MovieLens Capstone Project

Professional Certificate PH125.9x

Robert Gravelle

2023-11-13

MovieLens

Introduction

This is a report on the MovieLens data analysis and a recommendation model training and the models achieved performance. First the dataset is to be explored, possibly cleaned and inspected to evaluate possible training approaches. Next part is building a ML model to recommend movies to users.

Dataset

Grouplens created a movie rating dataset. The 10M dataset [@harper2015] used in this project is a subset of 10 million ratings of 10'000 movies by 72'000 random selected users.

Initial setup

Given is the loading of the MovieLens 10M dataset, split into an *edx* and a *final_holdout_test* set containing 10% of the MovieLens data only used for validating at the end. The dataset contains userId, movieId, rating, timestamp, title, and genre.

Goal

This dataset is used to explore and gain insight on how an effective recommendation algorithm could be developed. Such a machine learning algorithm is then developed and tested against the *final_holdout_test* set.

Summary

The analysis revealed interesting properties and correlations of the various features. Among other things, older films tended to be rated higher than newer ones, and some genres were generally rated slightly higher or lower. Above all, however, the films and the users play a decisive role in developing a model. Unfortunately, only an RMSE of about 0.879 was possible with this method. For further interesting model trainings neither the time nor the available computing power was sufficient, for example other training approaches like KNN or Decision Tree could be tried.

Analysis

Data Inspection and preprocessing

First lets take a closer look at the *edx* dataset structure and some of its content.

	userId	movieId	rating	timestamp	title	genres
1	1	122	5	838985046	Boomerang (1992)	Comedy|Romance
2	1	185	5	838983525	Net, The (1995)	Action|Crime|Thriller
4	1	292	5	838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller
5	1	316	5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi
6	1	329	5	838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi
7	1	355	5	838984474	Flintstones, The (1994)	Children|Comedy|Fantasy

There are several columns in this dataset:

- userId: an identifier for the individual user who rated the movie.
- movieId: an identifier for movie that was rated.
- rating: a rating that was given for this movie (from that user). The scale of the ratings is yet to be identified.
- timestamp: a timestamp in UNIX format
- title: the title of the movie including the year of release

- genres: a list of genres associated with this movie. Multiple genres are separated by '|'.

There are several aspects of the edx dataset to consider exploring:

- Several movies may be rated way above average because of a very good story or production.
- Some genres (or genre combinations) may be rated higher than others.
- User ratings are possibly biased or generally rated higher or lower.
- User ratings in relation to a genre. A user maybe likes Horror and Action but rates movies of other genres typically lower.
- User ratings in relation to movie release year.
- User ratings in relation to popularity of movies (indie vs blockbuster).

And probably lots more.

The dataset *edx* contains 0 missing values.

Lets list all genres.

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

We notice a unusual genre named *(no genres listed)*.

	userId	movieId	rating	timestamp	title	genres
1025055	7701	8606	5.0	1190806786	Pull My Daisy (1958)	(no genres listed)
1453345	10680	8606	4.5	1171170472	Pull My Daisy (1958)	(no genres listed)
4066835	29097	8606	2.0	1089648625	Pull My Daisy (1958)	(no genres listed)
6456906	46142	8606	3.5	1226518191	Pull My Daisy (1958)	(no genres listed)
8046611	57696	8606	4.5	1230588636	Pull My Daisy (1958)	(no genres listed)
8988750	64411	8606	3.5	1096732843	Pull My Daisy (1958)	(no genres listed)
9404670	67385	8606	2.5	1188277325	Pull My Daisy (1958)	(no genres listed)

On further investigation, only one movie (Pull My Daisy) has no genre listed. According to IMDB the genre of this movie from 1958 is "Short". Let's separate the genres listed and add the first three of each movie into separate columns.

As we have seen, there are 20 unique genres in the dataset.

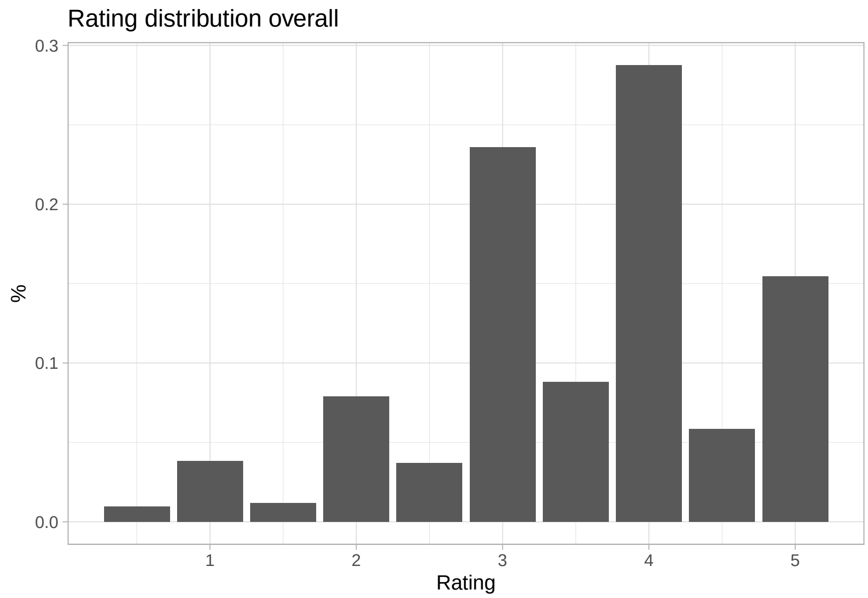
Then we transform the UNIX timestamps to date/time and for ease of use also extract the year the movie was rated. And separate the release year, embedded in the title, for possible further investigation.

