HarvardX - Data Science

Marco Schicker

2021

Contents

0.1 Executive summary

The goal of this project is to predict movie ratings based on a set of predictors. The target is to create a model which predictions minimize the deviations compared with the true rating, i.e. a low root mean square deviation (RMSE). The means to do so are analyzing a data set provided and training a model to make predictions for a validation set of data.

The edx-data set provided consists of 9000055 movie ratings. Each rating is categorized by the following variables: * userID - a unique identifier per user * movieID - a unique identifier per movie * title - The movie title, including the release year of the movie * timestamp - The date and time when the movie was rated by the user * genres - a list of genres to categorize movies

In general the steps performed to fulfill the task are: * importing data * cleaning data * exploring and visualizing data * training different models and comparing performance * validating the most promising model on the validation set

In the end a Model was found with a RMSE of `{r final result } final_model_rmse <- RMSE(validation$rating, final_predict)` running on the validation data set.

0.2 Methods & Analysis

The process to train the models consists of the following basic steps: * data import * data exploration, cleaning and visualization * data mutation * creating training and test set * training and comparing different models * validation of best model

0.2.1 data import

The data used to train the model is derived from the publicly available movielens database. According to provided code a data set "edx" and a dataset "validation" is automatically created. The "edx"-data set is used to train a model, while the second data set "validation" is used to validate the chosen model.

0.2.2 data exploration, cleaning and visualization

In order to work with the data set it is analyzed and altered

0.2.2.1 Overall analysis To get a first overview about the data the following table is created:

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35738   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres
##  Length:9000055     Length:9000055
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

The dimensions of the data file are `{r, echo=FALSE} dim(edx)` The data classes are shown in the following table

```
##      userId     movieId      rating   timestamp       title      genres
##   "integer"   "numeric"   "numeric"   "integer" "character" "character"
```

We can see that the data set edx consists of `{r rows} nrow(edx)` rows and six columns with numerical and character variables. There are no NA-Values and on first sight plausible min and max values, so the data quality looks good.

```
head(edx)
```

```
##    userId movieId rating timestamp                         title
## 1:      1     122      5 838985046              Boomerang (1992)
## 2:      1     185      5 838983525               Net, The (1995)
## 3:      1     292      5 838983421               Outbreak (1995)
## 4:      1     316      5 838983392               Stargate (1994)
## 5:      1     329      5 838983392 Star Trek: Generations (1994)
## 6:      1     355      5 838984474       Flintstones, The (1994)
##                            genres
## 1:                Comedy|Romance
## 2:          Action|Crime|Thriller
## 3:  Action|Drama|Sci-Fi|Thriller
## 4:       Action|Adventure|Sci-Fi
## 5: Action|Adventure|Drama|Sci-Fi
## 6:       Children|Comedy|Fantasy
```

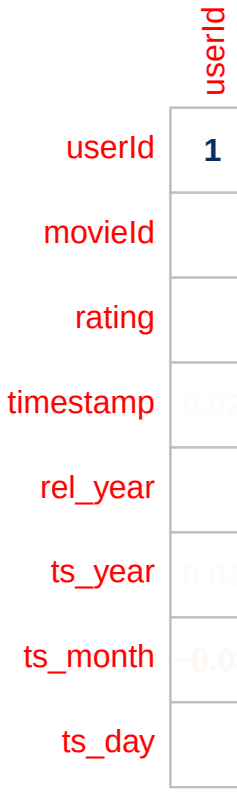
The column “title” contains the title and the release year of the movie. The columns “timestamp” is formatted as an integer, showing seconds since 1970-01-01. The data will be mutated to include the new columns: * “rel_year” - the release year of the movie, derived from the title column * “ts_year” - year of the timestamp of the rating * “ts_month” - month of the timestamp of the rating * “ts_day” - day of the timestamp of the rating

Since integer valus occupy only half space compared to numeric the columns with natural numbers will be converted to integer in order to reduce required data size for calculations later on.

The new summary of edx looks like this:

