MovieLens Project

Dr Daniel Mayenberger

May 2020

1 Executive Summary

Given over 10,000 movies and about 70,000 users that have rated some of these movies on a rating scale from half a star to five stars, the goal is to predict future ratings based on past ratings. The challenge is to do this for the over 700 millions different movie/user combinations based on only 1.2% (9 millions) of such combinations used for training the algorithm.

The available data of 10 million ratings are split into 9 million ratings to train different algorithms and 1 million ratings to evaluate the performance of these methods. Performance is measured by the root-mean squared error (RMSE) of ratings.

Based on the RMSE, the model that performs best with an RMSE of 0.8648 is

$$Y_{u,i} = \mu + b_i + b_u + \sum_{k=1}^{K} \delta_{i,k}\beta_k + \varepsilon_{u,i},$$

where $Y_{u,i}$ is the rating by user u of movie i, $\delta_{i,k}$ is an indicator which is 1 if the movie i is of genre k and zero otherwise, b_i is the movie effect, b_u the user effect and β_k the genre effect.

2 Introduction

The movielens project makes available millions of a movie ratings provided by anonymised users.

From the movielens website about the data the following further information is available:

- The data set contains just over 10 million ratings by users of the online movie recommender service MovieLens.
- Users were selected at random for inclusion. All users selected had rated at least 20 movies. Each user is represented by an id, and no other information is provided about users.

Denoting the rating by any of these users u of a certain movie i, the task is to predict ratings $Y_{u,i}$ of any such user/movie combination (u, i). The challenge is that there are overall 71,567 users and 10,681 movies (figures from the website), yielding $71,567 \times 10,681 = 764,407,127$ while only 10 million data points, so only about 1.3% (= 10/764.4) such data, are available in totality and 10% (1 million) of the data must be held out as test set, leaving 1.2% (= 9/764.4) of data to train a rating prediction algorithm.

To do so, the data are examined and modelled in section 3 which is further broken down into:

- subsection 3.1 to elaborate on the techniques used, in particular those coded in the later modelling subsection 3.5.
- subsection 3.2 to provide an overview of the basic structure of the rating data.
- subsection 3.3 to perform data cleaning.
- subsection 3.4 to visualise the most important properties of the ratings with respect to individual movies, their genres or individual users. These properties then inspire the modelling methods in the subseqeuent section.

- subsection 3.5 presents the models based on the most salient data properties and calibrates any free modelling parameters.

The results of all models are summarised in section 4 and the conclusions are drawn in section 5.

3 Methods and Analysis

3.1 Process and Techniques Used

Two modelling techniques will be described, first the regularisation in subsubsection 3.1.1 and then the local estimated scatterplot smoothing (LOESS) method in subsubsection 3.1.2.

3.1.1 Regularisation

The modelling techniques employed are regularised least squared estimates. As an example, a model of the form

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

for user u and movie i assumes that there is an overall average rating of μ across all movies. The b_i is based on the observation that some movies are better rated than others, as will be shown in Section 3.4.2. However, as there are some movies that are rated by hundreds or more users, the ratings of these movies are more reliable than the ratings of movies that are only rated by a few users. To calculate such an effect, denote the number of ratings awarded to movie i by n_i, the overall number of ratings as N and the overall number of movies by J. Further, denote the overall average rating with μ.

Without any weighing of ratings for a movie by the number of ratings awarded to it, the goal is to minimise the MSE given by

$$MSE = \frac{1}{N}\sum_{i=1}^{J}\sum_{u=1}^{n_i}(y_{u,i} - \hat{y}_{u,i})^2.$$

As for any user u and movie i the modelled rating is given by $\hat{y}_{u,i} = \mu + \hat{b}_i$, the MSE to minimise becomes

$$MSE = \frac{1}{N}\sum_{i=1}^{J}\sum_{u=1}^{n_i}\left(y_{u,i} - (\mu + \hat{b}_i)\right)^2.$$

The necessary condition for a minimum is $\frac{\partial MSE}{\partial b_j} = 0$ for all $j = 1, 2, \ldots, J$ which leads to

$$0 = -\frac{2}{N}\sum_{u=1}^{n_j}\left(y_{u,j} - (\mu + \hat{b}_j)\right) \quad (j = 1, \ldots, J).$$

This is equivalent to

$$\hat{b}_j = \frac{1}{n_j}\sum_{u=1}^{n_j}(y_{u,j} - \mu) \quad (j = 1, \ldots, J).$$

To penalise estimates for movies that are only rated by a few users, the MSE to minimise then becomes

$$MSE(\lambda) = \frac{1}{N}\left[\sum_{i=1}^{J}\sum_{u=1}^{n_i}(y_{u,i} - \hat{y}_{u,i})^2 + \lambda\sum_{i=1}^{J}b_i^2\right]$$

with a regularisation parameter $\lambda \geq 0$. Again, the necessary condition for a minimum of MSE is $\frac{\partial MSE(\lambda)}{\partial b_j} = 0$ for all $j = 1, 2, \ldots, J$ which implies

$$0 = -\sum_{u=1}^{n_j}\left(y_{u,j} - (\mu + \hat{b}_j)\right) + \lambda b_j \quad (j = 1, \ldots, J).$$

Solving for $\hat{b}_j$ yields

$$\hat{b}_j = \frac{1}{n_j + \lambda} \sum_{u=1}^{n_j} (y_{u,j} - \mu) \quad (j = 1, \ldots, J).$$

This solution is a minimum, since the second derivative (Hessian matrix) of the MSE function is a diagonal matrix, as $\frac{\partial^2 MSE}{\partial b_i \partial b_j} = 0$ for $i \neq j$ and $\frac{\partial^2 MSE}{\partial b_i^2} = 2\frac{n_i+\lambda}{N} > 0$. There are only positive entries on the diagonal of this matrix, and with that it is positively definitive. So the solution for $\hat{b}_j$ provided above is a local minimum and as there are no other local optima (there is only one point at which the first partial derivative vanishes), it is also a global minimum.

3.1.2 Local Estimated Scatterplot Smoothing (LOESS)

Unlike standard local regression that fits a line to the whole data set, the locally estimated scatterplot smoothing (LOESS) estimates a regression line in a local window that is progressively moved through the data. The parameter of LOESS is the width of this window.

Further details of the LOESS can be found on this Wikipedia website.

3.2 Data Structure and Loading

The raw data is provided in two sets, called `edx` and `validation`.

3.2.1 Sample Data

To facilitate code testing, 1,000 samples of data have been extracted from both datasets `edx` and `validation`, called `edx_1000` and `validation_1000` respectively. This is achieved with the following code:

```
set.seed(123, sample.kind = "Rounding")
edx_1000 <- sample_n(edx, size = 1000)
validation_1000 <- sample_n(validation, size = 1000)
```

3.2.2 Basic Data Structure

Both `edx` and `validation` have the same structure that can be displayed with the `str` function and is listed below for `edx`:

Column name	Type	First values
userId	integer	1 1 1
movieId	numeric	122 185 292
rating	numeric	5 5 5
timestamp	integer	838985046 838983525 838983421
title	character	Boomerang (1992) Net, The (1995) Outbreak (1995)
genres	character	Comedy|Romance Action|Crime|Thriller Action|Drama|Sci-Fi|Thriller

The data sets are both tidy - the count of `NA` values is zero for both sets:

```
edx %>% summarise_all(~sum(is.na(.)))
```

```
##   userId movieId rating timestamp title genres
## 1      0       0      0         0     0      0
```

```
validation %>% summarise_all(~sum(is.na(.)))
```

```
##   userId movieId rating timestamp title genres
## 1      0       0      0         0     0      0
```

The `userId`, `movieId` , `rating`, `title` and `genres` data can be used directly in the format provided.

The `timestamp` column is in raw data format of seconds since 1 January 1970 and will be converted into a date & time format.

3.3 Data Cleaning

The `timestamp` column is converted to a date and time using the `as_datetime` function for both the `edx` and the `validation` dataset and stored in a new column `ratingdate`. After the conversion the `timestamp` columns are no longer required and are discarded.

```
edx <- edx %>%
    mutate(ratingdate = as_datetime(timestamp)) %>%
    select(-timestamp)

validation <- validation %>%
    mutate(ratingdate = as_datetime(timestamp)) %>%
    select(-timestamp)
```

3.4 Data Exploration and Visualisation

3.4.1 General Data Distribution Properties

For better readability, the code pieces for the subsequent graphs in this Section are displayed in Appendix A.

The `edx` data set contains 9,000,055 data points, consisting of 10,677 distinct movies and 69,878 different users:

```
nrow(edx)
```

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

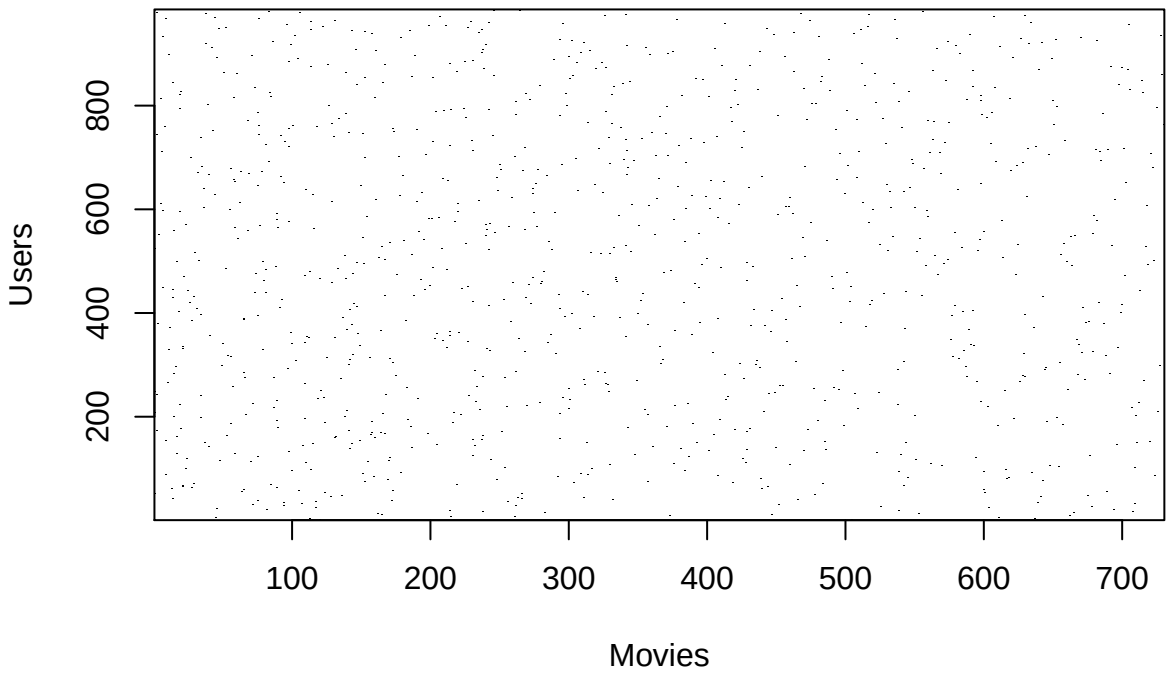
```
edx %>%
    summarise(n_movies = n_distinct(movieId),
              n_users = n_distinct(userId))
```

```
##   n_movies n_users
## 1    10677   69878
```

With that, the total number of diferent movie/user combinations is $10,677 \times 69,878 = 746,087,406$, much more than the overall 10 million data points from the combined `edx` and `validation` sets. While it is not feasible with the memory capacity of a personal computer to visualise all of the over 700 million combinations, it is possible to illustrate the sparsity of the data set on a representative sample.

