MovieLens - Movie Recommendation System

E. Willmott

07/16/2021

Contents

Please note: *The formatting of the document leaves something to be desired, but the formatting issues are unavoidable using R Markdown alone. Said formatting issues can be resolved by incorporating CSS (Cascading Style Sheets) in the Rmd file. As CSS is not taught in this program or a prerequisite, I have not incorporated any CSS into the Rmd file that produces this report.*

1. Introduction/Overview

This section describes the *dataset* used and summarizes the *goal* of the project and *key steps* performed.

1.1 Dataset - The MovieLens Dataset

1.1.1 The entire latest MovieLens Dataset The entire latest ***MovieLens dataset*** can be found ***Here*** (Harper and Konstan, 2015)

> "***GroupLens Research*** has collected and made available rating data sets from the **MovieLens web site**." (GroupLens, 2021a)

> "***GroupLens Research*** operates a movie recommender based on collaborative filtering, MovieLens, which is the source of these data." (GroupLens, 2009)

> "***GroupLens*** is a research lab in the Department of Computer Science and Engineering at the University of Minnesota, Twin Cities specializing in recommender systems, online communities, mobile and ubiquitous technologies, digital libraries, and local geographic information systems." (GroupLens, 2021b)

The entire MovieLens dataset includes about 27,000,000 ratings and about 1,100,000 tag applications applied to about 58,000 movies by about 280,000 users. As of the time of writing this report (7/2021), the entire MovieLens dataset was last updated 9/2018 (GroupLens, 2021c) The *README file* associated with this dataset states that –

> "This dataset describes 5-star rating and free-text tagging activity from MovieLens, a movie recommendation service. It contains 27,753,444 ratings and 1,108,997 tag applications across 58,098 movies. These data were created by 283,228 users between January 09, 1995 and September 26, 2018. This dataset was generated on September 26, 2018."

> "**Ratings are made on a 5-star scale, with half-star increments (0.5 stars - 5.0 stars).**" (GroupLens, 2018)

The *README file* for the entire latest ***MovieLens dataset*** can be accessed ***Here***.

The dataset used in this project is the ***10M version of the MovieLens dataset*** (Harper and Konstan, 2015)

1.1.2 The MovieLens 10M Dataset *(used in this project)*
The MovieLens 10M Dataset contains about 10 million ratings and about 100,000 tag applications applied to about 10,000 movies by about 72,000 users. It was released on 1/2009 (GroupLens, 2021d)

The *README file* associated with this dataset states that –

> "This data set contains 10000054 ratings and 95580 tags applied to 10681 movies by 71567 users of the online movie recommender service MovieLens." (GroupLens, 2009)

The README file of the 10M version of the MovieLens dataset can be accessed ***Here***.

The 10M version of the MovieLens dataset was downloaded from ***Here***.

The MovieLens 10M Dataset includes a **zip file** which contains three data files (and some *scrips* for generating subsets, not used in this project):

- **movies.dat**
- **ratings.dat**
- **tags.dat**

ratings.dat contains the ratings for each combination of a certain user (represented by a unique ***userId***), and a certain movie (represented by a unique ***movieId***). Each rating by a certain user to a certain movie also contains a ***timestamp*** marking the time a rating was given in seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970. Ratings are made on a 5-star scale, with half-star increments (GroupLens, 2009)

tags.dat contains a tag applied by a certain user to a certain movie and a timestamp (GroupLens, 2009) *tags.dat* was not used in this project.

movies.dat contains information about the movies, including a ***movieId***, a ***title*** and ***genres*** for each movie (GroupLens, 2009) The ***titles*** were not used by my model.

1.1.3 Creation of training (edx) and hold-out (validation) subsets

The edx and validation subsets were created from the **MovieLens 10M Dataset**.

Further to *data cleaning* and *data wrangling*, two datasets were created using the function *createDataPartition*, with 90% of the original data included in the edx dataset used for training the algorithm, and 10% of the original dataset included in the validation test used for testing the final model after the training was completed on the edx training set.

The **edx** dataset included all *users* and all *movies* that were included in the **validation** set. Thus, the **edx** dataset included all *users* and all *movies* included in the **MovieLens 10M Dataset**. This is an important feature of the **edx** dataset that enables making predictions for any *user/movie combination* included in the **MovieLens 10M Dataset**.

After wrangling, the **edx** and **validation** sets each included the following six columns:

- ***userId***
- ***movieId***
- ***rating***
- ***timestamp***
- ***title***
- ***genres***

Each row represented a rating given by a unique user (represented by ***userId***) to a unique movie (represented by ***movieId***) at a certain time (represented by ***timestamp***). Further information included a ***title*** (*not used by my model*) and a list of ***genres*** ascribed to each unique movie.

The table below presents the first six rows of the **edx** dataset. It presents six different *movies* that were all rated by the same *user* (*userId* = **1**). All of these movies received a rating = **5** from this particular *user*. The table also presents the *timestamp*, representing the time a rating was given in seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970. The table presents the *title* of each of these movies, and the *genre combination* ascribed to each one of them.

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

The edx dataset –

- Contained **9000055** rows, representing ratings given by a unique *user* to a unique *movie* (user/movie combinations).
- Included **10677** different *movies*.
- Included ratings by **69878** different *users*.

1.2 Goal

To create a movie recommendation system using the 10M version of the MovieLens dataset.

This included:

- Training an algorithm on the **edx** dataset in order to predict *movie ratings* in the **validation** set.
- Testing the performance of this algorithm on the validation set using RMSE (*Root Mean Squared Error*).

The data science book by prof. Irizarry states with regard to recommendation systems:

> "Recommendation systems use ratings that users have given items to make specific recommendations. … Items for which a high rating is predicted for a given user are then recommended to that user."

> "Netflix uses a recommendation system to predict how many stars a user will give a specific movie. One star suggests it is not a good movie, whereas five stars suggests it is an excellent movie." (Irizarry, 2019)

A movie recommendation system recommends *movies* to *users* who have not watched them yet based on a *prediction model*. My model described below predicted a rating for a given *user/movie combination* using effects that were computed based on the **edx** dataset. A movie recommendation system would recommend a certain movie to a certain user who has not watched it yet if the ***predicted rating*** this user would give such a movie was greater than a certain value, i.e. **4.0 stars**. For the purpose of this project, predictions of movie ratings were made for the **validation** set as if its ratings were unknown. The **edx** dataset was used to train the prediction algorithm and build a *prediction model*. **RMSE** was calculated to estimate the **model performance**. For most *user/movie combinations*, a rating does not exist in the **edx** dataset. The task can be seen as filling the blank dots in a *user/movie matrix* like the ones shown in the **data exploration and visualization** section below. **NAs** represent *user/movie combinations* that do not have a rating for them in the **edx** dataset. These *NAs* would be filled with a predicted rating based on the prediction model. Then, movies would be recommended to users that have not watched them yet, if the prediction model predicted for them a rating above a certain value, i.e. **4.0 stars**.

1.3 Key Steps

1. Download of the **10M version of the MovieLens dataset**.
2. Creation of a training subset (**edx**) and a **validation** subset based on the dataset that was downloaded.
3. Training a machine learning algorithm using the inputs in one subset (a train set carved out of the **edx** set) to predict movie ratings in a test subset (carved out of the **edx** set).
4. Testing the final algorithm by predicting movie ratings in the **validation** set (the final hold-out test set).
5. Using **RMSE** (*Root Mean Squared Error*) to evaluate how close my *predictions* were to the *true values* in the **validation** set (the final hold-out test set).

2. Methods/Analysis

This section explains the *process* and *techniques* used, including *data cleaning*, *data exploration and visualization*, *insights gained*, and the *modeling approach.*

2.1 Process and Techniques Used

2.1.1 Data Cleaning

The edx and validation sets were created from the **MovieLens 10M Dataset**. This was done using code that was provided as part of this project.

The following steps were performed:

- Download of a ***zip file*** from the ***MovieLens website***.
- Unzipping this file and reading the information from the unzipped files *ratings.dat* and *movies.dat* separately using the function *readLines()*.
- Replacing all occurrences of "`::`" with "`\t`" for a *ratings* object that was obtained from the *ratings.dat* file.
- Splitting the strings in a *movies* object that was obtained from the *movies.dat* file. The strings were split at the pattern "`\::`", to three pieces.
- Naming the columns of the *ratings* object - **userId**, **movieId**, **rating**, and **timestamp**.
- Naming the columns of the *movies* object - **movieId**, **title**, and **genres**.
- Converting the *movies* object into a data frame.
- Converting the columns of the *movies* object as follows: **movieId** to a *numerical* vector, **title** and **genres** to *character* vectors.
- Creation of an object ***movielens*** that was created by joining the *ratings* object with the *movies* object, matching by ***movieId***.

- Creation of two subsets of the **movielens** object with the function *createDataPartition*. The **edx** subset contained 90% of the data, and the **validation** subset contained 10% of the data.
- Removal of rows from the **validation** subset that had a ***movieId*** or a ***userId*** that were not included in the **edx** subset. These rows were then added back into the **edx** subset.
- Removal of temporary objects that were created during the process. At this stage, the only two objects left were the **edx** and the **validation** subsets.

I then removed the ***title*** column from all subsets. The ***title*** was not used by my model. It was removed in order to free space and limit the size of the datasets.

The **edx** dataset was tested, as follows –

Ratings

The ratings were supposed to be "**made on a 5-star scale, with half-star increments (0.5 stars - 5.0 stars)**" (GroupLens, 2018) I tested that no ratings of "**0**" were included in the **edx** dataset.

```
sum(edx$rating == 0)
```

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

There were no ratings in the edx dataset that were equal to ***zero***.

I made sure that all ratings were between *0.5* and *5*, as they should have been:

```
sum(edx$rating < 0.5 | edx$rating > 5)
```

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

All ratings in the **edx** dataset were between **0.5** and **5**.