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35738   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres             rel_year       ts_year
##  Length:9000055     Length:9000055     Min.   :1915   Min.   :1995
##  Class :character   Class :character   1st Qu.:1987   1st Qu.:2000
##  Mode  :character   Mode  :character   Median :1994   Median :2002
##                                        Mean   :1990   Mean   :2002
##                                        3rd Qu.:1998   3rd Qu.:2005
##                                        Max.   :2008   Max.   :2009
##     ts_month          ts_day
##  Min.   : 1.000   Min.   :1.000
##  1st Qu.: 4.000   1st Qu.:2.000
##  Median : 7.000   Median :4.000
##  Mean   : 6.786   Mean   :3.906
##  3rd Qu.:10.000   3rd Qu.:6.000
##  Max.   :12.000   Max.   :7.000
```

0.2.2.2 Find duplicate entries Duplicate entries could pose a problem for any algorithm so we need to check for them. `{r} length(problems)` duplicate entries could be found.



0.2.2.3 Find predictors that are highly correlated and remove if necessary

As the correlation matrix shows there are only few correlated numeric variables that should be excluded from the analysis. ts_year is highly correlated with timestamp because it is directly derived from the timestamp. That means we can work with only one of the two. To speed up data handling the timestamp will be dropped from the edx and validation dataset. The movieId and the timestamp have a slight correlation which shows, that most ratings happen after and rather close to the release date of a movie. However, movieId and title are highly correlated, as every movie has it´s fixed title. Since we cannot rule out misspelling we will exclude the title and only go with movieId as a predictor. Title will be kept in the data set for check purposes before regularization.

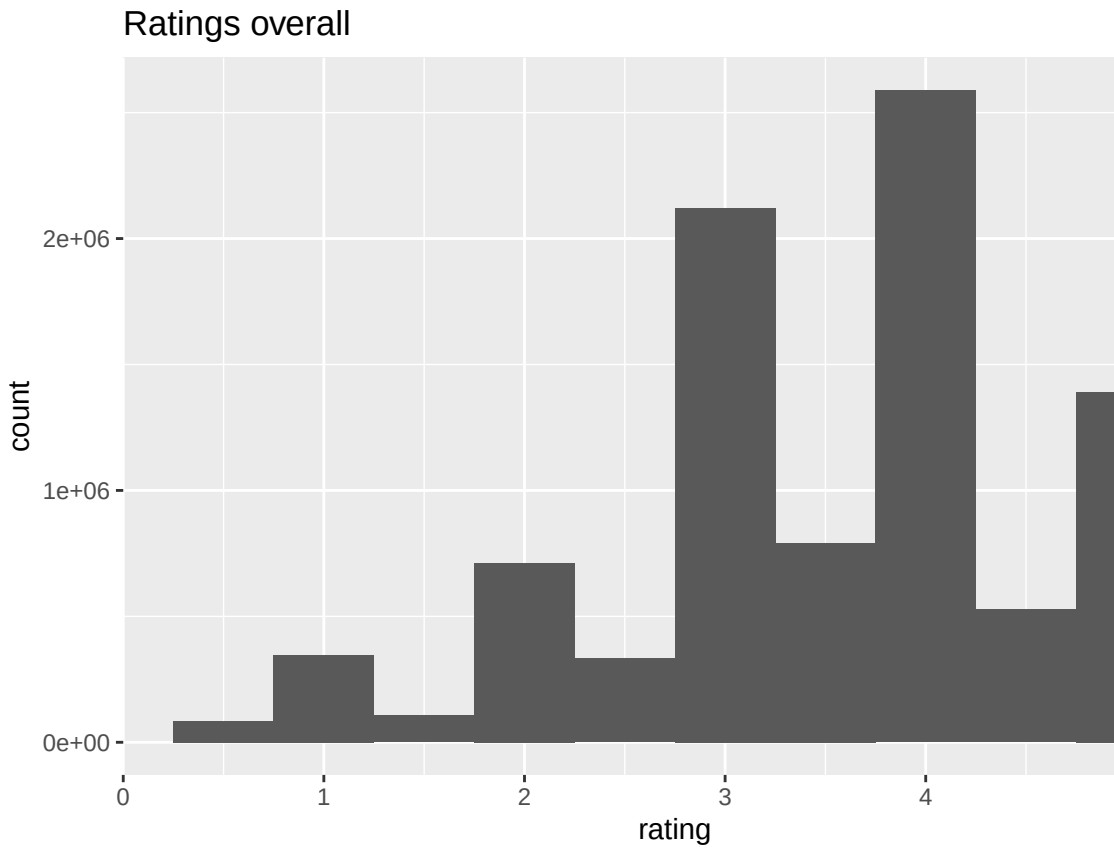
0.2.2.4 Remove predictors with near zero variation

Next we need to check if we have predictors in our data set that have no or very small variation.

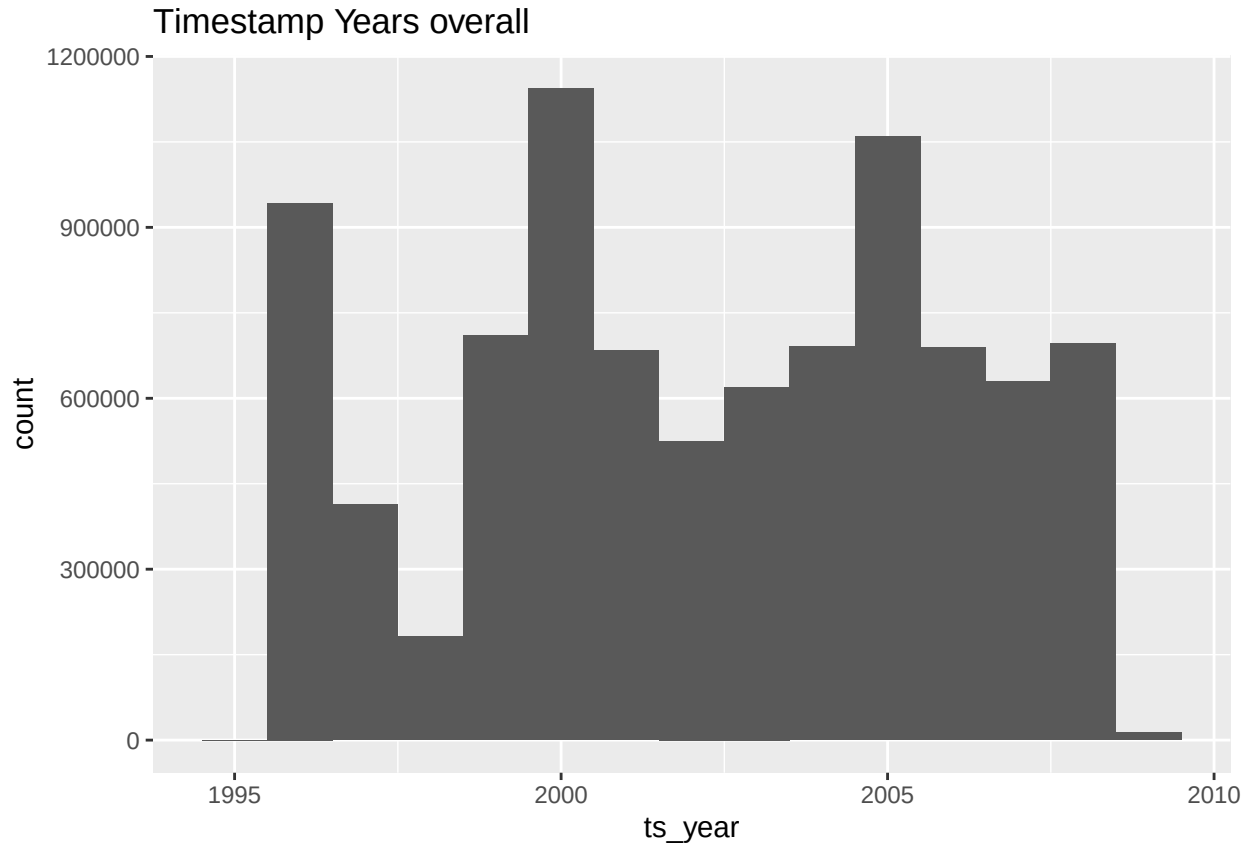
```