Now some columns are added for further inspection, lets see the table again.

	userId	movieId	rating	timestamp	title	genres	main_genre	side1_genre	side2_genre	date	yearrated	rele
1	1	122	5	838985046	Boomerang (1992)	Comedy|Romance	Comedy	Romance	NA	1996-08-02 11:24:06	1996	
2	1	185	5	838983525	Net, The (1995)	Action|Crime|Thriller	Action	Crime	Thriller	1996-08-02 10:58:45	1996	
4	1	292	5	838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller	Action	Drama	Sci-Fi	1996-08-02 10:57:01	1996	
5	1	316	5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi	Action	Adventure	Sci-Fi	1996-08-02 10:56:32	1996	
6	1	329	5	838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi	Action	Adventure	Drama	1996-08-02 10:56:32	1996	
7	1	355	5	838984474	Flintstones, The (1994)	Children|Comedy|Fantasy	Children	Comedy	Fantasy	1996-08-02 11:14:34	1996	

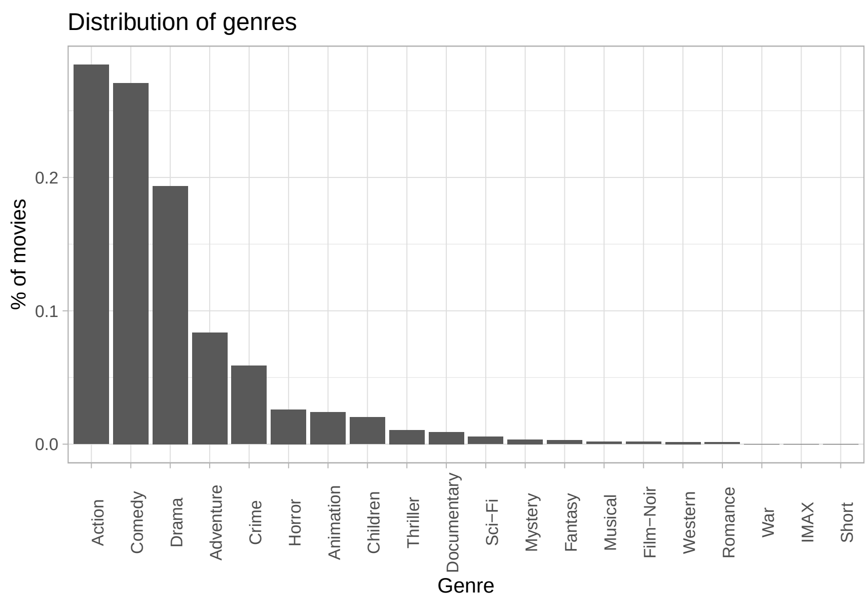
Distributions

Lets plot some distributions, starting the distribution of ratings.

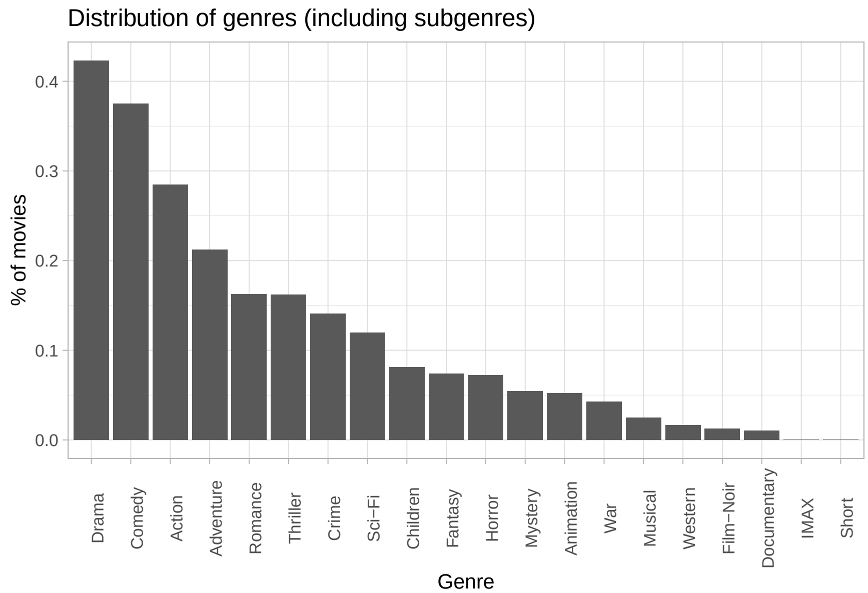


Most ratings are given as full number ratings (4, 3 or 5). Also the half-point ratings distribution follows a similar distribution as the full number ratings.

Now the distribution of the genres.



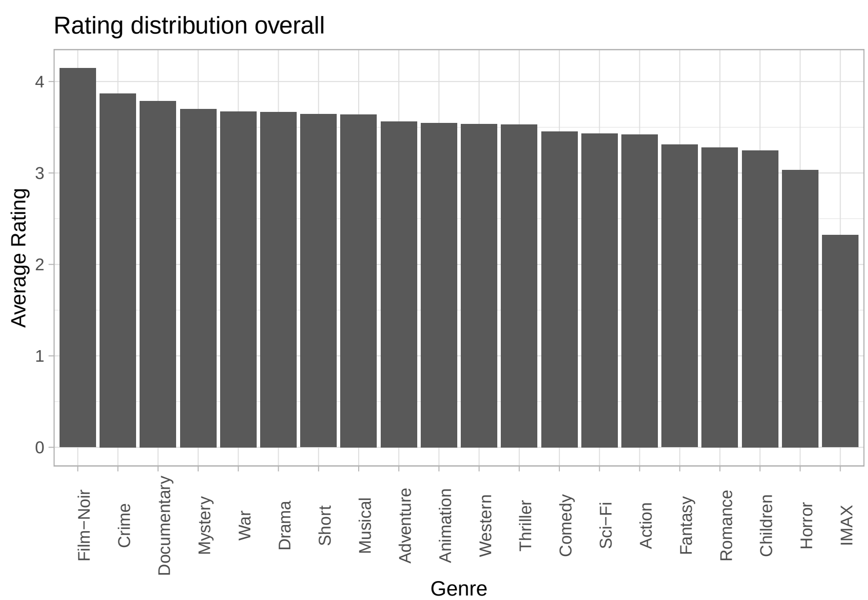
Most genres listed are Action, Comedy and Drama. What about when taking the sub genres into account.



When second and third listed genres are taken into account, the distribution is much finer. Top genres are still Drama, Comedy and Action but positions changed slightly. Drama is therefore an often used side genre, e.g. in combination with Romance, Thriller or others.

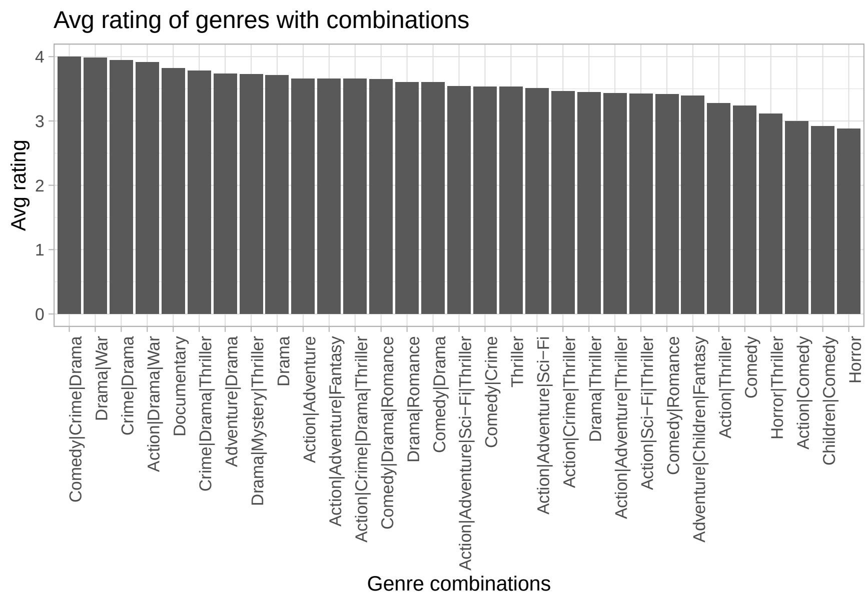
Genres

Lets plot the average rating against the genres.