All Values in the edx dataset

```
sum(is.na(edx))
```

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

There were no values that were NA (missing values) in the ***edx*** *dataset.*

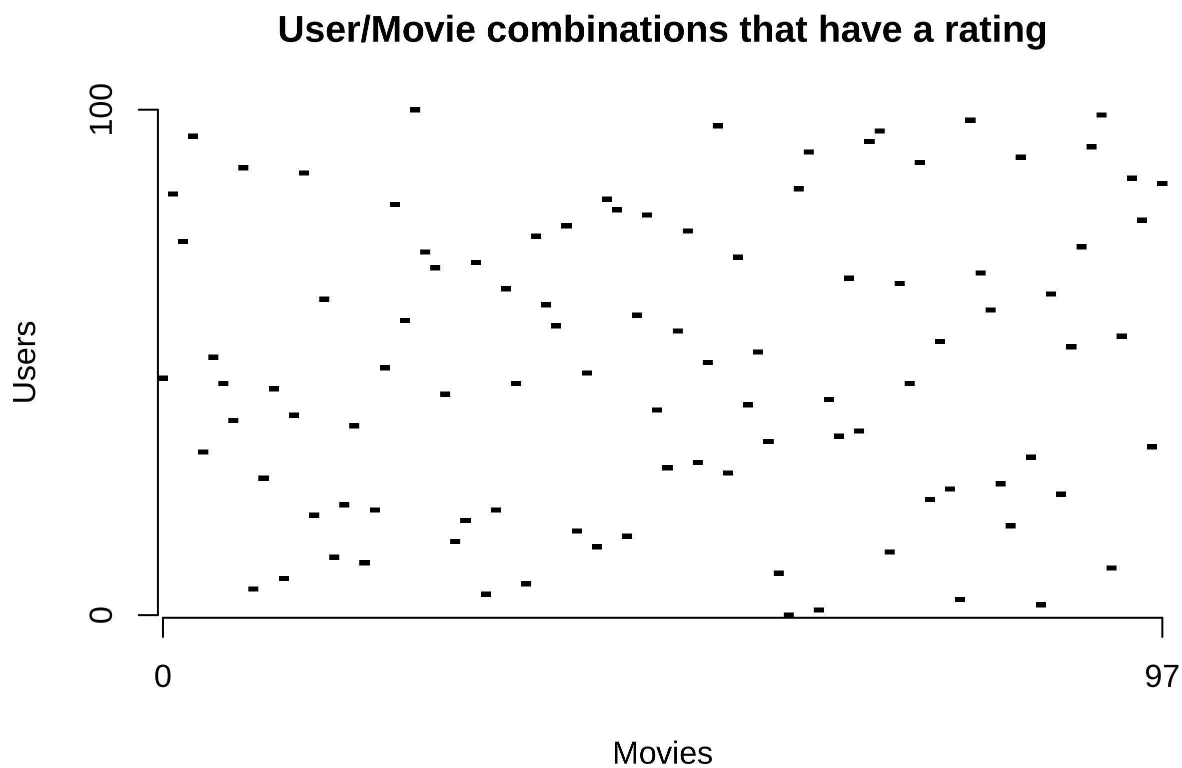
2.1.2 Data Exploration and Visualization

2.1.2.1 User/Movie Matrix

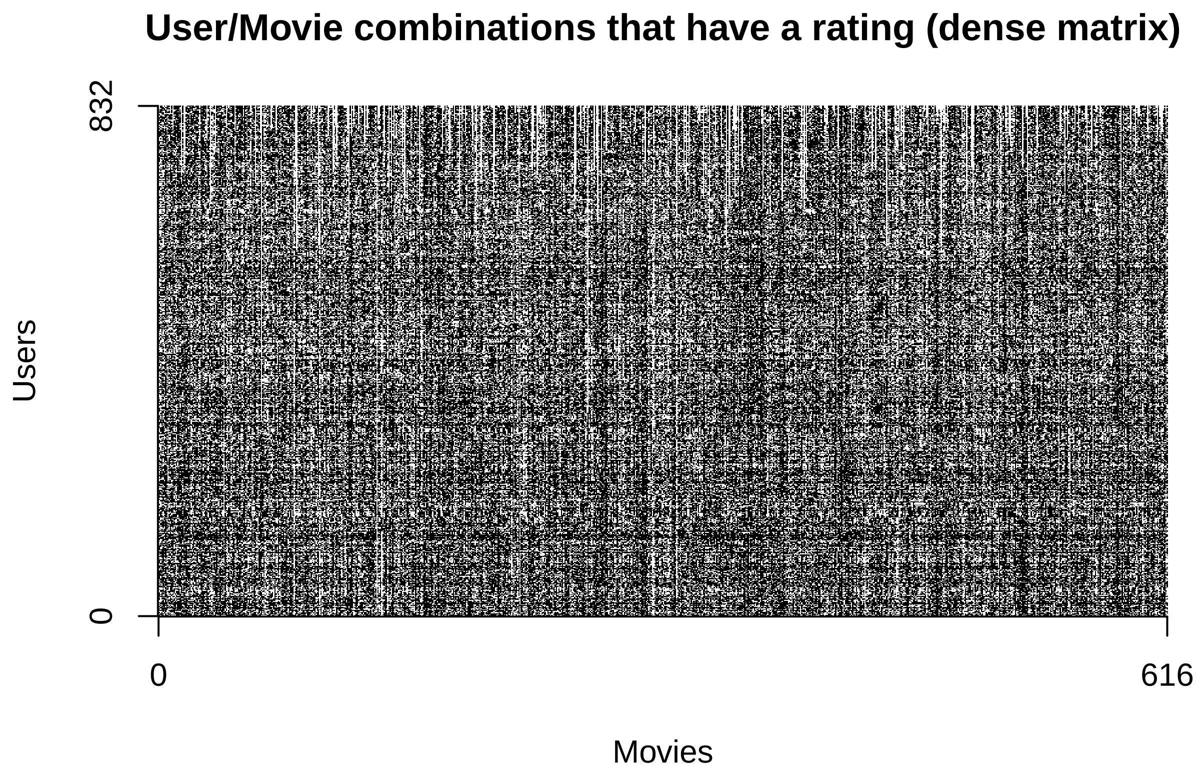
For observing the data, I created two small subsets of the **edx** dataset. These subsets were only used to visualize the data below and were not used in any other stage of the analysis. Using these small subsets, I created a *user/movie matrix* with ***users*** in the rows, ***movies*** in the columns, and ***ratings*** in the cells. I transformed all *ratings* to "**0**" and "**1**" - "**0**" for *no rating exists* and "**1**" for *rating exists*. Each dark dot in the images below represents a *user/movie combination* for which a *rating exists* in the **edx** dataset. The blank dots represent *user/movie combinations* for which *no rating exists* in the **edx** dataset. The matrices below are very sparse. For most *user/movie combinations*, no rating exists in the **edx** dataset. Each *user* rated a different set of *movies* and each *movie* was rated by a different set of *users*.

For the first demonstration, I filtered the **edx** dataset to include only *movies* with more than **50** ratings and only *users* that have rated more than **50** movies. I then selected **100** ratings by random. I ended up with **100** random users, and **97** random movies.

The *users* are on the rows, and the *movies* are on the columns. You can see how sparse the matrix is for these **100** random users and **97** random movies.



For the second demonstration, I filtered the **edx** dataset to include only *movies* that were rated more than **400** times, and only *users* that have rated more than **900** *movies*. I ended up with **832** *users* and **616** *movies*. Naturally, the matrix was less sparse now since **edx** has been filtered to include only *users* with many ratings and *movies* with many ratings. The image below presents this matrix, with black marking *user/movie combinations* for which a *rating exists* in the **edx** dataset. It can be seen that even with *users* and *movies* that have many *ratings*, still some dots are missing, representing *user/movie combinations* for which *no rating exists* in the **edx** dataset. The *image* shows the **832** most active *users* and **616** most rated *movies*. It shows that the most active *users* rated a similarly high number of the most rated *movies*. It also shows that even for the most active *users* and the most rated *movies*, not all *movies* were rated by all *users* (note the blank dots in the image).



The **goal** of this project was **to create a movie recommendation system using the 10M version of the MovieLens dataset**. This goal could be seen as filling the *NAs* in a *user/movie matrix* like the ones shown above. Once I have predictions for *user/movie combinations* that do not have a rating in the **edx** dataset, it is possible to create *recommendations* based on *predicted ratings* higher than a certain value (i.e., **4.0 stars**).

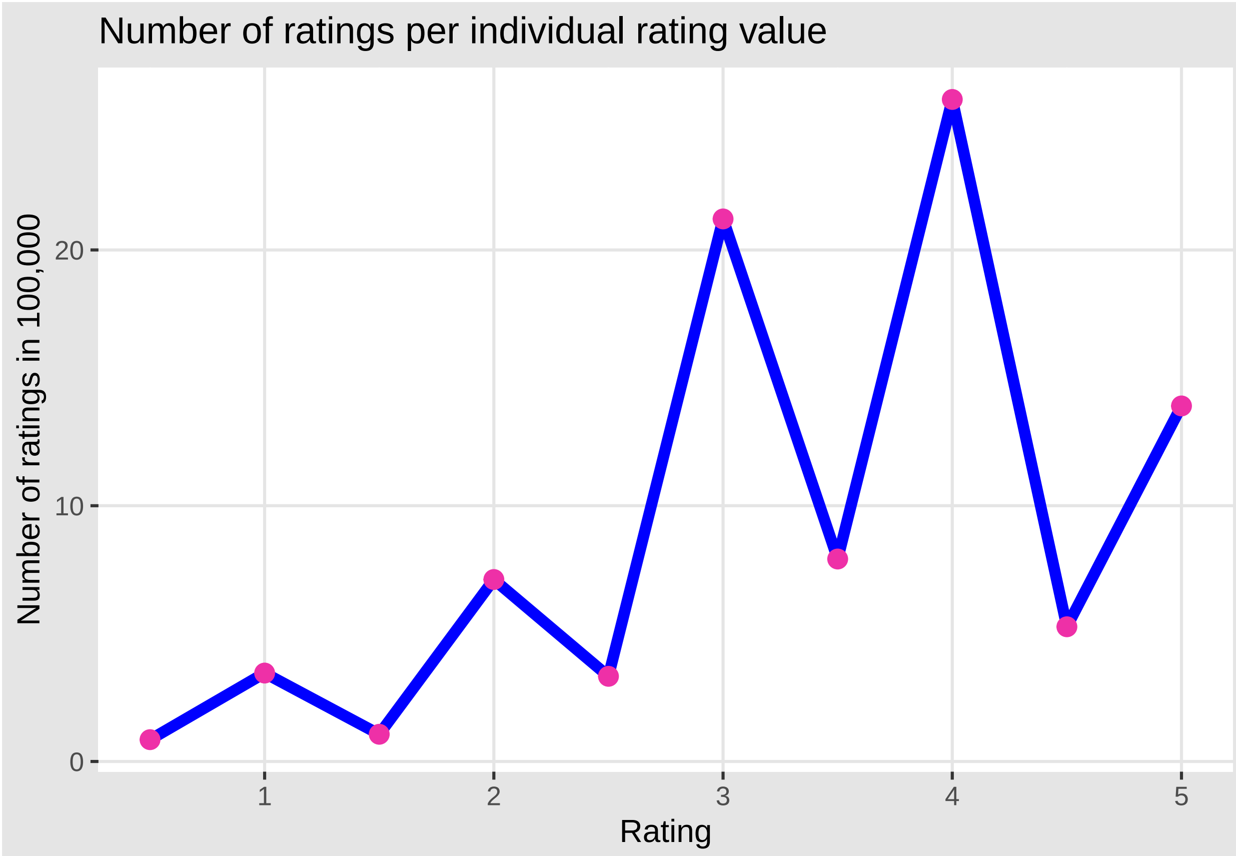
2.1.2.2 **Ratings**

Exploring different ratings given over ***all*** movies and ***all*** users in the **edx** dataset:

The table below presents the number of ratings for each rating value.

The ***graph*** below presents the number of ratings for each rating value.

Rating Value	Number of Ratings
0.5	85374
1	345679
1.5	106426
2	711422
2.5	333010
3	2121240
3.5	791624
4	2588430
4.5	526736
5	1390114



The most common rating (*the mode*) was **4**. Half-star ratings were less common than whole ratings. Also, bigger ratings (i.e., 3, 4, 5) were more common than smaller ratings (i.e., 1, 2).

