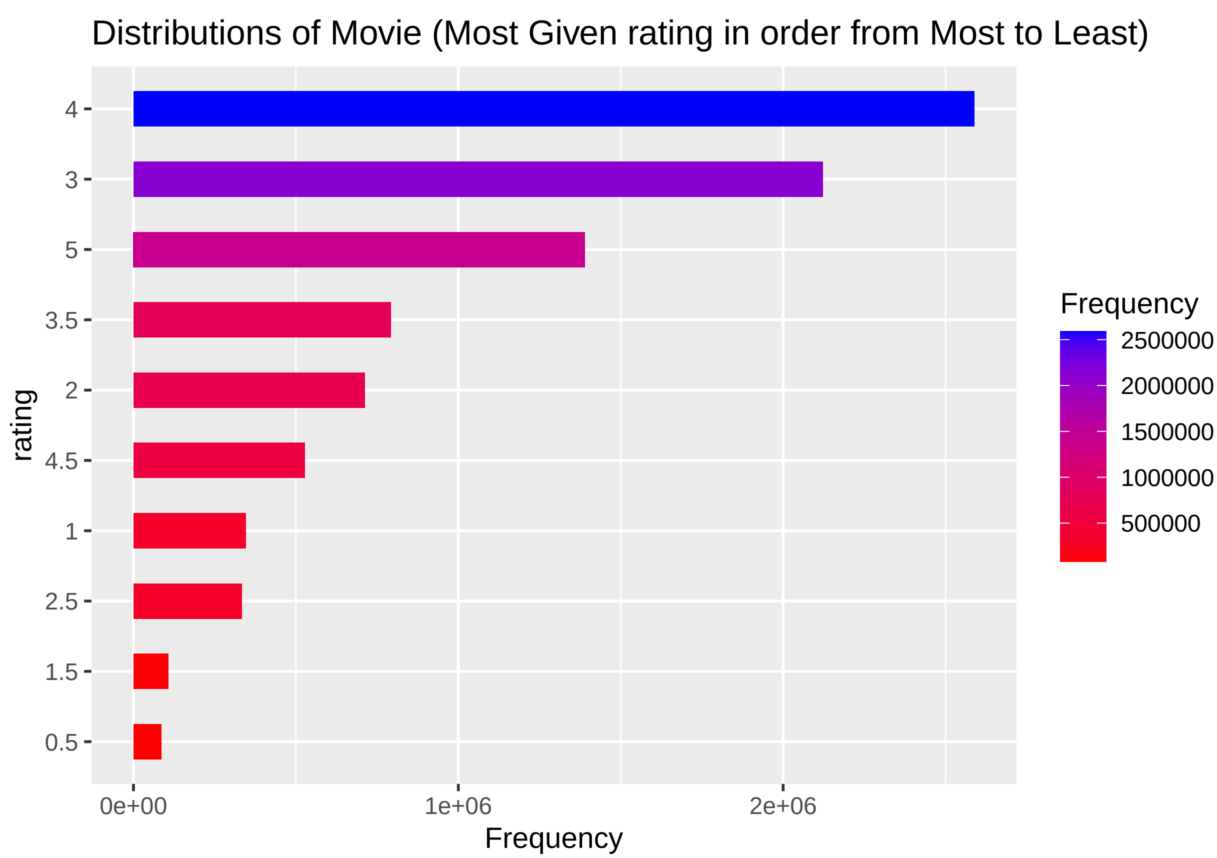
3.2.3 Distributions Analysis

3.2.3.1 Plot - rating Distributions of Movie



3.2.3.2 Plot - Distributions of Movie (Most Given rating in order from Most to Least)

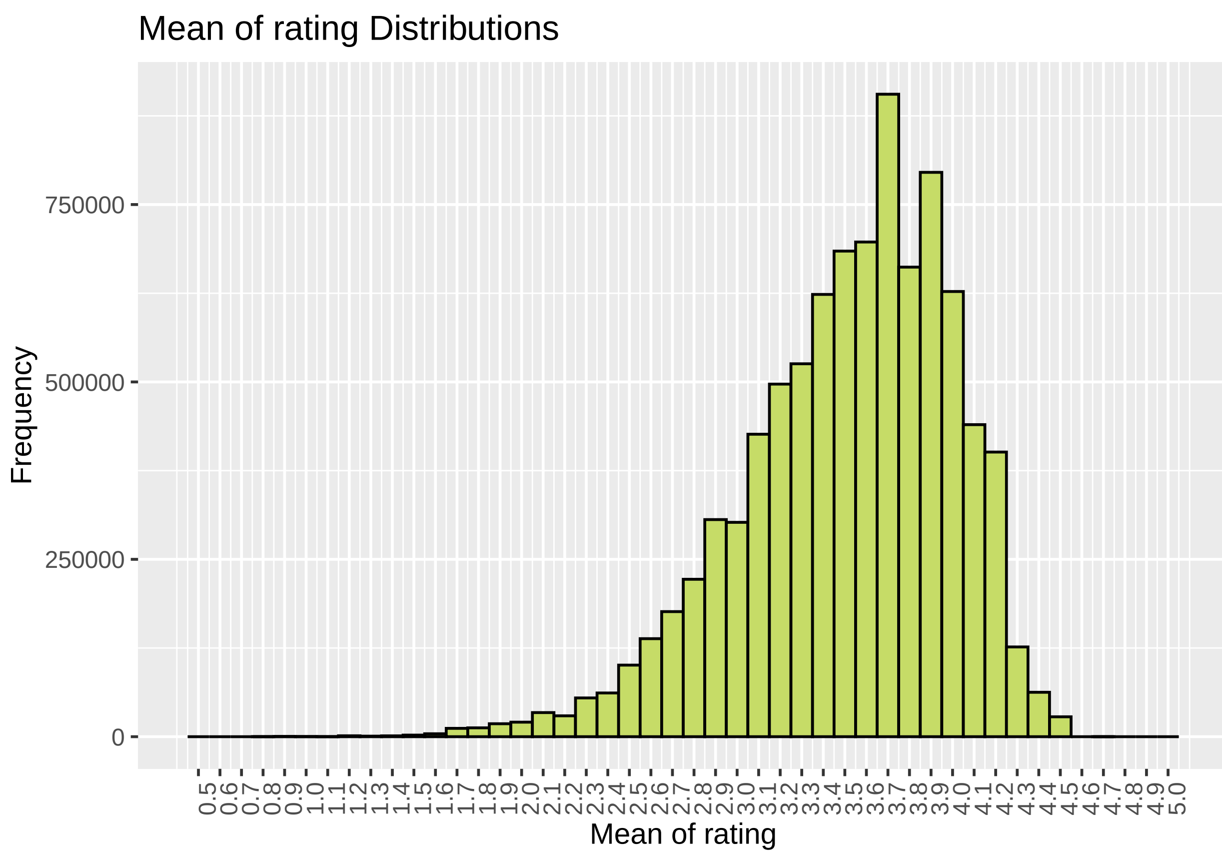
```
# generate frequency group by rating

edx_asc <- edx %>%
                group_by(rating) %>%
                summarise(Frequency = n()) %>%
                arrange(Frequency)

# ggplot distributions of movie from most given rating in order from most to least

edx_asc %>%
        ggplot(aes(x=reorder(rating, Frequency), y=Frequency)) +
        geom_bar(stat="identity", width=0.5, aes(fill=Frequency)) +
        scale_fill_gradient(low="red", high="blue") +
        coord_flip() +
        labs(y="Frequency",
             x="rating",
             title = "Distributions of Movie (Most Given rating in order from Most to Least)")
```



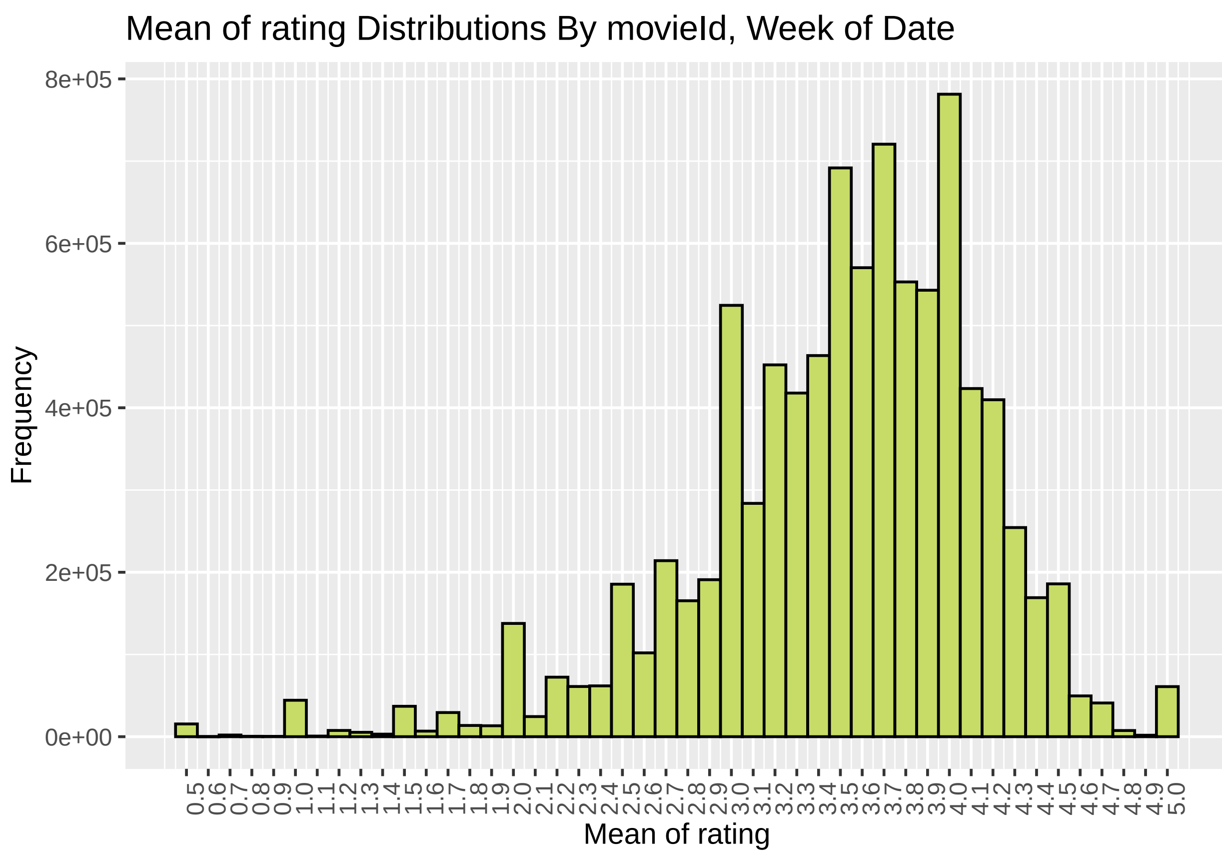
The Five most given rating in order from most to least are (4.0, 3.0, 5.0, 3.5, 2.0).

In general, half score **rating** are less common than whole score **rating**. For example, there are fewer **rating** of **3.5** than there are **rating** of **3.0** or **4.0**

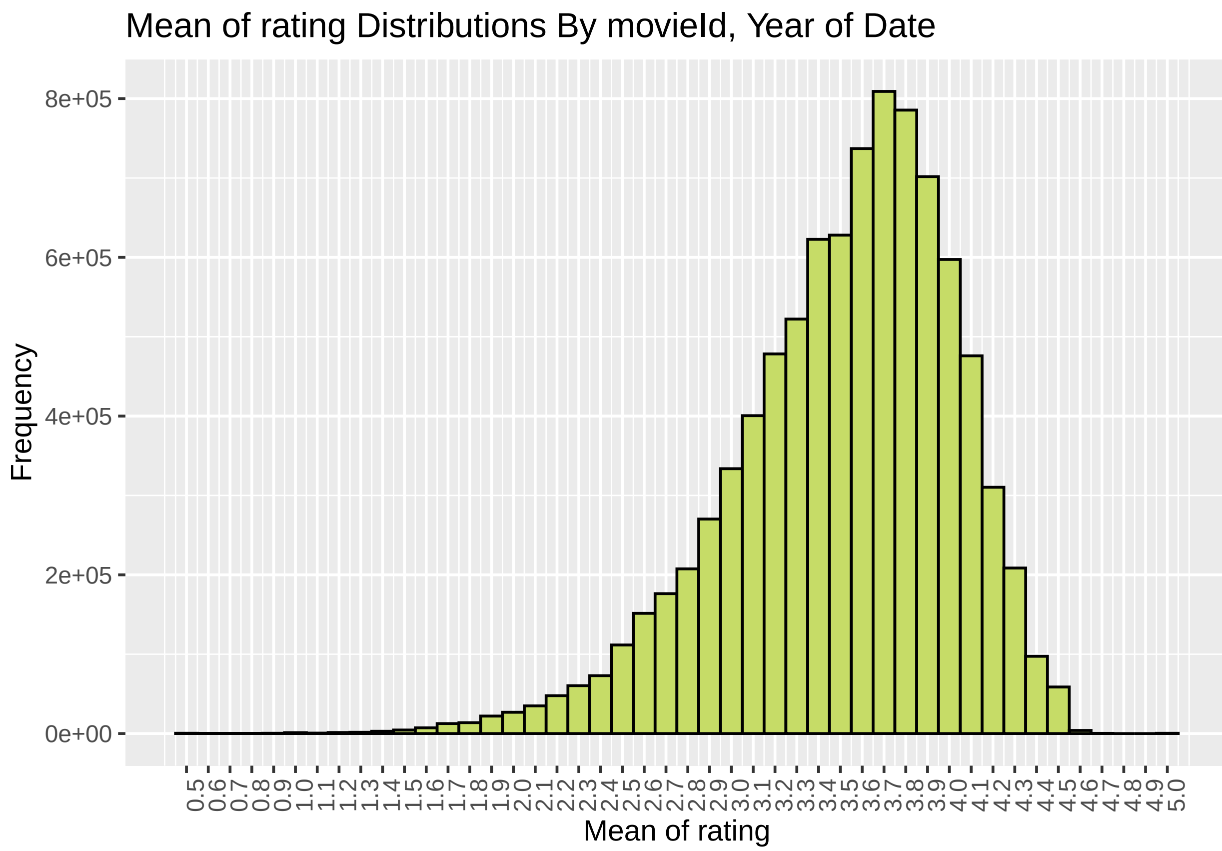
3.2.3.3 Plot - Mean of rating Distributions