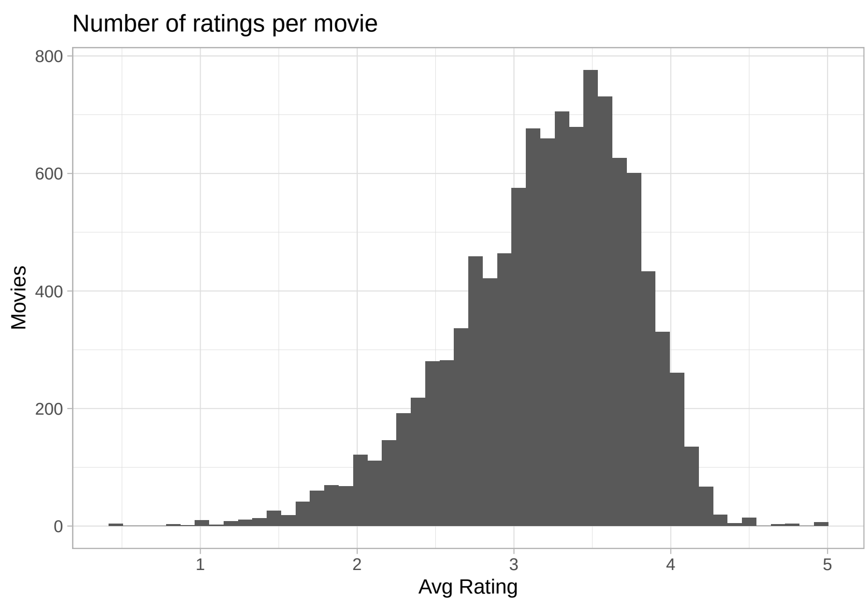
Ratings of genres show "intellectual" movie genres (e.g. Film-Noir, Crime, Drama..) are rated higher than movie genres associated with entertainment (e.g. Action, Fantasy, Horror)

And for average ratings of genre combinations with more than 50000 ratings.



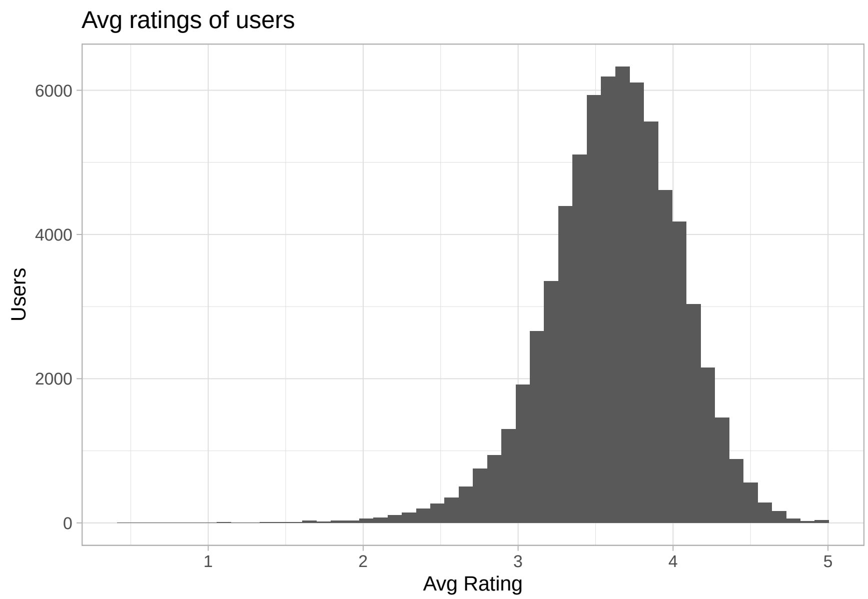
When taken genre combinations into account, a similar picture is painted, but some entertainment genre combinations, notably a Action combination, make it higher up the list (e.g. Action|Drama|War, Action|Adventure)

Average rating distribution of movies.



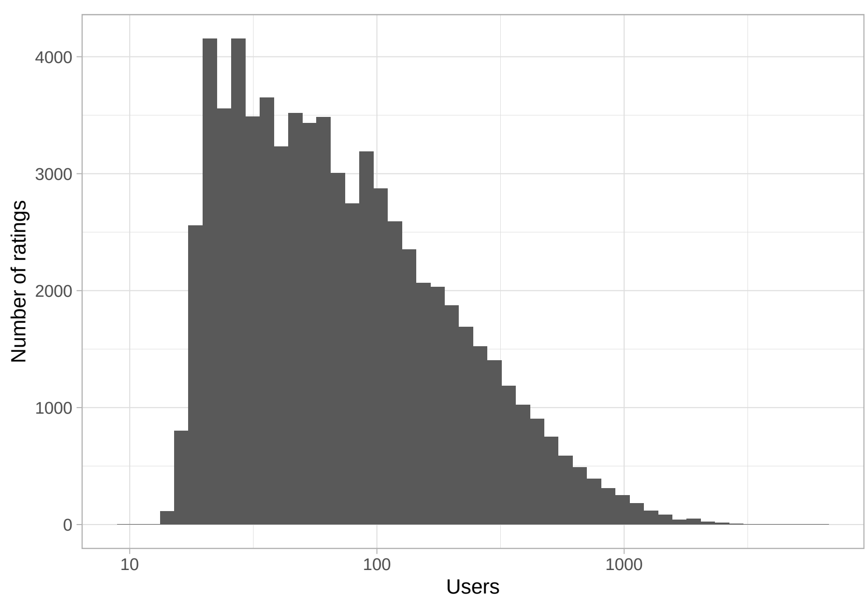
Only very few movies get an average rating above about 4.2. Most average ratings are between 2.5 and 4.