The table below presents a summary of the ratings in the **edx** dataset. "***n***" represents number of ratings.

rating	n
Min. :0	Min. : 85374
1st Qu.:2	1st Qu.: 336177
Median :3	Median : 619079
Mean :3	Mean : 900006
3rd Qu.:4	3rd Qu.:1240492
Max. :5	Max. :2588430

2.1.2.3 **Movie effect**

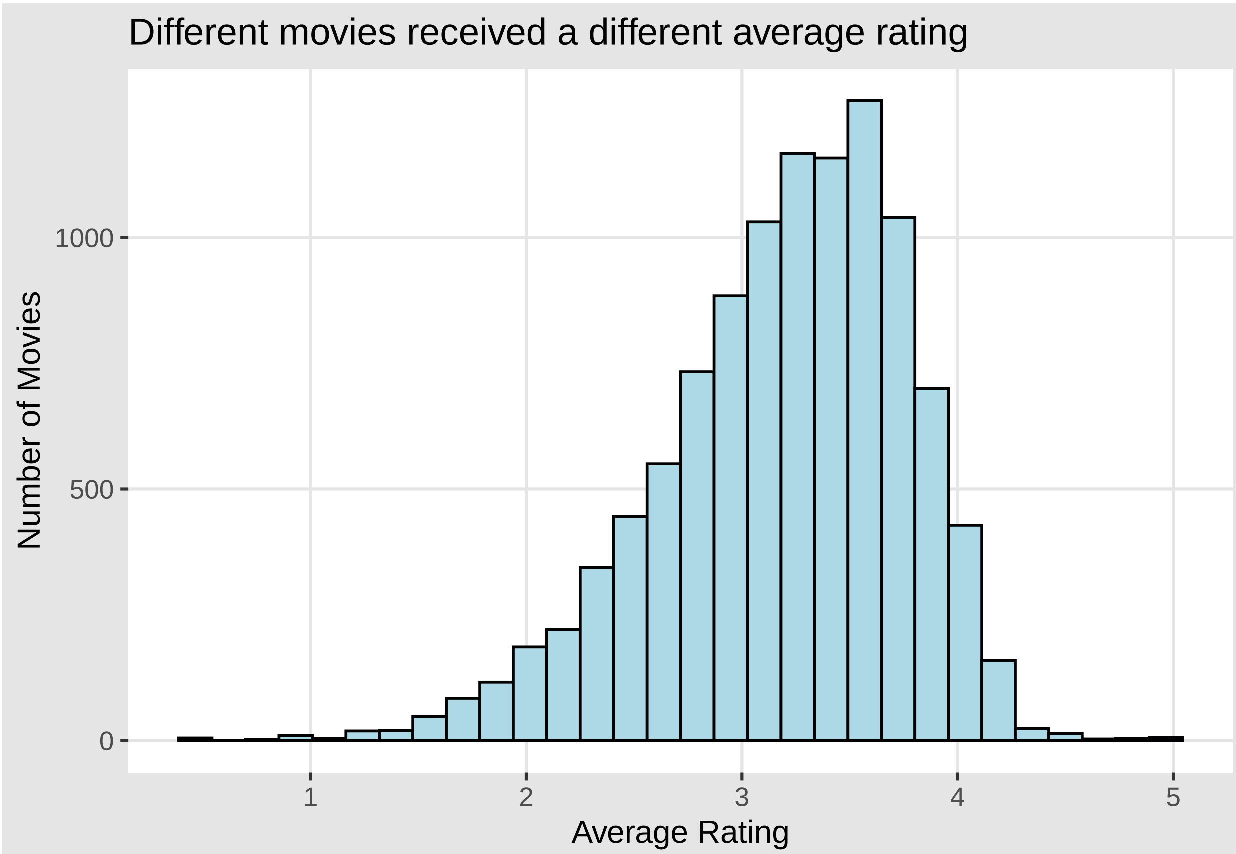
The table below presents a summary of the number of ratings given to different movies. "***n***" represents number of ratings.

movieId	n
Min. : 1	Min. : 1
1st Qu.: 2754	1st Qu.: 30
Median : 5434	Median : 122
Mean :13105	Mean : 843
3rd Qu.: 8710	3rd Qu.: 565

movieId	n
Max. :65133	Max. :31362

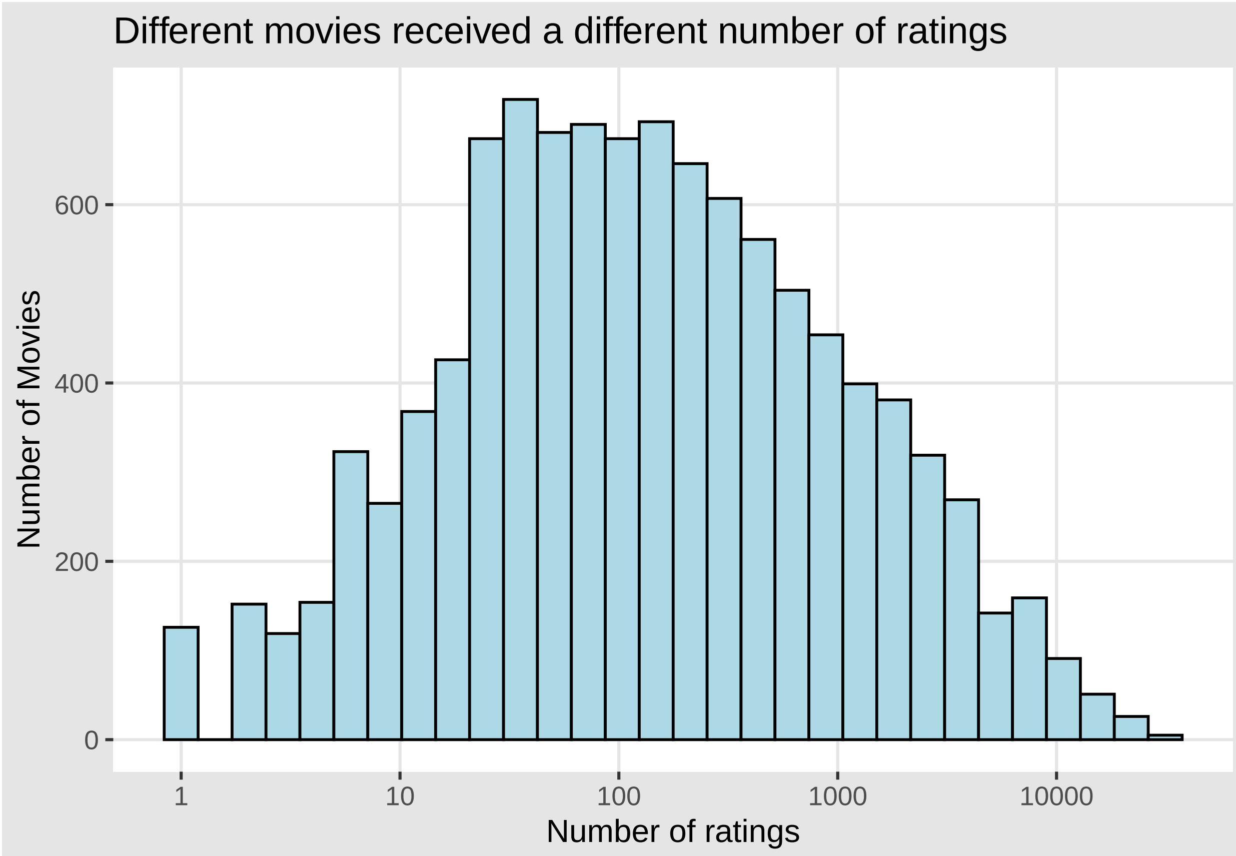
Different movies received a different number of ratings.

The ***histogram*** below shows that different *movies* were rated differently. They had a different ***average rating***. This was considered a ***movie effect*** and was included in the model.



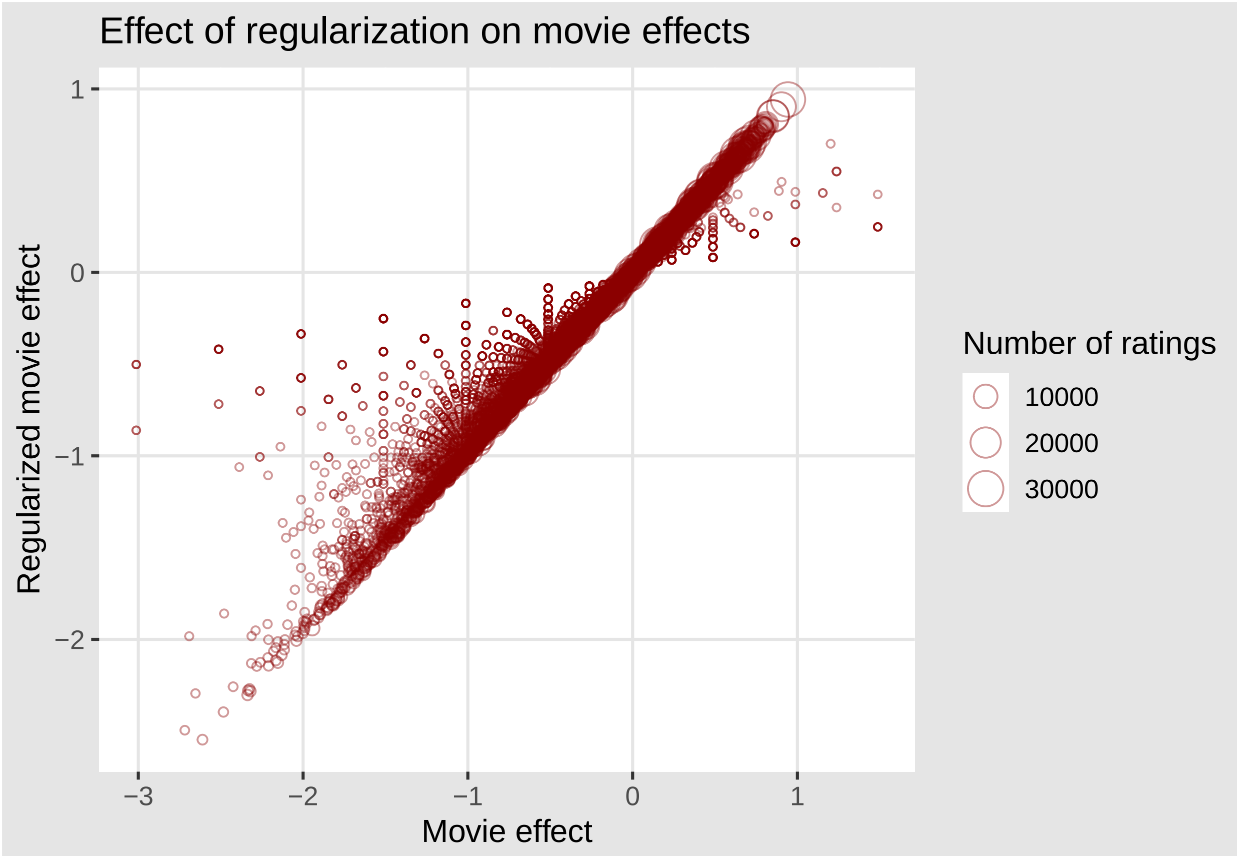
2.1.2.4 **Regularization**

The ***histogram*** below shows that some *movies* received many ratings, while other *movies* received very few ratings. Please note the *log scale* on the x-axis.



Regularization was applied to the *movie effects* (and to all other *effects* later included in the model), as explained in the ***modeling approach*** section below.

The ***graph*** below presents the effects of ***regularization*** on movie effect ***b__i***. It shows the ***b__i*** *movie effect* after regularization, *Vs* the ***b__i*** *movie effect* with no regularization. The *size* of the points represents the ***number of ratings*** given to a certain movie. Each point represents a single movie, with ***b__i without regularization*** on the x-axis and ***b__i with regularization*** on the y-axis. The graph shows that ***regularization*** affected more the ***smaller points*** - movies with less ***ratings***.



2.1.2.5 **Cross-validation**

The process of ***cross-validation*** is explained at the ***modeling approach*** section below.