3.4.1.1 Data Sparsity

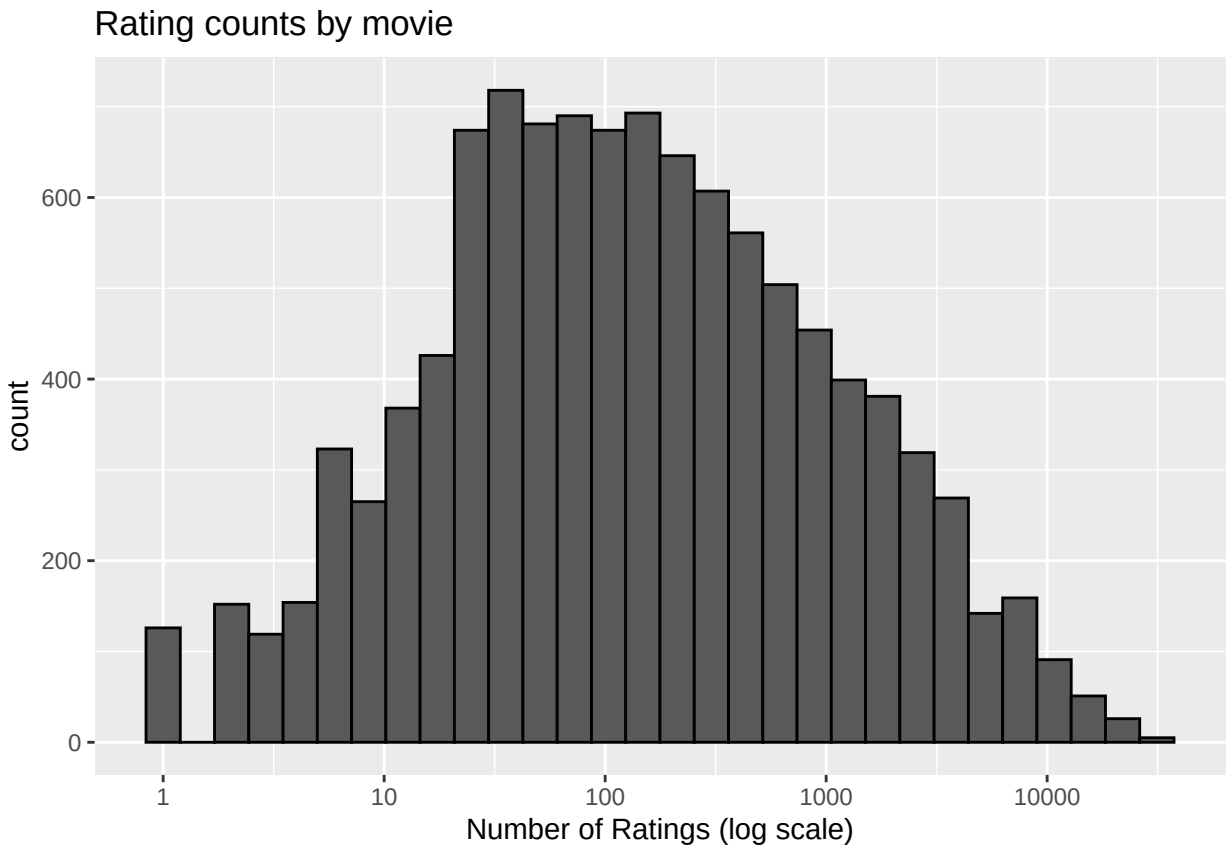
Within the sample of 1,000 ratings drawn as described in subsubsection 3.2.1 there are already 730 different movies and 985 different users. Plotting the data points of this matrix shows that the data given are very sparse but at least evenly distributed among movies and users.



The code for this figure is shown here.

3.4.1.2 Ratings by Movie

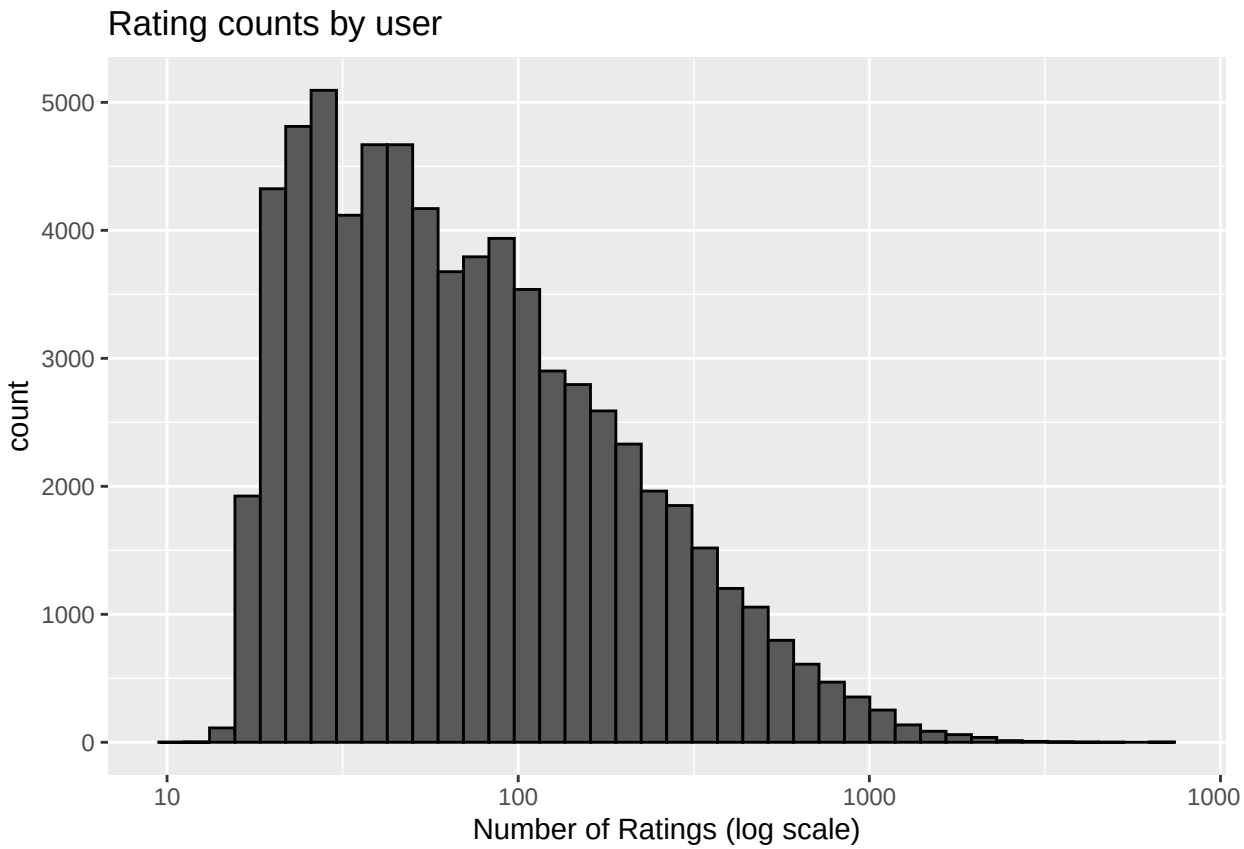
There is a wide variety of the number of ratings by movie. Some movies are rated only up to 10 times, while others are rated thousands of times:



The code for this chart can be found here.

3.4.1.3 Ratings by User

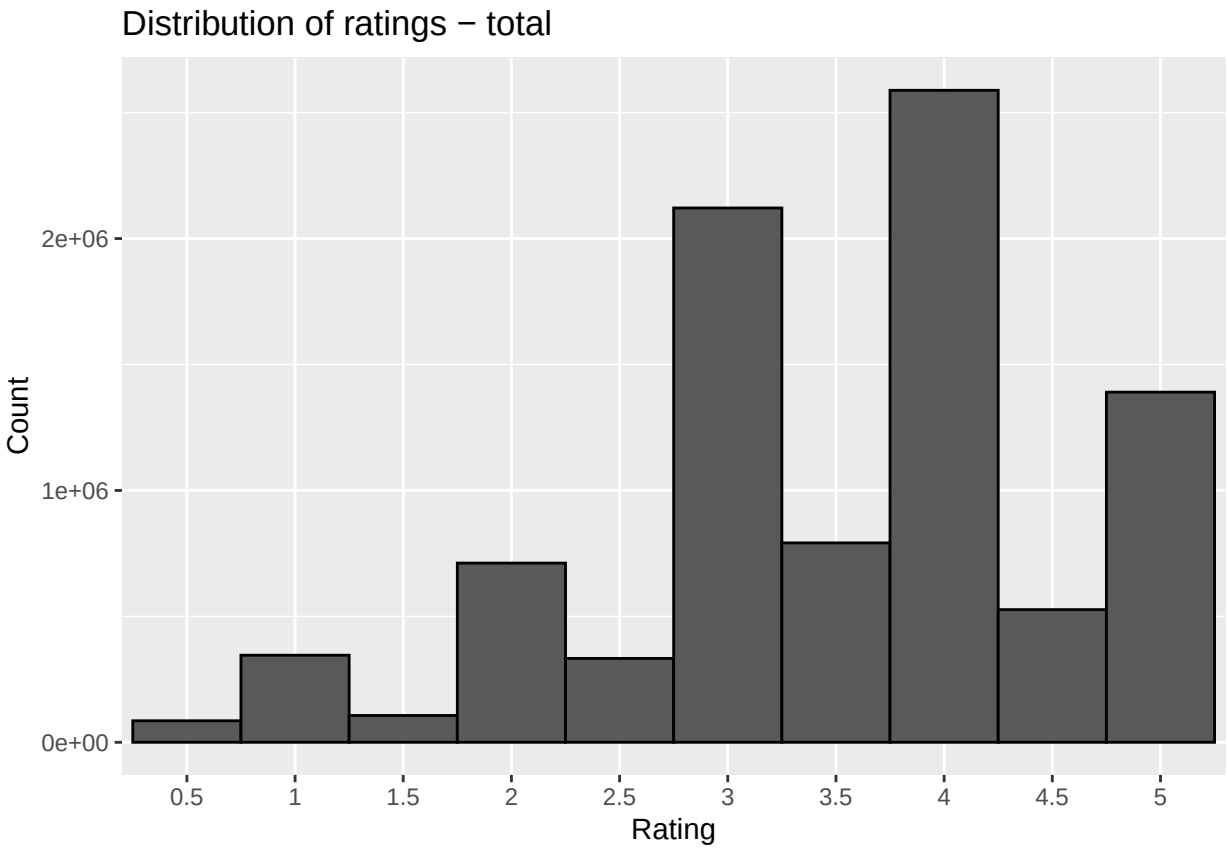
Similarly, there are users that only rate a few movies while other rate hundreds or even thousands of movies:



The code is shown here.

3.4.1.4 Distribution of Ratings

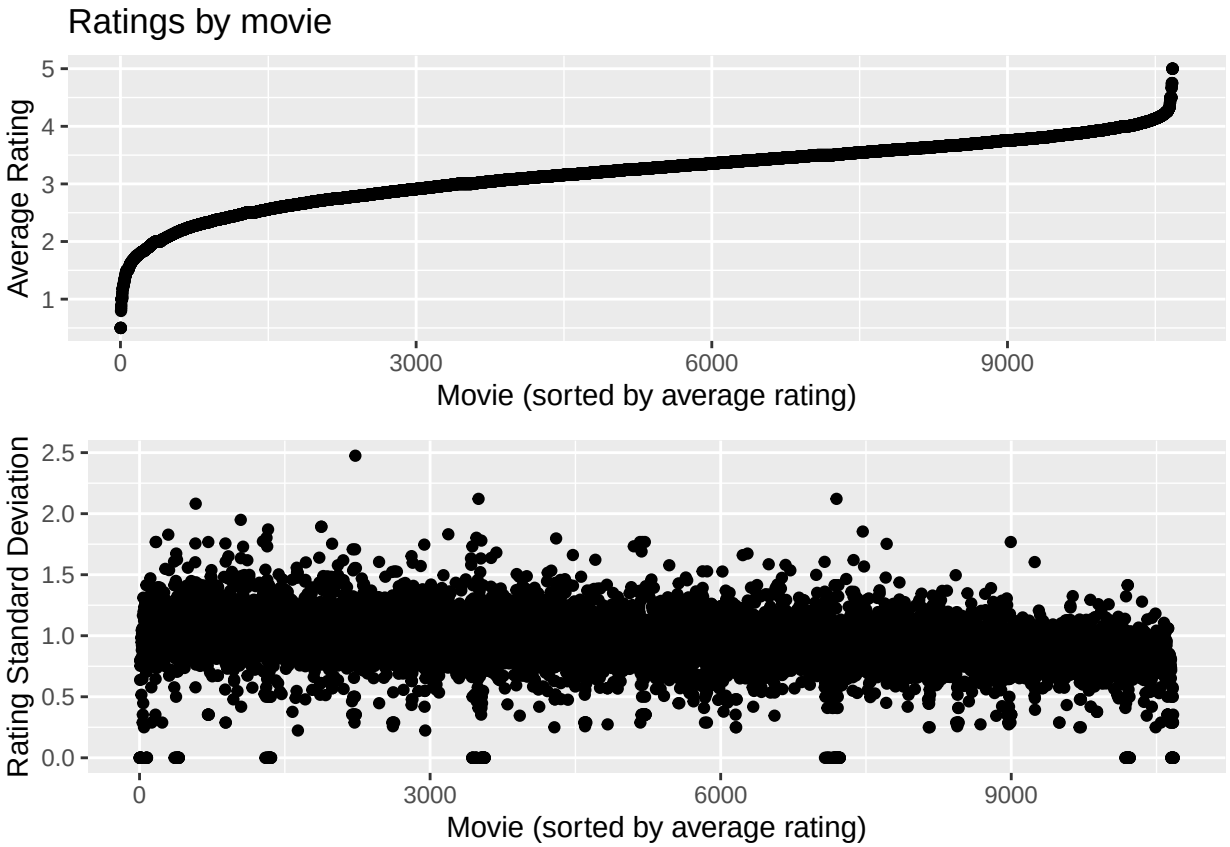
The ratings most awared are three and four stars. In general, whole-star ratings are more frequently given than half-star ratings:



The code for this histogram is displayed here.