Average rating distribution of users



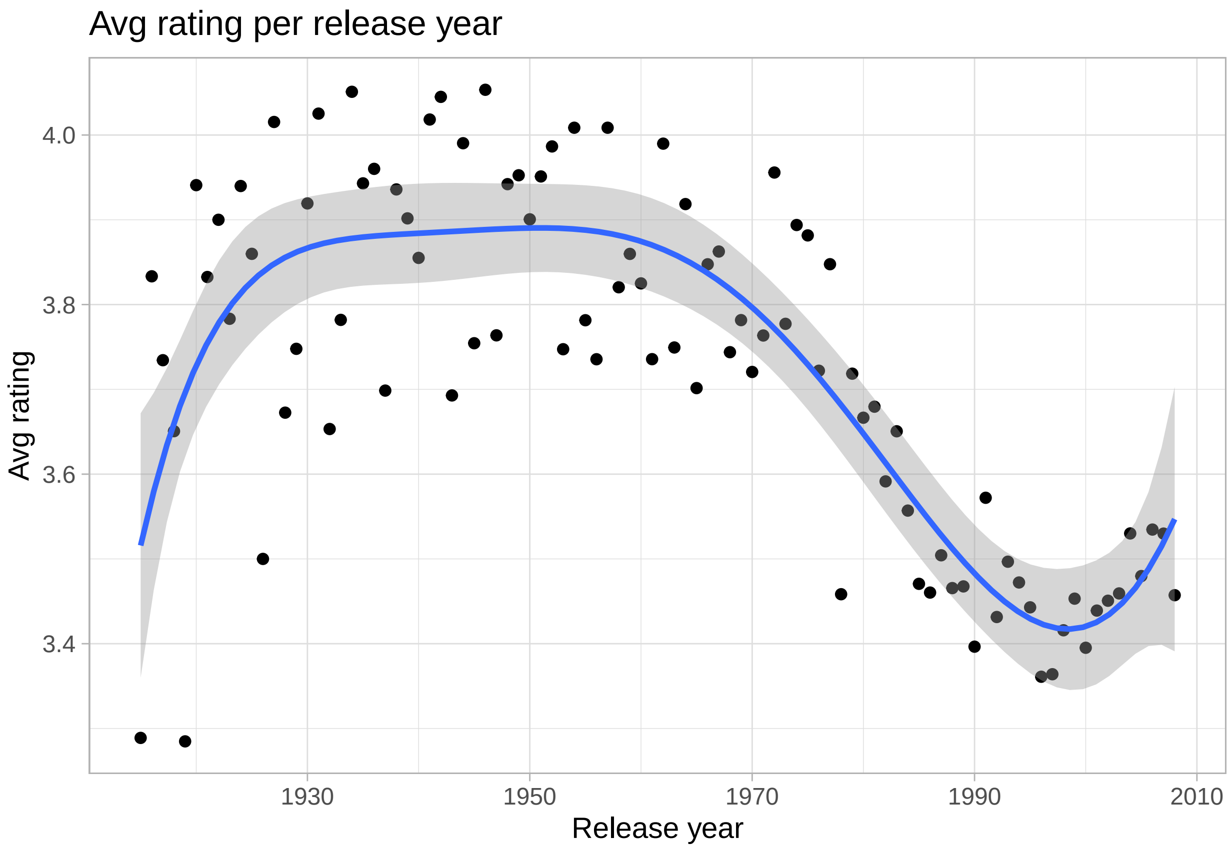
Most users rate movies between 3.5 and 4.2. This is higher than we've seen before on the average rating distribution of movies.

Number of ratings per user



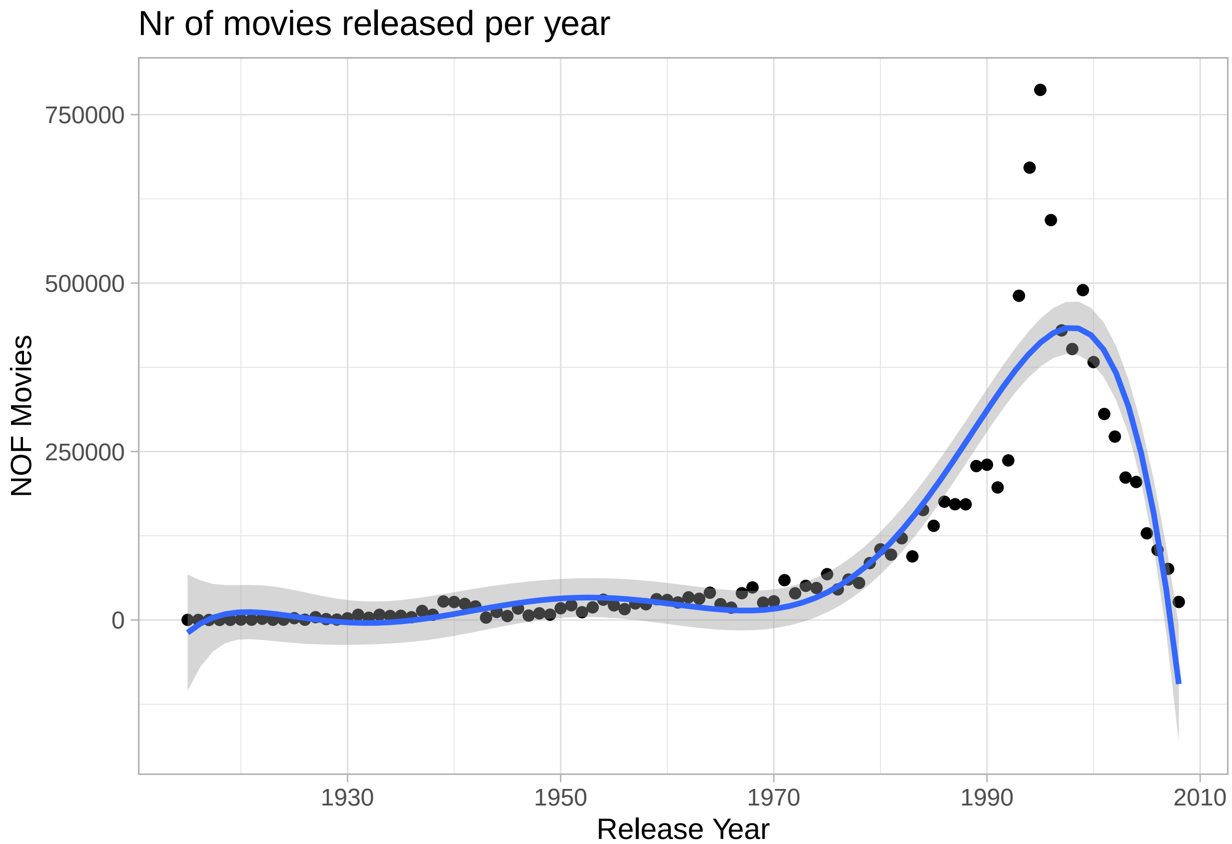
Most users rate between a few and about 100 movies.