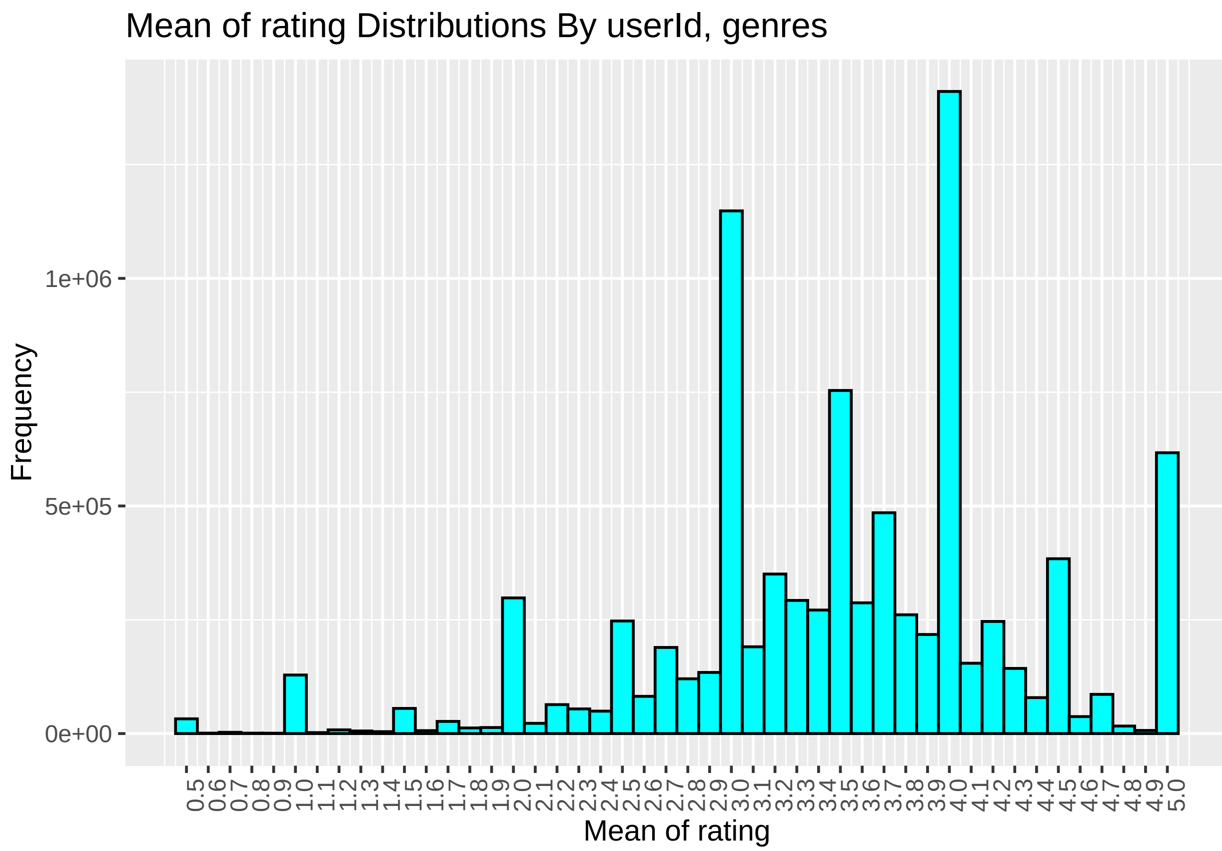
3.2.3.4 Plot - Mean of rating Distributions By movieId, Week of Date



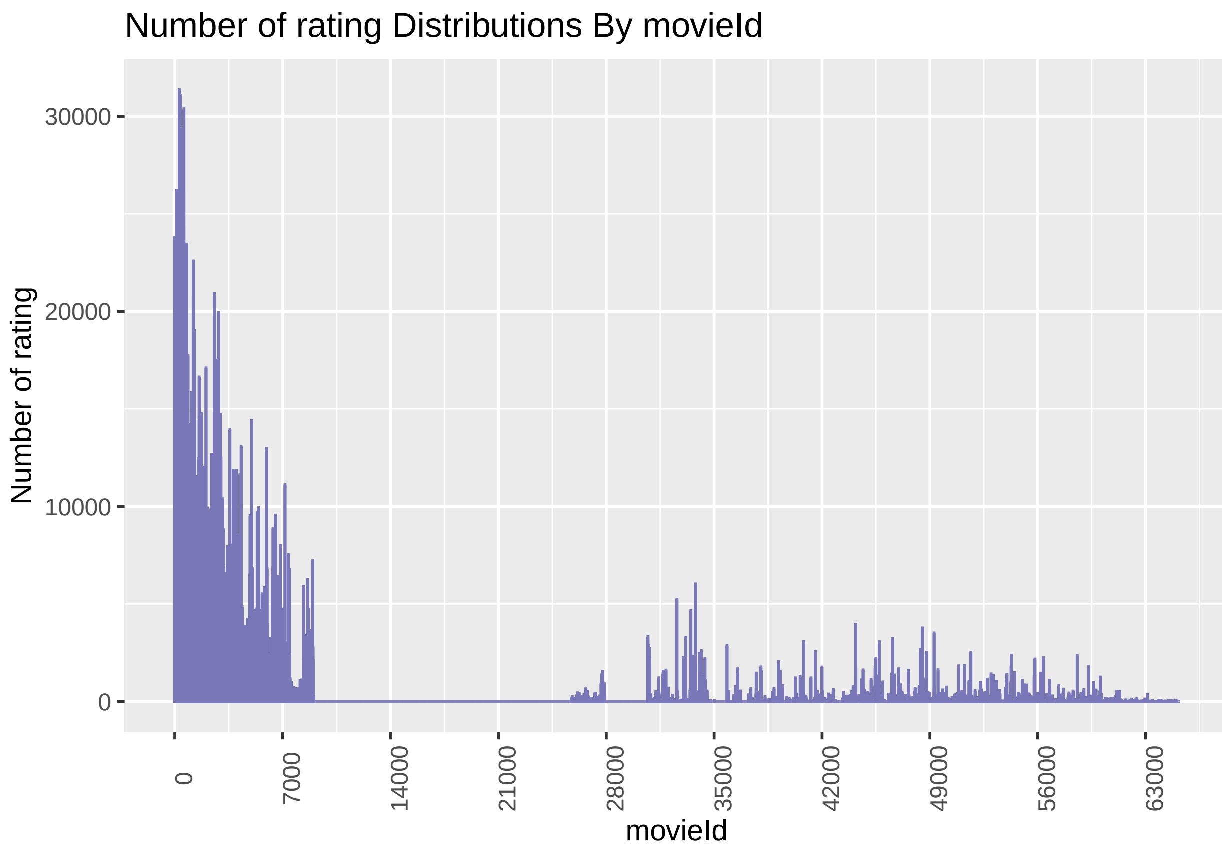
3.2.3.5 Plot - Mean of rating Distributions By movieId, Year of Date



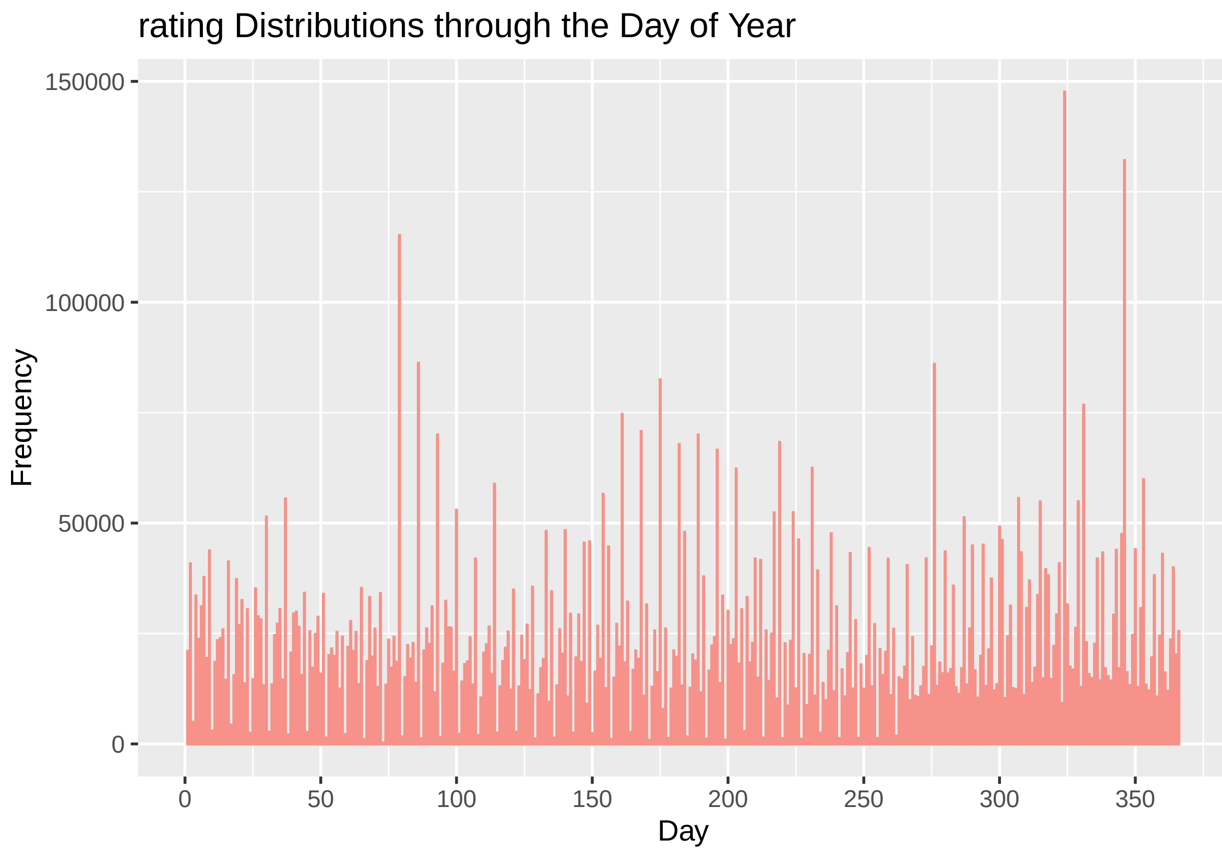
3.2.3.6 Plot - Mean of rating Distributions (By userId, genres)



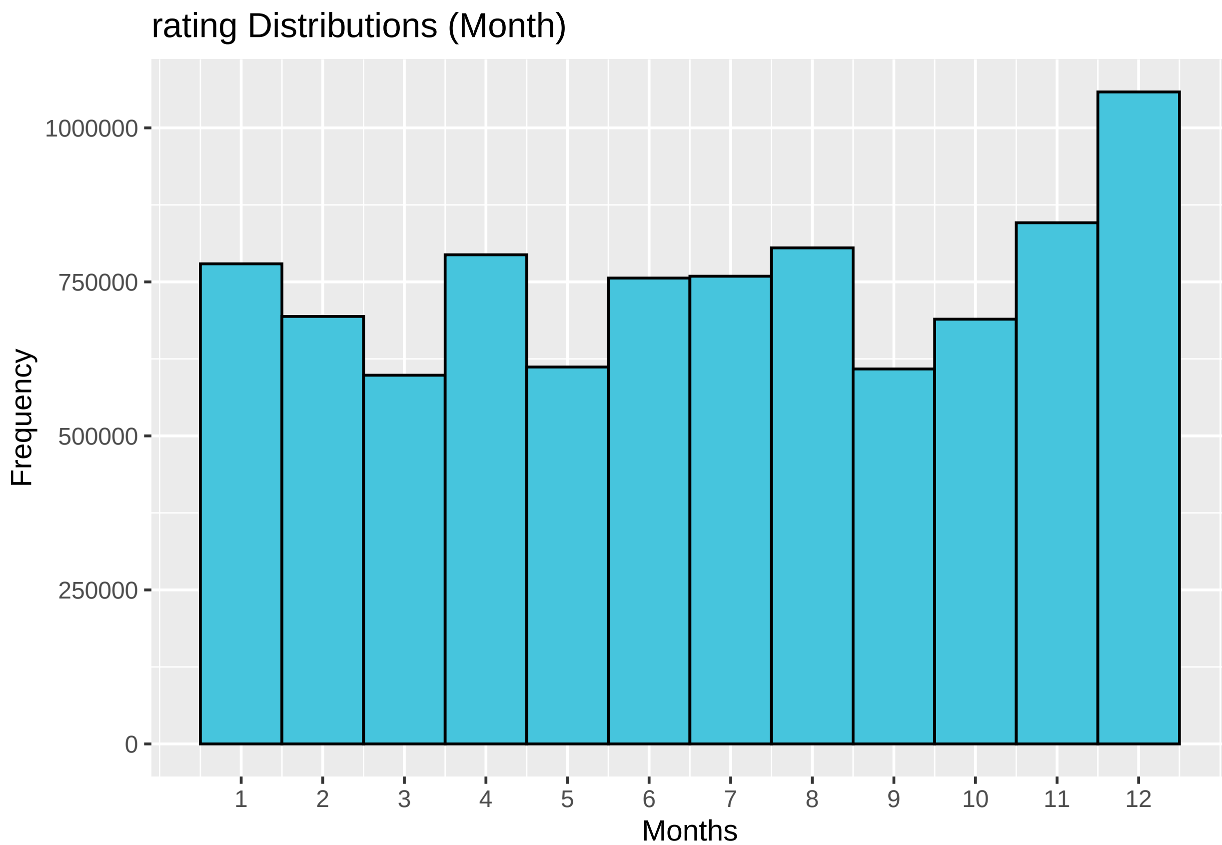
3.2.3.7 Plot - Number of rating Distributions By movieId