3.4.2 Movie Effect

From public movie ratings such as Rotten Tomatoes it is known that some movies are in general better rated than others. We group the ratings by movie ID to evaluate the same effect. In addition to the average rating for each movie, the standard deviation of ratings for the same movie is visualised in parallel.



The code for this chart is here.

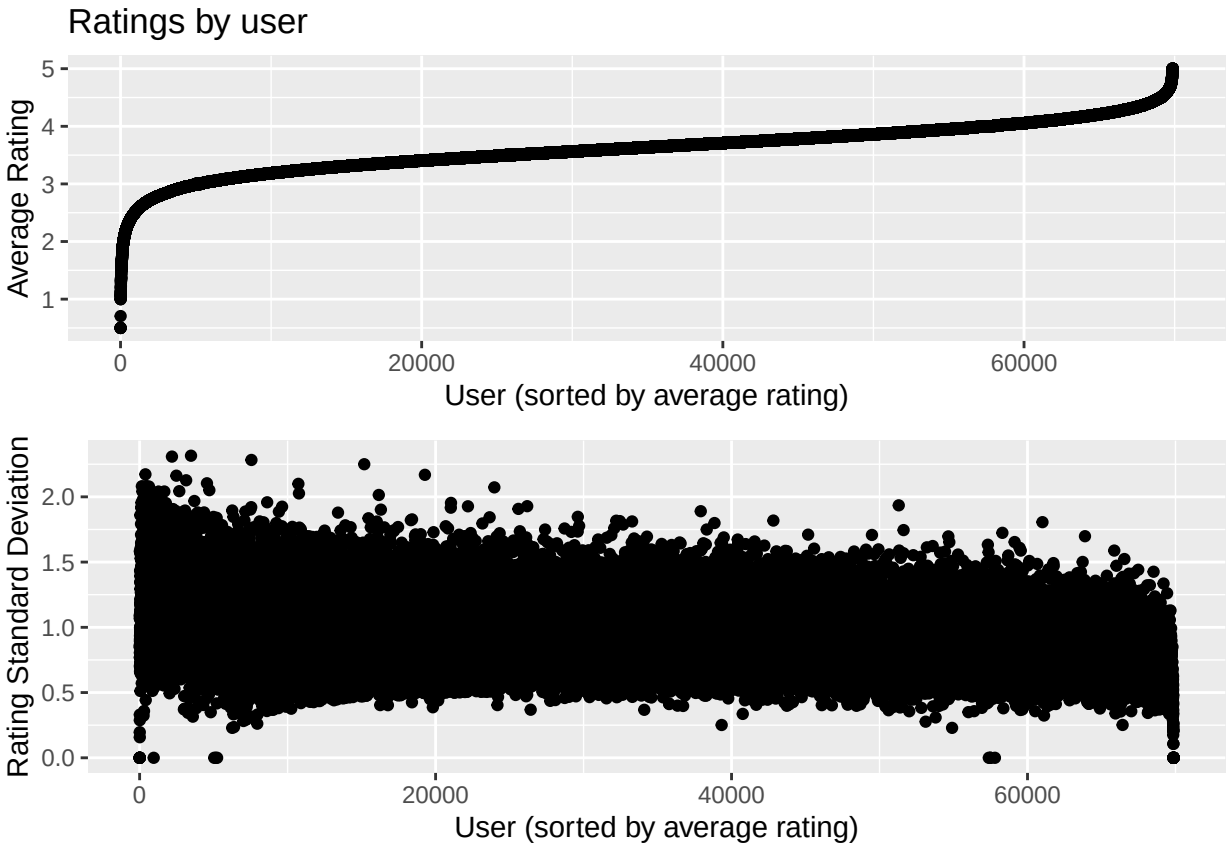
So there are indeed movies that, on average:

- are rated worse than other movies, on the left side of the chart.
- are rated better than other movies, towards the right of the chart.

At the same time, there is a high variability of this *movie effect*, illustrated by the broad range of the standard deviation in the lower half of the figure. Standard deviation of ratings by movie hovers around the value of one (star).

3.4.3 User Effect

Similar to the movie effect, different users may have the tendency to award higher or lower ratings, compared to other users. We examine this effect by grouping the ratings by user and arranging them by their average, along with the standard deviation of the same grouping by user:



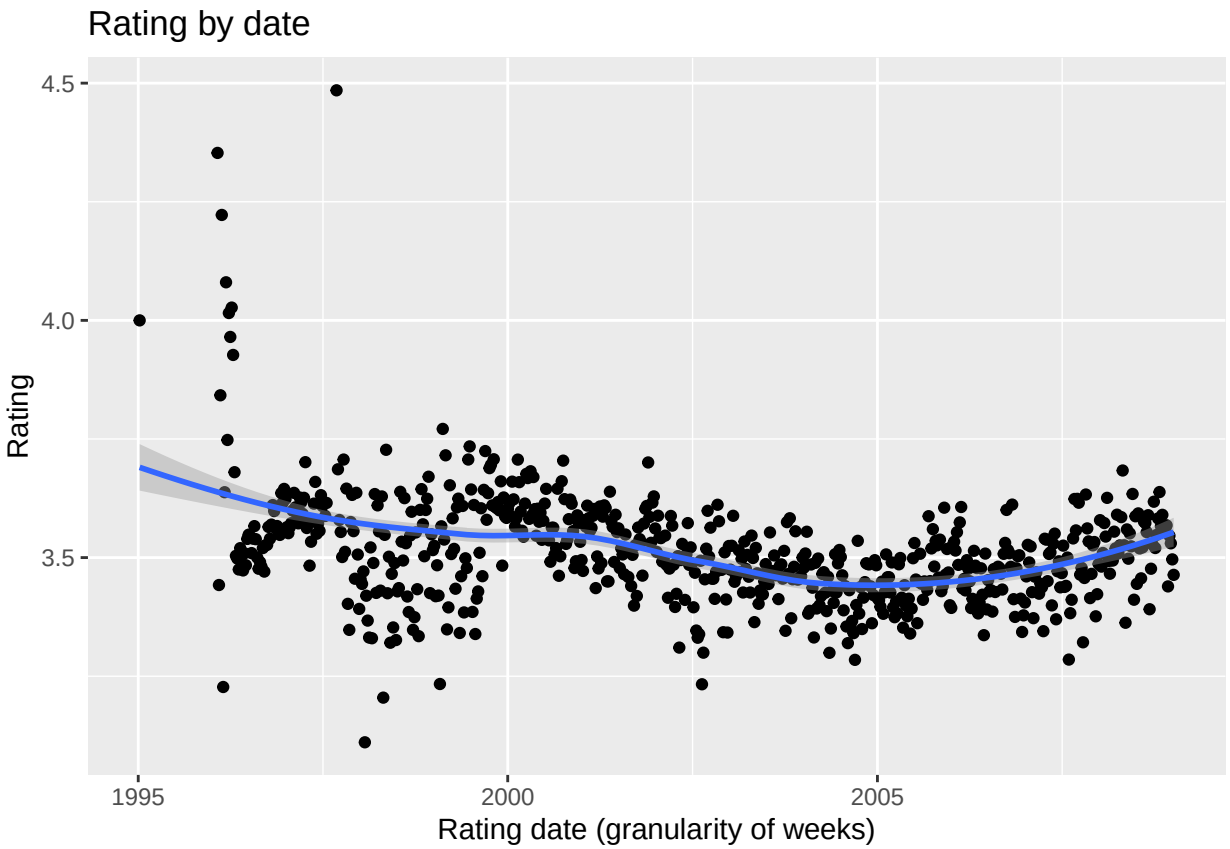
The code for this chart is here.

So similarly to the movie effect, there is a *user effect* with users that tend to assign movies lower rating on the left hand side of the chart and users that rate movies more highly towards the right. In constrast to the *movie effect* there is much higher variability of rating, consistent with users differentiating between good and bad movies.

3.4.4 Time of Rating

Another feature that may influence the rating is the date and time at which the rating was awarded. When averaging all ratings within a week and following this average with a LOESS function, the following pattern emerges:

3.4.4.1 Overall Time Effect

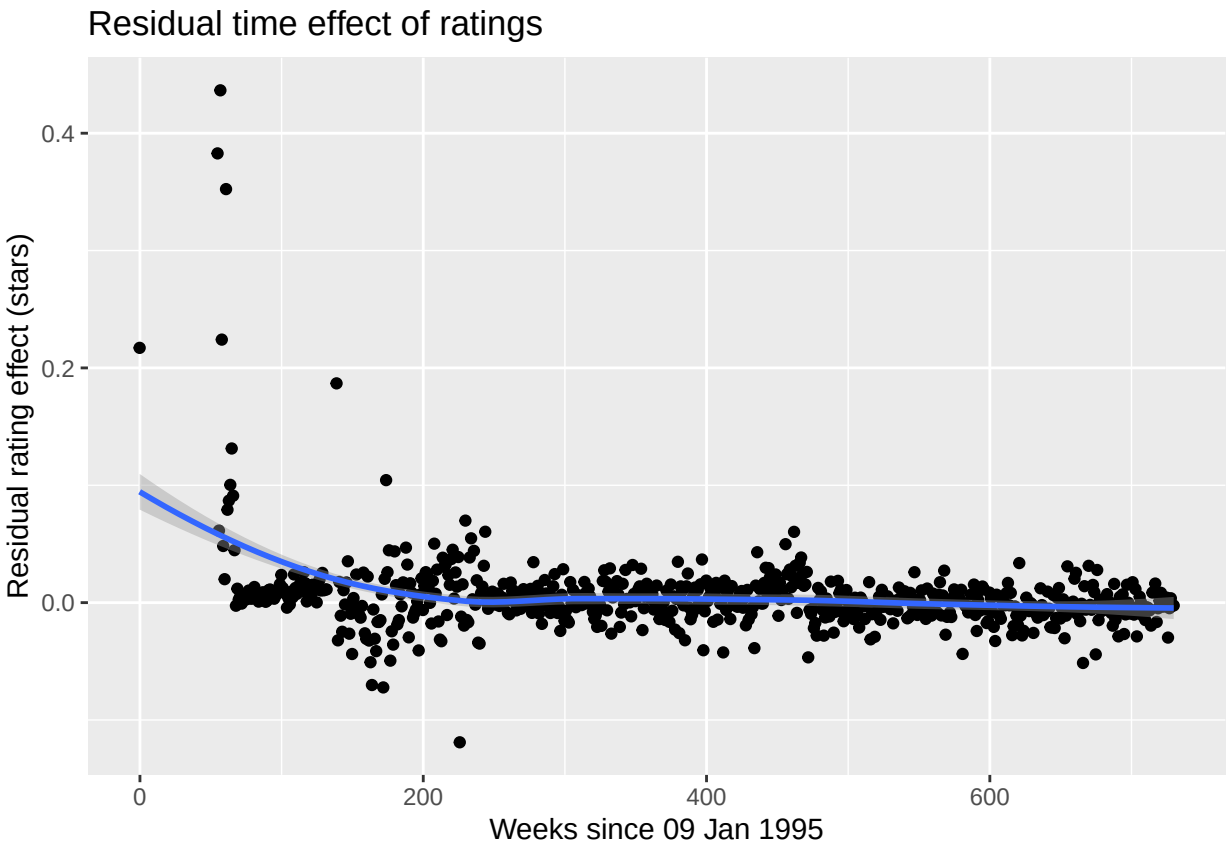


The code for this chart is here.

So there is a small effect that is worth capturing, as a smooth function of time.

3.4.4.2 Residual Time Effect

Since the effect will be modelled after movie effect and user effect are accounted for, it is also informative to plot the residual effect.



