# Contents



-9– title: “Nclode MovieLens Report” author: “Nicholas Clode” date: “07/06/2019” output: pdf_document —

# 1 Executive Summary

The purpose of this project is to build and train a machine learning model in R which predicts movie ratings. The model is trained on the MovieLens data set, which contains 10MM user movie ratings along with movie title, genres and timestamp.
The target of this model is to achieve a Root Mean Square Error (RMSE) of less than or equal to **0.87750** This is done by creating 6 individual models and generating an ensemble model which the RMSE is used to compare to the target.

The target is achieved as the RMSE score of the final ensemble model generated is **0.86416**

### 1.1 Data & Subsets

The MovieLens 10M dataset is split into subsets for the purpose of training, testing and cross validating.

**1.Validation set "Validation":** This subset is 10% of the total MovieLens data and is used for testing the final model and generating the final RMSE score **only**. This dataset will not be used until the final step.

**2.The Training set "edx":** This subset is 90% of the total MovieLens data and is used to train the final model. In addition, this set is split into additional subsets which are used to train and test the individual models and for cross validation when deciding on regularization parameters.
**edx training subset "train_set":** 90% of the edx subset and is used to train the individual models
**edx testing subset "test_set":** 10% of the edx subset and is used to test and compare the individual models

### 1.2 Approach

1. Generate 6 individual prediction models with the train_set, make predictions for the ratings on the test_set and calculate the RMSE of each. This will begin with a simple model which uses the average movie rating as the prediction every time. Each subsequent model will add one additional attribute.
2. Models will be regularized where appropriate using the penalized least squares method
3. Models will be compared using the RMSE score obtained when predicting using the test_set
4. The top 4 models will be used to generate a **final ensemble model** which is an optimized weighted average of the individual models
5. The Ensemble model will be used to make a final prediction using the Validation set. The final RMSE score which will be compared against the target will be generated using this model.

## 2 Data exploration

First examine the MovieLens data which was imported using the code provided by the Capstone Course

```
# Get the dimensions of the full MovieLens dataset
dim(movielens)
```

```
## [1] 9000055       6
```

```
# Names of each column
names(movielens)
```

```
## [1] "userId"    "movieId"   "rating"    "timestamp" "title"     "genres"
```

```
# Unique userIds, MovieIds and genre combinations
movielens %>%
  summarize(n_users = n_distinct(userId),
            n_movies = n_distinct(movieId),
            n_genres = n_distinct(genres))
```

```
##   n_users n_movies n_genres
## 1   69878    10677      797
```

```
# View the first 5 rows of the dataset
movielens[1:5,]
```

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

## 2.1 Additional Attributes

Viewing the first rows reveals that the title column contains information on the movie release year. This will be extracted from the title column.
In addition the MovieLens dataset readme ( link ) says the timestamp "represents seconds since midnight ~ Coordinated Universal Time (UTC) of **January 1, 1970**." therefore the year the rating was made will be extracted from the timestamp as a new column using an origin of **1970-01-01**

```
# extract the movie release year from the title and rating year from the timestamp
# into two new columns column called "movie_year" and "rating_year" Year".
edx <- edx %>%
  extract(title, regex = "([0-9][0-9][0-9][0-9])", into = "movie_year"
          , remove = FALSE,convert = TRUE) %>%
  mutate(rating_year = year(as.Date(as.POSIXlt(timestamp, origin="1970-01-01"))))

validation <- validation %>%
  extract(title, regex = "([0-9][0-9][0-9][0-9])", into = "movie_year"
          , remove = FALSE,convert = TRUE) %>%
  mutate(rating_year = year(as.Date(as.POSIXlt(timestamp, origin="1970-01-01"))))
```

Now that the additional columns added, split the edx set into the 2 further subsets *train_set* and *test_set*