##          zeroVar   nzv
## userId     FALSE FALSE
## movieId    FALSE FALSE
## rating     FALSE FALSE
## title      FALSE FALSE
## genres     FALSE FALSE
## rel_year   FALSE FALSE
## ts_year    FALSE FALSE
## ts_month   FALSE FALSE
## ts_day     FALSE FALSE
```

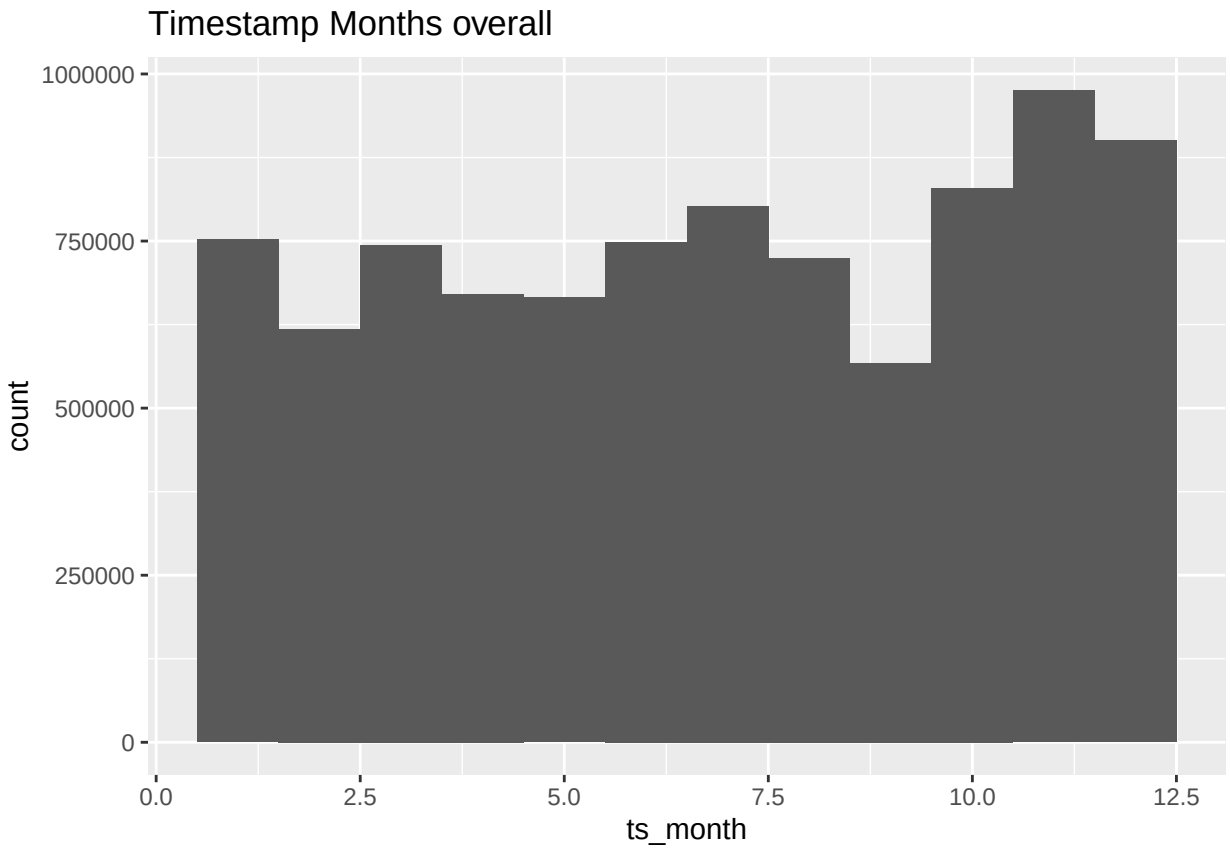
As the analysis shows there is no reason to exclude any predictors from the list.

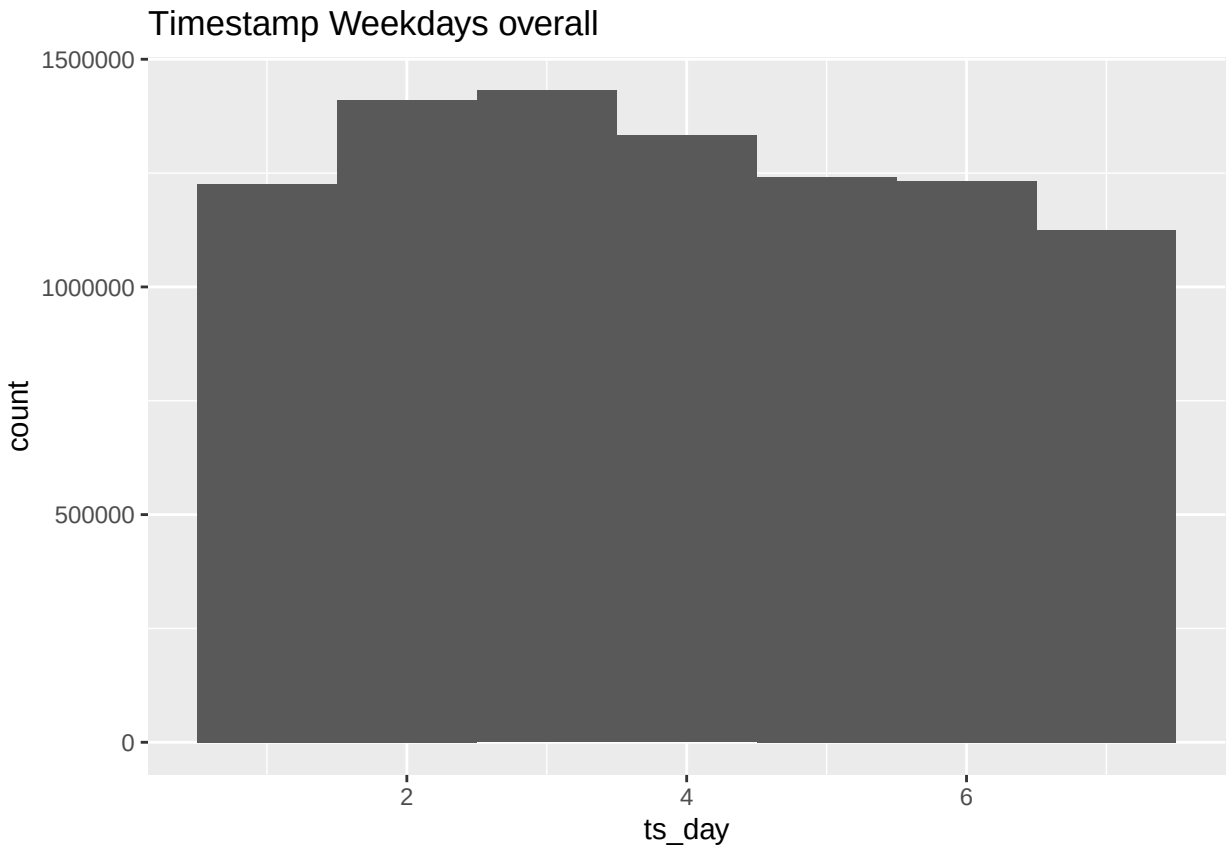
0.2.2.5 Average and distribution A first glance at the data shows the distribution of ratings with an overall average rating of `{r mean, echo=false} mean(edx$rating)`. To get a better grip on distributions of the target variable and the predictors we will have a look at histograms as well as on rating distributions for each predictor individually.

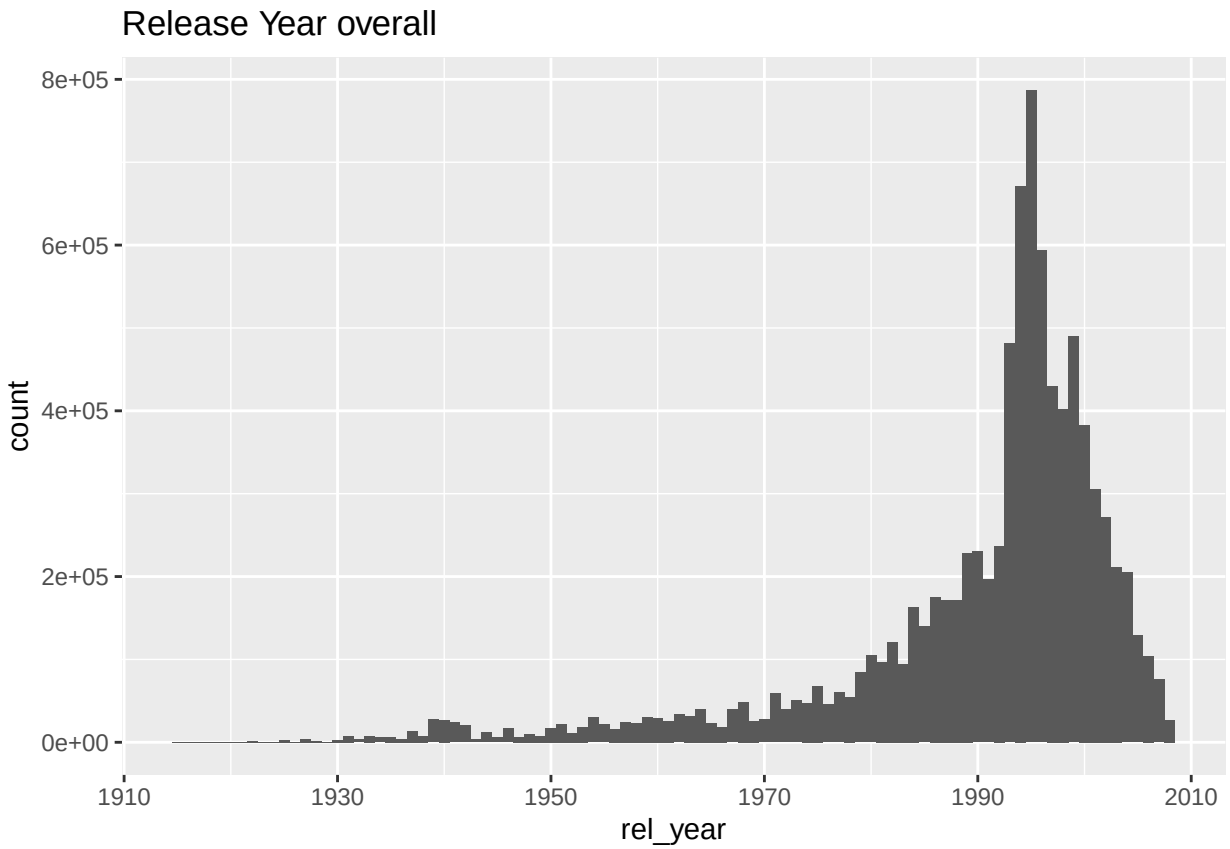


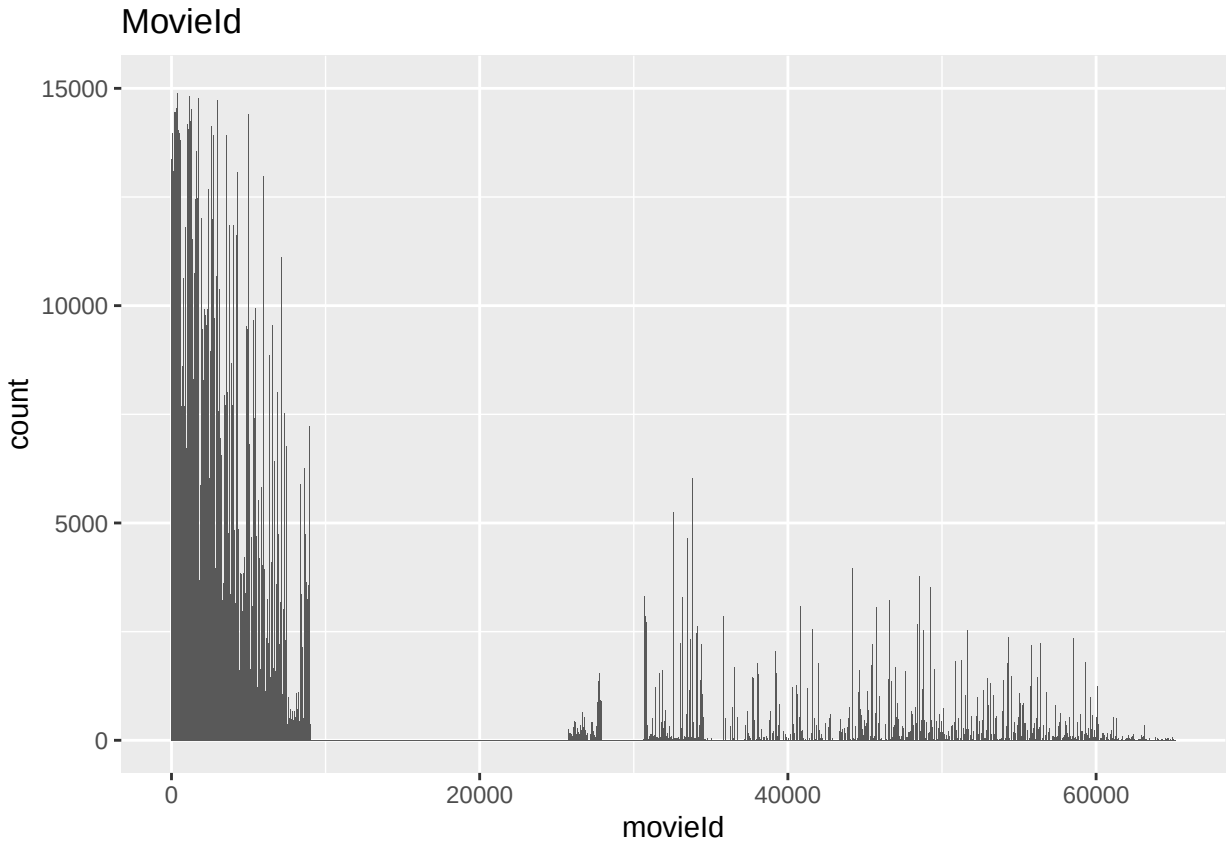
0.2.2.5.1 Histograms

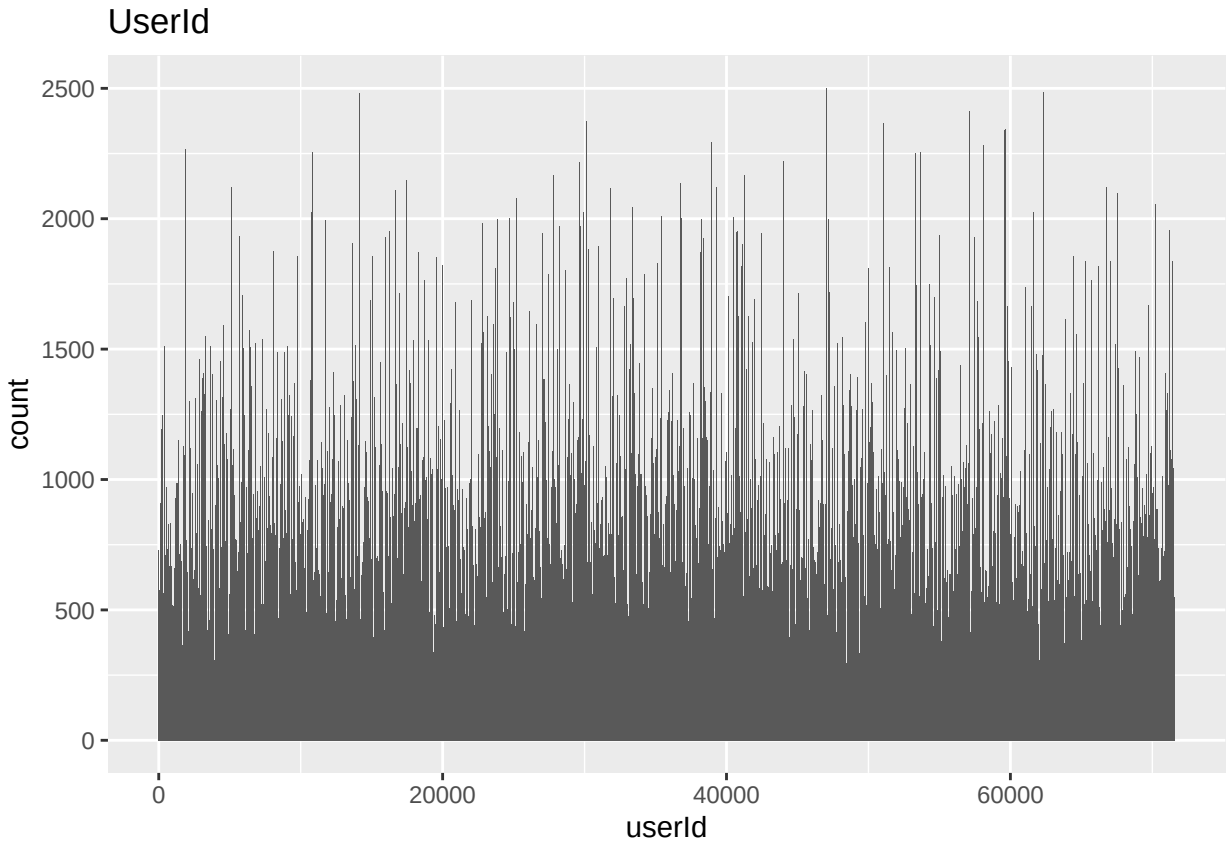



Timestamp Months overall
1000000
750000
500000
250000
0
count
0.0
2.5
5.0
7.5
10.0
12.5
ts_month


Timestamp Weekdays overall
1500000
1000000
500000
0
count
2
4
6
ts_day


Release Year overall
8e+05
6e+05
4e+05
2e+05
0e+00
count
1910
1930
1950
1970
1990
2010
rel_year


MovieId