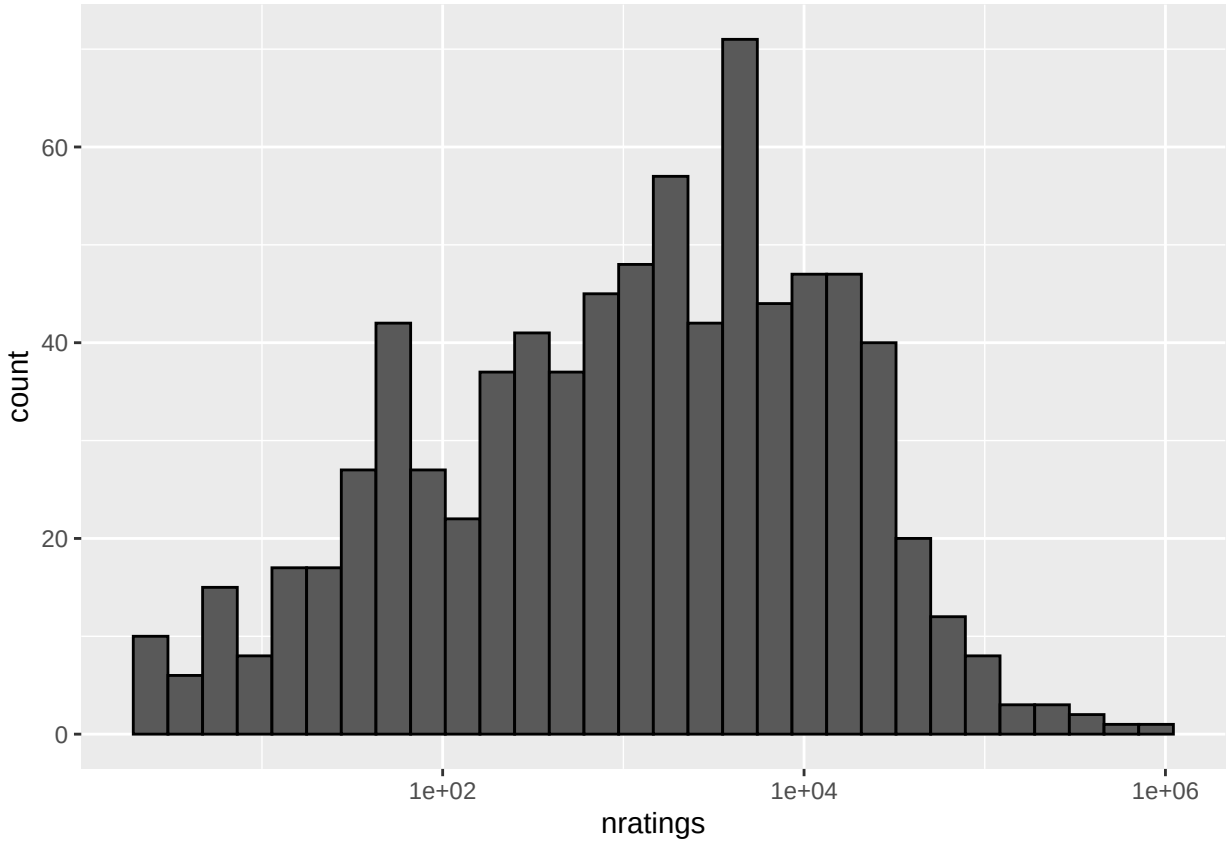
The code for this figure can be found here.

In comparison to the total effect, the residual time effect is:

- much stronger in the first 2 years, after 09 Jan 1995.
- considerable flatter after the the first 2 years.

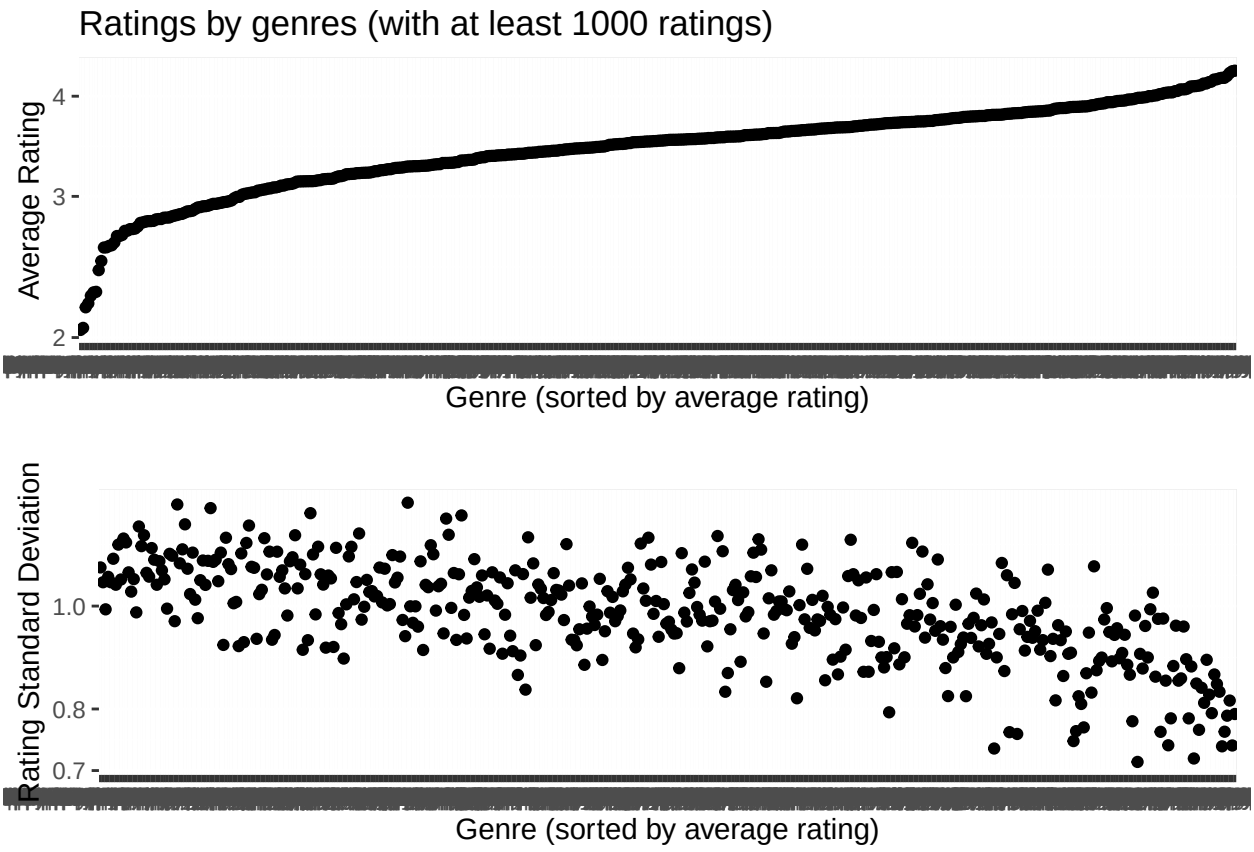
3.4.5 Genre Effect

As final feature, we investigate whether certain genres (or their combinations) are rated differently from others. First, we determine whether genres have sufficiently many ratings by examining the distribution of the number of ratings by genre:



The code for this figure is shown here.

So there are definitely sufficiently many genres with 1,000 or more ratings. In the same way as for movies and users, we group the rating by genres for all genres with over 1,000 rating and chart their average rating and standard deviation.



The code for this chart is shown here.

So the genre combination does have an effect on the rating, in a similar way as movies and users do.

3.4.6 Extrapolation Requirements Check

To predict movie ratings and calculate the final RMSE, it must be considered how the algorithm handles data in the `validation` data set to which it was not trained in the `edx` set. It will be shown that there is no need for such extrapolation for the provided data sets.

By construction, the `validation` set only contains movies and users that are also in the `edx` set, with the `semi_join` statements provided in the instructions. For verification, this code confirms that indeed no movies or users from the `validation` set are missing from the `edx` set:

```
validation %>%
    anti_join(edx, by = "movieId") %>%
    group_by(movieId, title) %>%
    summarise(n = n())
```

```
## # A tibble: 0 x 3
## # Groups:   movieId [0]
## # ... with 3 variables: movieId <dbl>, title <chr>, n <int>
```

```
validation %>%
    anti_join(edx, by = "userId") %>%
    group_by(userId) %>%
    summarise(n = n())
```

```
## # A tibble: 0 x 2
## # ... with 2 variables: userId <int>, n <int>
```

In addition, all genres, and even more specifically, all their combinations that are present in the `validation` set, are also found in the `edx` set, as this code shows:

```
validation %>%
    anti_join(edx, by = "genres") %>%
    group_by(genres) %>%
    summarise(n = n())
```

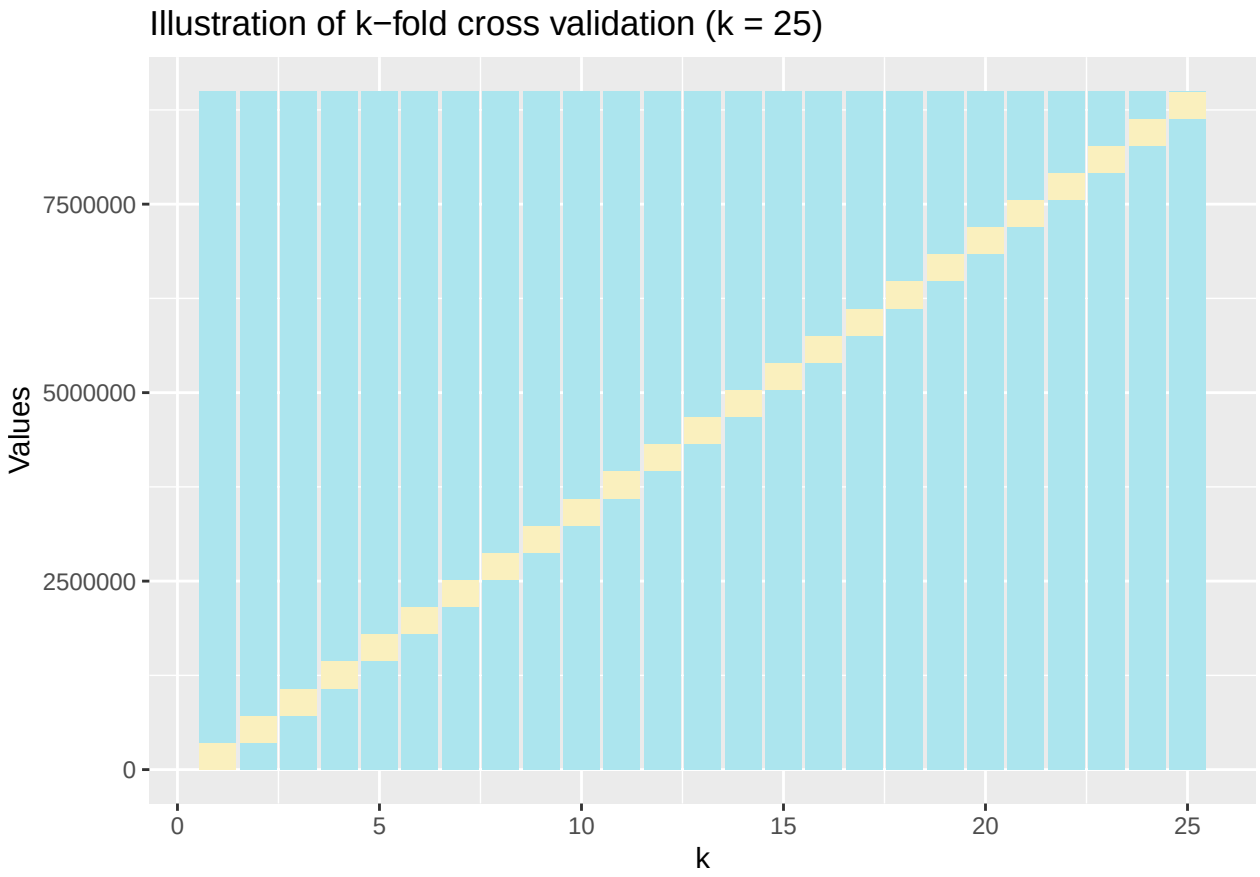
```
## # A tibble: 0 x 2
## # ... with 2 variables: genres <chr>, n <int>
```

So with the given data set, no extrapolation to movies, users or genres needs to be made in predicting ratings and determining model performance.

3.5 Modelling Approach

The training of model parameters is generally done using k-fold cross-validation. To do this, the `edx` data is split into a training and a validation set k times.

Graphically, k-fold validation can be illustrated by k subsequent splits of the overall `edx` set into training sets (shown in blue) and validation sets (shown in purple).



For example, the parameter λ of the regularisation method introduced in subsubsection 3.1.1 is applied to the training data and the RMSE that results from that particular values of λ is evaluated on the validation data. This is done $k = 25$ times to produce an estimate of the RMSE depending on λ.