Average rating per release year with trendline



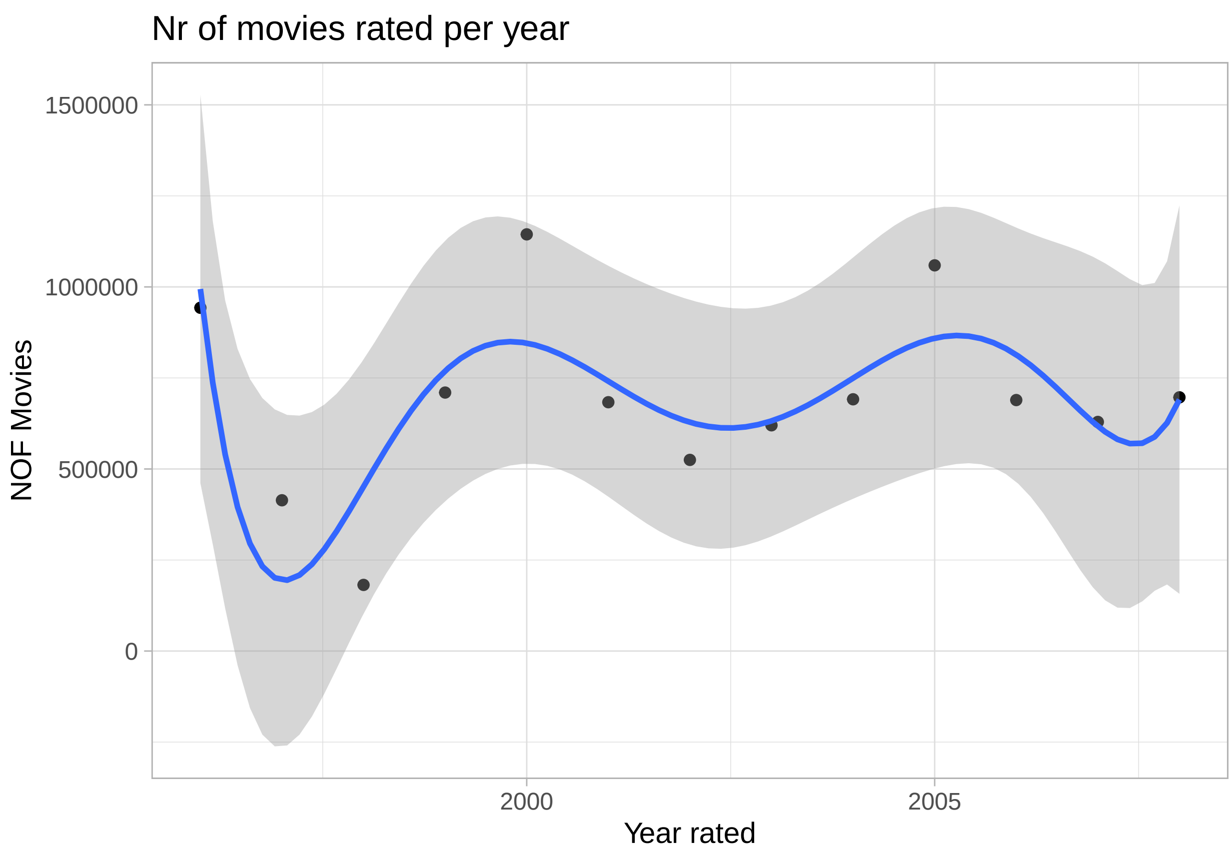
Movies released before 1970 are generally rated higher than movies released in the last two decades. This is a know effect (only good stuff survive the test of time).

Number of released movies per year



Movie releases were relative stable before the 1970, then picked up and almost exploded 1990 and the following years, probably due to advancements in technology, production and market (e.g. movie theaters, movie rentals, tv…). Since before 2000 the number of movies released collapsed as quickly as its explosion a decade earlier.

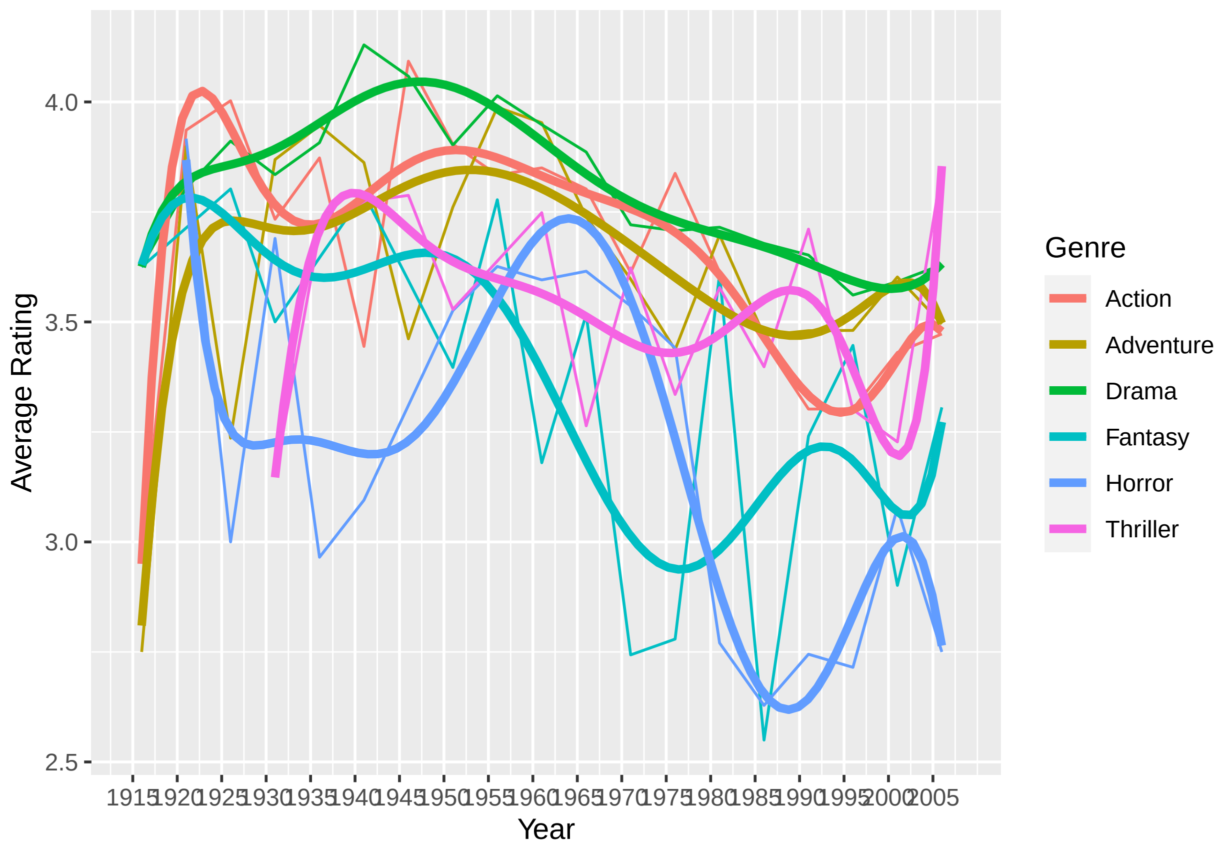
Number of ratings per year. Only include the years from 1996 to 2008, data before and after is 0.