15000
10000
5000
0
count
0
20000
40000
60000
movieId



We can see from the histograms that full ratings are much higher in number compared to half-point ratings. Ratings are within a range of 0.5 and 5 points. A rating of 4 has been given more often compared to any other rating and the distribution has a left tail. The histogram of *timestamp_years* shows that a few years are very low on ratings (1995, 1998, 2009). On the other hand the years 1996, 2000 and 2005 show a very high amount of ratings. This information will be regarded later after more thorough analysis when it comes to regularization.

The histogram of *timestamp_months* shows a rather evenly distributed amount of ratings throughout the year with a light drop in late summer and a rise during october until december.

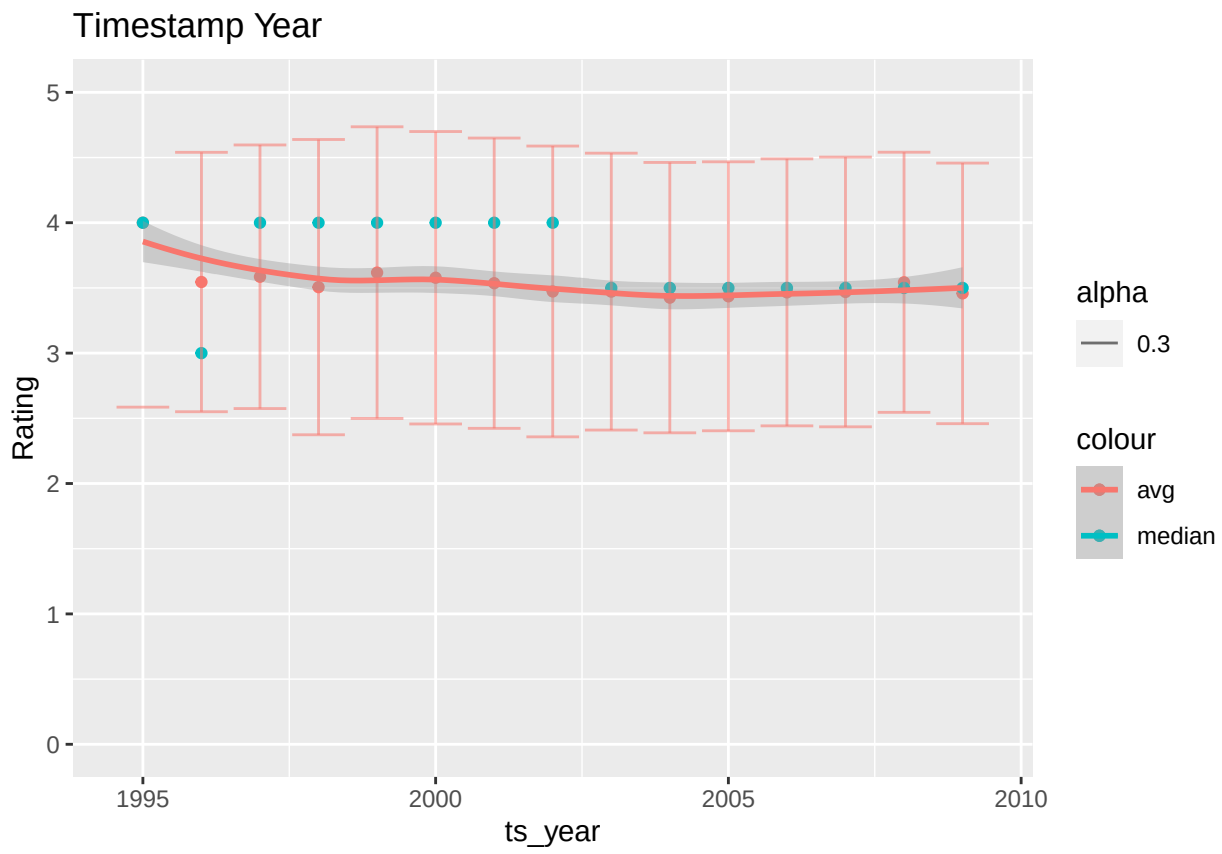
The histogram of *timestamp_days* also shows a widely evenly distribution with a high on tuesday and wednesday and a low on sunday.

The histogram of *release_year* shows a very distinctive distribution with most of the rated movies bein released around 1995. This information will come in handy when we come to regularization of the predictors.