The results of cross-validation conducted to select the optimal *lambda* (*"optimal"* meaning - the *lambda* that obtained the smallest **RMSE** for the model on the *cross-validation test set*) appear below.

The Optimal Lambda
5

2.1.2.6 **User Effect**

The table below presents a summary of the number of ratings given by different users. "***n***" represents number of ratings.

userId	n
Min. : 1	Min. : 10
1st Qu.:17943	1st Qu.: 32
Median :35798	Median : 62
Mean :35782	Mean : 129
3rd Qu.:53620	3rd Qu.: 141
Max. :71567	Max. :6616

The ***histogram*** below presents the *average residual* left after removing the overall average ***mu*** and the *movie effect* ***b__i*** from each rating, for different *users*. It can be observed that different users had different rating habits. Note the *log scale* of the y-axis.

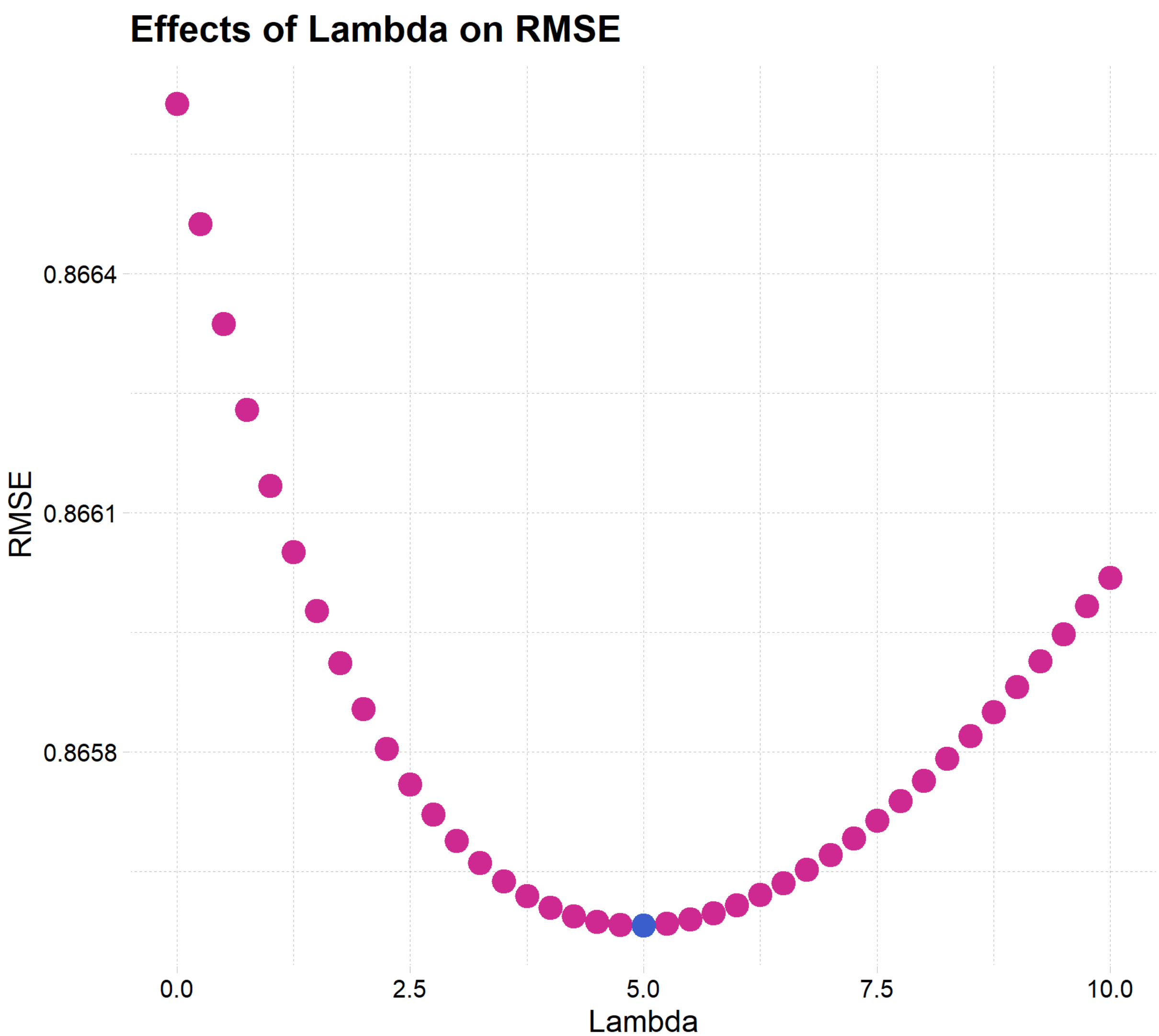
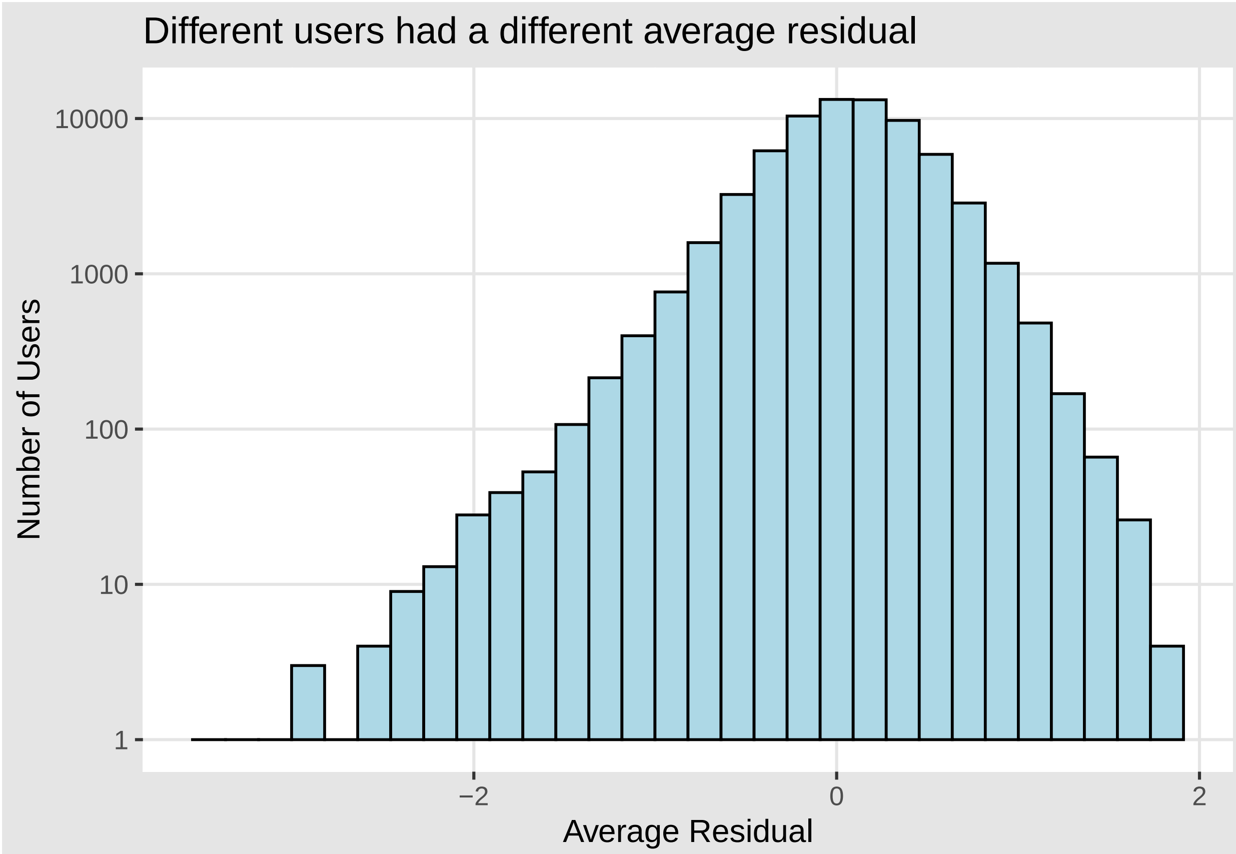


Figure 1: RMSE Vs Lambda



2.1.2.7 **Genre Combination Effect**

The **MovieLens dataset** includes a list of ***genres*** for each movie that is represented by a unique ***movieId***. The *README file* of the entire MovieLens dataset states that –

> "Genres are a pipe-separated list, and are selected from the following:

```
Action
Adventure
Animation
Children's
Comedy
Crime
Documentary
Drama
Fantasy
Film-Noir
Horror
Musical
Mystery
Romance
Sci-Fi
Thriller
War
Western
(no genres listed)"
```

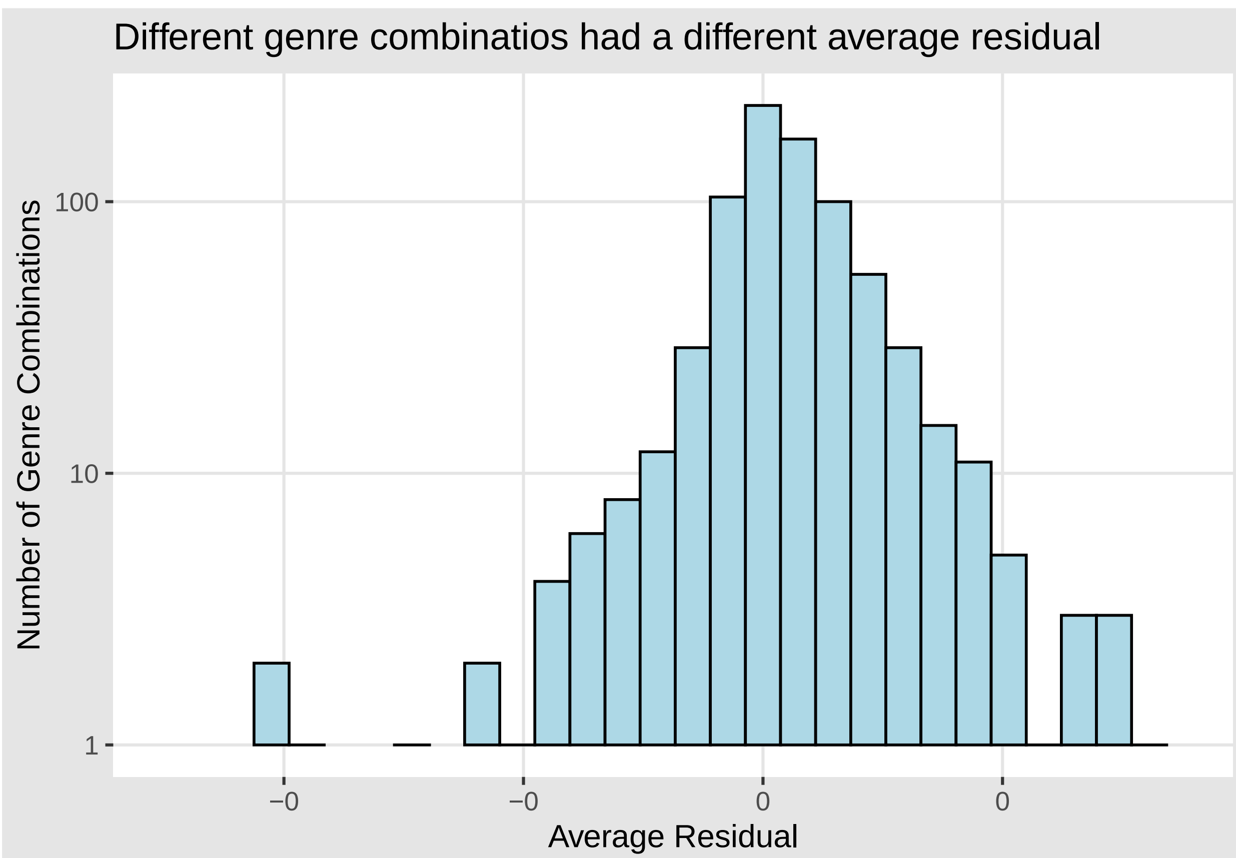
(GroupLens, 2018)

A total of **18** different *genres* were included in the dataset, as well as a category of *no genre listed*. The model included *genre combinations* ascribed to at least one movie in the **edx** dataset, rather than including each *genre* separately. Rather than **18** genres (+ 1 category of *no genres listed*), the **edx** dataset had **797** different *genre combinations*. An example of a *genre combination* is *Comedy|Romance*.