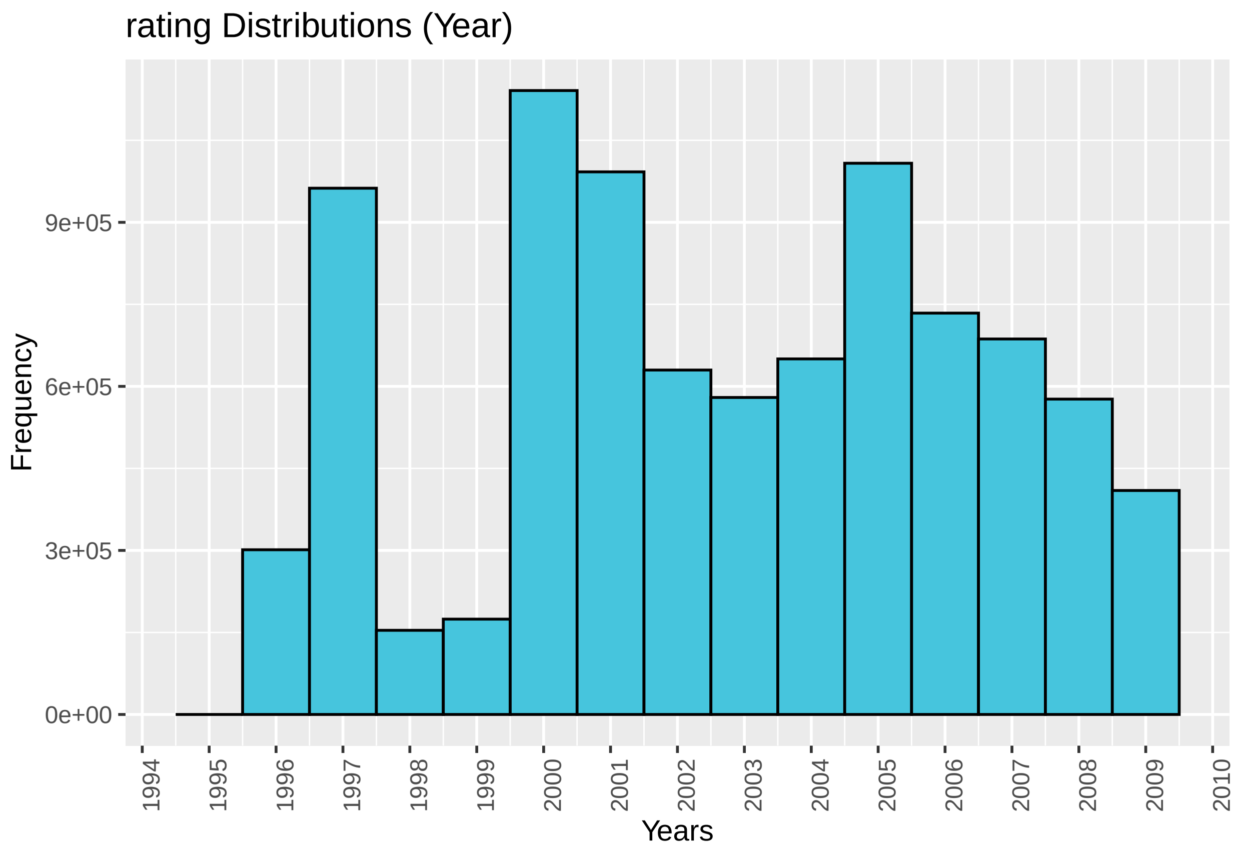
3.2.3.8 Plot - rating Distributions through the day of year



3.2.3.9 Plot - rating Distributions (Month)



3.2.3.10 Plot - rating Distributions (Year)



4 Machine Learning Modelling Approach and Algorithm

4.1 Modelling Approach

During Models development stage, **edx** datasest is split into separate **train_set** and **test_set** datasets for training and testing different Models. And then apply the relevant **Minimum Lambda** (λ) values on the **Algorithm** for the **Final Model building** using **edx** and **final_holdout_test** sets.

I use the **Collaborative Filtering Approach** to build our **Machine Learning Models** and compare different Models by evaluating their **loss functions**. The goal is to build a Ultimate Model that minimizes the loss. **Root Mean Squared Error (RMSE)** will be used as our **loss function**.

If N is the **number of User-Movie combinations**, $y_{u,i}$ is the **rating** for **Movie** i by **User** u, and $\hat{y}_{u,i}$ is our **Prediction**, the **RMSE** is defined as follow:

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

The estimate that minimizing **RMSE** represents the **Predicted Ratings** of all **Movies** across all **Users**. I will also use **Regularization Method** to penalize large estimates that arise from small sample sizes. Furthermore, the **Cross-Validation Method** will be adopted to mimic the **RMSE**.

4.1.1 Naive Mean Based Model

Assumes the same **rating** for all **Movies** and all **Users**, with all the differences explained by random variation. The simplest Model that someone can build, is a **Naive Mean Based Model** that **Predict** always the **Mean of rating** on **edx** datasets which represents the **true rating** for all **Movies** and **Users**. The **Mean** is approximately **3.512465**.

The **Naive Mean Based Model** is defined as follow:

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

with μ is the **Mean** and $\varepsilon_{u,i}$ is the **independent errors** sampled from the same distribution centered at **0**.

RMSE of **Naive Mean Based Model** on **final_holdout_test** sets is **1.061202**.
It is **NOT an Acceptable Result**.

4.1.2 Movie Effects Based Model

We can improve our Model by adding a term b_i, that represents the **Mean of rating** for **Movie** i
The first Non Naive Based Model takes into account the **Movie Specific Effects**. **Movies** are Rated higher or lower associated with each other.

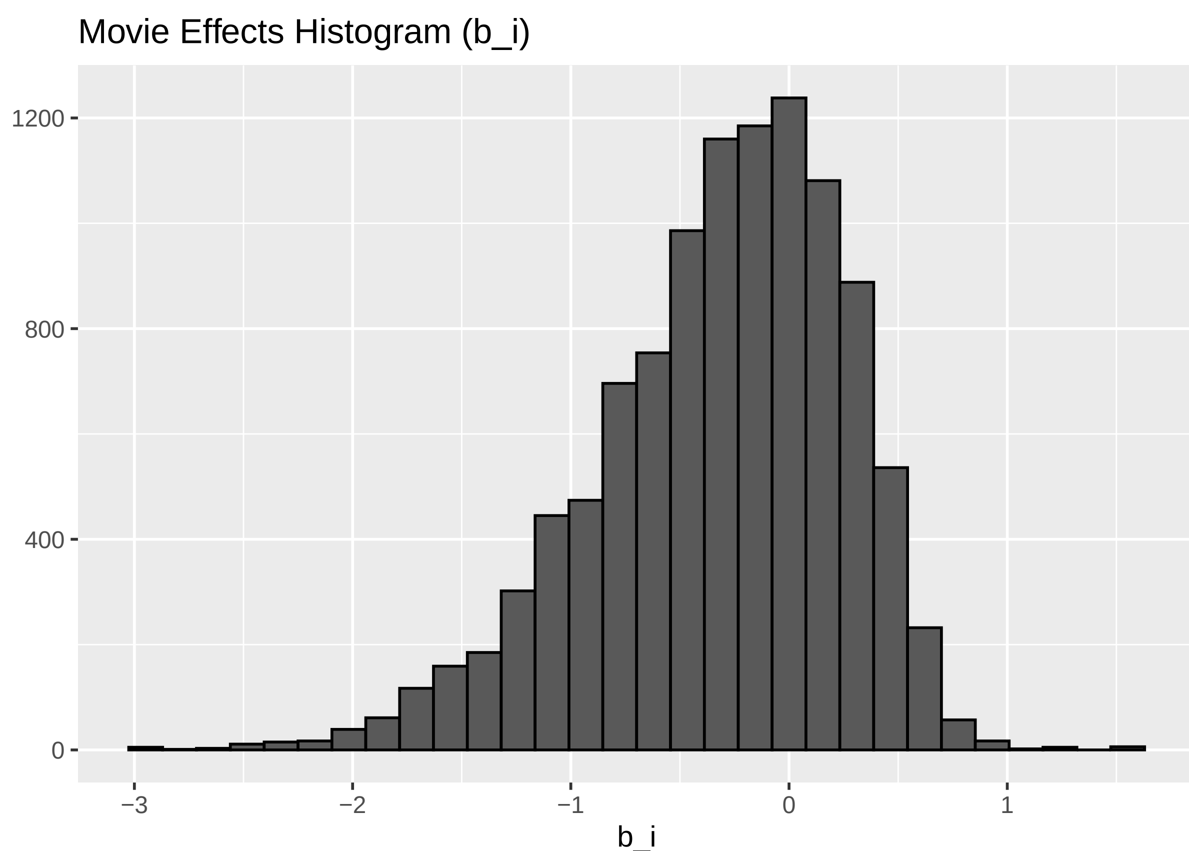
The **Movie Effects Based Model** is defined as follow:

$$Y_{u,i} = \mu + b_i + \varepsilon_{u,i}$$

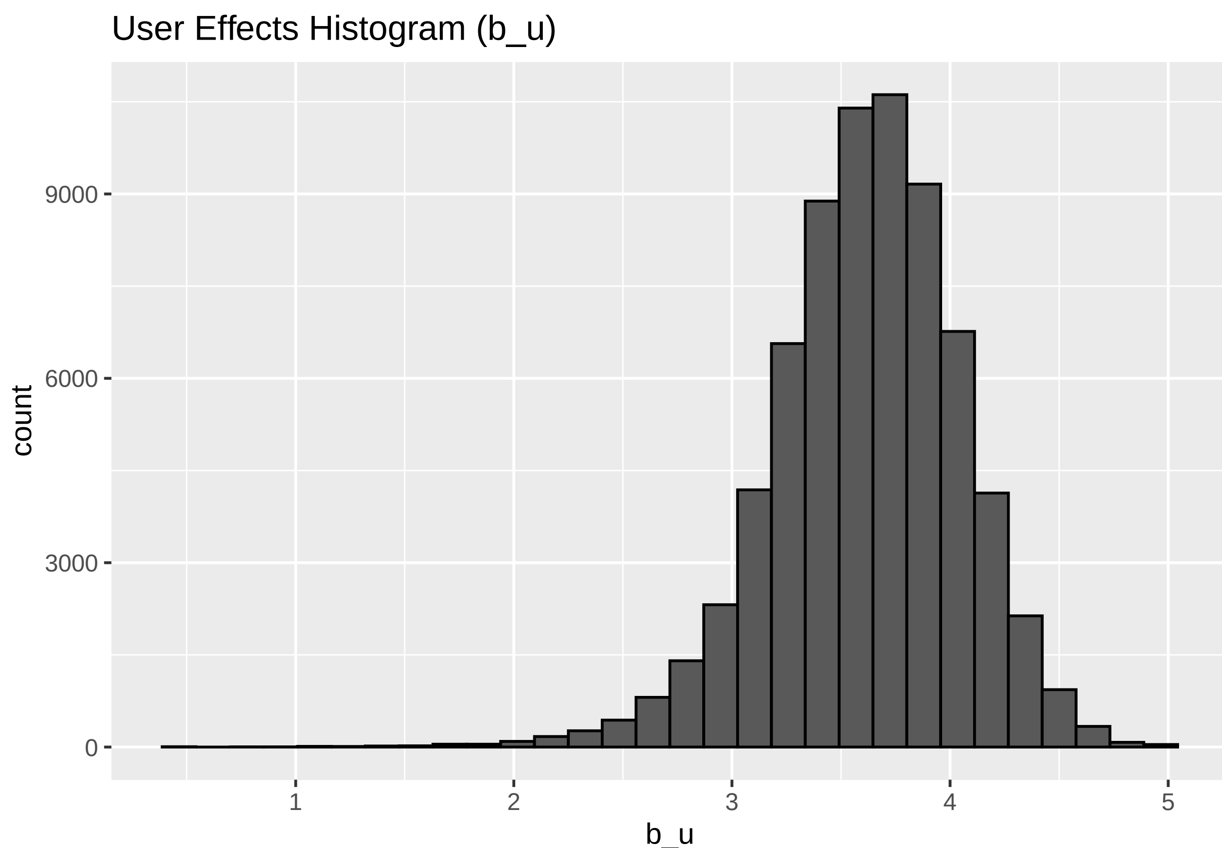
with μ is the **Mean** and $\varepsilon_{u,i}$ is the **independent errors** sampled from the same distribution centered at **0**. The b_i is a **measure for popularity** of **Movie** i.

The **RMSE** of **Movie Effects Based Model** on **final__holdout__test** sets is **0.943909**.
While this is an **improvement** over the **Naive Mean Based Model**, but it is still **Far** from **Satisfactory**.

4.1.2.1 Plot - Movie Effects (frequency vs b__i)



4.1.2.2 Plot - User Effects (frequency vs b_u)



4.1.3 Movie+User Effects Based Model

We can further improve our Model by adding b_u, the **User Specific Effects**. The Model considers that **Users** have different preference therefore some **Users** give higher **rating** than others.