Movie ratings per year are stable with some outliers.

Get the average rating of movie per genre but in 5 year bins.

```
## `summarise()` has grouped output by 'main_genre'. You can override using the
## `.groups` argument.
```



Some genres have pretty consistent average ratings over the years, others like e.g. Horror or Fantasy fluctuate a lot more.

Model

Prepare the training and test datasets and create a table for the RMSE results as we develop the model. I learned in the mean+movie approach, that all movieIds must be present in the training dataset, otherwise the test set will have movieId's where there was no training data on it. I guessed this will also be the case for userId and the main genre.

Guessing

Try stupid approach by guessing a rating from 0 to 5.

```
# create list of guessed ratings and make "guesses" for the size of the test set
guess_model <- sample(c(0, 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5), length(test_set$rating), replace=TRU
```

The resulting 2.1557864 is still far of the < 0.865, no suprise.

Model	RMSE
Guessing	2.155786

Mean

Get the average of all ratings on the training set and use this to predict a movie.

```
# Average mean of every movie in the training set
avg_model <- mean(train_set$rating)
```

The average of every movie used for predicting a rating results in 1.0598665. Closer but still far off.

Model	RMSE
Guessing	2.155786
Avg Model	1.059867

Genre bias

Evaluate the genre bias, the deviation from the movie average for each genre.

```
# average rating of all movies
movie_avg <- mean(train_set$rating)

# bias of genres. deviation from the average for each genre. e.g. Film-Noir is 0.638
# above average, Horror 0.48 below average.
genre_bias <- train_set %>%
  group_by(main_genre) %>%
  summarise(deviation_genre = mean(rating - movie_avg))

# combine genre bias with the test_set
mean_genre_model <- test_set %>%
  inner_join(genre_bias, by="main_genre")
```

The RMSE is a bit better at 1.0480846 than just take the average like before.

Model	RMSE
Guessing	2.155786
Avg Model	1.059867
Genre Model	1.048085

Movie bias

The movie_bias is the difference of the avg movie rating to the mean rating.

```r
movie_bias <- train_set %>%
  group_by(movieId) %>%
  summarise(deviation_movie = mean(rating - movie_avg))

# on the test set add the movie avg (3.512) with the difference the movie had
# to the avg in the training set and pull that column as a vector
mean_movie_model <- test_set %>%
  inner_join(movie_bias, by="movieId")
```

With and RMSE of 0.9431683 we are now in the sub 1 category.

Model	RMSE
Guessing	2.1557864
Avg Model	1.0598665
Genre Model	1.0480846
Movie Model	0.9431683

User bias

Lets inspect the user bias.

```r
user_bias <- train_set %>%
  group_by(userId) %>%
  summarise(deviation_user = mean(rating - movie_avg))

mean_user_model <- test_set %>%
  inner_join(user_bias, by="userId")
```

The user bias results in an RMSE of 0.9783862.

Model	RMSE
Guessing	2.1557864
Avg Model	1.0598665
Genre Model	1.0480846
Movie Model	0.9431683
User Model	0.9783862

Release year bias

Lets see if the release year will bring the RMSE down.

```r
releaseyear_bias <- train_set %>%
  group_by(releaseyear) %>%
  summarise(deviation_releaseyear = mean(rating - movie_avg))

mean_releaseyear_model <- test_set %>%
  inner_join(releaseyear_bias, by="releaseyear")
```

The release year of a movie bias results in an RMSE of 1.0489378.

Model	RMSE
Guessing	2.1557864
Avg Model	1.0598665
Genre Model	1.0480846
Movie Model	0.9431683
User Model	0.9783862
Release Year Model	1.0489378

User and Movie bias

Let's add the average rating of a user into the mix.

```r
# user_bias is the differnece of the avg user rating to the mean rating
user_bias <- train_set %>%
  group_by(userId) %>%
  summarise(deviation_user = mean(rating - movie_avg))

mean_movie_user_model <- test_set %>%
  inner_join(movie_bias, by="movieId") %>%
  inner_join(user_bias, by="userId")
```

The user and movie gets us below 0.9. But 0.8854123 is still not near the desired < 0.865.

Model	RMSE
Guessing	2.1557864
Avg Model	1.0598665
Genre Model	1.0480846
Movie Model	0.9431683
User Model	0.9783862
Release Year Model	1.0489378
Movie + User Model	0.8854123

User and Movie and Release Year bias

To the last model we add the release year.

```r
mean_movie_user_releaseyear_model <- test_set %>%
  left_join(movie_bias, by='movieId') %>%
  left_join(user_bias, by='userId') %>%
  left_join(releaseyear_bias, by='releaseyear')
```

With the release year added to the user and movie bias we get 0.9017918.

Model	RMSE
Guessing	2.1557864
Avg Model	1.0598665
Genre Model	1.0480846
Movie Model	0.9431683
User Model	0.9783862
Release Year Model	1.0489378
Movie + User Model	0.8854123
Movie + User + Release Year Model	0.9017918

User, Movie and Genre bias

Now combine user, movie and genre together in a single model.

```r
mean_movie_user_genre_model <- test_set %>%
  inner_join(movie_bias, by="movieId") %>%
  inner_join(user_bias, by="userId") %>%
  inner_join(genre_bias, by="main_genre")
```

This resulted in 0.9026264, which is worse than only using the movie and user. Maybe some tuning will fix it.