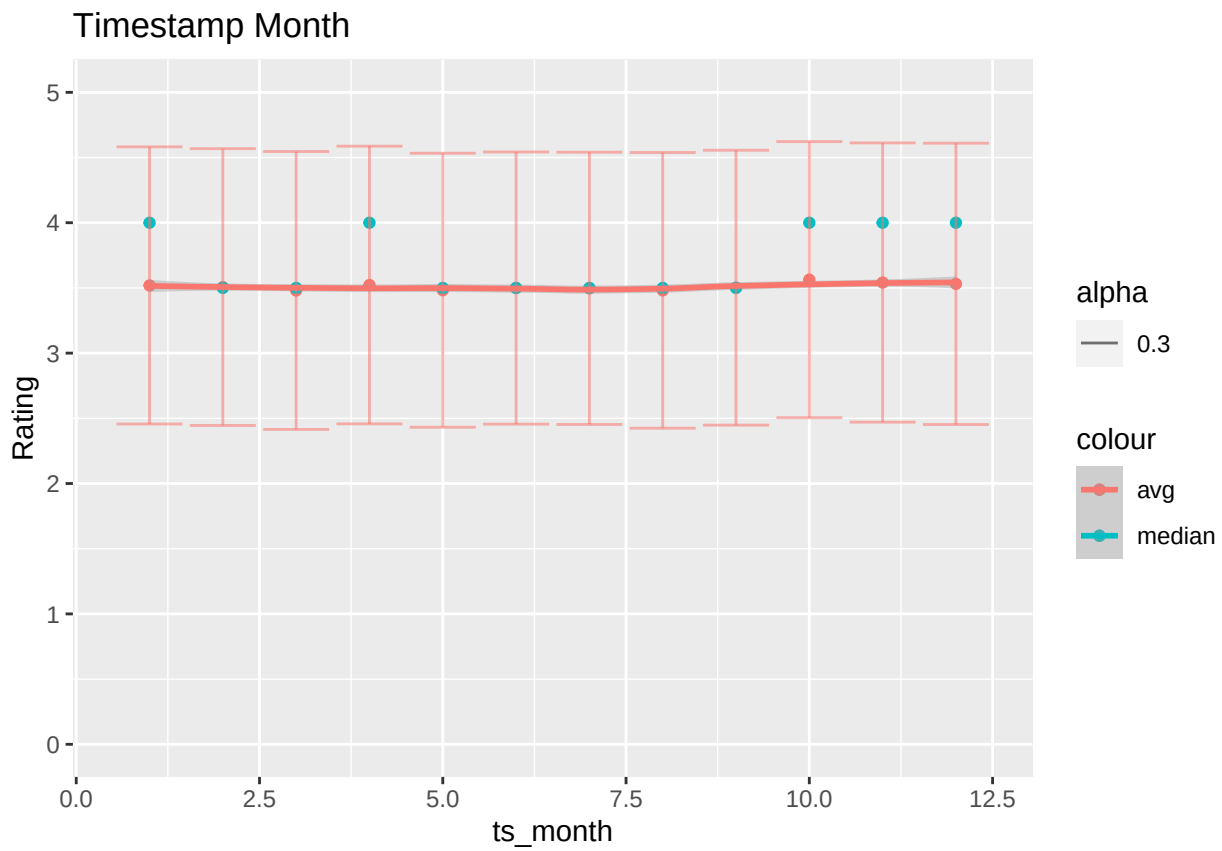
the two histograms of *movieId* and *userId* both show a wide range of occurances. That seems plausible since there are movies and users who have a high amount of ratings and those with very few ratings. In connection with the avg ratings per user and movie this will most likely become relevant when performing regularization.

0.2.2.5.2 Smoothplots for single predictors To dive deeper into the data graphs are created that show the average, the median and the standard error over each predictor. In addition to this a smoothplot is added to get a first idea about fitting based on single predictors using LOESS as a fitting algorithm.

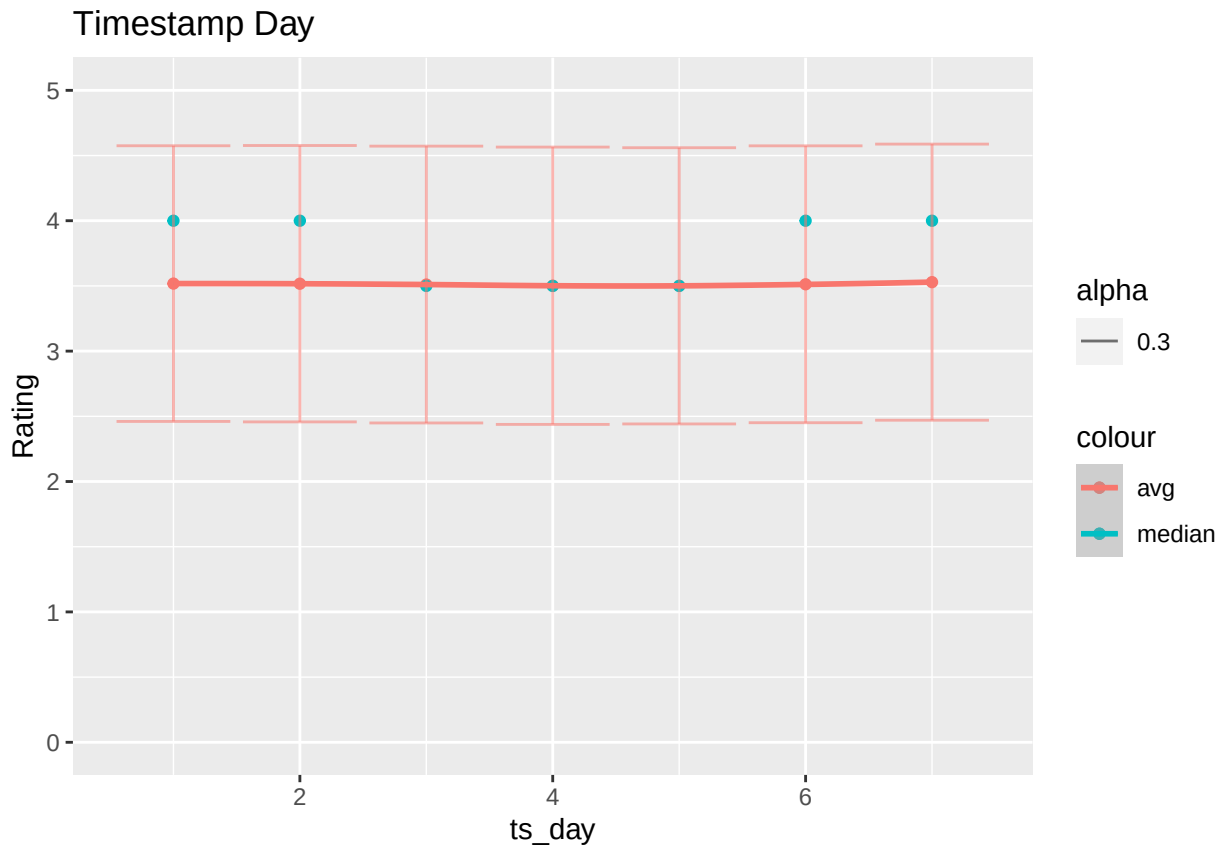
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



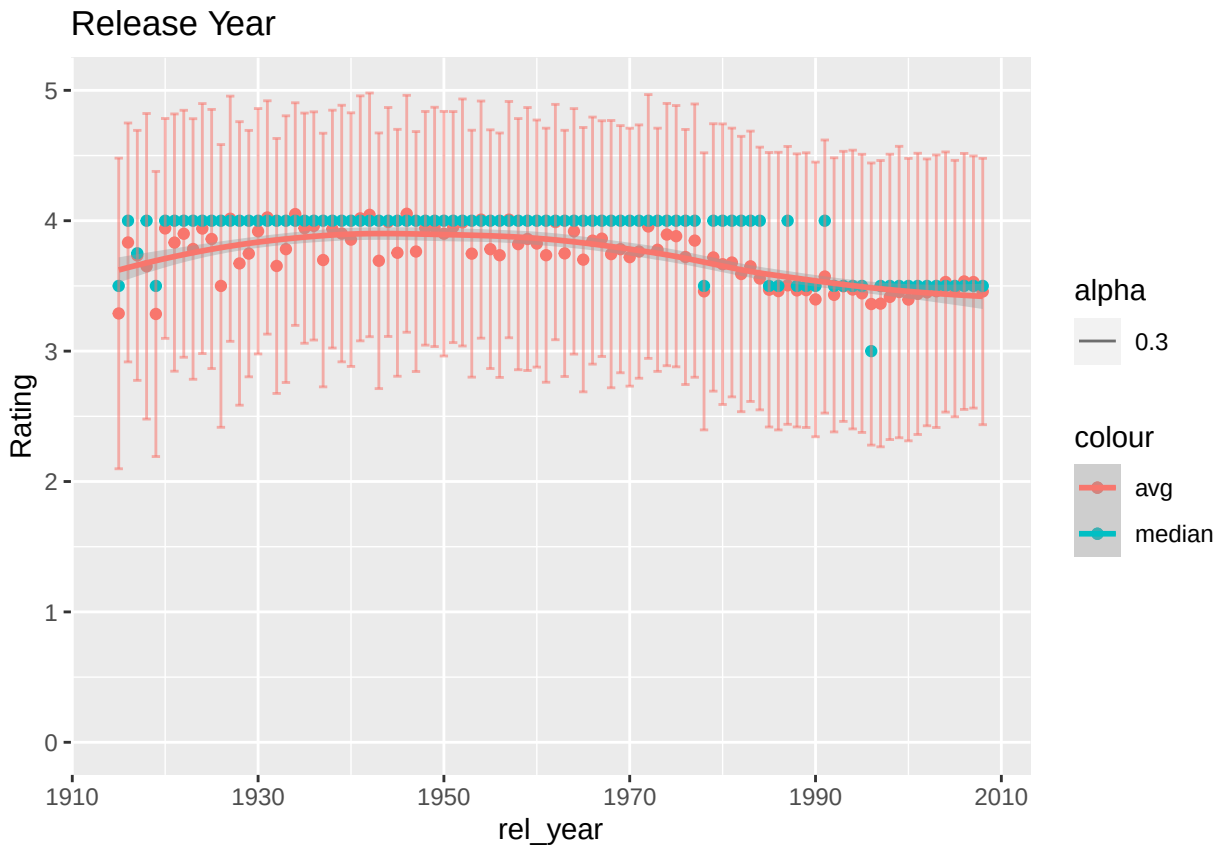

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

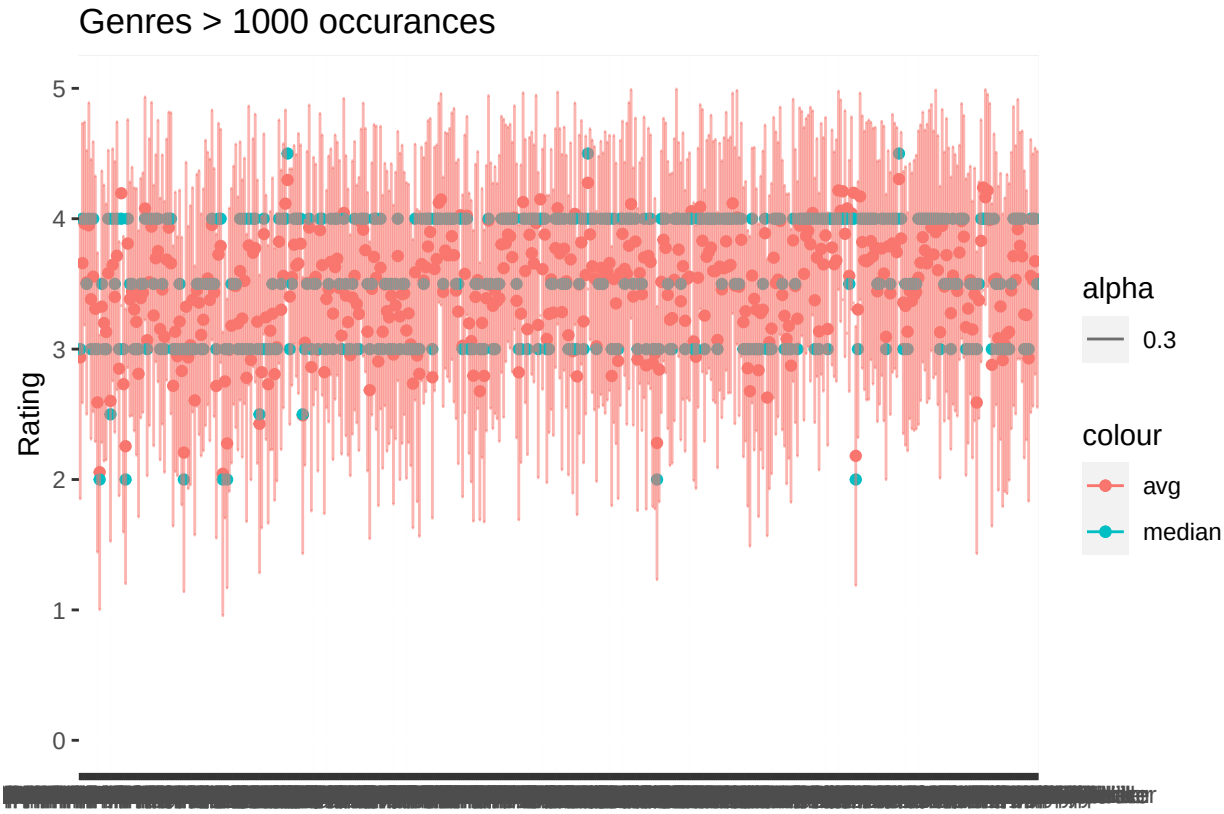


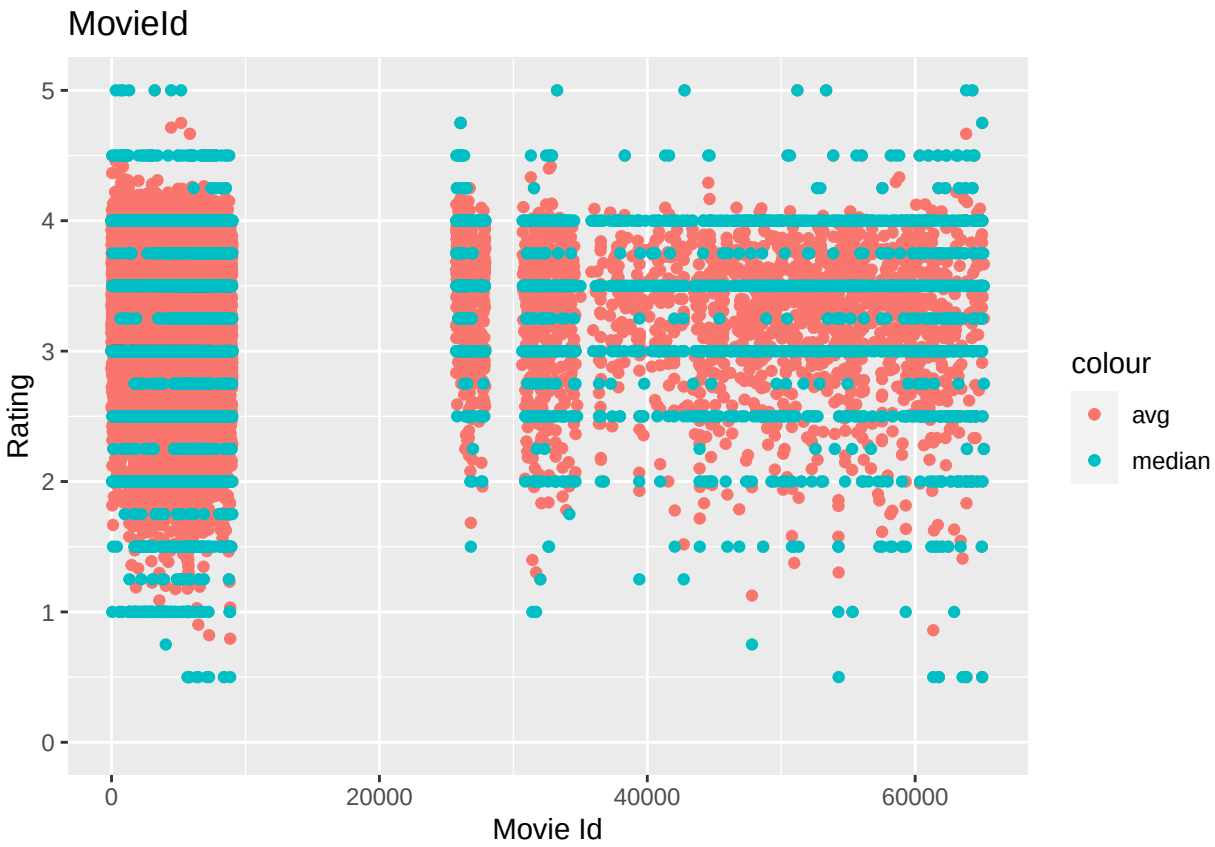

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```