The table below presents a summary of the number of ratings given for different *genre combinations*. "***n***" represents number of ratings.

genres	n
Length:797	Min. : 2
Class :character	1st Qu.: 185
Mode :character	Median : 1459
NA	Mean : 11292
NA	3rd Qu.: 7167
NA	Max. :733296

The ***histogram*** below presents the *average residual* left after removing the overall average ***mu***, the *movie effect* ***b_i***, and the *user effect* ***b_u*** from each rating, for different *genre combinations*. Different *genre combinations* had different *residuals*. Note the *log scale* on the y-axis.



Different genre combinations had different residuals.

2.1.2.8 **Time Effects**

A column *date* was added to the dataset based on the ***timestamp*** column. The total range of dates in the ***edx*** dataset was {1995-01-09 11:46:49, 2009-01-05 05:02:16}. The different time effects, *week*, *day-of-year* and *year*, are shown in the graphs below.

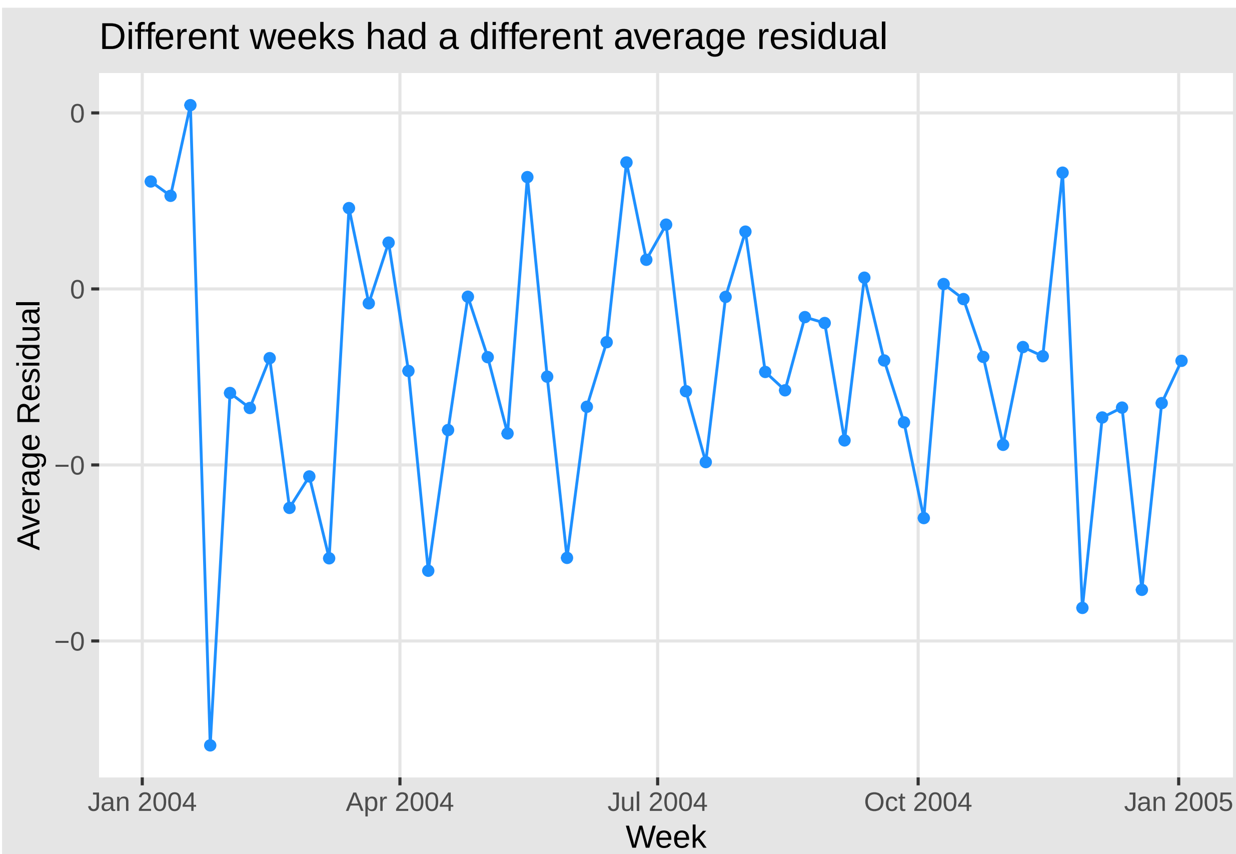
2.1.2.8.1 **Week Effect**

There were a total of **671** *different weeks* in the **edx** dataset, for the time epoch listed above. The *week* effect was computed separately for each of these **671** *weeks*.

The table below presents a summary of the number of ratings for different *weeks*. "***n***" represents number of ratings.

week	n
Min. :1995-01-08 00:00:00	Min. : 2
1st Qu.:1999-05-19 12:00:00	1st Qu.: 8634
Median :2002-08-04 00:00:00	Median : 11849
Mean :2002-07-29 11:48:11	Mean : 13413
3rd Qu.:2005-10-19 12:00:00	3rd Qu.: 15562
Max. :2009-01-04 00:00:00	Max. :133343

The ***graph*** below presents the effect of *week* on the *average residual* left after removing ***mu***, ***b__i***, ***b__u***, and ***b__g***, from each rating. To ease the viewing, the graph shows the *week* effect *only* for the year **2004**.



The graph shows a clear effect of *week* on the residuals left after removing previous effects from the ratings.

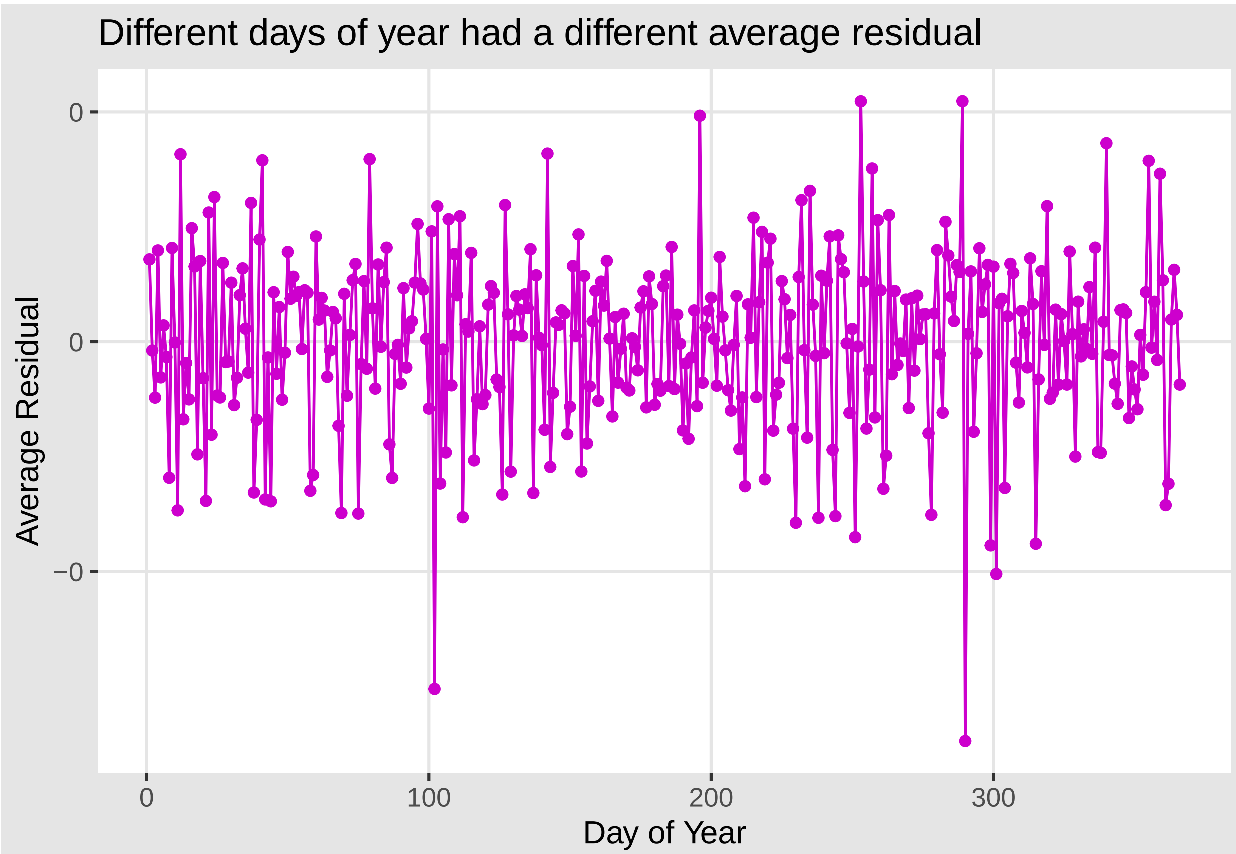
2.1.2.8.2 **Day-of-Year Effect**

Day of the year (*day-of-year*) is a number between **1** and **366** (applying also to *leap years*). January 1 is *day* 1.

The table below presents a summary of the number of ratings for different *days-of-year*. "***n***" represents number of ratings.

day	n
Min. : 1	Min. : 8043
1st Qu.: 92	1st Qu.:20139
Median :184	Median :23332
Mean :184	Mean :24590
3rd Qu.:275	3rd Qu.:26497
Max. :366	Max. :80899

The ***graph*** below presents the effect of *day-of-year* on the *average residual* left after removing ***mu***, ***b__i***, ***b__u***, and ***b__g***, as well the *week effect*, ***b__w***, from each rating. Note in particular the *residuals* for days **102** and **290**. These two days correspond to the **12$^{\text{th}}$ of April** and the **17$^{\text{th}}$ of October** respectively, on a *non-leap year*.