The **performance** of each algorithm is evaluated using the RMSE which is implemented in the function `RMSE_rating`:

```
RMSE_rating <- function(pred_ratings, true_ratings) {
    ifelse(length(pred_ratings) > 0,
           sqrt(mean((pred_ratings - true_ratings)^2)),
           NA)
    }
```

3.5.1 Constant Value

This is the simplest baseline model that assumes that the rating is the same every time. The model is

$$Y_{u,i} = \mu + \varepsilon_{u,i} \qquad (i = 1, \ldots, J; u = 1, \ldots, n_i),$$

The parameter μ is estimated as average across all ratings:

$$\hat{\mu} = \frac{1}{N} \sum_{u,i} y_{u,i}.$$

The R implementation of this model is given by a function that returns the same value for each record of the requested data set:

```
predict_const <- function(newdata) {
    mu <- mean(edx$rating)
    return(rep(mu, nrow(newdata)))
    }
```

3.5.2 Movie Effect Only

As seen in subsubsection 3.4.2, there is an effect of individual movies on the rating. In terms of the model,

$$Y_{u,i} = \mu + b_i + \varepsilon_{u,i} \qquad (i = 1, \ldots, J; u = 1, \ldots, n_i),$$

where b_i is the effect (or bias) of movie i across all users, and the residual value is $\varepsilon_{u,i}$. As shown in subsubsection 3.1.1, the estimator for the movie effect is given (for the special case of $\lambda = 0$) by

$$\hat{b}_i = \frac{1}{n_i} \sum_{u=1}^{n_i} \left(y_{u,i} - \hat{\mu} \right).$$

This estimate is implemented by the following code with calculates the average rating $\hat{\mu}$ on the training set and then calculates for each movie i the $\hat{b}_i$ according to the above formula. Finally, the estimates $\hat{b}_i$ are used to compute $\hat{y}_{u,i} = \hat{\mu} + \hat{b}_i$.

```
predict_movieb <- function(newdata) {
    # calculate mean $\mu$ of overall `edx` training set
    mu <- mean(edx$rating)

    # estimate $b_i = y_{u,i} - \mu$ for each movie $i$.
    b_i_tbl <- edx %>%
        group_by(movieId) %>%
        summarise(b_i = mean(rating - mu))

    # look up $b_i$ for each movie to predict $\hat{y}_{u,i} = \mu + b_i$
    pred_ratings <- newdata %>%
        left_join(b_i_tbl, by = "movieId") %>%
        mutate(y_hat = mu + b_i) %>% .$y_hat
    return(pred_ratings)
    }
```

3.5.3 User Effect Only

We observed in subsubsection 3.4.3 an effect of users on the rating. Analogously to the movie effect, the user effect is expressed through the model

$$Y_{u,i} = \mu + b_u + \varepsilon_{u,i} \qquad (i = 1, \ldots, J; u = 1, \ldots, n_i).$$

The user effect b_u is estimated by the equation following subsubsection 3.1.1:

$$b_u = \frac{1}{J_u} \sum_{i=1}^{J_u} (y_{u,i} - \hat{\mu}),$$

where J_u is the number of ratings awared by user u to all movies. The code for the estimation of the user effect is implemented in the same way as for the movie effect:

```
predict_userb <- function(newdata) {
    # calculate mean $\mu$ of overall `edx` training set
    mu <- mean(edx$rating)

    # estimate $\hat b_u = avg(y_{u,i} - \mu)$ for each movie $i$.
    b_u_tbl <- edx %>%
        group_by(userId) %>%
        summarise(b_u = mean(rating - mu))

    # look up $b_u$ for each movie to predict $\hat{y}_{u,i} = \mu + \hat b_u$
    pred_ratings <- newdata %>%
        left_join(b_u_tbl, by = "userId") %>%
        mutate(y_hat = mu + b_u) %>% .$y_hat
    return(pred_ratings)
    }
```

3.5.4 Combined Movie and User Effect

Both effects from the previous section can be simply combined in the model

$$Y_{u,i} = \mu + b_i + b_u + \varepsilon_{u,i} \qquad (i = 1, \ldots, J; u = 1, \ldots, n_i).$$

First, the estimate for b_i is calculate in the same way as for the movie effect only:

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

Then, the estimate for the user effect b_u is calculated as the average once the movie effect is already stripped out:

$$\hat{b}_u = \frac{1}{J_u} \sum_{i=1}^{J_u} \left(y_{u,i} - \hat{\mu} - \hat{b}_i \right).$$

These set of estimates $\hat{\mu}$, $\hat{b}_i$ and $\hat{b}_u$ are then used to predict the rating $\hat{y}_{u,i} = \hat{\mu} + \hat{b}_i + \hat{b}_u$. This successive estimation is implemented in the following code:

```
predict_movieuserb <- function(newdata) {
    # calculate mean $\mu$ of overall `edx` training set
    mu <- mean(edx$rating)

    # estimate $\hat b_i = avg(y_{u,i} - \mu)$ for each movie $i$.
```

```
    b_i_tbl <- edx %>%
        group_by(movieId) %>%
        summarise(b_i = mean(rating - mu))

    # estimate $\hat b_u = avg(y_{u,i} - \mu - \hat b_i)$
    b_u_tbl <- edx %>%
        left_join(b_i_tbl, by="movieId") %>%
        group_by(userId) %>%
        summarise(b_u = mean(rating - b_i - mu))

    # apply to new data to calculate
    # $\hat \y_{u,i} = \hat\mu + \hat b_i + \hat b_u$
    pred_ratings <- newdata %>%
        left_join(b_i_tbl, by = "movieId") %>%
        left_join(b_u_tbl, by = "userId") %>%
        mutate(y_hat = mu + b_i + b_u) %>% .$y_hat

    return(pred_ratings)
    }
```

3.5.5 Regularised Movie and User Effect

There are still the same movie and user effects in the model

$$Y_{u,i} = \mu + b_i + b_u + \varepsilon_{u,i} \qquad (i = 1, \ldots, J; u = 1, \ldots, n_i).$$

However, the estimate is given by regularisation with the parameter $\lambda > 0$. In the boundary case of $\lambda = 0$, the estimate is the simple average. The larger λ becomes, the more ratings that only occur a few times are penalised. The estimates for $\hat{\mu}$, $\hat{b}_i(\lambda)$ and $\hat{b}_u(\lambda)$ are calculated successively:

$$\hat{\mu} = \frac{1}{N} \sum_{u,i} y_{u,i}.$$

$$\hat{b}_i(\lambda) = \frac{1}{\lambda + n_i} \sum_{u=1}^{n_i} (y_{u,i} - \hat{\mu}) \qquad (i = 1, \ldots, J).$$

$$\hat{b}_u(\lambda) = \frac{1}{\lambda + J_U} \sum_{i=1}^{J_u} \left(y_{u,i} - \hat{\mu} - \hat{b}_i \right) \qquad \text{for all } u.$$

In addition to simply combining the effects, the parameter λ is tuned with k-fold cross-validation as explained at the beginning of subsection 3.5 in these steps:

1. Repeat for $l = 1, 2, \ldots, k = 25$:
   a) Estimate parameters $\hat{\mu}$, $\hat{b}_i(\lambda)$ and $\hat{b}_u(\lambda)$ on the l-th training set.
   b) Calculate estimated ratings $\hat{y}_{u,i}(\lambda)$ on the l-th validation set.
   c) Use estimated ratings to compute $RMSE_l(\lambda)$ on the l-th validation set.
2. Estimate the RMSE as $\hat{RMSE}(\lambda) = \frac{1}{k} \sum_{l=1}^{k} RMSE_l(\lambda)$.

Then repeat steps 1 through 2 for several values of λ to find the λ that minimises the (estimated) $RMSE$. Note that the final test set (which is confusingly named `validation`) is **not** used in this calculation to minimise the RMSE.

Steps 1 and 2 of this algorithm are implemented in the following code:

```r
#' Calculate RMSE using k-fold cross validation
#' for regularised movie-user effect with regularisation parameter lambda
#'
#' @param data Complete training data, to be split into (sub-)training
#'        and validation data.
#' @param ind_list List of $k$ indices of *validation* set.
#' @param lambda Regularisation parameter $\lambda$.
#' @return Vector of RMSEs of length $k$
RMSE_movieuser_kfold <- function(data, ind_list, lambda) {
    n <- length(ind_list)
    # iterate over indices
    rmse_v <- sapply(1:n, function (listIdx) {
        # split data int training and validation set
        train_set <- data[-ind_list[[listIdx]], ]
        validation_set <- data[ind_list[[listIdx]], ]

        # use training set for regularised movie + user effects
        mu <- mean(train_set$rating)
        b_i_tbl <- train_set %>%
            group_by(movieId) %>%
            summarise(b_i = sum(rating - mu)/(n()+lambda))
        b_u_tbl <- train_set %>%
            left_join(b_i_tbl, by="movieId") %>%
            group_by(userId) %>%
            summarise(b_u = sum(rating - b_i - mu)/(n()+lambda))

        # modify validation set so all movies and users from training
        # set are contained in it
        validation_set <- validation_set %>%
            semi_join(train_set, by = "movieId") %>%
            semi_join(train_set, by = "userId")

        # if nothing left, can already return NA
        if(length(validation_set) == 0) {
            return(NA)
        } else {
            # otherwise estimate ratings as \mu + b_i + b_u
            ratings_hat <-
                validation_set %>%
                left_join(b_i_tbl, by = "movieId") %>%
                left_join(b_u_tbl, by = "userId") %>%
                mutate(pred = mu + b_i + b_u) %>%
                .$pred
            #  and finally, calculate RMSE
            return(RMSE(ratings_hat, validation_set$rating))
            }
        })
    return(rmse_v)
    }
```

Then the RMSE estimation is repeated for several values of $\lambda = 0, 0.25, \ldots, 10.0$:

```r
# define sequence of $\lambda = 0, 0.25, \ldots, 10.0$
lambdas <- seq(0, 10, 0.25)
```

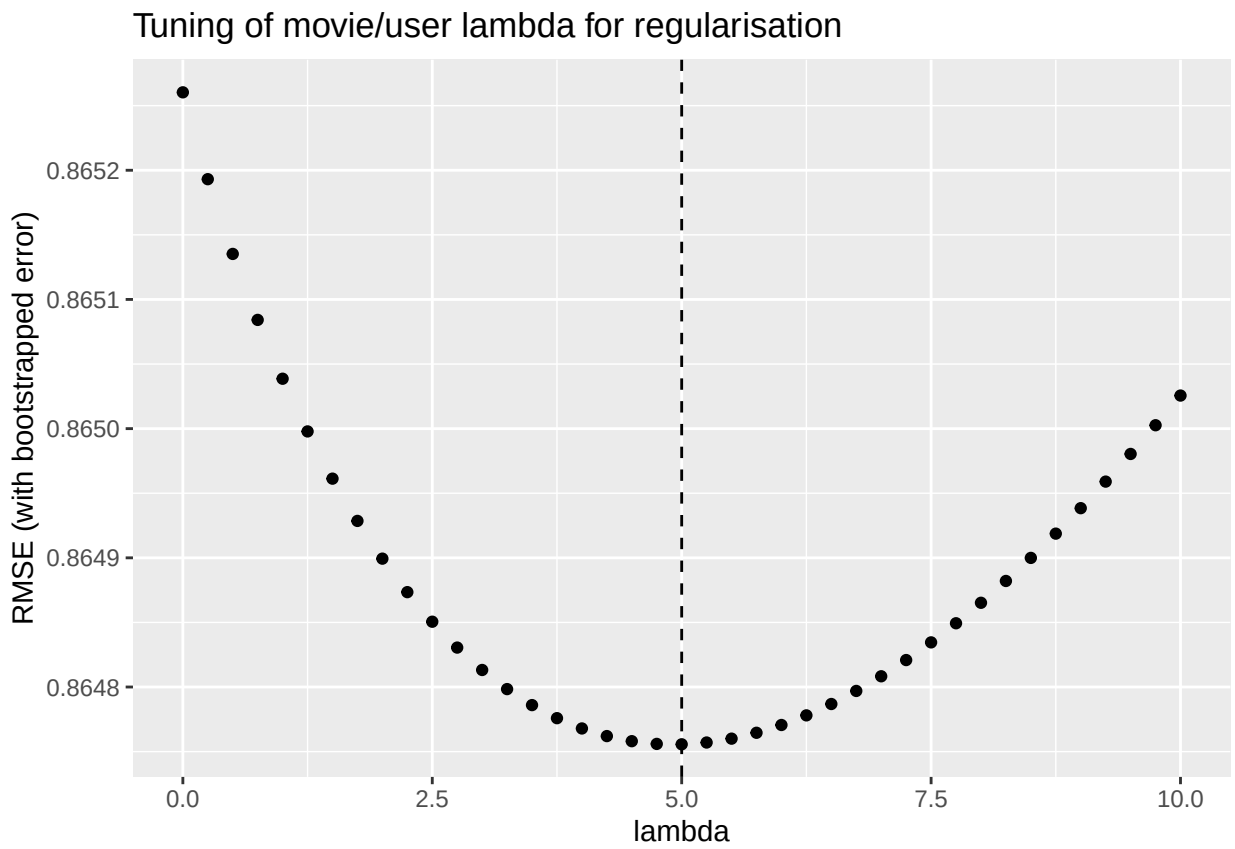
```
# calculate RMSEs for all these values of $\lambda$
RMSE_data_mur <- map_df(lambdas, function(lambda) {
    # calculate vector of RMSEs
    rmse_vec <- RMSE_movieuser_kfold(edx, index_list, lambda)
    # strip out the NA values
    rmse_vec <- na.omit(rmse_vec)
    # calculate mean and standard deviation of RMSEs
    list(RMSE_avg = mean(rmse_vec),
         RMSE_sd = sd(rmse_vec))
    })

# add lambda as first columns
RMSE_data_mur <- cbind(data.frame(lambda = lambdas),
                       RMSE_data_mur)

# look up lambda for which the RMSE is minimal
lambda_mur_opt <- lambdas[which.min(RMSE_data_mur$RMSE_avg)]
RMSE_lambda_opt <- RMSE_data_mur$RMSE_avg[which.min(RMSE_data_mur$RMSE_avg)]
```