```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```


Release Year
Rating
rel_year
5
4
3
2
1
0
1910
1930
1950
1970
1990
2010
alpha
0.3
colour
avg
median


Genres > 1000 occurances
Rating
5
4
3
2
1
0
alpha
0.3
colour
avg
median


MovieId
Rating
Movie Id
0
20000
40000
60000
0
1
2
3
4
5
colour
avg
median

0.2.3 Insights gained

As the graphs show there are big differences between the ratings depending on the different predictors. The standard error is comparable for all numeric variables around 1 and varies more widely when we look at genre as a predictor. That shows us that any of these predictors alone

0.2.3.1 timestamp_year The average rating doesn't change very much in different days and can be estimated pretty well.We can see that 1995 has no calculated average and standard error. Further analysis shows that we have only

```
## [1] 2
```

ratings from 1995 in the data set. The smoothplot shows that this causes the uncertainty to rise and also impacts the RMSE in 1996 negatively.

0.2.3.2 timestamp month and day We can see from the analyzed data that we have hardly any change of rating or standard error looking at the predictors * ts_month * ts_day As we have seen in the histogram there is also hardly change in numbers, so these two variables will be dumped from the edx and validation data set.

0.2.3.3 release year of movie The data provides a very useful insight to the predictor. We can see that avg and standard error change significantly based on the release year of the movie. Further we notice a significant drop in avg and median rating after 1980. Compared with the information seen in the istogram

we can see, that obviously movies with an early release date receive less ratings but on average better ratings. release year seems to be a very good predictor to differentiate ratings of movies. However, the smoothcurve shows that we have quite a few outliers which shows that we need additional predictors to reach the desired RMSE.

0.2.3.4 Genres To get a better overview we limit the graph to genres with more than 1000 occurances. Looking at the results we can see that there are significant differences between different genres. Using genres as a predictor would enable us to account for quite some variation in the data set. As it is categorical data we don't see a smoothplot.

0.2.3.5 MovieId Analyzing rating over MoviId we can see the strongest spread between avg and median ratings. We can see the full range between 0.5 and 5 in the median range. Using the movieId as a predictor should account for most of the variation.

0.2.3.6 UserId The userId is also showing a lot of variation. Similar to MOvieId as a predictor we can observe the full range of possible ratings. Next to MOvieId this should be the second predictor to start with.

0.2.3.7 Concluding insights To create the models we will use the five predictors 1. movieId 2. userId 3. genres 4. rel_year (release year of the movie) 5. ts_year (year of the rating)

0.2.4 modeling approach

In order to find the best performing model we will use the predictors as described in the prior chapter individually or in combination and apply different methods to train the models.

0.2.4.1 Initialization of modelling The data set edx is split into a training set (90% of the data) and a test set (10% of the data).

Secondly the target function is defined. As we are aiming to minimize the rooted mean square error (RMSE) the function is defined as follows:

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

The last two intitialisation procedures include calculating the mean rating (as it is used for quite few of the model calculations) and setting up a dataframe to compare the results of the different models.

```
#Calculate Mu (average)
mu <- mean(edx_train$rating)

#set up results dataframe
rmse_results <- data.frame(method = character(),
                           RMSE = numeric())
```

0.2.4.2 List of prediction models

The models used are described by the following code:

Model 1 - Average Model 2 - Linear Models Model 2.1 - Models with 1 predictor Model 2.1.1 - movieId Model 2.1.2 - userId Model 2.2 - Models with 2 predictors Model 2.2.1 - movieId + userId Model 2.2.2 - movieId + ts_year Model 2.2.3 - userId + genres Model 2.2.4 - userId + rel_year Model 2.3 - Models with 3 predictors Model 2.3.1 - movieId + userId + genres Model 2.3.2 - movieId + userId + rel_year MOdel 2.3.3 - movieId + userId + ts_year Model 2.4 - Models with 4 predictors Model 2.4.1 - movieId + userId + genres + rel_year Model 2.4.2 - movieId + userId + genres + ts_year Model 2.5 - Model with 5 predictors Model 2.6 - Top performing linear model with regularized predictors Model 3 - knn Model 4 - decision tree Model 5 - loess Model 6 - random forest Model 7 - GLM

Some of the models didn't run due to computing power constraints. Those models are marked as *"not able to run"* and are not included in the comparison of best performing model. However, they are kept in the overview and code, ready to run for anybody with a more potent computer.

0.2.4.3 Model 1 - Average

The average is the starting model and is the basis that we always need to beat. It results in a RMSE of

method	RMSE
1 - Average	1.060054

0.2.4.4 Model 2 - Linear models

The group of linear models is the biggest in this project and is divided in groups defined by the number of used predictors. The best performing model is finally optimized by using regularization. In general the models always work by analyzing the difference between the avg rating of a certain predictor and the overall avg rating. This value is then used to calculate the prediction by adding it to the mean.

0.2.4.4.1 Linear models with one predictor

The two most promising predictors are MovieId and UserId and are compared by running a model based on a single predictor.

method	RMSE
1 - Average	1.0600537
2.1.1 - linear movieId	0.9429615
2.1.2 - linear userId	0.9777091

As we can see movieId has a slightly lower RMSE and is therefore the better predictor.

0.2.4.4.2 Linear models with two predictors

In this group of predictors we follow the same approach but use two predictors. The predictor mix is always chosen in a way that either movieId or userId are the first one and will be paired with one of the remaining other three predictors. Recalling certain correlations it doesnot make sense to try out every possible combination as for example release year and movieId are highly correlated.

method	RMSE
1 - Average	1.0600537
2.1.1 - linear movieId	0.9429615
2.1.2 - linear userId	0.9777091
2.2.1 - linear movieId + userId	0.8646844
2.2.2 - linear movieId + ts_year	0.9408085
2.2.3 - linear userId + genres	0.9456509

method	RMSE
2.2.4 - linear userId + rel_year	0.9807695

The results show that the model with the lowest RMSE is the combination of movieId and userId (2.2.1). This combination will also be the basis for all next models.

0.2.4.4.3 Linear models with three predictors The two predictors movieId and userId are combined with the genre in the first model and with the release year in the second model.

method	RMSE
1 - Average	1.0600537
2.1.1 - linear movieId	0.9429615
2.1.2 - linear userId	0.9777091
2.2.1 - linear movieId + userId	0.8646844
2.2.2 - linear movieId + ts_year	0.9408085
2.2.3 - linear userId + genres	0.9456509
2.2.4 - linear userId + rel_year	0.9807695
2.3.1 - linear movieId + userId + genre	0.8643242
2.3.2 - linear movieId + userId + rel_year	0.8643302
2.3.3 - linear movieId + userId + rel_year	0.8646812

The results show that the best performing model is now the one with movieId, userId and genre as predictors (2.3.1). It seems that genre accounts for more separate uncertainty compared to release year. The reason could be that release year and movieId are correlated to some degree and account for the same uncertainty.

0.2.4.4.4 Linear models with four predictors The best performing model with three predictors is complemented with rel_year and ts_year and the final performance is shown in the following overview:

method	RMSE
1 - Average	1.0600537
2.1.1 - linear movieId	0.9429615
2.1.2 - linear userId	0.9777091
2.2.1 - linear movieId + userId	0.8646844
2.2.2 - linear movieId + ts_year	0.9408085
2.2.3 - linear userId + genres	0.9456509
2.2.4 - linear userId + rel_year	0.9807695
2.3.1 - linear movieId + userId + genre	0.8643242
2.3.2 - linear movieId + userId + rel_year	0.8643302
2.3.3 - linear movieId + userId + rel_year	0.8646812
2.4.1 - linear movieId + userId + genre + rel_year	0.8641262
2.4.2 - linear movieId + userId + genre + ts_year	0.8643142