The **Movie+User Effects Based Model** is defined as follow:

$$Y_{u,i} = \mu + b_i + b_u + \varepsilon_{u,i}$$

with μ is the **Mean** and $\varepsilon_{u,i}$ is the **independent errors** sampled from the same distribution centered at **0**. The b_i is a measure for **popularity of Movie** i. The b_u is a measure due to effects associated with **User** u. The Model accounts for **Movie** to **Movie** difference through b_i and **User** to **User** differences through b_u.

The **RMSE** of **Movie+User Effects Based Model** on **final_holdout_test** sets is **0.865349**.
The result has **Improved Significantly** and it is **Almost Reaching** the **Desired Performance**.

4.1.4 Movie+User+Genres Effects Based Model

The **Movie+User+Genres Effects Based Model** is defined as follow:

$$Y_{u,i} = \mu + b_i + b_u + \sum_{k=1}^{K} x_{u,i}^k \beta_k + \varepsilon_{u,i} \text{ with } x_{u,i}^k = 1 \text{ if } g_{u,i} \text{ is genre } k$$

with μ is the **Mean** and $\varepsilon_{u,i}$ is the **independent errors** sampled from the same distribution centered at **0**. The b_i is a **measure for popularity** of **Movie** i . The b_u is a **measure for mildness** of **User** u . I define $g_{u,i}$ as the **genres** for **User** u **rating** of **Movie** i .

RMSE of **Movie+User+Genres Effects Based Model** on **final_holdout_test** sets is **0.864947**.
It's reaching the **Desired Performance** but it represents only a **very Little Improvement** over **Movie+User Effects Based Model**.

4.1.5 Movie+User+Time Effects Based Model

The **Movie+User+Time Effects Based Model** is defined as follow:

$$Y_{u,i} = \mu + b_i + b_u + f(bt_i) + \varepsilon_{u,i} \text{ with } f \text{ a smooth function of } bt_i$$

with μ is the **Mean** and $\varepsilon_{u,i}$ is the **independent errors** sampled from the same distribution centered at **0**. The b_i is a **measure for popularity** of **Movie** i. The b_u is a **measure for mildness** of **User** u. We define $f(bt_i)$ a **smooth function** of bt_i.

The **RMSE** of **Movie+User+Time Effects Based Model** on **final_holdout_test** sets is **0.864097**.
While it performs **Slightly Better** than **Movie+User+Genres Effects Model**, the **Improvement is Modest**.

4.1.6 Regularization Method

The **Regularization Method** introduces a penalty term (often denoted as **Lambda** λ) to address the issue of **Overfitting**. Specifically, it penalizes **Movies** with large estimates based on a small sample size. By doing so, it helps prevent the Model from fitting noise in the data and encourages more **robust Predictions**.

In order to optimize b_i, **Regularized Movie Effects Based Model** is defined as follow:

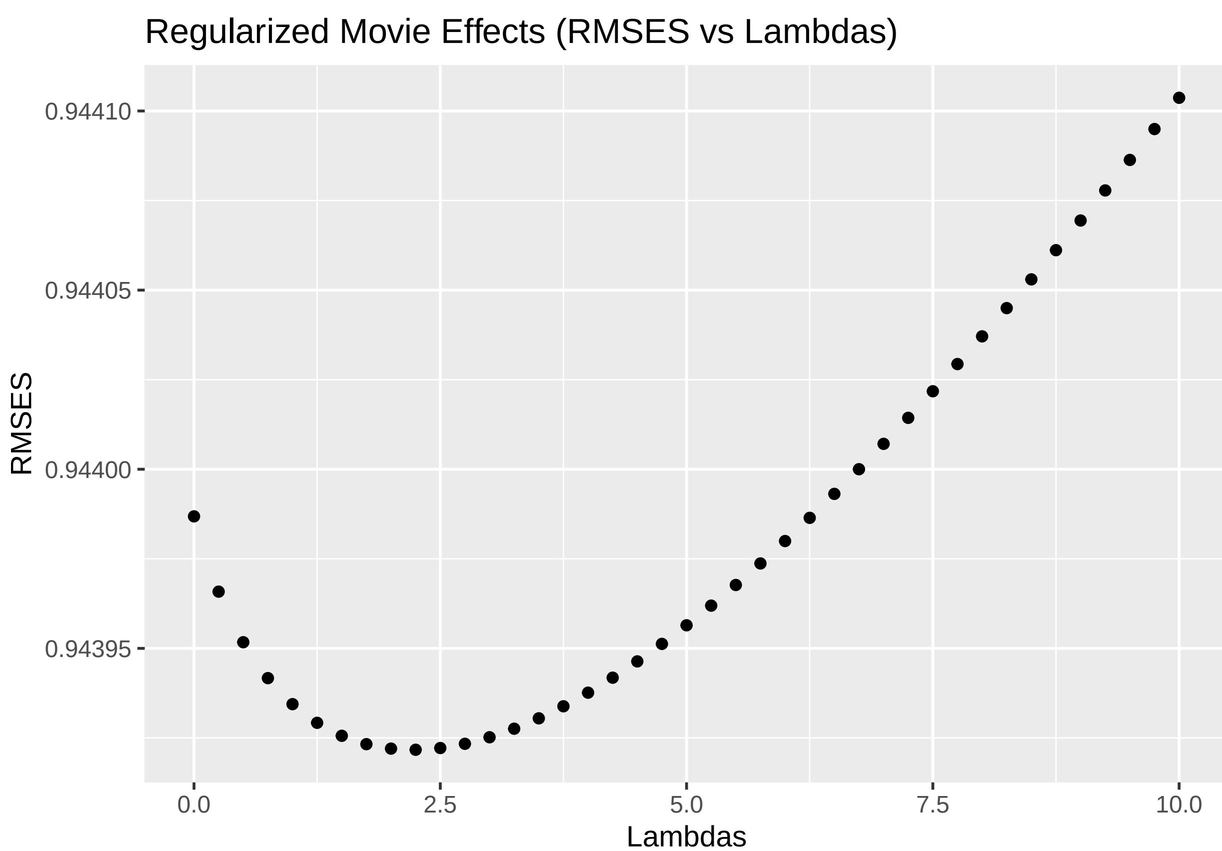
$$\frac{1}{N}\sum_{u,i}(y_{u,i}-\mu-b_i)^2+\lambda\sum_i b_i^2$$

reduced to this Model as follow:

$$\hat{b}_i(\lambda)=\frac{1}{\lambda+n_i}\sum_{u=1}^{n_i}(Y_{u,i}-\hat{\mu})$$

The **RMSE** of **Regularized Movie Effects Based Model** on **final_holdout_test** sets is **0.943852**. However, the **Improvement** over the **Non-Regularized Movie Effects Based Model** is **Minimal**.

4.1.6.1 Plot - Regularized Movie Effects (RMSES vs Lambdas)



4.1.7 Regularized Movie+User Effects Based Model

In order to optimize b_u, it is defined as follow:

$$\frac{1}{N}\sum_{u,i}(y_{u,i}-\mu-b_i-b_u)^2+\lambda\left(\sum_i b_i^2+\sum_u b_u^2\right)$$

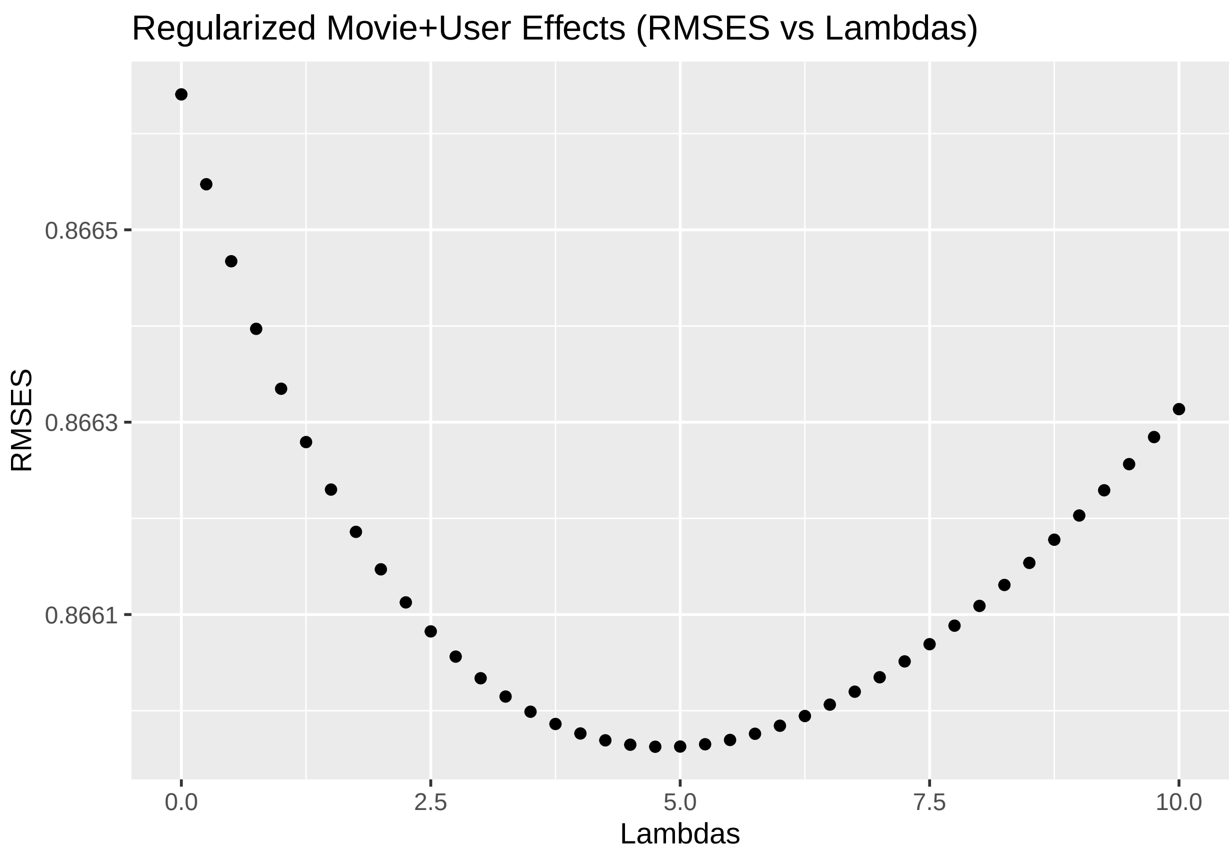
reduced to this Model as follow:

$$\hat{b}_u(\lambda)=\frac{1}{\lambda+n_u}\sum_{i=1}^{n_u}(Y_{u,i}-\hat{\mu}-\hat{b}_i(\lambda))$$

The **RMSE** of **Regularized Movie+User Effects Based Model** on **final_holdout_test** sets is **0.86482**.

While it shows **Substantial Improvement** compared to the **Regularized Movie Effects Based Model** but the **Improvement** over **Non-Regularized Movie+User Effects Based Model** is only **Marginal**.

4.1.7.1 Plot - Regularized Movie+User Effects (RMSES vs Lambdas)



4.1.8 Regularized Movie+User+Genres Effects Based Model

In order to optimize b_g, it is defined as follow:

$$\frac{1}{N}\sum_{u,i}(y_{u,i}-\mu-b_i-b_u-b_g)^2+\lambda\left(\sum_i b_i^2+\sum_u b_u^2+\sum_g b_g^2\right)$$

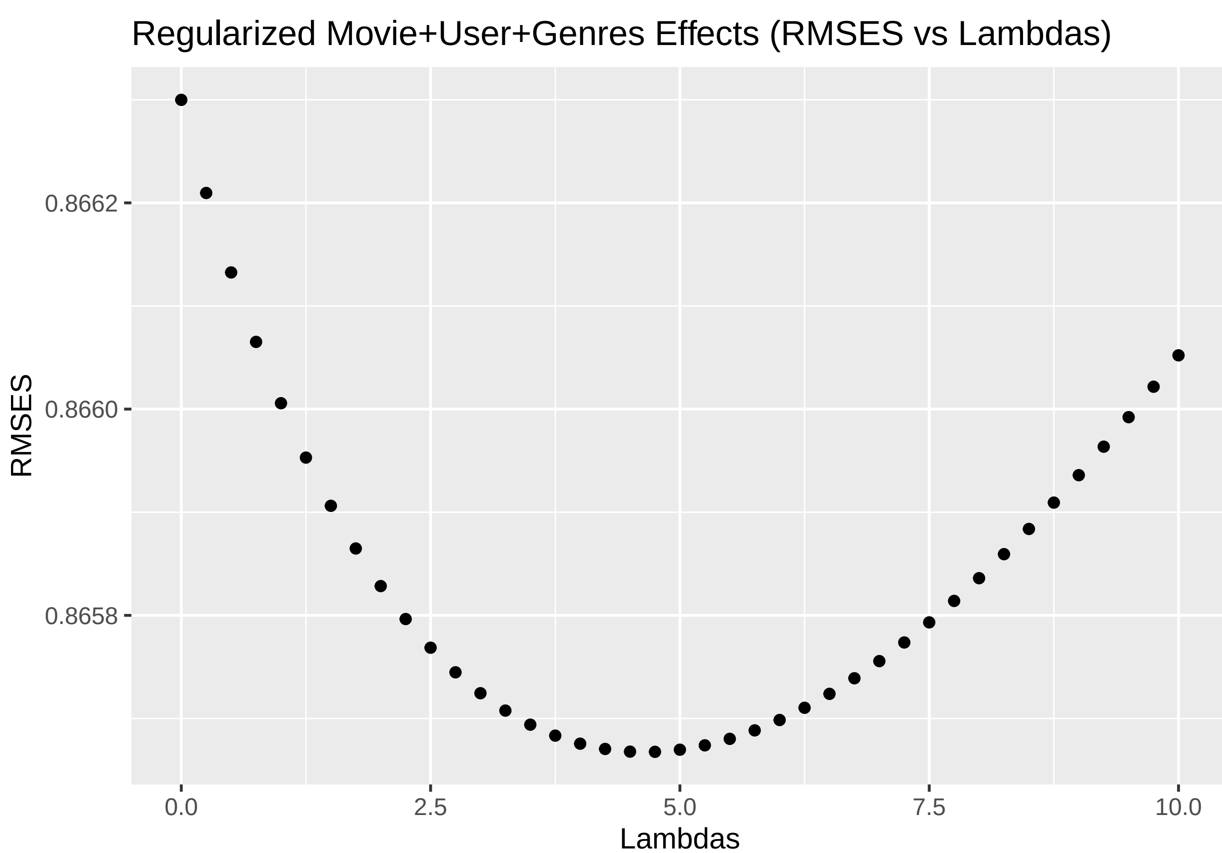
reduced to this Model as follow:

$$\hat{b}_g(\lambda)=\frac{1}{\lambda+n_g}\sum_{i=1}^{n_g}(Y_{u,i}-\hat{\mu}_g-\hat{b}_u(\lambda))$$

The **RMSE** of **Regularized Movie+User+Genres Effects Based Model** on **final_holdout_test** sets is **0.864456**.