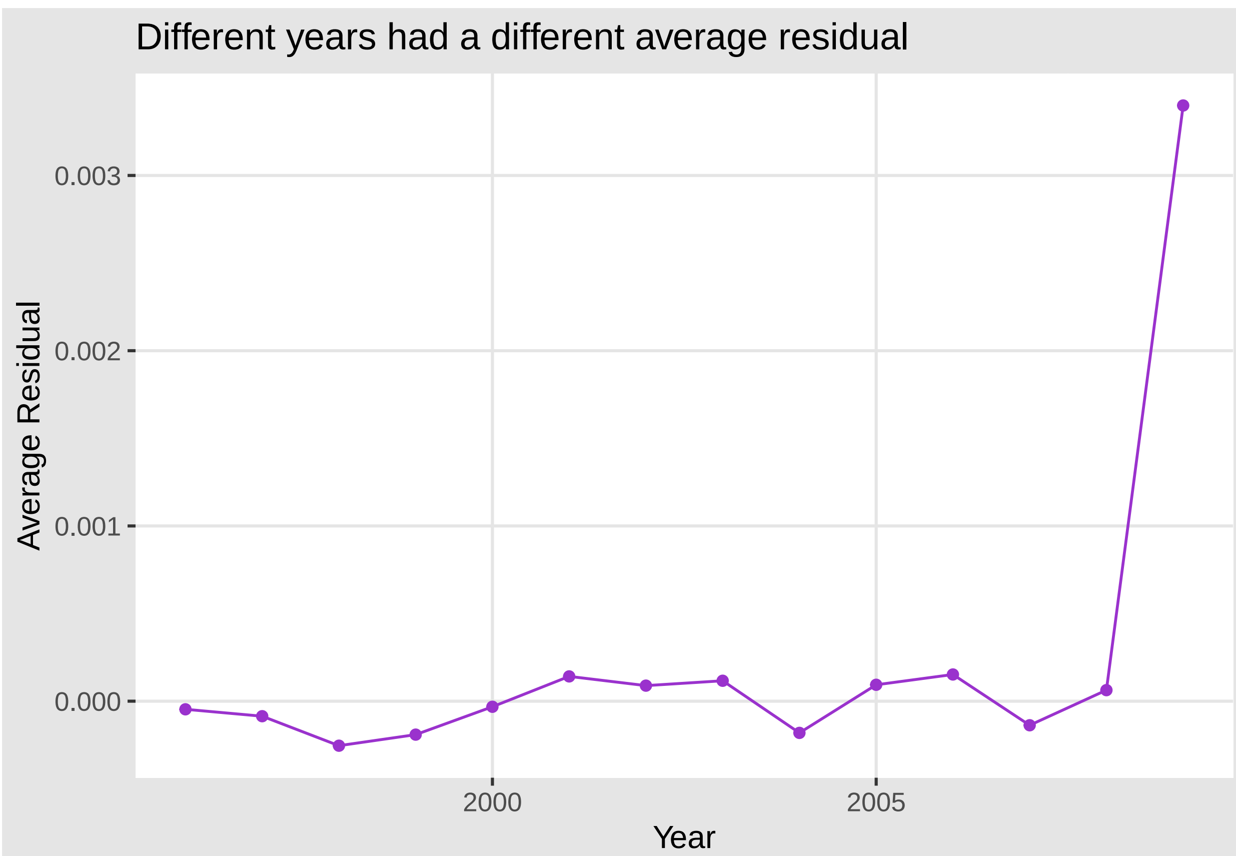
2.1.2.8.3 **Year Effect**

There was a total of **15** *years* in the **edx** dataset.

The table below presents a summary of the number of ratings for different *years*. "***n***" represents number of ratings.

year	n
Min. :1995	Min. : 2
1st Qu.:1998	1st Qu.: 469530
Median :2002	Median : 683355
Mean :2002	Mean : 600004
3rd Qu.:2006	3rd Qu.: 703316
Max. :2009	Max. :1144349

The ***graph*** below presents the effect of *year* on the *average residual* left after removing ***mu***, ***b__i***, ***b__u***, ***b__g***, and ***b__w***, as well as the *day-of-year* effect, ***b__d***, from each rating. *The year 1995, the first year for which ratings were obtained, was removed from this graph.* There were only **2** ratings obtained for that year. The *average residual* for *that year* was much greater than for *other years*. In order to allow viewing the effects of other years on the same scale, the year **1995** is not shown in the graph below. Also note that since *that year* had only **2** ratings, *regularization* with lambda = **5** could reduce significantly the effect computed for that year. Note also that there are only **13123** ratings included for the last year in the **edx** dataset, **2009**. This is in comparison with a mean number of **600004** ratings for all years in the **edx** dataset, as shown in a table at the **data exploration and visualization** section above. This explains the much greater *average residual* for this year (2009) compared to other years presented in the graph. *Regularization* with a much greater *lambda* could reduce effects obtained from that year as well, as is suggested for future work.



2.1.3 Insights Gained

Data exploration and visualization revealed the following effects on the *ratings*: *movie* effect, *user* effect, *genre combination* effect, *week* effect, *day-of-year* effect, and *year* effect. Also, the value of *regularization* was recognized, and *regularization* with the same *lambda* was applied to all these effects. *Lambda* was selected through a process of *cross-validation* on *cross-validation train and test sets*. It was illuminating to be able to identify and observe the different effects influencing the ratings.

It was transforming to observe that even after many effects had already been included in the model, it was still possible to identify new effects influencing the residuals (the value left after removing all previous effects from each rating in the train set).

It was especially illuminating to observe the different *time* effects, that were indeed different from each other. The trend of each of these time effects was different than the other two *time* effects. These effects can be seen in the graphs showing *time effects* above.

The *year* effect was particularly interesting, where the first year of rating, **1995**, had only **2** ratings overall. The effect computed for that year was much greater than effects computed for any other given year. This demonstrated the importance of *regularization*. **Data exploration and visualization** also demonstrated the importance of *lambda selection* for regularization. The *optimal lambda* selected for regularization could depend on the random *cross-validation train and test sets* used. As future work, potentially different *optimal lambdas* could be selected for different random *cross-validation train and test sets*. Then, a *mean optimal lambda* could be selected and used for the model.

Another insight gained was enhancing my understanding of the difference between an *apparent error* and a *true error*. A random partition of the data to subsets could affect the RMSE calculated on the test set. This could be resolved through *cross-validation*. Partitioning the dataset multiple times to multiple train and test sets, then using a mean RMSE over these random train and test subsets as an estimate of the *true error*. This could be done as part of future work.

Working on this project was inspiring, and taught me skills of building, developing and organizing a data science project, as well as producing reports. My R programming skills have improved. The process of identifying new features for a machine learning algorithm was illuminating.

2.1.4 The Modeling Approach

The *modeling approach* was based on insights gained through **data exploration and visualization**. These insights led to a modeling approach that accounted for six different effects. At first, the overall average rating ***mu*** was ascribed to each *user/movie combination*. Then, different effects were added to each *user/movie combination*. Some of the effects were based on the **time** when a certain rating was given, based on the ***timestamp*** column. The effects included ***movie***, ***user***, ***genre combination***, ***week***, ***day-of-year***, and ***year***. Predictions were made for the **test** set and then for the **validation** set using the following formula:

Predicted = mu + b_i + b_u + b_g + b_w + b_d + b_y

with –

- ***mu*** - *overall average rating* for all movies and all users
- ***b_i*** - *movie effect*
- ***b_u*** - *user effect*
- ***b_g*** - *genre combination effect*
- ***b_w*** - *week effect*
- ***b_d*** - *day-of-year effect*
- ***b_y*** - *year effect*

All effects were regularized using the same **lambda = 5**. The optimal *lambda* for regularizing all effects was selected in a process of *cross-validation* on *cross-validation train and test sets*.

Effects were computed using the **train** set to make predictions for the **test** set, and using the **edx** dataset to make predictions for the **validation** set.

The **modeling approach** included the steps described below.

2.1.4.1 **Datasets**

1. **Train** and **test** sets were carved out of the **edx** dataset using the function *createDataPartition*. These sets were used to *train* and *test* the model. 80% of the **edx** dataset was included in the **train** set, and 20% of the **edx** dataset was included in the **test** set.

The data science book by Prof. Irizarry states in this regard:

> "A standard way of generating the training and test sets is by randomly splitting the data. The caret package includes the function createDataPartition that helps us generates indexes for randomly splitting the data into training and test sets".
>
> "…we carve out a piece of our dataset and pretend it is an independent dataset: we divide the dataset into a training set … and a test set … . We will train our algorithm exclusively on the training set and use the test set only for evaluation purposes."
>
> "We usually try to select a small piece of the dataset so that we have as much data as possible to train. However, we also want the test set to be large so that we obtain a stable estimate of the loss without fitting an impractical number of models. Typical choices are to use 10%-20% of the data for testing."

In this case, I chose to use 80% of the data for *training*, so that I "have as much data as possible to train", and at the same time for use 20% of the data for *testing*, in order for "the test set to be large" so that I "obtain a stable estimate of the loss" (Irizarry, 2019) Therefore, I chose to use 80% of the data for *training* (the **train** set), and 20% of the data for *testing* (the **test** set).

2. *Cross-validation train and test sets* were carved out of the **train** set using the function *createDataPartition*. These sets were used for *cross-validation* conducted in order to select the optimal *lambda* for **regularization** of the effects in the model. 80% of the **train** set was included in the *cross-validation **train** set*, and 20% of the **train** set was included in the *cross-validation **test** set.*

*Once the final model has been built and selected, the final effects were computed over the **edx** dataset before creating **predictions** for the **validation** set.*

2.1.4.2 **RMSE function**

RMSE (*Root Mean Squared Error*) is defined and discussed below.

> "Root mean squared error (RMSE) is the square root of the mean of the square of all of the error. The use of RMSE is very common, and it is considered an excellent general-purpose error metric for numerical predictions. … RMSE is a good measure of accuracy, but only to compare prediction errors of different models." (Neill and Hashemi, 2018)

The **RMSE** (*Root Mean Squared Error*) function appearing below was written in order to evaluate **model performance** and calculate an **RMSE** on the **test** set. The RMSE function below was based on the following formula:

$$\sqrt{\frac{1}{N}\sum_{u,i}(Predicted\ Ratings\ _{u,i} - True\ Ratings\ _{u,i})^2}$$

with *Predicted Rating*$_{u,i}$ the predicted rating by user u to movie i, and *True Rating*$_{u,i}$ the *true rating* by *user u* to *movie i*, N the number of *user/movie combinations*, and the *sum* occurring over all these *user/movie combinations*.

The RMSE function I wrote appears below.

```
RMSE
```

```
## function(predicted, actual){
##   sqrt(mean((predicted - actual)^2))
## }
## <bytecode: 0x0000000026b2ab28>
```

2.1.4.3 **Ratings**

Please note: All ratings were "**made on a 5-star scale, with half-star increments (0.5 stars - 5.0 stars)**" (GroupLens, 2018) However, for simplicity of the model, I assumed the ratings were on a *continuous numerical* scale, and could take any numerical value, rather than being *discrete*, or *ordinal*.

The model included computing the overall average rating ***mu***, across all *users* and all *movies*:

Mu
3.512465

Please note: The ***mu*** above was computed over the **edx** dataset. The model included computing a different ***mu*** several times at different stages of the model over different subsets of the **edx** dataset.

2.1.4.4 **Movie effect**

Data exploration and visualization demonstrated that different *movies* were rated differently. They had a different ***average rating***. This was considered a ***movie effect***. It was computed separately for each movie. The ***movie effect*** was represented in the model by the term ***b__i***. This effect was first computed as an average of the term ***rating - mu*** over all *ratings* received by a certain *movie*. Later, ***regularization*** was applied to the movie effect ***b__i***.

2.1.4.5 **Regularization**

The data science book by prof. Irizarry states –

> "Regularization permits us to penalize large estimates that are formed using small sample sizes."
>
> "The general idea behind regularization is to constrain the total variability of the effect sizes."
>
> "The general idea of penalized regression is to control the total variability of the movie effects".