Model	RMSE
Guessing	2.1557864
Avg Model	1.0598665
Genre Model	1.0480846
Movie Model	0.9431683
User Model	0.9783862
Release Year Model	1.0489378
Movie + User Model	0.8854123
Movie + User + Release Year Model	0.9017918
Movie + User + Genre Model	0.9026264

User, Movie, Release Year and Genre bias

Now combine user, movie, release year and genre together in a single model.

```
mean_movie_user_genre_releaseyear_model <- test_set %>%
  inner_join(movie_bias, by="movieId") %>%
  inner_join(user_bias, by="userId") %>%
  inner_join(genre_bias, by="main_genre") %>%
  inner_join(releaseyear_bias, by="releaseyear")
```

This resulted in 0.9203211.

Model	RMSE
Guessing	2.1557864
Avg Model	1.0598665
Genre Model	1.0480846
Movie Model	0.9431683
User Model	0.9783862
Release Year Model	1.0489378
Movie + User Model	0.8854123
Movie + User + Release Year Model	0.9017918
Movie + User + Genre Model	0.9026264
Movie + User + Genre + Release Year Model	0.9203211

User, Movie, Release Year and first three listed genres bias

Same as before but instead of the whole genres list as a whole the first three listed genres are used.

```
main_genre_bias <- train_set %>%
  group_by(main_genre) %>%
  summarise(deviation_main_genre = mean(rating - movie_avg))

side1_genre_bias <- train_set %>%
  group_by(side1_genre) %>%
  summarise(deviation_side1_genre = mean(rating - movie_avg))

side2_genre_bias <- train_set %>%
  group_by(side2_genre) %>%
  summarise(deviation_side2_genre = mean(rating - movie_avg))

mean_movie_user_all_genre_model <- test_set %>%
  inner_join(movie_bias, by="movieId") %>%
  inner_join(user_bias, by="userId") %>%
  inner_join(main_genre_bias, by="main_genre") %>%
```

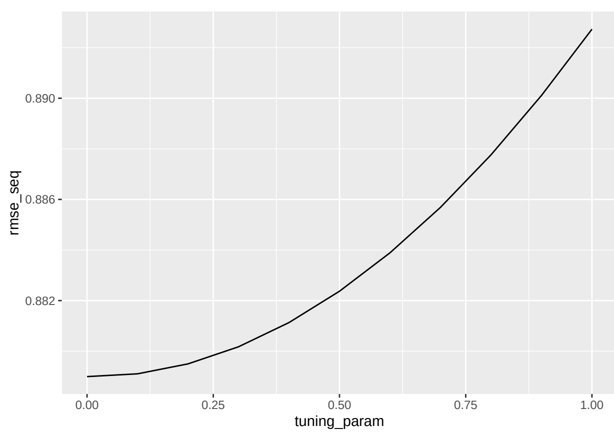
```r
inner_join(side1_genre_bias, by="side1_genre") %>%
inner_join(side2_genre_bias, by="side2_genre")
```

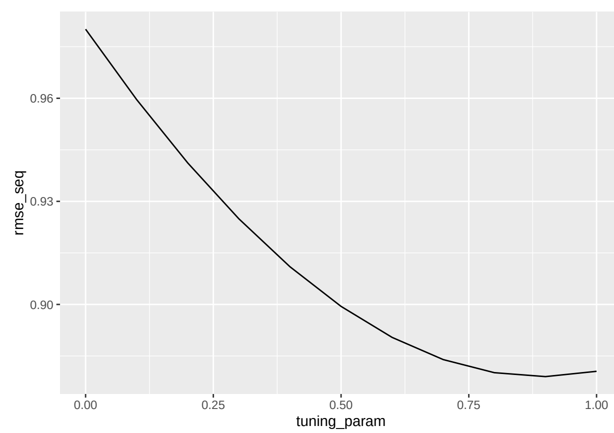
This resulted in 0.9305867.

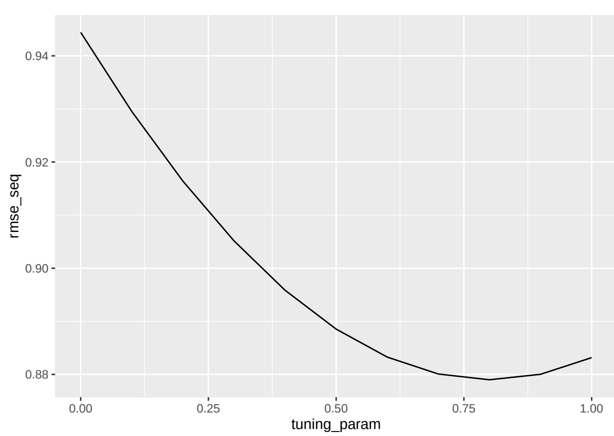
Model	RMSE
Guessing	2.1557864
Avg Model	1.0598665
Genre Model	1.0480846
Movie Model	0.9431683
User Model	0.9783862
Release Year Model	1.0489378
Movie + User Model	0.8854123
Movie + User + Release Year Model	0.9017918
Movie + User + Genre Model	0.9026264
Movie + User + Genre + Release Year Model	0.9203211
Movie + User + All Genre Model	0.9305867

Tuning

Lets try to tune the three biases, each individually and plot the tuning parameter and the resulting RMSE.







After searching for each tuning parameter individually these are the final tuning parameters:

- genre_bias: 0.0055
- movie_bias: 0.8944
- user_bias: 0.798

The resulting tuned function with the individual bias tuning factors:

```
tuned_movie_user_genre_model <- function(t) {
  avg <- mean(train_set$rating)

  genre_bias <- train_set %>%
    group_by(main_genre) %>%
    summarise(deviation_genre = 0.0055 * sum(rating - movie_avg)/n())

  movie_bias <- train_set %>%
    group_by(movieId) %>%
    summarise(deviation_movie = 0.8944 * sum(rating - movie_avg)/n())

  user_bias <- train_set %>%
    group_by(userId) %>%
    summarise(deviation_user = 0.798 * sum(rating - movie_avg)/n())

  model <- test_set %>%
    inner_join(genre_bias, by="main_genre") %>%
    inner_join(movie_bias, by="movieId") %>%
    inner_join(user_bias, by="userId")

  model$predicted_rating <- model$deviation_genre +
                            model$deviation_user +
                            model$deviation_movie +
                            movie_avg

  return(RMSE(test_set$rating, model$predicted_rating))
}

# add result to table
ml_results <- ml_results %>%
  bind_rows(tibble(Model="Tuned model", RMSE=tuned_movie_user_genre_model()))
```

Model	RMSE
Guessing	2.1557864
Avg Model	1.0598665
Genre Model	1.0480846
Movie Model	0.9431683
User Model	0.9783862
Release Year Model	1.0489378
Movie + User Model	0.8854123
Movie + User + Release Year Model	0.9017918
Movie + User + Genre Model	0.9026264
Movie + User + Genre + Release Year Model	0.9203211
Movie + User + All Genre Model	0.9305867
Tuned model	0.8789933