While it is **Slight Improvement** over the **Non-Regularized Movie+User+Genres Effects Based Model**, the **RMSE** remains **Very Close** to the **Regularized Movie+User Effects Based Model**.

4.1.8.1 Plot - Regularized Movie+User+Genres Effects (RMSES vs Lambdas)



4.1.9 Regularized Movie+User+Time Effects Based Model

In order to optimize bt_i (bt_i is **Time Specific Effects**), it is defined as follow:

$$\frac{1}{N}\sum_{u,i}(y_{u,i}-\mu-b_i-b_u-bt_i)^2+\lambda\left(\sum_i b_i^2+\sum_u b_u^2+\sum_i bt_i^2\right)$$

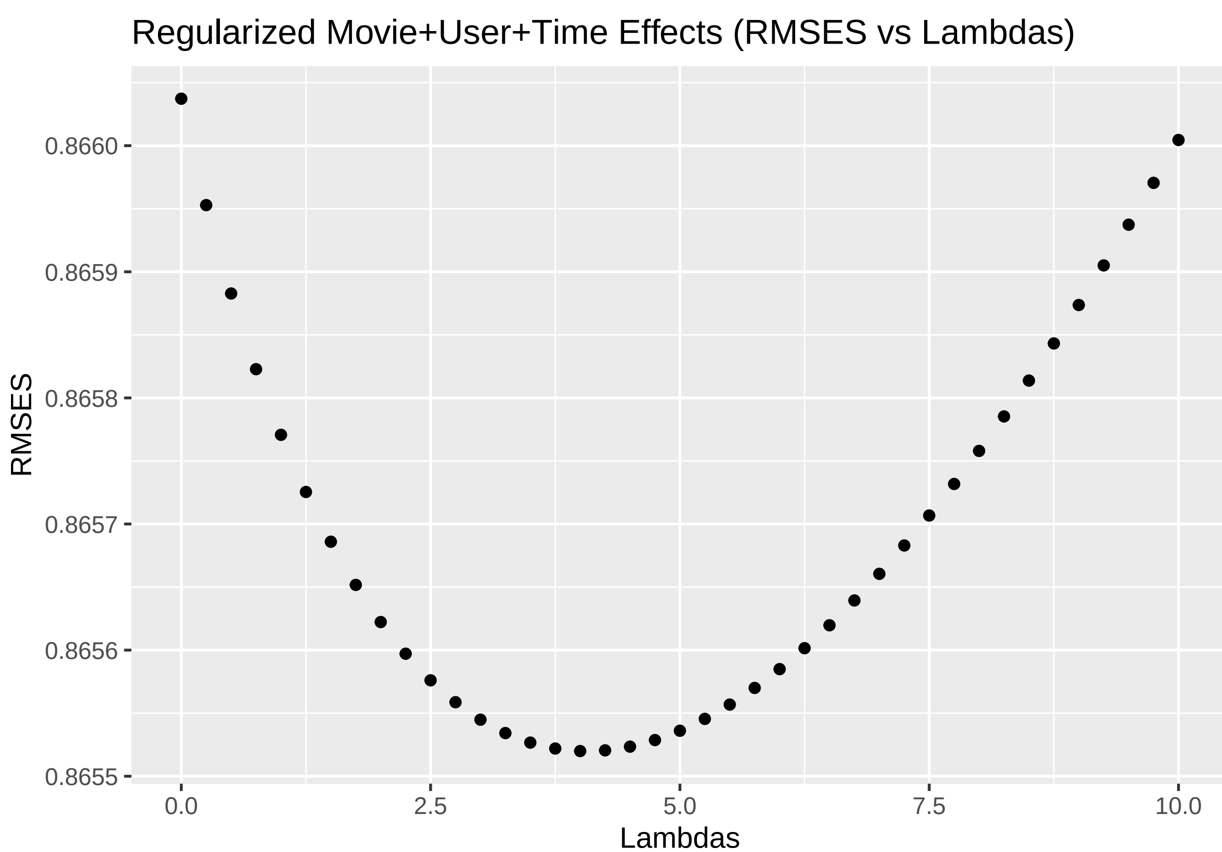
reduced to this Model as follow:

$$\hat{bt}_i(\lambda)=\frac{1}{\lambda+n_i}\sum_{u=1}^{n_i}(Y_{u,i}-\hat{\mu t}_i-\hat{b}_u(\lambda)) \text{ ,with } \hat{\mu t}_i=\hat{f}(x_0)=\frac{1}{N_0}\sum_{i\in A_0}Y_i \quad ,|x_i-x_0|\leq 7$$

The **RMSE** of **Regularized Movie+User+Time Effects Based Model** on **final_holdout_test** sets is **0.863759**.

Although the **Improvement** in **RMSE** is **Not Substantial**, but this Model **Outperforms All other Models** by achieving the **lowest RMSE** value. Consequently, the **Regularized Movie+User+Time Effects Based Model** demonstrates a **Very Good Performance**.

4.1.9.1 Plot - Regularized Movie+User+Time Effects (RMSES vs Lambdas)



R codes for Final Regularized Movie+User+Time Effects Based Model Building

```
# 1) build regularized movie+user+time effects based model:
#
#                      a) training of edx datasets
#                      b) testing on final_holdout_test sets
#
# 2) apply relevant minimum lambda value on algorithm and compute final model RMSE


# minimum lambda value of regularized movie+user+time effects based model from wrangled data
lambd <- min_lambda$movie_user_time[1]

# compute mean of ratings of edx datasets
mu <- mean(edx$rating, na.rm=TRUE)

# compute b_i (movie effects) of edx datasets
b_i <- edx %>%
          group_by(movieId) %>%
          summarize(b_i = sum(rating - mu)/(n()+lambd))

# compute b_u (user effects) of edx datasets
b_u <- edx %>%
          left_join(b_i, by="movieId") %>%
          group_by(userId) %>%
          summarize(b_u = sum(rating - b_i - mu)/(n()+lambd))

# compute b_t (time effects) of edx datasets
b_t <- edx %>%
          left_join(b_u, by="userId") %>%
          group_by(movieId) %>%
          summarize(b_t = sum( rating - b_u - m_rw ) /(n()+lambd))

# compute predicted ratings (mean of ratings + movie Effects + user Effects + time effects)
predicted_ratings <-  final_holdout_test %>%
         left_join(b_i, by = "movieId") %>%
         left_join(b_u, by = "userId")  %>%
         left_join(b_t, by = "movieId") %>%
         mutate(pred = mu + b_i + b_u + b_t ) %>%
         pull(pred)

# compute RMSE on final_holdout_test
final_model_rmse_rmut <- RMSE(predicted_ratings, final_holdout_test$rating)

# generate regularized movie+user+time effects based model final models RMSE table
final_model_rmse_table <- bind_rows(final_model_rmse_table,
                              data_frame(MODEL = "Regularized Movie+User+Time Effects Based Model",
                                         RMSE  = final_model_rmse_rmut))
```

4.2 Algorithm - Key Points

- Use **Collaborative Filtering Approach** for Models Building.
- Design and formula the **linear relationship** between the **Predictors/Features**.
- Fit our **Machine Learning Models** with **Movie, User, Genres or Time Metrics** for **Prediction**.
- **Regularization Method** is used to penalize magnitudes of **Parameters** to avoid **Overfitting**.
- Use **Regularization** to estimate **Movie Effects, User Effects, Genres Effects and Time Effects**.
- Determine **Minimum Lambda** values for different Models during training and testing process using **Cross Validation Method**.
- **Predict rating** of **Movie** by applying relevant **Minimum Lambda** on **Algorithm**.
- Compute **Root Mean Squared Errors (RMSES)** of different Models.
- Retain the relevant **Minimum Lambda** values to Compute **RMSES** for **Final Models**.
- Select the Ultimate **Best Model** with an **lowest RMSE** value among all of the **Final Models**.

5 Result

5.1 Metric and Algorithm Evaluation

The **RMSE** Metric is used to evaluate an **Algorithm**. These can all be derived from using values of **Lambda** (λ) from **0** to **10** increment by **0.25** and then find **Lambda** values that minimizes the **RMSES**.

Lower the **Root Mean Squared Error** values indicate better Model Performance. The Goal is to minimize the **RMSE** between **Predicted** and **Actual rating**. **Regularization Method** is used to shrink deviations from the average towards **0**. To apply the Method, I subtract the overall average before shrinking since using **Regularization Method** is shrinking values towards to **0**. **Predicted rating** for each Model is divided by **n + Lambda**, with n the **size** and **Lambda** a **Regularization Parameter**.

5.2 RMSES of Final Models/Algorithm

On average, RMSE Does Not Change much as n gets Larger, but the Variability of RMSES Decreases.

A) Following table show Lambdas Minimum RMSE for Regularized Final Models

Table 12: Lambdas give the Minimun RMSE

Effects	Lambdas
Regularized Movie Effects	2.25
Regularized Movie+User Effects	4.75
Regularized Movie+User+Genres Effects	4.75
Regularized Movie+User+Time Effects	4.00

B) Following table show Summary Results of RMSE for all Final Models

Table 13: Result of Final Models - RMSE

MODEL	RMSE
Naive Mean Based Model	1.061202
Movie Effects Based Model	0.943909
Movie+User Effects Based Model	0.865349
Movie+User+Genres Effects Based Model	0.864947
Movie+User+Time Effects Based Model	0.864097
Regularized Movie Effects Based Model	0.943852
Regularized Movie+User Effects Based Model	0.864820
Regularized Movie+User+Genres Effects Based Model	0.864456
Regularized Movie+User+Time Effects Based Model	0.863759

Final Model with Lowest RMSE is "Regularized Movie+User+Time Effects Based Model".

Table 14: Final Model with Lowest RMSE

MODEL	RMSE
Regularized Movie+User+Time Effects Based Model	0.863759

After training different Models on the **edx** datasets and evaluating them on **final__holdout__test** sets, The **Regularized Movie+User+Time Effects Based Model** achieved an **Lowest RMSE** of **0.863759**. This Result also Demonstrates a **Very Strong Performance**.

6 Conclusion

Collaborative Filtering Approach captures the interactions between **Users** and **Movies** that result in diverse **ratings**. However, observed **ratings** are also influenced by Effects associated with **Users** and **Movies**, such as **Genres Effects** and **Time Effects**. During the analysis process, I explore and ascertain **Time Effects** account for a portion of the **Variability of ratings**. A observed trends indicates that **More Frequently a Movie is rated, the Higher its Average rating**. Through Data Visualization, Stratification Analysis, Distributions Analysis and Frequency Analysis, I determine that **Mean of rating** by **Week of Date** is the most appropriate **Parameter** for Modelling **Time Effects**.

The **Regularization Method** is employed to shrunk the **Mean of rating** towards **Zero**, thereby avoiding **Overfitting**. The Algorithm demonstrates **Robust Performance** in **Predicting** the Movie **rating** as evidenced by **RMSE** value of **Final Model**.

6.1 Limitations

However, after incorporating **Regularization** and **Cross Validation Method**, the Improvement Over some Other Regularized Models is still **Minimal**. It may also be a **Limitation** of **Regularization** and **Cross Validation Method** when using the **Collaborative Filtering Approach**.

6.2 Final Regularized Movie+User+Time Effects Based Model

RMSE of Regularized Movie+User+Time Effects Based Model(**0.863759**) **Outperforms** Non-Regularized Movie+User+Time Effects Based Model(**0.864097**), as well as Regularized Movie+User+Genres Effects Based Model(**0.864456**) and Regularized Movie+User Effects Based Model(**0.86482**).

Regularized Movie+User+Time Effects Based Model is the Optimal Model, achieving an **Lowest Root Mean Square Error (RMSE)** of **0.863759**. I am **Optimistic** about the **Final Machine Learning Algorithm** that we have Selected.

The **Final Regularized Movie+User+Time Effects Based Model** is defined as follow:

$$Y_{u,i} = \mu + b_i + b_u + f(bt_i) + \varepsilon_{u,i} \text{ with } f \text{ a smooth function of } bt_i$$

$$\hat{bt}_i(\lambda) = \frac{1}{\lambda + n_i} \sum_{u=1}^{n_i} (Y_{u,i} - \hat{\mu t}_i - \hat{b}_u(\lambda)) \text{ ,with } \hat{\mu t}_i = \hat{f}(x_0) = \frac{1}{N_0} \sum_{i \in A_0} Y_i \quad , |x_i - x_0| \leq 7$$

The **Best Model** is **Regularized Movie+User+Time Effects Based Model**, which achieved an **Lowest RMSE**. I am satisfied with the results, and the **Reliability** and **Trustworthiness** of the Model are Validated by the **RMSE Metric Evaluation**.