Data exploration and visualization demonstrated that some *movies* received many ratings, while other *movies* received very few ratings. The process of ***regularization*** included "penalizing" the effects obtained, such that effects obtained from **stratas** (i.e., a certain *movieId*) that included less ratings were affected more by *regularization*. The *penalty* term was applied to all movies but its effect was more noticeable for movies that did not get many ratings. The penalty was applied through a parameter called *lambda*. Instead of computing an average effect ***b__i*** for each movie, a **sum** was computed for the term ***rating - mu*** over all ratings received for a certain movie. This **sum** was then divided by the **number of ratings** received for that movie ***n*** + the parameter ***lambda***. Movies that had many ratings had a large ***n***, and the effect of ***lambda*** on the computation was negligent. For such movies, the result of the computation above was closer to the effect ***b__i*** obtained by averaging the term ***rating - mu***, as described above. Movies that had less ratings had a smaller ***n***. For these movies the effect of ***lambda*** was more noticeable. The prediction made for such movies was closer to the average ***mu***, with a smaller movie effect ***b__i*** included in the prediction. This process is called ***regularization***.

The data science book by prof. Irizarry explains the effect of ***regularization***:

> "[W]here $\mathrm{n_i}$ is the number of ratings made for movie i. ... [W]hen our sample size $\mathrm{n_i}$ is very large, a case which will give us a stable estimate, then the penalty λ is effectively ignored since $\mathrm{n_i} + \lambda \approx \mathrm{n_i}$. However, when the $\mathrm{n_i}$ is small, then the estimate $\hat{b}_\mathrm{i}(\lambda)$ is shrunken towards 0. The larger λ, the more we shrink."

Please note: ***Regularization*** was applied to all effects in the model. The same ***lambda*** was used for all these effects.

2.1.4.6 **Cross-validation**

Cross-validation train and test sets were carved out of the **train** set. Theses sets were used to select the optimal ***lambda*** for regularizing the effects in the model. The same ***lambda*** was selected and used for all effects included in the model.

The ***cross-validation train and test sets*** were created in order not to use the **test** set for selection of *lambda* and not to *overtrain* the model on the **test** set.

The data science book by Prof. Irizarry states that -

> "If I train an algorithm on the same dataset that I use to compute the apparent error, I might be overtraining. In general, when I do this, the apparent error will be an underestimate of the true error." (Irizarry, 2019)

The results of cross-validation to select the optimal *lambda* appear below. A graph that shows the effect of *lambda* on the **RMSE** obtained for the *cross-validation test set* appear at the **data exploration and visualization** section above.

The Optimal Lambda
5

2.1.4.7 **User Effect**

Data exploration and visualization demonstrated that different *users* had different rating habits. The *average residual* for a certain *user* was considered a ***user*** effect. After removing the overall average rating ***mu*** and the movie effect ***b__i***, the user effect ***b__u*** was computed for each user. It was computed as the **sum** of the term ***rating - mu - b__i*** over all ratings by that user, divided by the ***number of ratings*** that user gave ***n*** + ***lambda***.

2.1.4.8 **Genre Combination Effect**

Please note: In the model I used each *genre combination* ascribed to at least one movie in the **edx** dataset, rather than using each *genre* by itself. Rather than **18** genres (+ 1 category of *no genres listed*), the **edx** dataset had **797** different *genre combinations*. An example of a *genre combination* is *Comedy|Romance*.

Data exploration and visualization demonstrated that different *genre combinations* had different *average residuals* left after removing ***mu***, *movie effect* ***b__i*** and *user effect* ***b__u*** from each rating. The *average residual* for a certain *genre combination* was considered a ***genre combination*** effect. A *genre combination* effect ***b__g*** was computed for each *genre combination*. It was computed as the **sum** of the term ***rating - mu - b__i - b__u*** over all ratings for a certain *genre combination*, divided by the ***number of ratings*** for that *genre combination* ***n*** + ***lambda***.

2.1.4.9 Time Effects

After removing all previous effects, I looked for **time** effects on the residuals. Time effects were computed using the *timestamp*, which represented the time when the rating was given, in seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970.

A ***date*** column was added using the function *as_datetime()* on the *timestamp*. Then, the columns ***week***, ***day*** (*day-of-year*) and ***year*** were added to the dataset, using the functions *round_date(date, unit = "week")*, *yday(date)* and *year(date)* respectively. The total range of dates in the **edx** dataset was {1995-01-09 11:46:49, 2009-01-05 05:02:16}.

The different time effects, *week*, *day-of-year* and *year*, were computed on the residuals left after removing from each rating - ***mu***, ***b__i***, ***b__u***, and ***b__g***. For the *day-of-year* effect, also the *week* effect ***b__w*** was removed. For the *year* effect, in addition to ***b__w***, also the *day-of-year* effect ***b__d*** was removed.

2.1.4.9.1 Week Effect

There were a total of **671** different *weeks* in the **edx** dataset. The *week* effect was computed separately for each week. **Data exploration and visualization** demonstrated a clear effect of *week* on the *average residuals* left after removing previous effects from the ratings. The *average residual* for any given week was considered a ***week*** effect. A *week* effect ***b__w*** was computed for each week. It was computed as the **sum** of the term ***rating - mu - b__i - b__u - b__g*** over all ratings for any given week, divided by the ***number of ratings*** for that particular week ***n*** + ***lambda***.

2.1.4.9.2 Day-of-Year Effect

There were a total of **366** different *days-of-year* in the **edx** dataset (applying also to leap years). **Data exploration and visualization** suggested *day-of-year* had an effect on the *average residuals* left after removing previous effects from the ratings. The *average residual* for a given *day-of-year* was considered a ***day effect***. A *day* effect ***b__d*** was computed for each given *day-of-year*. It was computed as the **sum** of the term ***rating - mu - b__i - b__u - b__g - b__w*** over all *ratings* for any given *day-of-year*, divided by the ***number of ratings*** for that particular *day-of-year* ***n*** + ***lambda***. *Note* the minimal *number of ratings* for a given *day-of-year* (as can be seen in a table at the **data exploration and visualization** section above) was **8043**. This is quite a large *number of ratings* in a single *strata*. Therefore the influence of lambda = 5 on this great *number of ratings* ***n*** in a single strata was negligible. Selecting a different *lambda* for the *day-of-year* effect could be sensible. This could be done as part of future work.

2.1.4.9.3 Year Effect

There were a total of **15** different *years* in the **edx** dataset. The *year* effect was computed separately for each year. **Data exploration and visualization** demonstrated that after accounting for all previous effects, a *year* effect was apparent on the *average residuals* left after removing previous effects from the ratings. The *average residual* for a given year was considered a ***year effect***. A *year* effect ***b__y*** was computed for each given year. It was computed as the **sum** of the term ***rating - mu - b__i - b__u - b__g - b__w - b__d*** over all ratings given at a certain *year*, divided by the ***number of ratings*** given in that year ***n*** + ***lambda***.

2.1.4.10 Modeling Approach - Summary

Based on **data exploration and visualization**, a unique effect was assumed to each of the following – ***movieId***, ***userId***, ***genre combination***, ***week***, ***day-of-year***, and ***year***.

The **edx** dataset included:

- **10677** unique *movieId's*
- **69878** unique *userId's*

- **797** unique *genre combinations*
- **671** unique *weeks* (*over the period for which ratings were given {1995-01-09 11:46:49, 2009-01-05 05:02:16}*)
- **366** unique *days of year* (*including leap years*)
- **15** unique *years* (*over the same period, as above*)

An *effect* meant that after removing ***mu*** and *previous effects*, a certain feature (i.e. *genre combination*) had an effect on the residuals left after removing ***mu*** and previous effects from the rating.

An average *effect* was computed for each of these features, for each strata (i.e. day of year = 1, the first day of any calendar year).

All of the effects were ***regularized***. A *lambda* parameter was used for *regularization. Lambda* was applied to all strata, but its effect was more noticeable on strata that did not have many ratings. ***Lambda*** was selected using *cross-validation* on *cross-validation train and test sets.*

The final model was used to make *predictions* for each unique *user/movie combination* in the **test** set. The final model was then used to make ***predictions*** for the **validation** set.

Predictions were made using the following formula:

> ***Predicted = mu + b_i + b_u + b_g + b_w + b_d + b_y***

Effects were computed on the *train* set for predictions made for the *test* set. Effects were computed on the *edx* dataset for predictions made for the *validation* set.

3. Results

This section presents the *modeling results* and discusses the *model performance.*

3.1 Modeling Results

The ***Final Model*** included the following *effects*, computed for each *user/movie combination*:

1. ***Overall average rating*** for all movies and all users - ***mu***.
2. ***Movie Effect* - *b_i***.
3. ***User Effect* - *b_u***.
4. ***Genre Combination Effect* - *b_g***.
5. ***Week Effect* - *b_w***.
6. ***Day-Of-Year Effect* - *b_d***.
7. ***Year Effect* - *b_y***.

All effects except ***mu*** were ***regularized*** using the same ***lambda***. The optimal ***lambda*** for regularizing all of the effects was selected using *cross-validation* on *cross-validation train and test sets.*

The optimal ***lambda*** selected was:

The Optimal Lambda
5

Predictions for the **validation** set were made using the following formula –

Predicted = mu + b_i + b_u + b_g + b_w + b_d + b_y

with –

- ***mu*** - *overall average rating* for all movies and all users
- ***b_i*** - *movie effect*
- ***b_u*** - *user effect*
- ***b_g*** - *genre combination effect*
- ***b_w*** - *week effect*
- ***b_d*** - *day-of-year effect*
- ***b_y*** - *year effect*

Please note: *All effects were regularized using the same lambda* = 5.

Predictions were made for all *user/movie combinations* that had an actual rating for them in the validation set, as if these ratings were unknown. **Model performance** was evaluated using **RMSE**, as described below.

The table below presents **10 predictions** made using the final model on the validation set.

The table has the following columns: *userId*, *movieId*, *rating*, and *predicted*.

userId	movieId	rating	predicted
1	231	5.0	4.291164
1	480	5.0	5.004174
1	586	5.0	4.395282
2	151	3.0	3.371963
2	858	2.0	4.274493
2	1544	3.0	2.765103
3	590	3.5	3.921511
3	4995	4.5	4.123380
4	34	5.0	4.246182
4	432	3.0	3.277863

3.2 Model Performance

The **RMSE** (*Root Mean Squared Error*) calculated for the final model on the validation set was:

RMSE - Final Model
0.8642704

This RMSE is ***excellent***, considering the project instructions suggested an **RMSE < 0.86490** was a desirable result.

The RMSE calculated on the validation set implies that my prediction error when making predictions of ratings was about **0.8642704** stars - a prediction error of less than 1.0 star.

With regard to **RMSE** as a measure of model performance:

> "Root mean squared error (RMSE) is the square root of the mean of the square of all of the error. The use of RMSE is very common" (Neill and Hashemi, 2018)

The data science book by Prof. Irizarry (Irizarry, 2019) states –

> "[I]t is the typical error we make when predicting a movie rating." (Irizarry, 2019)

The RMSE was used to estimate the prediction error of my final model on the validation set.

> "[RMSE] is considered an excellent general-purpose error metric for numerical predictions" (Neill and Hashemi, 2018)

The RMSE of my predictions for the validation set was about 0.8642704.

> "RMSE is a good measure of accuracy, but only to compare prediction errors of different models." (Neill and Hashemi, 2018)

The prediction error of different models was compared on the **test** set. The final model that created the smallest **RMSE** on the **test** set was selected. The final model was then used to make predictions for the **validation** set, and **RMSE** was calculated on the **validation** set. The accuracy of my predictions for the validation set was excellent.