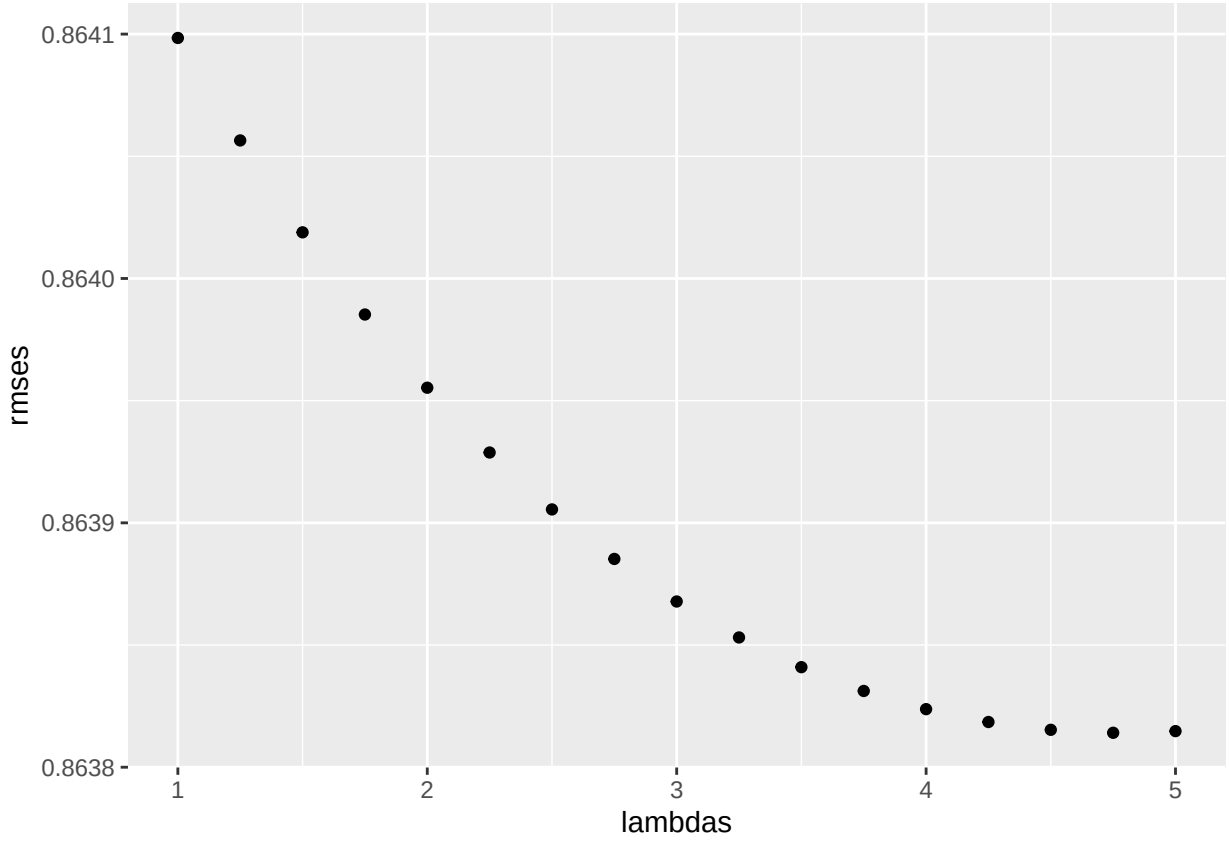
As we see the best performing model so far is 2.4.1 - using movieId + userId + genre + rel_year as predictors. This model will be regularized to account for the different weight of adjustments based on the number of ratings.

0.2.4.4.5 Linear models with five predictors *"not able to run"*

0.2.4.4.6 Regularized linear model In order to regularize the impact of the four predictors we will need to penalize predictor values with very few occurrences. In order to do so we define a value lambda that leads to an overall minimized RMSE. That means minimizing the following formula:

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

The following graph shows the lambda that minimizes the RMSE and the corresponding RMSE in the comparison table.



method	RMSE
1 - Average	1.0600537
2.1.1 - linear movieId	0.9429615
2.1.2 - linear userId	0.9777091
2.2.1 - linear movieId + userId	0.8646844
2.2.2 - linear movieId + ts_year	0.9408085
2.2.3 - linear userId + genres	0.9456509
2.2.4 - linear userId + rel_year	0.9807695
2.3.1 - linear movieId + userId + genre	0.8643242
2.3.2 - linear movieId + userId + rel_year	0.8643302
2.3.3 - linear movieId + userId + rel_year	0.8646812
2.4.1 - linear movieId + userId + genre + rel_year	0.8641262
2.4.2 - linear movieId + userId + genre + ts_year	0.8643142
Regularized 2.4.1 - movie+user+genre+rel_year	0.8638141

As you can see from the graph the minimum RMSE is reached with a lambda of `{r} lambda`

0.2.4.5 Model 3 - knn *"not able to run"* #### Model 4 - decision tree *"not able to run"* #### Model 5 - loess The local weighted regression (loess model) is being used on the two predictors *movieId* and *userId*, takes a long time to run with results not much better compared to the overall rating average.

method	RMSE
1 - Average	1.0600537
2.1.1 - linear movieId	0.9429615
2.1.2 - linear userId	0.9777091
2.2.1 - linear movieId + userId	0.8646844
2.2.2 - linear movieId + ts_year	0.9408085
2.2.3 - linear userId + genres	0.9456509
2.2.4 - linear userId + rel_year	0.9807695
2.3.1 - linear movieId + userId + genre	0.8643242
2.3.2 - linear movieId + userId + rel_year	0.8643302
2.3.3 - linear movieId + userId + rel_year	0.8646812
2.4.1 - linear movieId + userId + genre + rel_year	0.8641262
2.4.2 - linear movieId + userId + genre + ts_year	0.8643142
Regularized 2.4.1 - movie+user+genre+rel_year	0.8638141
5 - gamLOESS	1.0585858

0.2.4.6 Model 6 - random forest *"not able to run"* #### Model 7 - GLM As a last model for comparison the generalized linear model is used. Due to computing power we use a model with only two predictors (*movieId* and *userId*). As expected the results are not as good as the ones from a regularized linear model with four predictors.

```
##   parameter
## 1      none
```

method	RMSE
1 - Average	1.0600537
2.1.1 - linear movieId	0.9429615
2.1.2 - linear userId	0.9777091
2.2.1 - linear movieId + userId	0.8646844
2.2.2 - linear movieId + ts_year	0.9408085
2.2.3 - linear userId + genres	0.9456509
2.2.4 - linear userId + rel_year	0.9807695
2.3.1 - linear movieId + userId + genre	0.8643242
2.3.2 - linear movieId + userId + rel_year	0.8643302
2.3.3 - linear movieId + userId + rel_year	0.8646812
2.4.1 - linear movieId + userId + genre + rel_year	0.8641262
2.4.2 - linear movieId + userId + genre + ts_year	0.8643142
Regularized 2.4.1 - movie+user+genre+rel_year	0.8638141
5 - gamLOESS	1.0585858
Model 7 - GLM	1.0600317

0.3 Results

Following the approach mentioned above we can see that we reach the best result by running a linear regularized model with the 4 predictors *movieId*, *userId*, *genres*, *rel_year* This model is validated on the

validation data set. As lambda we use the evaluated value of 4.75. As there are very few rows that didn't show up in the training data the prediction function creates a few NA-values that can be identified in the summary. A check shows the titles of those few movies. In order to make the model work these prediction values are substituted with mu and the final RMSE is calculated.

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##  -0.432   3.138   3.564   3.512   3.942   5.965       4
```

```
## [1] "Rowing with the Wind (Remando al viento) (1988)"
## [2] "Young Unknowns, The (2000)"
## [3] "Big Fella (1937)"
## [4] "Rowing with the Wind (Remando al viento) (1988)"
```

method	RMSE
1 - Average	1.0600537
2.1.1 - linear movieId	0.9429615
2.1.2 - linear userId	0.9777091
2.2.1 - linear movieId + userId	0.8646844
2.2.2 - linear movieId + ts_year	0.9408085
2.2.3 - linear userId + genres	0.9456509
2.2.4 - linear userId + rel_year	0.9807695
2.3.1 - linear movieId + userId + genre	0.8643242
2.3.2 - linear movieId + userId + rel_year	0.8643302
2.3.3 - linear movieId + userId + rel_year	0.8646812
2.4.1 - linear movieId + userId + genre + rel_year	0.8641262
2.4.2 - linear movieId + userId + genre + ts_year	0.8643142
Regularized 2.4.1 - movie+user+genre+rel_year	0.8638141
5 - gamLOESS	1.0585858
Model 7 - GLM	1.0600317
VALIDATION - Regularized 2.4.1 - movie+user+genre+rel_year	0.8648557

This is expected as we have seen in the analyzed data that the rankings vary largely across the predictor´s range of values. GLM performs lower because it assumes a linear context for all predictors. The local weighted regression (loess model) assumes that the within very small windows of data the data is linear resp. parabolic. That allows for an overall non-linear context looking at the whole range of data. However this approach considers data points within a span and weights them differently. Looking at predictors like *movieId* or *userId* this approach would not work very well because the Ids have a rather categorical character and two movies or users with very close Ids have independent ratings.

0.4 Conclusion

To sum up the results of this report it was possible to train an algorithm and reach an RMSE of

```
## [1] 0.8648557
```

. The best performance was reached by using Model *2.6* with a regularized approach.

The limitations of this report lie in the limited data used and in the approach chosen, with a limited amount of pre-chosen predictors. Secondly some of the methods couldn´t be used due to a limitation of computing power. In this analysis we only use a fraction of the possible combinations of predictors and training models.

Separate predictors with separate models depending on the distribution and for example using an ensemble of those could also be an option to improve performance.
Next steps to improve performance would be to use other predictors like director, actors, country of origin, languages available, production cost, movie length, amount of awards, and more... One could also analyze for trigger words within the title and check if there are correlations with ratings if the title contains words like “Love”, “reloaded”, “fight” or others. Another way would also be to increase training data size, or use neural networks to find new predictors using deep learning algorithms. In order to do so we would need to provide a broader data base.