The Final Machine Learning Model achieved an Lowest RMSE of 0.863759 which can described as Exceptional Outcome. As a Result, I am Confident in adopting the Ultimate Model and Algorithm to build our Movie Recommendation System.

6.3 Future Work/Research

In this project, we refer our **Movie Recommendation System** to the **Machine Learning tasks** as **Prediction** since the **predicted ratings** output is **Continuous**. Thus, **RMSE** Metric is used to evaluate our **Model/Algorithm**. We can conduct research on Machine Learning Algorithm for assigning **Predicted rating** to the appropriate **Classes of rating**.

The 10 Classes of ratings are (0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0) because the **ratings** of **Movie** range from **0.5** to **5.0** with increment of **0.5**.

Then, We can refer the **Machine Learning task** as **Classification** since the outcome is **Categorical**. Therefore, Metrics such as **Accuracy**, **F1 Score**, **Sensitivity** and **Specificity** can also be adopted to evaluate our **Machine Learning Model** in the **Future**.

Appendix

All Code Chunks of Rmd Report

```
knitr::opts_chunk$set(
    message = FALSE,
    warning = FALSE
)

# Note: It takes approx. 4 Minutes to generate the Data Science Report.


############################################
# Installing Packages and Loading Libraries #
############################################

# Install Necessary Packages if required

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

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

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

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

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

if(!require(ggplot2)) install.packages("ggplot2",
                                       repos = "http://cran.us.r-project.org")

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

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

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

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

if(!require(tinytex)){
  install.packages("tinytex", repos = "http://cran.us.r-project.org")
  tinytex::install_tinytex()
}
```

```r
# Loading Necessary Libraries

  library(dslabs)
  library(tidyverse)
  library(dplyr)
  library(tidyr)
  library(lubridate)
  library(stringr)
  library(ggplot2)
  library(gridExtra)
  library(knitr)
  library(rstudioapi)
  library(caret)
  library(tinytex)


# Set number of significant digits=6 globally

  options(digits=6)

  set.seed(1990)

# Loading rda files which previously generated after running recommendation-system.R

  load("rda/edx-original-rda.rda")

  load("rda/edx-rda.rda")

  load("rda/final-holdout-test.rda")

  load("rda/min-lambda.rda")


# - generate dimension of edx: number of rows and columns
# - display output using kable function

dim_df <- data.frame(Rows = dim(edx_original)[1], Columns = dim(edx_original)[2])

dim_df %>% knitr::kable(caption="edx datasets")


# - generate column name and class of edx
# - display output using kable function

class_results <- data.frame(Class=sapply(edx_original, class))

class_results %>% knitr::kable(caption="edx datasets")


# compute number of unique movieId, userId and genres

u_movie_n  <- length(unique(edx_original$movieId))
```

```r
u_user_n   <- length(unique(edx_original$userId))

u_genres_n <- length(unique(edx_original$genres))


# - generate table - Number of Unique movieId, userId and genres.
# - display output using kable function

um_result <- data.frame(Description="Number of Unique movieId", Count=u_movie_n )

um_result <- bind_rows(um_result,
                       data.frame(Description="Number of unique userId", Count=u_user_n))

um_result <- bind_rows(um_result,
                       data.frame(Description="Number of unique combined genres", Count=u_genres_n))

um_result %>% knitr::kable(caption="edx datsets")


# - generate 10 examples - counting the Occurrences of rating by movieId
# - display output using kable function

m_result <- edx %>% count(movieId,name="Occurency") %>%
                     count(Occurency, name="Count") %>%
                     arrange(desc(Count)) %>%
                     dplyr::slice(1:10)
m_result %>% knitr::kable(caption="edx datasets")


# - generate 10 examples - counting the occurrences of rating by userId
# - display output using kable function

u_result <- edx %>% count(userId,name="Occurency") %>%
                     count(Occurency, name="Count") %>%
                     dplyr::slice(30:40)
u_result %>% knitr::kable(caption="edx datasets")


# - generate distributions (number of occurrence) using movieId and userId
# - ggplot histogram - m1 and m2

m1 <- edx %>%
          dplyr::count(movieId) %>%
          ggplot(aes(n)) +
          geom_histogram(bins = 50, fill="#006EBB", color="black") +
          labs(y="Count",x="Number of Occurrence") +
          scale_x_log10() +
          ggtitle("Movies")

m2 <- edx %>%
          dplyr::count(userId) %>%
          ggplot(aes(n)) +
          geom_histogram(bins = 200, fill="cyan", color="#D883B7"   ) +
```

```
          labs(y="Count",x="Number of Occurrence") +
          scale_x_log10()+
          ggtitle("Users")

grid.arrange(m1, m2, ncol = 2)


# - generate missing value (if any) in column rating of edx
# - display output using kable function

na_results  <- edx[apply(is.na(edx),1,any),]

na_results %>% select(userId,movieId,rating,timestamp,title,genres) %>%
               knitr::kable(caption="Columns with NA values in edx datasets")


# generate and display zeros value (if any) in column rating of edx

length(which(edx$rating==0))


# - generate 8 examples: the original edx - rating given by one User to one movie
# - display output of edx using kable function

    edx %>%
          group_by(title) %>%
          mutate(f=length(rating)) %>% ungroup() %>%
          filter(f > 10000) %>%
          distinct(title, .keep_all=TRUE) %>%
          arrange(desc(f)) %>%
          select(movieId,title,genres,userId,rating,timestamp) %>%
          dplyr::slice(1:8) %>%
          knitr::kable(caption="edx datasets")


# - mean of rating by genres bEFORE combined genres separation
# - generate 20 examples-mean of rating by genres
#                                in descending order by number of rating(count > 10000)

genres_avg <- edx %>%
                   group_by(genres) %>%
                   summarize(average_rating_genres=mean(rating, na.rm=TRUE ),
                   se=sd(rating, na.rm=TRUE)/sqrt(n()), count=n()) %>%
                   filter(count > 10000) %>%
                   arrange(desc(average_rating_genres)) %>%
                   mutate(rank=rownames(.))

# display output of genres_avg

genres_avg %>% select(rank, genres, average_rating_genres, count) %>%
               dplyr::slice(1:20)

# prepare genres_avg_-plot for ggplot
```

```r
genres_avg_plot <- genres_avg %>% dplyr::slice(1:20)


# display mean of rating by genres (genres="Drama")

genres_avg %>% select(rank, genres, average_rating_genres, count) %>%
               dplyr::slice(48)

# display mean of rating by genres (genres="Comedy")

genres_avg %>% select(rank, genres, average_rating_genres, count) %>%
               dplyr::slice(131)

# generate and display rank, Combined genres, Mean of rating by genres
#                      and number of rating of "Comedy" and "Drama"

genres_t1 <- genres_avg %>%
                         filter(genres%in%c("Comedy","Drama")) %>%
                         select(rank, genres, average_rating_genres, count)


# ggplot mean of rating by genres with error bars before genres separation

genres_avg_plot %>%
                ggplot(aes(x=genres, y=average_rating_genres)) +
                geom_errorbar(aes(ymin=average_rating_genres - se,
                                  ymax=average_rating_genres + se),color="#7977B8") +
                geom_point(color="#F89E78") +
                theme(axis.text.x=element_text(angle=90, vjust=0.5, hjust=1)) +
                labs(x="Combined genres", y="Mean of rating",
                title="(Original Combined genres) Mean of rating by genres with Error Bars")


# - mean of rating by genres after Combined genres separation

# - data Wrangling: 1) separate combined genres into several Rows
#                      and each row contains only one genre
#                   2) use data frame edx_original

# - generate mean of rating by genres in descending order(count > 20000)

genres_avg <- edx_original %>%
                 separate_longer_delim(genres, delim="|") %>%
                 group_by(genres) %>%
                 summarize(average_rating_genres=mean(rating, na.rm=TRUE ),
                 se=sd(rating, na.rm=TRUE)/sqrt(n()), count=n()) %>%
                 filter(count > 200000) %>%
                 arrange(desc(average_rating_genres)) %>%
                 mutate(rank=rownames(.))

# display output of genres_avg

   genres_avg %>% select(rank, genres, average_rating_genres, count) %>%
```

```r
                  dplyr::slice(1:20)

# generate rank, combined genres, mean of rating by genres
#           and number of rating of "Comedy" and "Drama"

  genres_t2 <- genres_avg %>%
                          filter(genres%in%c("Comedy","Drama")) %>%
                          select(rank, genres, average_rating_genres,count)

# Prepare genres_avg_plot for ggplot

  genres_avg_plot <- genres_avg %>% dplyr::slice(1:20)


# ggplot mean of rating by genres with error bars after genres separated into several rows

  genres_avg_plot %>%
        ggplot(aes(x=genres, y=average_rating_genres)) +
        geom_errorbar(aes(ymin= average_rating_genres- se,
                          ymax= average_rating_genres+ se),color="#7977B8") +
        geom_point(color="#F89E78") +
        theme(axis.text.x=element_text(angle=90, vjust=0.5, hjust=1)) +
        labs(x="Separated genres", y="Mean of rating",
        title="(After genres Separated) Mean of rating by genres with Error Bars") +
        scale_y_continuous(breaks=seq(0.5,5,by=0.1))


# display output table of genres_t1 using kable function

genres_t1 %>% knitr::kable(caption="BEFORE Combined genres Separated")


# display output table of genres_t2 using kable function

genres_t2 %>% knitr::kable(caption="After Combined genres Separated")


# data Wrangling: generate new columns (d_w, d_m, d_y) from column timestamp

   edx <- edx %>% mutate(d_w=format(round_date(as_datetime(timestamp),"week"),"%Y-%m-%d"))

   edx <- edx %>% mutate(d_m=format(round_date(as_datetime(timestamp),"month"),"%Y-%m-%d"))

   edx <- edx %>% mutate(d_y=format(round_date(as_datetime(timestamp),"year"),"%Y-%m-%d"))


# data Wrangling: generate new columns (m_r, m_rw, m_ry, m_rg, tot_nr) from column rating

edx <- edx %>% group_by(movieId) %>% mutate(m_r = mean(rating, na.rm=TRUE))

edx <- edx %>% group_by(movieId,d_w) %>% mutate(m_rw= mean(rating, na.rm=TRUE))

edx <- edx %>% group_by(movieId,d_y) %>% mutate(m_ry= mean(rating, na.rm=TRUE))
```

```r
edx <- edx %>% group_by(userId,genres) %>% mutate(m_rg=mean(rating, na.rm=TRUE))

edx <- edx %>% group_by(movieId) %>% mutate(tot_nr=n()) %>% ungroup()


# data wrangling: generate new column release from column title

edx <- edx %>% mutate(release = str_extract(title,"\\d{4}")) %>%
               mutate(title = str_replace(title,"\\s*\\(\\d{4}\\)",""))


# generate and display 10 examples encompassing new features (d_w)

edx %>% group_by(title) %>%
        mutate(f=length(rating)) %>% ungroup() %>%
        filter(f > 15000) %>%
        arrange(desc(f),timestamp) %>%
        distinct(title, .keep_all=TRUE) %>%
        select(userId,movieId,title,d_w,rating) %>%
        dplyr::slice(10:20)


# generate 8 examples encompassing new features (release, m_r, m_rw, m_ry, m_rg, tot_nr)

edx %>% group_by(title) %>%
        mutate(f=length(rating)) %>% ungroup() %>%
        filter(f > 20000) %>%
        arrange(desc(f),timestamp) %>%
        distinct(title, .keep_all=TRUE) %>%
        select(userId,release,title,rating,m_r,m_rw,m_ry,m_rg) %>%
        dplyr::slice(1:8)


# generate 20 movies: most frequently rated by User with frequency > 10000

c_1 <- edx %>%
           group_by(title) %>%
           mutate(frequency=length(rating)) %>% ungroup() %>%
           filter(frequency > 10000) %>%
           distinct(title, .keep_all=TRUE) %>%
           arrange(desc(frequency)) %>%
           mutate(rank=rownames(.)) %>%
           select(rank,title,frequency) %>%
           dplyr::slice(1:20)

# display c_1 using kable function

c_1 %>% knitr::kable(caption="Frequency of rating By title")


# ggplot frequency of rating By title

c_1 %>% mutate(title=reorder(title,frequency)) %>%
```

```r
        ggplot(aes(title,frequency)) +
        geom_bar(stat="identity",fill="#D883B7",color="white")  +
        labs(y="Frequency",x="Moive title",title="Frequency of User rating By Movie title") +
        theme(axis.text.x=element_text(angle=90)) +
        scale_y_continuous(breaks=seq(0,33000,by=3000)) +
        coord_flip()


# generate 20 genres: most frequently rated by User with frequency > 10000

c_2 <- edx %>%
           group_by(genres) %>%
           mutate(frequency=length(rating)) %>% ungroup() %>%
           filter(frequency > 10000) %>%
           distinct(genres, .keep_all=TRUE) %>%
           arrange(desc(frequency)) %>%
           mutate(rank=rownames(.)) %>%
           select(rank,genres,frequency) %>%
           dplyr::slice(1:20)

# display c_2

print_df <- function(title,df)
              {
                 cat(title, "\n\n")
                 cat(capture.output(print(n=50,df)), sep="\n")
              }
print_df("List of 20 genres - Frequency of rating By genres ",c_2)


# ggplot frequency of rating By genres

c_2 %>% mutate(genres=reorder(genres,frequency)) %>%
        ggplot(aes(genres,frequency)) +
        geom_bar(stat="identity",fill="#584298",color="white")  +
        labs(y="Frequency",x="genres",title="Frequency of User rating By genres") +
        theme(axis.text.x=element_text(angle=90)) +
        scale_y_continuous(breaks=seq(0,800000,by=80000)) +
        coord_flip()


# compute correlation coefficient (r) of rating and m_r with 6 decimals place
r <- edx %>%
         summarize(r=cor(rating,m_r)) %>%
         pull(r)
r <- round(r,6)

# ggplot correlation of Normalized Mean of rating and rating
edx %>% mutate(rating=scale(rating),m_r=scale(m_r)) %>%
        group_by(rating) %>%
        summarize(m_r=mean(m_r)) %>%
        ggplot(aes(rating,m_r)) + geom_point()+
        geom_abline(intercept=0, slope=r,color="#F89E78")+
```

```r
        labs(x="scale(rating)",y="scale(m_r)",
        title=paste("Normalized Mean of rating vs rating [cor =",r,"]"))


# compute correlation coefficient (r) of rating and m_rw with 6 decimals place

r <- edx %>%
         summarize(r=cor(rating,m_rw)) %>%
         pull(r)
r <- round(r,6)

# ggplot correlation of normalized mean by week of date and rating

edx %>% mutate(rating=scale(rating),m_rw=scale(m_rw)) %>%
        group_by(rating) %>%
        summarize(m_rw=mean(m_rw)) %>%
        ggplot(aes(rating,m_rw)) + geom_point()+
        geom_abline(intercept=0, slope=r,color="#F89E78")+
        labs(x="scale(rating)",y="scale(m_rw)",
        title=paste("Normalized Mean of rating by Week of Date vs rating [cor =",r,"]"))


# compute correlation coefficient (r) of rating and m_ry with 6 decimals place

r <- edx %>%
         summarize(r=cor(rating,m_ry)) %>%
         pull(r)
r <- round(r,6)

# ggplot correlation of normalized mean by year of date and rating

edx %>% mutate(rating=scale(rating),m_ry=scale(m_ry)) %>%
        group_by(rating) %>%
        summarize(m_ry=mean(m_ry)) %>%
        ggplot(aes(rating,m_ry)) + geom_point()+
        geom_abline(intercept=0, slope=r,color="#F89E78")+
        labs(x="scale(rating)",y="scale(m_ry)",
        title=paste("Normalized Mean of rating by Year of Date vs rating [cor =",r,"]"))


# compute correlation coefficient (r) of rating and m_rg with 6 decimals place

r <- edx %>%
         summarize(r=cor(rating,m_rg)) %>%
         pull(r)
r <- round(r,6)

# ggplot correlation of normalized mean of rating by genres and rating

edx %>% mutate(rating=scale(rating),m_rg=scale(m_rg)) %>%
        group_by(rating) %>%
        summarize(m_rg=mean(m_rg)) %>%
        ggplot(aes(rating,m_rg)) + geom_point()+
```

```r
        geom_abline(intercept=0, slope=r,color="#F89E78") +
        labs(x="scale(rating)",y="scale(m_rg)",
        title=paste("Normalized Mean of rating by genres vs rating [cor =",r,"]"))


# generate correlation coefficient table (m_r, m_rw, m_ry, m_rg) of edx

avg_r_all <- edx %>%
                 select(rating,m_r,m_rw,m_ry,m_rg)
cor_r_all <- cor(na.omit(avg_r_all[, unlist(lapply(avg_r_all, is.numeric))]))

# display correlation coefficient table using kable function

cor_r_all %>% knitr::kable(caption="Correlation - new Features of edx dataset")


# ggplot rating distributions of movie

edx %>% ggplot(aes(rating)) +
        geom_histogram(binwidth=0.5,fill="#7977B8",color="black") +
        labs(y="Frequency",x="rating",title="rating Distributions of Movie") +
        scale_x_continuous(breaks=seq(0.5,5,by=0.5))


# generate frequency group by rating

edx_asc <- edx %>%
               group_by(rating) %>%
               summarise(Frequency = n()) %>%
               arrange(Frequency)

# ggplot distributions of movie from most given rating in order from most to least

edx_asc %>%
        ggplot(aes(x=reorder(rating, Frequency), y=Frequency)) +
        geom_bar(stat="identity", width=0.5, aes(fill=Frequency)) +
        scale_fill_gradient(low="red", high="blue") +
        coord_flip() +
        labs(y="Frequency",
             x="rating",
             title = "Distributions of Movie (Most Given rating in order from Most to Least)")


# ggplot mean of rating distributions

edx %>% ggplot(aes(m_r)) +
        geom_histogram(binwidth=0.1,fill="#C6DC67",color="black") +
        labs(y="Frequency",x="Mean of rating",title="Mean of rating Distributions") +
        theme(axis.text.x=element_text(angle=90)) +
        scale_x_continuous(breaks=seq(0.5,5,by=0.1))


# ggplot mean of rating distributions by movieId, week of date
```

```r
edx %>% ggplot(aes(m_rw)) +
        geom_histogram(binwidth=0.1,fill="#C6DC67",color="black") +
        labs(y="Frequency",x="Mean of rating",
             title="Mean of rating Distributions By movieId, Week of Date") +
        theme(axis.text.x=element_text(angle=90)) +
        scale_x_continuous(breaks=seq(0.5,5,by=0.1))


# ggplot mean of rating distributions by movieId, year of date

edx %>% ggplot(aes(m_ry)) +
        geom_histogram(binwidth=0.1,fill="#C6DC67",color="black") +
        labs(y="Frequency",x="Mean of rating",
             title="Mean of rating Distributions By movieId, Year of Date") +
        theme(axis.text.x=element_text(angle=90)) +
        scale_x_continuous(breaks=seq(0.5,5,by=0.1))


# ggplot mean of rating distributions by userId, genres

edx %>% ggplot(aes(m_rg)) +
        geom_histogram(binwidth=0.1,fill="cyan",color="black") +
        labs(y="Frequency",x="Mean of rating",
             title="Mean of rating Distributions By userId, genres") +
        theme(axis.text.x=element_text(angle=90)) +
        scale_x_continuous(breaks=seq(0.5,5,by=0.1))


# ggplot number of rating distributions by movieId

edx %>% ggplot(aes(movieId)) +
        geom_histogram(binwidth=1,color="#7977B8") +
        labs(y="Number of rating",x="movieId",title="Number of rating Distributions By movieId") +
        theme(axis.text.x=element_text(angle=90)) +
        scale_x_continuous(breaks=seq(0,70000,by=7000))


# ggplot rating distributions through the day of year

edx %>% ggplot(aes(yday_dw)) +
        geom_histogram(binwidth=0.05,color="#F69289") +
        labs(y="Frequency",x="Day",title="rating Distributions through the Day of Year") +
        scale_x_continuous(breaks=seq(0,400,by=50))


# ggplot rating distributions (month)

edx %>% ggplot(aes(month_dm)) +