The Netflix Prize

The data science book by Prof. Irizarry (Irizarry, 2019) states with regard to the *Netflix Prize* –

> "In October 2006, Netflix offered a challenge to the data science community: improve our recommendation algorithm by 10% and win a million dollars."

> "To win the grand prize of $1,000,000, a participating team had to get an RMSE of about 0.857." (Irizarry, 2019)

Considering an RMSE of about **0.857** was required to win the grand prize of the *Netflix challenge*, the RMSE of my final model on the validation set was compared quite favorably.

A summary written about the winning algorithm quoted the winners of the Netflix challenge, stating that their winning score was RMSE = 0.8712. They also stated that RMSE < 0.8800 "**would land in the top 10**" (Chen, 2011) The RMSE of 0.8712 relates to the RMSE that won the Netflix $50,000 Progress Prize in 2007 (Wikipedia, 2021)

In this context, the **RMSE** of my final model on the validation set, **0.8642704** was lower than the one that won the Netflix $50,000 Progress Prize in 2007. This is excellent.

The RMSE could be possibly improved if additional features were added to the model, i.e. *user/movie interactions*, as suggested in the future work section below.

4. Conclusion

This section gives a brief *summary* of the report, its *limitations* and *future work*.

4.1 Summary

The ***Introduction/Overview*** section presents the dataset I worked with - the **10M version of the MovieLens dataset** (Harper and Konstan, 2015)

The **goal** of the project was *to create a movie recommendation system using the 10M version of the MovieLens dataset.*

The ***Methods/Analysis*** section presents the *process* and *techniques* used, including *data cleaning*, *data exploration and visualization*, *insights gained*, and the *modeling approach.* The process of developing the model and techniques used are presented.

The ***Results*** section presents the *modeling results* and discusses the *model performance.* The final model included the effects of *movie*, *user*, *genre combination*, *week*, *day-of-year*, and *year.* The final model was used to make predictions for the **validation** set. The **RMSE** of the final model calculated for the validation set was **0.8642704**.

The ***Conclusion*** section gives a brief *summary* of the report, its *limitations* and *future work.*

4.2 Limitations

4.2.1 **User/movie interactions** were not incorporated into the model. These could impact the ratings, and including them could improve the **model performance**. User/movie interactions could be incorporated into the model through *matrix factorization* using **PCA** (*Principal Component Analysis*) or **SVD** (*Single Value Decomposition*). This could be done as part of future work.

4.2.2 **Memory limits** Several computations attempted during the development of the model could not be carried out due to ***memory limits***, with a message showing such as "*Error: cannot allocate vector of size 68.7 Mb*". One of the solutions I found was to keep removing objects I did not use from the environment. The dimensions of the edx dataset were {**9000055, 6**}. This could explain a problem with *memory limits.* A newer and more powerful computer could help resolve this problem as well.

4.2.3 The parameter ***lambda*** was used for *regularization.* The model used the same *lambda* for all *effects.* Selection of a different *lambda* for *different effects* could improve **model performance**. This is in particular as some of the effects, i.e. *time effects*, included a great *number of ratings* in each strata. A different *lambda* for regularizing these effects could yield better **model performance**. This could be done as part of future work. Note that although many ratings were included in most *stratas* of the *time effects*, the smallest *number of ratings* included in a given strata for these effects was **2** for the *week* effect, and **2** for the *year* effect. For these two stratas, a $lambda = 5$ could make sense. For the *day of year* effect, the minimal *number of ratings* included in a certain strata was **8043**. This might imply that selection of a different *lambda* for this effect could possibly yield better **model performance**. *Lambda* could be selected for these effects separately. Future work could employ *cross-validation* to select different *lambdas* for different effects.

4.2.4 The random partitioning of the data to random subsets could affect the **RMSE** calculated on the **test** set. This could be resolved through *cross-validation.* This would include partitioning the dataset multiple times to multiple train and test sets (this could be done through applying the argument $times = 10$ for example in the createDataPartition function; "The argument times is used to define how many random samples of indexes to return" (Irizarry, 2019)), and then using a *mean* RMSE over these random train and test sets as an estimate of the *true error.* This could be done as part of future work. This is an example of the *apparent error* computed for a particular subset Vs the *true error* of the model. Better estimates of the *true error* could be obtained through *cross-validation.*

4.2.5 **Discretization of the ratings** The ratings in the **edx** dataset were *discrete/ordinal.* All ratings were "**made on a 5-star scale, with half-star increments (0.5 stars - 5.0 stars)**" (GroupLens, 2018) For simplicity of the model, I assumed the ratings were on a *continuous numerical* scale, and could take any numerical value, rather than being *discrete*, or *ordinal.* My predictions were thus continuous numerical, and not discrete. Future work could develop and test a model that accounts for this *feature* of the data, by forcing the *predictions* to be discrete, through rounding them to the nearest half-point. The *predictions* could also be forced to be in the range of $\{\mathbf{0.5}, \mathbf{5}\}$.

4.2.6 **Time effects** The model incorporated several *time* effects based on the ***timestamp*** assigned to each rating. These effects would not be able to be used to create future predictions for ratings that do not exist in the dataset, and therefore have no timestamp. However, understanding the influence of *time* on the ratings in the **edx** dataset and possibly removing such effects when making predictions could still be important for future models. Removing time effects when computing other effects (i.e. *movie*, *user*) could remove the ***time bias*** and enhance **model performance** by making more accurate future predictions for ratings that do not exist in the dataset and thus have no timestamp.

4.3 Future Work

Please note: *Some of the future work that could be done is mentioned throughout the report at different sections, and especially at the* ***limitations*** *section above.*

4.3.1 **Matrix Factorization** *User/movie interactions* were not part of the model. This could be added to the model through *matrix factorization* using **PCA** (*Principal Component Analysis*) or **SVD** (*Singular Value Decomposition*). Such analysis could be performed on the residuals matrix left after removing from the ratings all previous effects. Then, data could be wrangled to include *users* in the *rows*, *movies* in the *columns*, and *ratings* in the *cells*. Creation of such a *user/movie matrix* might be limited by the size of this matrix. This matrix would include a total of **9000055** ratings, each for a unique *user/movie combination* in the **edx** dataset. This user/movie matrix would include **69878** rows (*users*) and **10677** columns (*movies*). Furthermore, this matrix would be very sparse. Each user rated a different set of movies and each movie was rated by a different set of users. The sparsity of such a matrix is demonstrated in the **data exploration and visualization** section above.

Matrix factorization could help explain the residuals left after removing all previous effects. This could help achieve more accurate predictions for the validation set and improve **model performance**.

Note that as the user/movie matrix would be very sparse, as can be seen in the demonstrations shown in the **data exploration and visualization** section above, it would contain many **NAs** - user/movie combinations for which no rating exists in the **edx** dataset. These NAs should be replaced with care. *user/movie combinations* for which no rating exists in the dataset may be *user/movie combinations* that would potentially receive a lower rating than the overall average rating across all movies and all users ***mu***. If this was the case, the value to replace the NAs with should be lower then ***mu***. How much lower than ***mu***? Different values to replace the *NAs* with could possibly be tried with *cross-validation.*

4.3.2 **Cross-Validation** The model included partitioning the **edx** dataset once, to create a single **train** set on which the models were trained, and a single **test** set on which predictions were made and **RMSE** tested. Future work could incorporate further *cross-validation.* The **edx** dataset could be randomly partitioned multiple times to multiple random train and test sets (the argument *times* could be used to create multiple random train and test sets with the function *createDataPartition*), models developed, and RMSE tested on each random test set. The *mean* RMSE would then be calculated. The same model could be applied to all these random train and test sets, and RMSE tested on each random test set, using the function *sapply.* The *mean* RMSE of the model would then be calculated. Such a *mean RMSE* could be a better estimate of the *true error* of the model. *Lambda* could also be selected by multiple random partitions of the **train** set to multiple random cross-validation train and test sets and using cross-validation to select a *lambda* for each of these random cross-validation train and test sets, then eventually averaging the optimal *lambda* over all random subsets tested. This could be incorporated into the same *sapply* function described above. Selection of a mean *lambda* over multiple cross-validations with multiple random cross-validation train and test sets could improve **model performance**. However, this could increase computing time considerably. This step of *cross-validation* could improve **model performance** of future models.

4.3.3 **Genre Effect** The model computed a *genre combination* effect ***b_g*** for each *genre combination* ascribed to at least one movie in the **edx** dataset, i.e. Comedy|Romance. This allowed for **797** distinct *genre* *combinations.* Future work could test the effect of each *single genre* and compute a ***b__g*** effect for each *single genre* separately.

4.3.4 **Regularization** The *lambda* parameter used in the model was selected based on *cross-validation* for all of the effects together. Future work could employ *cross-validation* to select different *lambdas* for different effects. Selecting a different *lambda* for *different effects* could improve **model performance**. This is in particular as some of the effects, i.e. *time effects*, included a great *number of ratings* in each strata. Selecting a different *lambda* for regularizing *time effects* could yield better **model performance**. This could be done as part of future work.

4.3.5 **Discretization of the ratings** The ratings included in the **edx** dataset were *discrete/ordinal.* Future work could develop and test a model that accounts for this *feature* of the data. Such model could force the *predictions* to be *discrete.* This could be done by rounding the *predictions* to the nearest half-point. The *predictions* could also be forced to be in the range of $\{\mathbf{0.5}, \mathbf{5}\}$. This could improve **model performance**, as it may reduce the prediction error.

To conclude – the model included the following effects: ***overall average rating mu***, ***movie b__i***, ***user b__u***, ***genre combination b__g***, ***week b__w***, ***day-of-year b__d***, and ***year b__y***.

The model was used to make *predictions* for the **validation** set. The following RMSE was calculated:

RMSE - Final Model
0.8642704

Limitations and *future work* are discussed above. The *insights gained* are discussed in the *insights gained* section above.

5. References

Chen, E 2011 Winning the Netflix Prize: A Summary. Available at http://blog.echen.me/2011/10/24/winning-the-netflix-prize-a-summary/ [Last accessed 15July2021]

GroupLens 2009 MovieLens 10M Dataset README. Available at https://files.grouplens.org/datasets/movielens/ml-10m-README.html [Last accessed 12July2021].

GroupLens 2018 MovieLens Latest Full Dataset README. Available at https://files.grouplens.org/datasets/movielens/ml-latest-README.html [Last accessed 12July2021].

GroupLens 2021a MovieLens. Available at https://grouplens.org/datasets/movielens/ [Last accessed 12July2021]

GroupLens 2021b What is GroupLens? Available at https://grouplens.org/about/what-is-grouplens/ [Last accessed 12July2021].

GroupLens 2021c MovieLens Latest Datasets. Available at https://grouplens.org/datasets/movielens/latest/ [Last accessed 16July2021]

GroupLens 2021d MovieLens 10M Dataset. Available at https://grouplens.org/datasets/movielens/10m/ [Last accessed 16July2021]

Harper, F M and **Konstan, J A** 2015 The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5(4.19):1-19. DOI: http://dx.doi.org/10.1145/2827872

Irizarry, R A 2019 *Introduction to Data Science.* 1st Edition. Chapman & Hall/CRC Data Science Series. Available at https://leanpub.com/datasciencebook [Last accessed 12July2021].

Neill, S P and **Hashemi M R** 2018 *Fundamentals of Ocean Renewable Energy.* Elsevier. DOI: https://doi.org/10.1016/C2016-0-00230-9

Wikipedia 2021 Netflix Prize. Available at https://en.wikipedia.org/wiki/Netflix_Prize [Last accessed 15July2021]