Note that regularisation RMSE data are loaded from the file `RMSE_data_mur.Rdata` generated by the R script that accompanies this report, as re-running the tuning takes several hours:

```
load(file = "data//RMSE_data_mur.Rdata")
```

Plotting the RMSE for these values of λ shows that the optimal values is $\lambda_{mur} = 5$.

```
RMSE_data_mur %>%
    ggplot(aes(x = lambda, y = RMSE_avg)) +
    geom_point() +
    geom_vline(xintercept = lambda_mur_opt, linetype = "dashed") +
    labs(y = "RMSE (with bootstrapped error)",
         title = "Tuning of movie/user lambda for regularisation")
```



Finally, since the optimal values of λ has been determined, the prediction function is implemented:

```
#' Prediction function for regularised movie/user effect
#'
#' @param newdata Data for which to predict the ratings.
#' @param lambda Regularisation parameter $\lambda$.
#' @return Predicted ratings
predict_regmovieuser <- function(newdata, lambda) {
    # overall mean on training data
    mu <- mean(edx$rating)
    # tabulate movie effects "b_i" with regularisation
    b_i_tbl <- edx %>%
        group_by(movieId) %>%
        summarise(b_i = sum(rating - mu)/(n()+lambda))
    # tabulate user effects "b_u" with regularisation
    b_u_tbl <- edx %>%
        left_join(b_i_tbl, by="movieId") %>%
        group_by(userId) %>%
        summarise(b_u = sum(rating - b_i - mu)/(n()+lambda))
    # calculate rating prediction by looking up "b_i" and "b_u"
    # from the tables just created and compute on the NEW data
    # \mu + b_i + b_u
    pred_ratings <-
        newdata %>%
        left_join(b_i_tbl, by = "movieId") %>%
        left_join(b_u_tbl, by = "userId") %>%
        mutate(pred = mu + b_i + b_u) %>%
```

```
        .$pred
    return(pred_ratings)
    }
```

3.5.6 Additional Time Effect

After movie and user effect had been stripped out, an additional time effect was observed in Section 3.4.4. The model then becomes

$$Y_{u,i} = \mu + b_i + b_u + f(d_{u,i}) + \varepsilon_{u,i} \qquad (i = 1, \ldots, J; u = 1, \ldots, n_i),$$

where $d_{u,i}$ is the date rounded to weeks of the rating and f is a smooth function. Since the plotting already used LOESS, the same method will be used here, see also subsubsection 3.1.2 The model information for the `gamLoess` method

```
modelLookup("gamLoess")
```

```
##      model parameter  label forReg forClass probModel
## 1 gamLoess      span   Span   TRUE     TRUE      TRUE
## 2 gamLoess    degree Degree   TRUE     TRUE      TRUE
```

says that `span` and `degree` can be tuned. The degree of the local regression is fine to leave at 1 (linear), and the span will be varied. The tuning of the `span` parameters is achieved with the `train` function of the `caret` package, using the values for $d_{u,i}$ calculated in paragraph 3.4.4.2. Note that the $d_{u,i}$ values are the residual time effects once regularised movie and user effect have already been taking into account

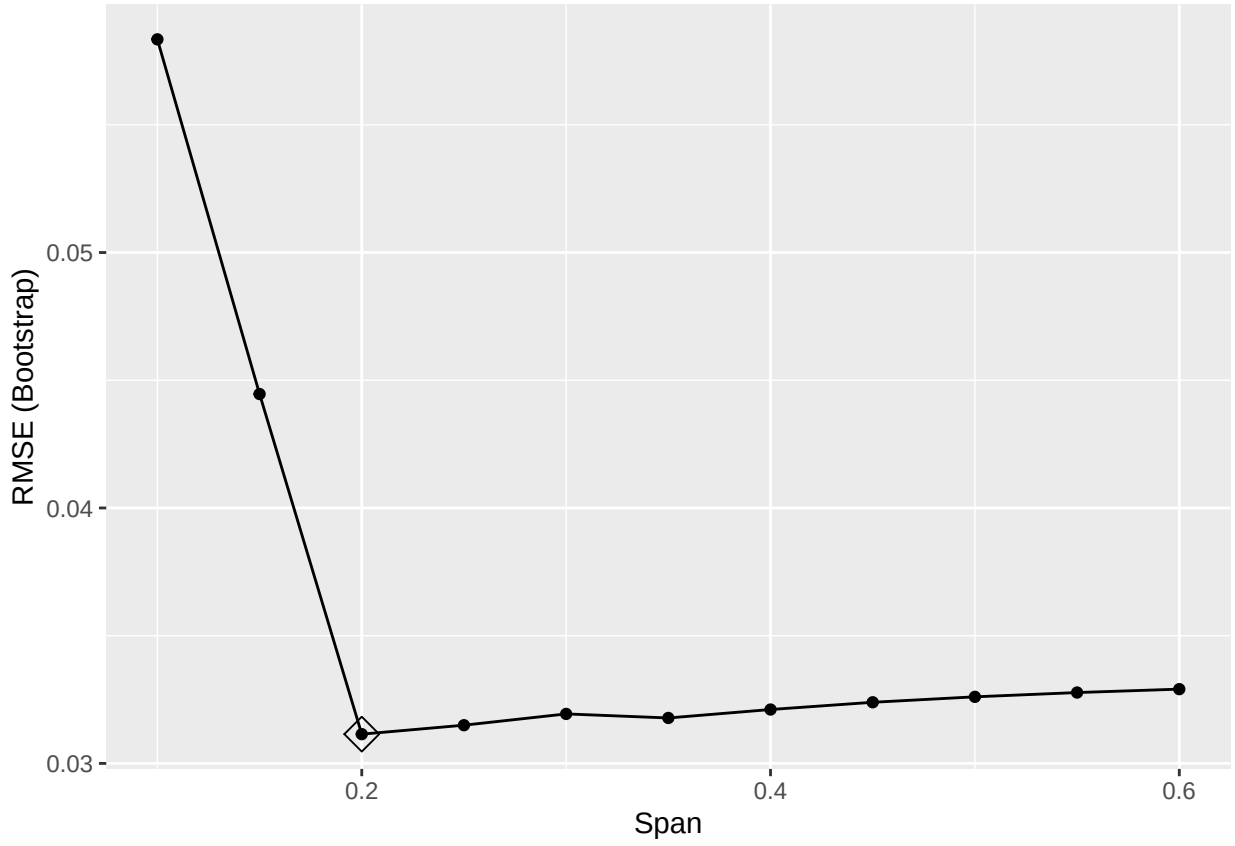
```
train_loess <- train(d_ui ~ rating_wk, data = d_ui_tbl,
                     method = "gamLoess",
                     tuneGrid = data.frame(degree = 1,
                                           span = seq(0.10, 0.60, 0.05)))
```

Note that since this tuning takes some time, the results are stored for this report and loaded from the file `train_loess.Rdata`:

```
load(file = "data//train_loess.Rdata")
```

The optimal span parameters is $span_{opt} = 0.2$ as can be seen from the plot of the RMSE against this parameter:

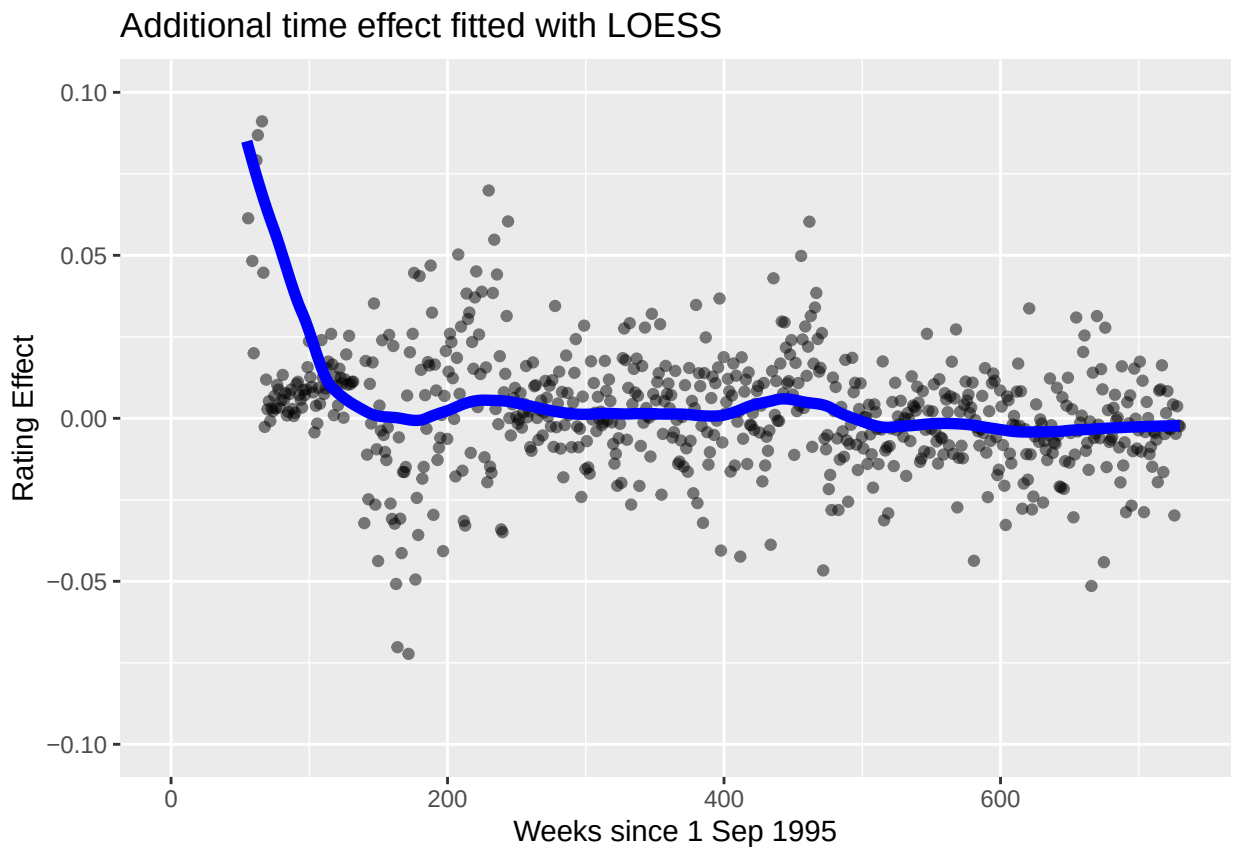
```
# plot and look up best span parameter
ggplot(train_loess, highlight = TRUE)
```




```
span_opt <- train_loess$bestTune$span
```

The fit of the smoothing loess function through the residual time effect is illustrated in the following plot.

```
# plot fit of best model to time effect
d_ui_tbl %>%
    mutate(d_ui_hat = predict(train_loess, newdata = .)) %>%
    ggplot(aes(x = as.numeric(rating_wk))) +
    geom_point(aes(y = d_ui), alpha = 0.5) +
    geom_line(aes(y = d_ui_hat), col = "blue", size = 2) +
    ylim(c(-0.1, 0.1)) +
    labs(title = "Additional time effect fitted with LOESS",
         x = "Weeks since 1 Sep 1995",
         y = "Rating Effect")
```