        geom_histogram(binwidth=1,fill="#46C5DD",color="black") +
        labs(y="Frequency",x="Months",title="rating Distributions (Month)") +
        scale_x_continuous(breaks=seq(1,12,by=1))
```

```
# ggplot rating distributions (year)

edx %>% ggplot(aes(year_dy)) +
        geom_histogram(binwidth=1,fill="#46C5DD",color="black") +
        labs(y="Frequency",x="Years",title="rating Distributions (Year)") +
        theme(axis.text.x=element_text(angle=90)) +
        scale_x_continuous(breaks=seq(1993,2020,by=1))


# RMSE Function
   RMSE <- function(predicted_ratings, true_ratings){
                     sqrt(mean((true_ratings - predicted_ratings)^2))
}


######################################################################################
#     Build Final Models - Training of edx and testing on final_holdout_test sets    #
#                                                                                    #
######################################################################################
#                                                                                    #
# 1. Models Training of edx datasets.                                                #
#                                                                                    #
# 2. Apply relevant Minimum Lambda value.                                            #
#                                                                                    #
# 3. Final Models Testing on final_holdout_test sets.                                #
#                                                                                    #
# 4. Compute RMSE for Final Models.                                                  #
#                                                                                    #
# 5. Generate Final Model RMSE table.                                                #
#                                                                                    #
######################################################################################

# build naive mean based model": 1) training of edx datasets
#                                2) testing on final_holdout_test sets
#                                3) compute final model RMSE
#

set.seed(755)

# compute mean of ratings using edx
mu  <- mean(edx$rating, na.rm=TRUE)

# compute RMSE on final_holdout_test
final_model_rmse_naive <- RMSE(mu, final_holdout_test$rating)

# generate naive mean based model final model RMSE table
final_model_rmse_table  <- data_frame(MODEL = "Naive Mean Based Model",
                                      RMSE = final_model_rmse_naive)


# build movie effects based model: 1) training of edx datasets
#                                  2) testing on final_holdout_test sets
#                                  3) compute final model RMSE
```

```r
# compute mean of ratings of edx
mu <- mean(edx$rating,na.rm=TRUE )

# compute b_i (movie effects) of edx
movie_avgs <- edx %>%
                  group_by(movieId) %>%
                  summarize(b_i = mean(rating - mu))

# compute predicted ratings (mean of ratings + movie effects)
predicted_ratings <- final_holdout_test %>%
                          left_join(movie_avgs, by='movieId') %>%
                          mutate(pred = mu + b_i ) %>%
                          pull(pred)

# compute RMSE on final_holdout_test
final_model_rmse_m <- RMSE(predicted_ratings, final_holdout_test$rating)

# generate movie effects based model final models RMSE table
final_model_rmse_table <- bind_rows(final_model_rmse_table,
                          data_frame(MODEL ="Movie Effects Based Model",
                                     RMSE = final_model_rmse_m ))


# qplot histogram: movie effects (frequency vs b_i)
movie_avgs %>%
            qplot(b_i,geom ="histogram",bins=30,data =.,color=I("black")) +
            labs(title="Movie Effects Histogram (b_i)")


# ggplot histogram: user effects (frequency vs b_u)
edx %>%
     group_by(userId) %>%
     summarize(b_u = mean(rating)) %>%
     ggplot(aes(b_u)) +
     geom_histogram(bins = 30, color = "black") +
     labs(title="User Effects Histogram (b_u)")


# build movie+user effects based model: 1) training of edx datasets
#                                       2) testing final_holdout_test sets
#                                       3) compute final model RMSE

# compute b_u (user effects) of edx
user_avgs <- edx %>%
                 left_join(movie_avgs, by='movieId') %>%
                 group_by(userId) %>%
                 summarize(b_u = mean(rating - mu - b_i))

# compute predicted ratings (mean of ratings + movie effects + user effects)
predicted_ratings <- final_holdout_test %>%
                          left_join(movie_avgs, by='movieId') %>%
                          left_join(user_avgs, by='userId') %>%
                          mutate(pred = mu + b_i + b_u ) %>%
```

```r
                          pull(pred)

# compute RMSE on final_holdout_test
final_model_rmse_mu <- RMSE(predicted_ratings,final_holdout_test$rating)

# generate movie+user effects based model final models RMSE table
final_model_rmse_table <- bind_rows(final_model_rmse_table,
                          data_frame(MODEL="Movie+User Effects Based Model",
                                     RMSE = final_model_rmse_mu))


# build movie+user+genres effects based model: 1) training of edx datasets
#                                              2) testing on final_holdout_test sets
#                                              3) compute final model RMSE

# compute mean of ratings of edx
mu <- mean(edx$rating, na.rm=TRUE)

# compute b_i (movie effects) of edx
b_i <- edx %>%
           group_by(movieId) %>%
           summarize(b_i = mean(rating - mu))

# compute b_u (user effects) of edx
b_u <- edx %>%
           left_join(b_i, by="movieId") %>%
           group_by(userId) %>%
           summarize(b_u = mean(rating - b_i - mu))

# compute b_g (genres effects) of edx
b_g <- edx %>%
           left_join(b_u, by="userId") %>%
           group_by(genres) %>%
           summarize(b_g = mean(rating - b_u - m_rg))

# compute predicted ratings (mean of ratings + movie effects + user Effects + genres effects)
predicted_ratings <-  final_holdout_test %>%
             left_join(b_i, by = "movieId") %>%
             left_join(b_u, by = "userId")  %>%
             left_join(b_g, by = "genres") %>%
             mutate(pred = mu + b_i + b_u + b_g )  %>%
             pull(pred)

# compute RMSE on final_holdout_test
final_model_rmse_mug <-  RMSE(predicted_ratings, final_holdout_test$rating)

# generate movie+user+genres effects based model final models RMSE table
final_model_rmse_table <- bind_rows(final_model_rmse_table,
                          data_frame(MODEL ="Movie+User+Genres Effects Based Model",
                                     RMSE = final_model_rmse_mug))


# build movie+user+time effects based model: 1) training of edx datasets
```

```r
#                                            2) testing of final_holdout_test sets
#                                            3) compute final model RMSE


# compute mean of ratings of edx
mu <- mean(edx$rating, na.rm=TRUE)

# compute b_i (movie effects) of edx
b_i <- edx %>%
           group_by(movieId) %>%
           summarize(b_i = mean(rating - mu))

# compute b_u (user Effects) of edx
b_u <- edx %>%
           left_join(b_i, by="movieId") %>%
           group_by(userId) %>%
           summarize(b_u = mean(rating - b_i - mu))

# compute b_t (time effects) of edx
b_t <- edx %>%
           left_join(b_u, by="userId") %>%
           group_by(movieId) %>%
           summarize(b_t = mean(rating - b_u - m_rw))

# compute predicted ratings (mean of ratings + movie effects + user effects + time effects)
predicted_ratings <- final_holdout_test %>%
             left_join(b_i, by = "movieId") %>%
             left_join(b_u, by = "userId")  %>%
             left_join(b_t, by = "movieId") %>%
             mutate(pred = mu + b_i + b_u + b_t )  %>%
             pull(pred)

# compute RMSE on final_holdout_test
final_model_rmse_mut <-  RMSE(predicted_ratings, final_holdout_test$rating)

# generate movie+user+time effects based model final models RMSE table
final_model_rmse_table <- bind_rows(final_model_rmse_table,
                          data_frame(MODEL ="Movie+User+Time Effects Based Model",
                                     RMSE = final_model_rmse_mut))


# 1) build regularized movie effects based model: - training of edx datasets and
#                                                 - testing on final_holdout_test sets
#
# 2) apply relevant minimum lambda value on algorithm and compute final model RMSE


# minimum lambda value of regularized movie effects based model from wrangled data
lambd <- min_lambda$movie[1]

# compute mean of ratings of edx
mu <- mean(edx$rating,na.rm=TRUE )
```

```r
# compute b_i (movie effects) of edx
movie_reg_avgs <- edx %>%
                      group_by(movieId) %>%
                      summarize(b_i = sum(rating - mu)/(n()+lambd), n_i = n())

# compute predicted ratings (mean of ratings + movie effects)
predicted_ratings <- final_holdout_test %>%
                          left_join(movie_reg_avgs, by='movieId') %>%
                          mutate(pred = mu + b_i ) %>%
                          pull(pred)

# compute RMSE on final_holdout_test
final_model_rmse_rm <-  RMSE(predicted_ratings, final_holdout_test$rating)

# generate regularized movie effects based model final models RMSE table
final_model_rmse_table <- bind_rows(final_model_rmse_table,
                          data_frame(MODEL ="Regularized Movie Effects Based Model",
                                     RMSE = final_model_rmse_rm))


# lambdas = sequence from 0 to 10 increment by 0.25
lambdas <- seq(0, 10, 0.25)

# qplot - Regularized Movie Effects (RMSES vs Lambdas)
qplot(lambdas, min_lambda$rmses_rm) +
  labs(x="Lambdas",y="RMSES",title="Regularized Movie Effects (RMSES vs Lambdas)")


# 1) build regularized movie+user effects based model:
#
#                     a) training of edx datasets.
#                     b) testing on final_holdout_test sets
#
# 2) apply relevant minimum lambda value on algorithm and compute final model RMSE


# minimum lambda value of regularized movie+user effects based model from wrangled data
lambd <- min_lambda$movie_user[1]

# compute mean of ratings of edx datasets
mu <- mean(edx$rating, na.rm=TRUE)

# compute b_i (movie effects) of edx datasets
b_i <- edx %>%
           group_by(movieId) %>%
           summarize(b_i = sum(rating - mu) /(n()+lambd))

# compute b_u (user effects) of edx datasets
b_u <- edx %>%
          left_join(b_i, by="movieId") %>%
          group_by(userId) %>%
          summarize(b_u = sum(rating - b_i - mu) /(n()+lambd))
```

```r
# compute predicted ratings (mean of ratings + movie effects + user Effects)
predicted_ratings <- final_holdout_test %>%
            left_join(b_i, by = "movieId") %>%
            left_join(b_u, by = "userId") %>%
            mutate(pred = mu + b_i + b_u ) %>%
            pull(pred)

# compute RMSE on final_holdout_test
final_model_rmse_rmu <- RMSE(predicted_ratings, final_holdout_test$rating)

# generate regularized movie+user effects based model final models RMSE table
final_model_rmse_table <- bind_rows(final_model_rmse_table,
                          data_frame(MODEL = "Regularized Movie+User Effects Based Model",
                                     RMSE  = final_model_rmse_rmu))


# lambdas = sequence from 0 to 10 increment by 0.25
lambdas <- seq(0, 10, 0.25)

# qplot regularized movie+user effects (RMSES vs Lambdas)
qplot(lambdas, min_lambda$rmses_riu) +
  labs(y="RMSES",x="Lambdas",title="Regularized Movie+User Effects (RMSES vs Lambdas)")


# 1) build regularized movie+user+genres effects based model:
#
#                     a) training of edx datasets
#                     b) testing on final_holdout_test sets
#
# 2) apply relevant minimum lambda value on algorithm and compute final model RMSE


# minimum lambda value of regularized movie+user+genres effects based model from wrangled data
lambd <- min_lambda$movie_user_genres[1]

# compute mean of ratings of edx
mu <- mean(edx$rating, na.rm=TRUE)

# compute b_i (movie effects) of edx datasets
b_i <- edx %>%
           group_by(movieId) %>%
           summarize(b_i = sum(rating - mu)/(n()+lambd))

# compute b_u (user effects) of edx datasets
b_u <- edx %>%
          left_join(b_i, by="movieId") %>%
          group_by(userId) %>%
          summarize(b_u = sum(rating - b_i - mu)/(n()+lambd))

# compute b_g (genres effects) of edx datasets
b_g <- edx %>%
         left_join(b_u, by="userId") %>%
         group_by(genres) %>%
```

```r
         summarize(b_g = sum(rating - b_u - m_rg)/(n()+lambd))

# compute predicted ratings (mean of ratings + movie effects + user effects + genres effects)
predicted_ratings <-  final_holdout_test %>%
        left_join(b_i, by = "movieId") %>%
        left_join(b_u, by = "userId")  %>%
        left_join(b_g, by = "genres") %>%
        mutate(pred = mu + b_i + b_u + b_g ) %>%
        pull(pred)

# compute RMSE on final_holdout_test
final_model_rmse_rmug <- RMSE(predicted_ratings, final_holdout_test$rating)

# generate regularized movie+user+genres effects based model final models RMSE table
final_model_rmse_table <- bind_rows(final_model_rmse_table,
                          data_frame(MODEL = "Regularized Movie+User+Genres Effects Based Model",
                                     RMSE  = final_model_rmse_rmug))


# lambdas = sequence from 0 to 10 increment by 0.25
lambdas <- seq(0, 10, 0.25)

# qplot regularized movie+user+genres effects (RMSES vs Lambdas)
qplot(lambdas, min_lambda$rmses_rmug) +
  labs(y="RMSES",x="Lambdas",title="Regularized Movie+User+Genres Effects (RMSES vs Lambdas)")


# 1) build regularized movie+user+time effects based model:
#
#                     a) training of edx datasets
#                     b) testing on final_holdout_test sets
#
# 2) apply relevant minimum lambda value on algorithm and compute final model RMSE


# minimum lambda value of regularized movie+user+time effects based model from wrangled data
lambd <- min_lambda$movie_user_time[1]

# compute mean of ratings of edx
mu <- mean(edx$rating, na.rm=TRUE)

# compute b_i (movie effects) of edx datasets
b_i <- edx %>%
          group_by(movieId) %>%
          summarize(b_i = sum(rating - mu)/(n()+lambd))

# compute b_u (user effects) of edx datasets
b_u <- edx %>%
          left_join(b_i, by="movieId") %>%
          group_by(userId) %>%
          summarize(b_u = sum(rating - b_i - mu)/(n()+lambd))

# compute b_t (time effects) of edx datasets
```

```r
b_t <- edx %>%
           left_join(b_u, by="userId") %>%
           group_by(movieId) %>%
           summarize(b_t = sum( rating - b_u - m_rw ) /(n()+lambd))

# compute predicted ratings (mean of ratings + movie Effects + user Effects + time effects)
predicted_ratings <-  final_holdout_test %>%
          left_join(b_i, by = "movieId") %>%
          left_join(b_u, by = "userId")  %>%
          left_join(b_t, by = "movieId") %>%
          mutate(pred = mu + b_i + b_u + b_t ) %>%
          pull(pred)

# compute RMSE on final_holdout_test
final_model_rmse_rmut <- RMSE(predicted_ratings, final_holdout_test$rating)

# generate regularized movie+user+time effects based model final models RMSE table
final_model_rmse_table <- bind_rows(final_model_rmse_table,
                         data_frame(MODEL = "Regularized Movie+User+Time Effects Based Model",
                                    RMSE  = final_model_rmse_rmut))


# lambdas = sequence from 0 to 10 increment by 0.25
lambdas <- seq(0, 10, 0.25)

# qplot regularized movie+user+time effects (RMSES vs Lambdas)
qplot(lambdas, min_lambda$rmses_rmut) +
  labs(y="RMSES",x="Lambdas",title="Regularized Movie+User+Time Effects (RMSES vs Lambdas)")


# 1) build regularized movie+user+time effects based model:
#
#                     a) training of edx datasets
#                     b) testing on final_holdout_test sets
#
# 2) apply relevant minimum lambda value on algorithm and compute final model RMSE


# minimum lambda value of regularized movie+user+time effects based model from wrangled data
lambd <- min_lambda$movie_user_time[1]

# compute mean of ratings of edx datasets
mu <- mean(edx$rating, na.rm=TRUE)

# compute b_i (movie effects) of edx datasets
b_i <- edx %>%
           group_by(movieId) %>%
           summarize(b_i = sum(rating - mu)/(n()+lambd))

# compute b_u (user effects) of edx datasets
b_u <- edx %>%
           left_join(b_i, by="movieId") %>%
           group_by(userId) %>%
```

```r
          summarize(b_u = sum(rating - b_i - mu)/(n()+lambd))

# compute b_t (time effects) of edx datasets
b_t <- edx %>%
          left_join(b_u, by="userId") %>%
          group_by(movieId) %>%
          summarize(b_t = sum( rating - b_u - m_rw ) /(n()+lambd))

# compute predicted ratings (mean of ratings + movie Effects + user Effects + time effects)
predicted_ratings <-  final_holdout_test %>%
         left_join(b_i, by = "movieId") %>%
         left_join(b_u, by = "userId")  %>%
         left_join(b_t, by = "movieId") %>%
         mutate(pred = mu + b_i + b_u + b_t ) %>%
         pull(pred)

# compute RMSE on final_holdout_test
final_model_rmse_rmut <- RMSE(predicted_ratings, final_holdout_test$rating)

# generate regularized movie+user+time effects based model final models RMSE table
final_model_rmse_table <- bind_rows(final_model_rmse_table,
                          data_frame(MODEL = "Regularized Movie+User+Time Effects Based Model",
                                     RMSE  = final_model_rmse_rmut))


####################################################################################
# summary of lambda values that give the minimum RMSE for regularized final models #
####################################################################################


# determine minimum lambdas from wrangled data

min_lambda_m  <- lambdas[which.min(min_lambda$rmses_rm)]

min_lambda_mu  <- lambdas[which.min(min_lambda$rmses_riu)]

min_lambda_mug  <- lambdas[which.min(min_lambda$rmses_rmug)]

min_lambda_mut  <- lambdas[which.min(min_lambda$rmses_rmut)]


# generate data frame of min_lambda_result

min_lambda_result <- data.frame(Effects = "Regularized Movie Effects",
                                Lambdas = min_lambda_m )

min_lambda_result <- bind_rows(min_lambda_result,
                               data.frame(Effects="Regularized Movie+User Effects",
                                          Lambdas=min_lambda_mu))

min_lambda_result <- bind_rows(min_lambda_result,
                               data.frame(Effects="Regularized Movie+User+Genres Effects",
                                          Lambdas=min_lambda_mug))
```

```r
min_lambda_result <- bind_rows(min_lambda_result,
                               data.frame(Effects="Regularized Movie+User+Time Effects",
                                          Lambdas=min_lambda_mut))


# display min_lambda_result table using kable function

min_lambda_result %>% knitr::kable(caption="Lambdas give the Minimun RMSE")


############################################
# summary RMSES of final models/algorithm #
############################################

# display final_model_rmse_table using kable function

final_model_rmse_table %>% knitr::kable(caption="Result of Final Models - RMSE")


##########################################################################
# final regularized movie+user+time effects based model with lowest RMSE #
##########################################################################


# generate data frame of lowest_rmse_model

lowest_rmse_model <-data_frame(
                          MODEL=final_model_rmse_table$MODEL[which.min(final_model_rmse_table$RMSE)],
                          RMSE =final_model_rmse_table$RMSE[which.min(final_model_rmse_table$RMSE)])

# display lowest_rmse_model using kable function

lowest_rmse_model %>% knitr::kable(caption="Final Model with Lowest RMSE")
```