Results

RMSE

Lets test the final model with its tuning in place against the verification set.

```
final_model_prediction <- function() {
  avg <- mean(train_set$rating)
```

```r
  #genre_bias <- train_set %>%
  #  group_by(main_genre) %>%
  #  summarise(deviation_genre = 0.0055 * sum(rating - movie_avg)/n())

  movie_bias <- train_set %>%
    group_by(movieId) %>%
    summarise(deviation_movie = 0.8944 * sum(rating - movie_avg)/n())

  user_bias <- train_set %>%
    group_by(userId) %>%
    summarise(deviation_user = 0.798 * sum(rating - movie_avg)/n())

  model <- final_holdout_test %>%
    #inner_join(genre_bias, by="main_genre") %>%
    inner_join(movie_bias, by="movieId") %>%
    inner_join(user_bias, by="userId")

  model$predicted_rating <- #model$deviation_genre +
    model$deviation_user +
    model$deviation_movie +
    movie_avg
  return(model$predicted_rating)
}
```

This are all the achieved model RMSE:

Model	RMSE
Guessing	2.1557864
Avg Model	1.0598665
Genre Model	1.0480846
Movie Model	0.9431683
User Model	0.9783862
Release Year Model	1.0489378
Movie + User Model	0.8854123
Movie + User + Release Year Model	0.9017918
Movie + User + Genre Model	0.9026264
Movie + User + Genre + Release Year Model	0.9203211
Movie + User + All Genre Model	0.9305867
Tuned model	0.8789933
Final Model Verification	0.8798800

Conclusion

Even with all this tuning, lower than 0.87988 is not possible with this approach. More training and maybe separating different features is needed.

Future Improvements

Further information about the users could improve accuracy, e.g. shopping preferences, music taste, background information like education level. But there privacy concern about the usage of user personal data has to be considered. Training with larger dataset would be beneficial but would require more capable systems (e.g. with GPU). Even with this dataset and no more features or data other model algorithms could be tried, for example KNN, SVM or Decision Tree could be tried. For further interesting model trainings neither the time nor the available computing power was sufficient.

Resources

[1] Rafael Irizarry. 2018. Introduction to Data Science.https://rafalab.dfci.harvard.edu/dsbook/

System

Hardware

All above computations are done with an 12th Gen Intel(R) Core(TM) i7-1255U CPU with 12 and 32.00 GB of RAM.

Software

This report is compiled using R markdown with RStudio.

```
sessionInfo()
```

```
## R version 4.2.2 (2022-10-31 ucrt)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 22621)
##
## Matrix products: default
##
## Random number generation:
##  RNG:     Mersenne-Twister
##  Normal:  Inversion
##  Sample:  Rounding
##
## locale:
## [1] LC_COLLATE=English_United States.utf8
## [2] LC_CTYPE=English_United States.utf8
## [3] LC_MONETARY=English_United States.utf8
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United States.utf8
##
## attached base packages:
## [1] parallel  stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
##  [1] caret_6.0-94      lattice_0.22-5    forcats_1.0.0     dplyr_1.1.3
##  [5] purrr_1.0.2       readr_2.1.4       tidyr_1.3.0       tibble_3.2.1
##  [9] tidyverse_2.0.0   kableExtra_1.3.4  stringr_1.5.0     scales_1.2.1
## [13] lubridate_1.9.3   rmarkdown_2.25    benchmarkme_1.0.8 devtools_2.4.5
## [17] usethis_2.2.2     ggplot2_3.4.4     pacman_0.5.1
##
## loaded via a namespace (and not attached):
##   [1] colorspace_2.1-0      ellipsis_0.3.2        class_7.3-22
##   [4] fs_1.6.3              rstudioapi_0.15.0     farver_2.1.1
##   [7] listenv_0.9.0         remotes_2.4.2.1       bit64_4.0.5
##  [10] prodlim_2023.08.28    fansi_1.0.5           xml2_1.3.5
##  [13] codetools_0.2-19      splines_4.2.2         doParallel_1.0.17
##  [16] cachem_1.0.8          knitr_1.45            pkgload_1.3.3
##  [19] pROC_1.18.5           shiny_1.7.5.1         compiler_4.2.2
##  [22] httr_1.4.7            Matrix_1.6-2          fastmap_1.1.1
```

```
##  [25] cli_3.6.1             later_1.3.1          htmltools_0.5.7
##  [28] prettyunits_1.2.0     tools_4.2.2          gtable_0.3.4
##  [31] glue_1.6.2            reshape2_1.4.4       Rcpp_1.0.11
##  [34] vctrs_0.6.4           svglite_2.1.2        nlme_3.1-163
##  [37] iterators_1.0.14      timeDate_4022.108    gower_1.0.1
##  [40] xfun_0.41             globals_0.16.2       ps_1.7.5
##  [43] rvest_1.0.3           timechange_0.2.0     mime_0.12
##  [46] miniUI_0.1.1.1        lifecycle_1.0.4      future_1.33.0
##  [49] MASS_7.3-60           ipred_0.9-14         vroom_1.6.4
##  [52] hms_1.1.3             promises_1.2.1       yaml_2.3.7
##  [55] memoise_2.0.1         rpart_4.1.21         stringi_1.7.12
##  [58] highr_0.10            foreach_1.5.2        hardhat_1.3.0
##  [61] pkgbuild_1.4.2        lava_1.7.3           benchmarkmeData_1.0.4
##  [64] rlang_1.1.2           pkgconfig_2.0.3      systemfonts_1.0.5
##  [67] evaluate_0.23         labeling_0.4.3       recipes_1.0.8
##  [70] htmlwidgets_1.6.2     bit_4.0.5            processx_3.8.2
##  [73] tidyselect_1.2.0      parallelly_1.36.0    plyr_1.8.9
##  [76] magrittr_2.0.3        R6_2.5.1             generics_0.1.3
##  [79] profvis_0.3.8         mgcv_1.9-0           pillar_1.9.0
##  [82] withr_2.5.2           survival_3.5-7       nnet_7.3-19
##  [85] future.apply_1.11.0   crayon_1.5.2         utf8_1.2.4
##  [88] tzdb_0.4.0            urlchecker_1.0.1     grid_4.2.2
##  [91] data.table_1.14.8     callr_3.7.3          ModelMetrics_1.2.2.2
##  [94] digest_0.6.33         webshot_0.5.5        xtable_1.8-4
##  [97] httpuv_1.6.12         stats4_4.2.2         munsell_0.5.0
## [100] viridisLite_0.4.2     sessioninfo_1.2.2
```