Finally, the model for the prediction of the residual time effect is extracted from the training object. This object will later be used for the final prediction of the ratings on the `validation` data set.

```
# fit model for prediction of RESIDUAL time effect
fit_loess <- train_loess$finalModel

# prediction function for regularised movie and user effect plus
# additional time effect

#' Prediction function for regularised movie/user and time effect
#'
#' @param newdata Data for which to predict the ratings.
#' @param lambda Regularisation parameter $\lambda$ for movie and user effects.
#' @param d_ui_loess LOESS model object to predict residual time effect
#'        based on week in which rating was awared.
#' @return Predicted ratings
predict_rmu_t <- function(newdata, lambda, d_ui_loess) {
    # overall mean on training data
    mu <- mean(edx$rating)
    # tabulate movie effects "b_i" with regularisation and given lambda
    b_i_tbl <- edx %>%
        group_by(movieId) %>%
        summarise(b_i = sum(rating - mu)/(n()+lambda))
    # tabulate user effects "b_u" with regularisation and given lambda
    b_u_tbl <- edx %>%
        left_join(b_i_tbl, by="movieId") %>%
        group_by(userId) %>%
```

```
        summarise(b_u = sum(rating - b_i - mu)/(n()+lambda))

    # calculate rating prediction by looking up "b_i" and "b_u",
    # and taking the model $f(d_ui)$ provided by d_ui_loess
    # from the tables just created and compute on the NEW data
    # $\mu + b_i + b_u + f(d_ui)$
    pred_ratings <-
        newdata %>%
        left_join(b_i_tbl, by = "movieId") %>%
        left_join(b_u_tbl, by = "userId") %>%
        mutate(ratingdate_wk = round_date(ratingdate, unit = "week")) %>%
        mutate(rating_wk = (ratingdate_wk - min_date)/7) %>%
        mutate(f_d_ui = predict(d_ui_loess, newdata = .)) %>%
        mutate(pred = mu + b_i + b_u + f_d_ui) %>%
        .$pred
    return(pred_ratings)
    }
```

3.5.7 Additional Genre Effect

As established in subsubsection 3.4.5, genres have an impact on the rating. A model that contains the movie and user effect in addition to the genre effect is of the following form:

$$Y_{u,i} = \mu + b_i + b_u + \sum_{k=1}^{K} \delta_{i,k}\beta_k + \varepsilon_{u,i} \qquad (i = 1, \ldots, J; u = 1, \ldots, n_i),$$

where μ is the overall rating average, b_i the movie effect of movie i, b_u the user effect of user u and β_k the effect of genre k. The $\delta_{i,k}$ is an indicator function

$$\delta_{i,k} = \begin{cases} 1, & \text{if movie } i \text{ has genre } k. \\ 0, & \text{otherwise.} \end{cases}$$

The estimation of all these effects is done successively, as detailed in subsubsection 3.5.5, with the optimal regularisation parameter $\lambda_{mur} = 5$ for $\hat{b}_i$ and $\hat{b}_u$. In addition, the genre effects β_k are estimated as with a separate regularisation parameter λ_g as

$$\hat{\beta}_k = \frac{1}{\lambda_g + J_U} \sum_{i=1}^{J_u} \delta_{i,k} \left(y_{u,i} - \hat{\mu} - \hat{b}_i - \hat{b}_u \right)$$

The k-fold cross validation follows the same algorithm described in subsubsection 3.5.5:

- Estimate $\hat{RMSE}(\lambda_g)$ for $\lambda_g = 0, 0.25, \ldots, 10.0$ in k-fold cross-validation with $k = 25$.
- Select the λ_g for which the $\hat{RMSE}(\lambda_g)$ is minimal.

The algorithm is implemented by the following code:

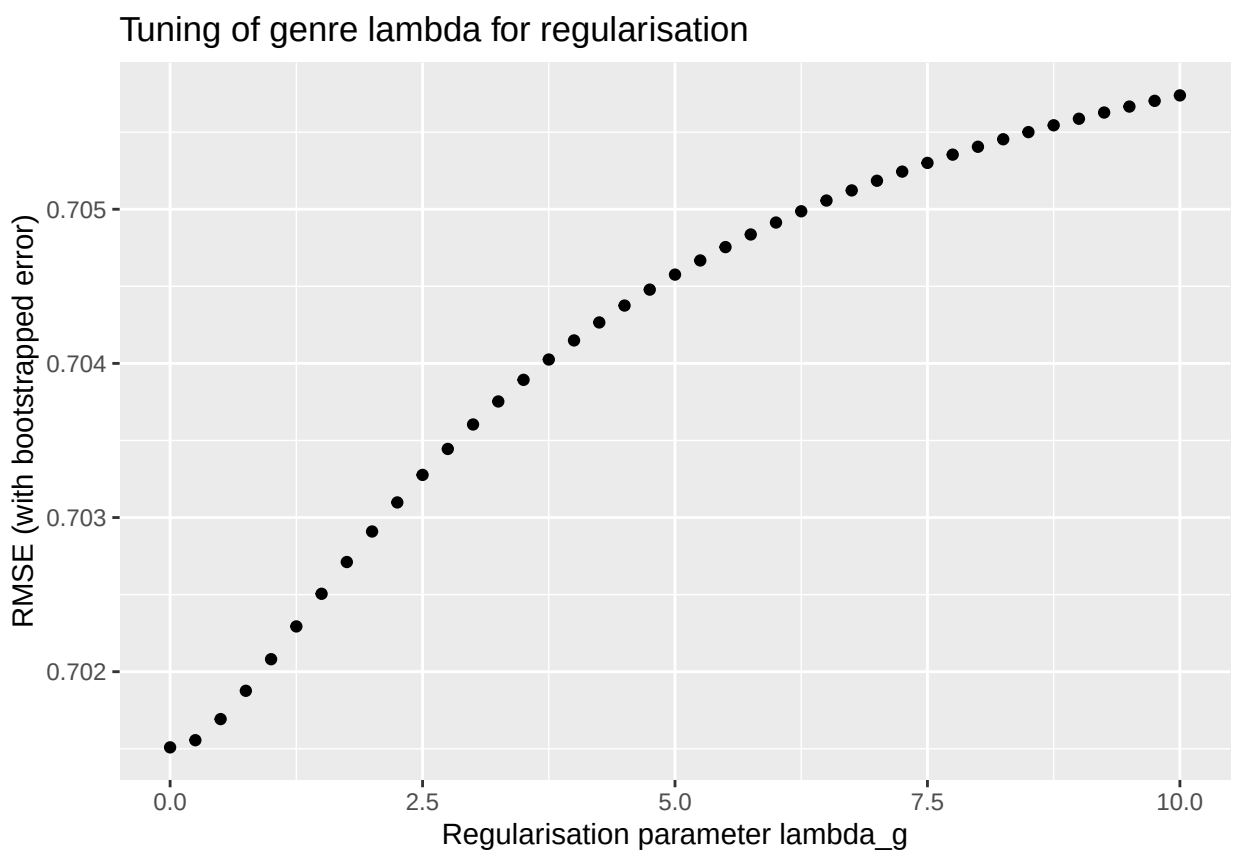
```
#' Calculate RMSE using k-fold cross validation
#' for regularised genre effect with regularisation parameter lambda
#' based on already regularised movie-user effect
#'
#' @param data Complete training data, to be split into (sub-)training
#'        and validation data.
#' @param ind_list List of $k$ indices of *validation* set.
#' @param lambda_mur Regularisation parameter $\lamdba$ for the regularised
```

```
#'        movie and user effects.
#' @param lambda_g Regularisation parameter $\lambda$ for the genre effect.
#' @return Vector of RMSEs of length $k$
RMSE_rmu_g_kfold <- function(data, ind_list, lambda_mur, lambda_g) {
    k <- length(ind_list)
    # iterate over indices
    rmse_v <- sapply(1:k, function (listIdx) {
        # split data int training and validation set
        train_set <- data[-ind_list[[listIdx]], ]
        validation_set <- data[ind_list[[listIdx]], ]
        # use training set for regularised movie + user effects
        mu <- mean(train_set$rating)
        b_i_tbl <- train_set %>%
            group_by(movieId) %>%
            summarise(b_i = sum(rating - mu)/(n()+lambda_mur))
        b_u_tbl <- train_set %>%
            left_join(b_i_tbl, by="movieId") %>%
            group_by(userId) %>%
            summarise(b_u = sum(rating - b_i - mu)/(n()+lambda_mur))
        # also add on genre effect
        g_ik_tbl <- train_set %>%
            left_join(b_i_tbl, by = "movieId") %>%
            left_join(b_u_tbl, by = "userId") %>%
            group_by(genres) %>%
            summarise(g_ik = mean(rating - mu - b_i - b_u)/(n()+lambda_g))

        # modify validation set so all moviess, users and genres from training
        # set are contained in it
        validation_set <- validation_set %>%
            semi_join(train_set, by = "movieId") %>%
            semi_join(train_set, by = "userId") %>%
            semi_join(train_set, by = "genres")

        # if nothing left, can already return NA
        if(length(validation_set) == 0) {
            return(NA)
        } else {
            # otherwise estimate ratings as \mu + b_i + b_u + \sum_k g_ik
            ratings_hat <-
                validation_set %>%
                left_join(b_i_tbl, by = "movieId") %>%
                left_join(b_u_tbl, by = "userId") %>%
                left_join(g_ik_tbl, by = "genres") %>%
                mutate(pred = mu + b_i + b_u + g_ik) %>%
                .$pred
            #  and finally, calculate RMSE
            return(RMSE(ratings_hat, validation_set$rating))
            }
        })
    return(rmse_v)
    }

# apply RMSE estimation for a list of lambdas
```

```
lambdas <- seq(0, 10, 0.25)
RMSE_data_g <- map_df(lambdas, function(lambda) {
    # calculate vector of RMSEs
    rmse_vec <- RMSE_rmu_g_kfold(edx, index_list, lambda_mur_opt,
                                 lambda)
    # strip out the NA values
    rmse_vec <- na.omit(rmse_vec)
    # calculate mean and standard deviation of RMSEs
    list(RMSE_avg = mean(rmse_vec),
         RMSE_sd = sd(rmse_vec))
    })

# add lambda as first column
RMSE_data_g <- cbind(data.frame(lambda = lambdas),
                     RMSE_data_g)
```

Since this tuning runs several hours, the results are saved to the file `RMSE_data_g.Rdata` and loaded from there for this report to plot the estimated RMSE against λ_g:

```
load(file = "data//RMSE_data_g.Rdata")
RMSE_data_g %>%
    ggplot(aes(x = lambda, y = RMSE_avg)) +
    geom_point() +
    labs(x = "Regularisation parameter lambda_g",
         y = "RMSE (with bootstrapped error)",
         title = "Tuning of genre lambda for regularisation")
```



```
# and select optimal lambda
lambda_g_opt <- RMSE_data_g$lambda[which.min(RMSE_data_g$RMSE_avg)]
```

The plot reveals that the optimal regularisation parameter is $\lambda_{g,opt}$ = `lambda_g_opt`, proving that no regularisation was necessary for this particular set of training data `edx`, but this may not always be the case.

Finally, the prediction of the model with regularised movie, user and genre effect is implemented by the following function which takes two different regularisation parameters, λ_{mur} for the movie and user effects and λ_g for the genre effect.

```
#' Prediction function for regularised movie/user and genre effect
#'
#' @param newdata Data for which to predict the ratings.
#' @param lambda_mur Regularisation parameter $\lambda_{mur}$ for movie and user
#'        effects.
#' @param lamdba_g Regularisation parameter $\lambda_g$ for genre effect.
#' @return Predicted ratings
predict_rmu_g <- function(newdata, lamdba_mur, lambda_g) {
    # use total training set to first calculate overall average rating $\mu$
    mu <- mean(edx$rating)

    # Then construct table of regularised movie effects
    b_i_tbl <- edx %>%
        group_by(movieId) %>%
        summarise(b_i = sum(rating - mu)/(n()+lambda_mur))

    # Then tabulate the regularised user effects
    b_u_tbl <- edx %>%
        left_join(b_i_tbl, by="movieId") %>%
        group_by(userId) %>%
        summarise(b_u = sum(rating - b_i - mu)/(n()+lambda_mur))

    # Last, tabulate the regularised genre effect
    g_ik_tbl <- edx %>%
        left_join(b_i_tbl, by = "movieId") %>%
        left_join(b_u_tbl, by = "userId") %>%
        group_by(genres) %>%
        summarise(g_ik = mean(rating - mu - b_i - b_u)/(n()+lambda_g))

    # With that, look up all three effect (movie, user, genre) to compute
    # $\hat{y}_{u,i} = \mu + b_i + b_u + #\sum_{k=1}^k \delta_{i,k}\beta_k$
    ratings_hat <-
        newdata %>%
        left_join(b_i_tbl, by = "movieId") %>%
        left_join(b_u_tbl, by = "userId") %>%
        left_join(g_ik_tbl, by = "genres") %>%
        mutate(pred = mu + b_i + b_u + g_ik) %>%
        .$pred

    return(ratings_hat)
    }
```

4 Results

The RMSEs are estimated for each of the seven models presented in subsection 3.5 and collected in a table for comparison. All values are computed to the full seven digits to show also marginal differences in performance. The code follows these steps for each method:

1. Calculate estimate according to the method on the `validation` data set.
2. Compute RMSE for this estimate on the `validation` data set.
3. Store results in table `rmse_results`.

```
# 1 Constant Value
ratings_hat_const <- predict_const(validation)
rmse_const <- RMSE_rating(ratings_hat_const, validation$rating)
# create table with first row of results
rmse_results <- tibble(Method = "Simple Average",
                       RMSE = rmse_const)

# 2 Movie Effect Only
ratings_hat_movieb <- predict_movieb(validation)
rmse_movieb <- RMSE_rating(ratings_hat_movieb, validation$rating)
rmse_results <- rbind(rmse_results,
                      tibble(Method = "Movie effect",
                             RMSE = rmse_movieb))

# 3 User Effect Only
ratings_hat_userb <- predict_userb(validation)
rmse_userb <- RMSE_rating(ratings_hat_userb, validation$rating)
rmse_results <- rbind(rmse_results,
                     tibble(Method = "User effect",
                            RMSE = rmse_userb))

# 4 Combined Movie and User Effects
ratings_hat_movieuserb <- predict_movieuserb(validation)
rmse_movieuserb <- RMSE_rating(ratings_hat_movieuserb, validation$rating)
rmse_results <- rbind(rmse_results,
                      tibble(Method = "Combined Movie and User effect",
                             RMSE = rmse_movieuserb))

# 5 Regularised Movie and User Effect
ratings_hat_regmovieuser <- predict_regmovieuser(validation, lambda_mur_opt)
rmse_regmovieuser <- RMSE_rating(ratings_hat_regmovieuser, validation$rating)
rmse_results <- rbind(rmse_results,
                      tibble(Method = "Regularised Movie and User effect",
                             RMSE = rmse_regmovieuser))

# 6 Regularised Movie and User Effect, Additional Time Effect
ratings_hat_rmu_t <- predict_rmu_t(validation, lambda_mur_opt, fit_loess)
rmse_rmu_t <- RMSE_rating(ratings_hat_rmu_t, validation$rating)
rmse_results <-
    rbind(rmse_results,
          tibble(Method = "Reg. Movie and User effect, add. time effect",
                 RMSE = rmse_rmu_t))

# 7 Regularised Movie and User Effect, Additional Genre Effect
ratings_hat_rmu_g <- predict_rmu_g(validation, lambda_mur_opt, lambda_g_opt)
```

```
rmse_rmu_g <- RMSE_rating(ratings_hat_rmu_g, validation$rating)
rmse_results <-
    rbind(rmse_results,
          tibble(Method = "Reg. Movie, User and Genre effect",
                 RMSE = rmse_rmu_g))

# output table of results
knitr::kable(rmse_results)
```

Method	RMSE
Simple Average	1.0612018
Movie effect	0.9439087
User effect	0.9783360
Combined Movie and User effect	0.8653488
Regularised Movie and User effect	0.8648177
Reg. Movie and User effect, add. time effect	0.8649112
Reg. Movie, User and Genre effect	0.8648170

The mathematical formulation of the model is shown in the following overview:

Method	Model Formula
Simple Average	$Y_{u,i} = \mu + \varepsilon_{u,i}$
Movie effect	$Y_{u,i} = \mu + b_i + \varepsilon_{u,i}$
User effect	$Y_{u,i} = \mu + b_u + \varepsilon_{u,i}$
Combined Movie and User effect	$Y_{u,i} = \mu + b_i + b_u + \varepsilon_{u,i}$
Regularised Movie and User effect	$Y_{u,i} = \mu + b_i + b_u + \varepsilon_{u,i}$
Reg. Movie and User effect, add. time effect	$Y_{u,i} = \mu + b_i + b_u + f(d_{u,i}) + \varepsilon_{u,i}$
Reg. Movie, User and Genre effect	$Y_{u,i} = \mu + b_i + b_u + \sum_{k=1}^{K} \delta_{i,k}\beta_k + \varepsilon_{u,i}$

As expected, the simple average performs worst as it contains no specific information from the movies or users. The incorporation of the movie or the user effects each provide an improvement. The movie effect alone performs better than the user effect as there is higher variability in the user effect, as observed in subsubsection 3.4.3.

Combining the movie and user effects has a large incremental effect. That the regularisation only leads to a marginal improvement is due to the fact that the vast majority of movies have 10 or more ratings (see also paragraph 3.4.1.2).

Using the regularised movie and user effect model as building block to add more information it can be observed that:

- Adding the time of the rating actually produces a worse performance than leaving it out.
- Adding the genre of the movie leads to only a slight improvment in performance, but does result in the **best-performing model** with an RMSE of 0.8648.

5 Conclusion

With elementary methods such as regularised average and local regression (LOESS), ratings can be forecast with an accuracy of 0.8648 stars (as measured by RMSE) based on just over 1% of the total combinations of movie and user ratings using only the elementary information of movie and user effects. The genre carries only little additional information, while adding the time of the rating even slightly deteriorates the performance.

So the overall best model among those tested is the regularised movie, user and genre effect, with the same regularisation parameter for movie and user and a different regularisation parameter for the genre:

$$Y_{u,i} = \mu + b_i + b_u + \sum_{k=1}^{K} \delta_{i,k}\beta_k + \varepsilon_{u,i}$$

The RMSE is 0.8648.

It may be possible to boost the performance further by

- Separating the genres that have been used cumulatively into their constituents.
- Performing a principle component analysis (PCA) of the matrix of ratings arranged by user for rows and movie for columns and taking the first components as additional factors in the model. This may become unwieldy, though since such a matrix has about 750 million entries.

Appendix A - Code of Figures

To enhance readibility of the report, the code for most figures in Section 3.4 is shown in this Appendix.

Figures in Section 3.4.1 - General Data Distribution

Code for *data sparsity* figure shown here:

```
edx_1000 %>%
    mutate(rating = 1) %>%
    select(movieId, userId, rating) %>%
    spread(movieId, rating) %>%
    column_to_rownames(var = "userId") %>%
    as.matrix() %>% t() %>%
    image(x = 1:n_movies_sub, y = 1:n_users_sub, z = .,
          xlab = "Movies", ylab = "Users",
          col = grey.colors(n = 1, start = 0, end = 1))
```

Code for *ratings counts by movie* figure shown here:

```
edx %>%
    group_by(movieId) %>%
    summarise(n = n()) %>%
    ggplot(aes(x = n)) +
    scale_x_log10() +
    geom_histogram(bins = 30, col = "black") +
    labs(title = "Rating counts by movie",
         x = "Number of Ratings (log scale)")
```

Code for *rating counts by user* figure shown here:

```
edx %>%
    group_by(userId) %>%
    summarise(n = n()) %>%
    ggplot(aes(x = n)) +
    scale_x_log10() +
    geom_histogram(bins = 40, col = "black") +
    labs(title = "Rating counts by user",
         x = "Number of Ratings (log scale)")
```

Code for *distribution of ratings - total* histogram shown here:

```
edx %>%
    count(rating) %>%
    ggplot(aes(x = factor(rating), y = n)) +
    geom_bar(stat = "identity", width = 1, col = "black")+
    labs(title = "Distribution of ratings - total",
         x = "Rating", y = "Count")
```

Figure in Section 3.4.2 - Movie Effect

Code for *movie effect* chart shown here:

```
# distribution of ratings by movie
movie_avgs <- edx %>%
    group_by(movieId) %>%
    summarise(movie_avg = mean(rating),
              movie_sd = ifelse(n()>1,sd(rating),0),
              n_ratings_bymovie = n()) %>%
    arrange(movie_avg) %>%
    mutate(row = row_number(movie_avg))

# plot average of ratings and their
# standard deviation, horizontally aligned
p1 <- movie_avgs %>%
    ggplot(aes(x = row, y = movie_avg)) +
    geom_point() +
    labs(x = "Movie (sorted by average rating)",
         y = "Average Rating", title = "Ratings by movie")
p2 <- movie_avgs %>%
    ggplot(aes(x = row, y = movie_sd)) +
    geom_point() +
    labs(x = "Movie (sorted by average rating)",
         y = "Rating Standard Deviation")
grid.arrange(p1, p2, nrow = 2)
```

Figure in Section 3.4.3 - User Effect

Code for *user effect* chart shown here:

```
# distribution of ratings by user - in appendix
user_avgs <- edx %>%
    group_by(userId) %>%
    summarise(user_avg = mean(rating),
              user_sd = ifelse(n()>1,sd(rating),0),
              n_ratings_byuser = n()) %>%
    arrange(user_avg) %>%
    mutate(row = row_number(user_avg))

# plot average of ratings and their
# standard deviation, horizontally aligned
p1 <- user_avgs %>%
    ggplot(aes(x = row, y = user_avg)) +
    geom_point() +
    labs(x = "User (sorted by average rating)",
         y = "Average Rating", title = "Ratings by user")
p2 <- user_avgs %>%
    ggplot(aes(x = row, y = user_sd)) +
```

```
    geom_point() +
    labs(x = "User (sorted by average rating)",
         y = "Rating Standard Deviation")
grid.arrange(p1, p2, nrow = 2)
```

Figures in Section 3.4.4 - Time of Rating

Code for *time effect* chart shown here:

```
# tabulate rating by time
 edx %>%
    mutate(ratingdate_wk = round_date(ratingdate, unit = "week")) %>%
    group_by(ratingdate_wk) %>%
    summarize(rating_wk = mean(rating))
    ggplot(aes(x = ratingdate_wk, y = rating_wk)) +
    geom_point() +
    geom_smooth()
```

Code for *residual time effect* once movie and user effect are stripped out, in chart shown here:

```
min_date = min(edx$ratingdate)
lambda_mur <- 5.0
# Calculate regularised movie user and time effect, see
mu <- mean(edx$rating)
# Movie effect
b_i_tbl <- edx %>%
    group_by(movieId) %>%
    summarise(b_i = sum(rating - mu)/(n()+lambda_mur))
# User effect
b_u_tbl <- edx %>%
    left_join(b_i_tbl, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+lambda_mur))
# Residual time effect
d_ui_tbl <- edx %>%
    mutate(ratingdate_wk = round_date(ratingdate, unit = "week")) %>%
    mutate(rating_wk = (ratingdate_wk - min_date)/7) %>%
    left_join(b_i_tbl, by = "movieId") %>%
    left_join(b_u_tbl, by = "userId") %>%
    group_by(rating_wk) %>%
    summarise(d_ui = mean(rating - mu - b_i - b_u))

# plot the overall time effect once movie and user effect are accounted for
d_ui_tbl %>%
    ggplot(aes(x = as.numeric(rating_wk), y = d_ui)) +
    geom_point() +
    geom_smooth(formula = y ~ x, method = "loess") +
    labs(title = "Residual time effect of ratings",
```

```
         x = paste0("Weeks since ",format(min_date, "%d %b %Y")),
         y = "Residual rating effect (stars)")
```

Figures in Section 3.4.5 - Genre Effect

The code for the first chart in subsubsection 3.4.5 is:

```
edx %>%
    group_by(genres) %>%
    summarise(rating_avg = mean(rating),
              nratings = n()) %>%
    arrange(nratings) %>%
    ggplot(aes(x = nratings)) +
    scale_x_log10() +
    geom_histogram(bins = 30, col = "black")
```

The code for the second chart is:

```
# First chart - average ratings by genres, in ascending order
p1 <- edx %>%
    select(genres, rating) %>%
    mutate(genres = reorder(genres, rating, mean)) %>%
    group_by(genres) %>%
    summarise(rating_avg = mean(rating),
              rating_sd = sd(rating),
              nratings = n()) %>%
    filter(nratings >= 1000) %>%
    ggplot(aes(x = genres, y = rating_avg)) +
    scale_y_log10() +
    theme(axis.text.x = element_text("")) +
    geom_point() +
    labs(title = "Ratings by genres (with at least 1000 ratings)",
         x = "Genre (sorted by average rating)",
         y = "Average Rating")

# Second chart - standard dev of ratings by genre, genres need to
# be in same order as in first chart
p2 <- edx %>%
    select(genres, rating) %>%
    mutate(genres = reorder(genres, rating, mean)) %>%
    group_by(genres) %>%
    summarise(rating_avg = mean(rating),
              rating_sd = sd(rating),
              nratings = n()) %>%
    filter(nratings >= 1000) %>%
    ggplot(aes(x = genres, y = rating_sd)) +
    scale_y_log10() +
    theme(axis.text.x = element_text("")) +
```

```
    geom_point() +
    labs(title = "",
         x = "Genre (sorted by average rating)",
         y = "Rating Standard Deviation")

grid.arrange(p1, p2, nrow = 2)
```