References

1. Harvard Professor "Rafael A. Irizarry".(2019). INTRODUCTION TO DATA SCIENCE.
2. https://rafalab.dfci.harvard.edu/dsbook/
3. https://rafalab.github.io/dsbook/
4. https://topepo.github.io/caret/available-models.html
5. https://topepo.github.io/caret/train-models-by-tag.html
6. Steve,L.(2009). Netflix Awards $1Million Prize and Starts a New Contest. Bits. The New York Times.
7. https://bits.blogs.nytimes.com/2009/09/21/netflix-awards-1-million-prize-and-starts-a-mew-contest/
8. https://blog.echen.me/2011/10/24/winning-the-netflix-prize-a-summary/
9. https://www.netflixprize.com/assets/GrandPrize2009_BPC_BellKor.pdf
10. Comprehensive R Archive Network. https://cran.r-project.org/doc/manuals/r-release/R-lang.html
11. RDocumentation. https://www.rdocumentation.org
12. Microsoft. https://copilot.microsoft.com
13. Kaggle. https://www.kaggle.com
14. UCI Machine Learning Repository. https://archive.ics.uci.edu
15. Stack Overflow. https://stackoverflow.com
16. Statology. https://www.statology.org

Acknowledgements

I would like to express my gratitude to Harvard Professor “Rafael Irizarry” for this excellence in teaching. His expertise and insights were instrumental in shaping the direction of my learning in this Data Science Professional Certificate program.

Additionally, I also thanks all Teaching Assistants of this program at Harvard University for providing necessary feedback and assistance for my study. Special thanks to “Sapei” and Friends for their spiritual support.

Lastly, I extend my heartfelt thanks to my wife “Jess” for her unwavering understanding during the course of my professional study.