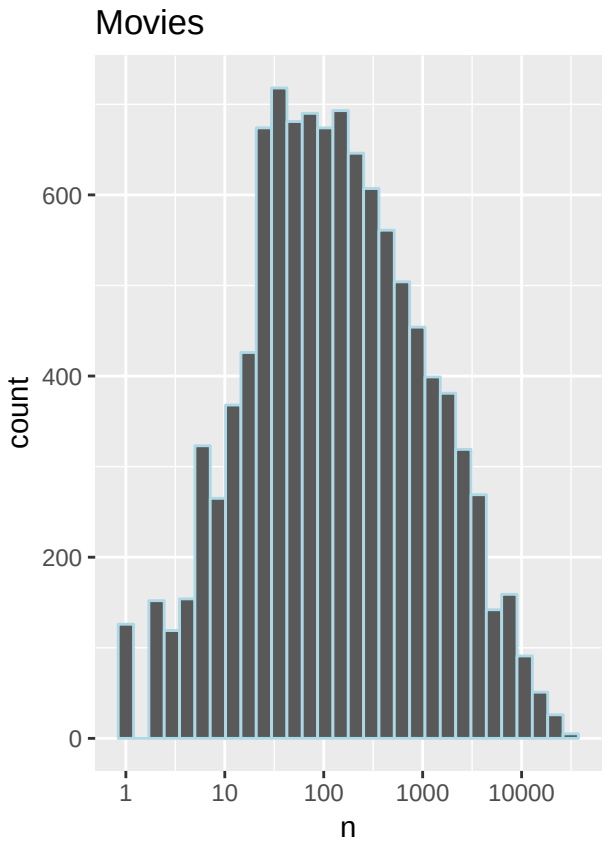
```
# Create data partitions on the 'edx' training data
set.seed(1)
test_index <- createDataPartition(edx$rating, times = 1, p = 0.1, list = FALSE)
train_set <- edx %>% slice(-test_index)
test_set_temp <- edx %>% slice(test_index)

set.seed(1)
test_set <- test_set_temp %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")

removed <- anti_join(test_set_temp, test_set)
train_set <- rbind(train_set, removed)
```

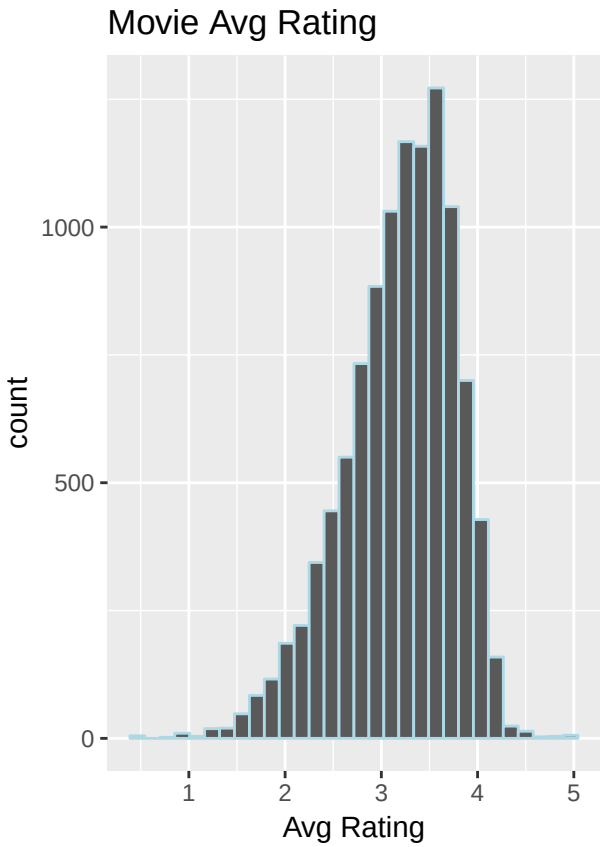
# 3 Use plots to further explore the data

Now further explore the data by generating plots

## 3.1 Ratings by MovieId

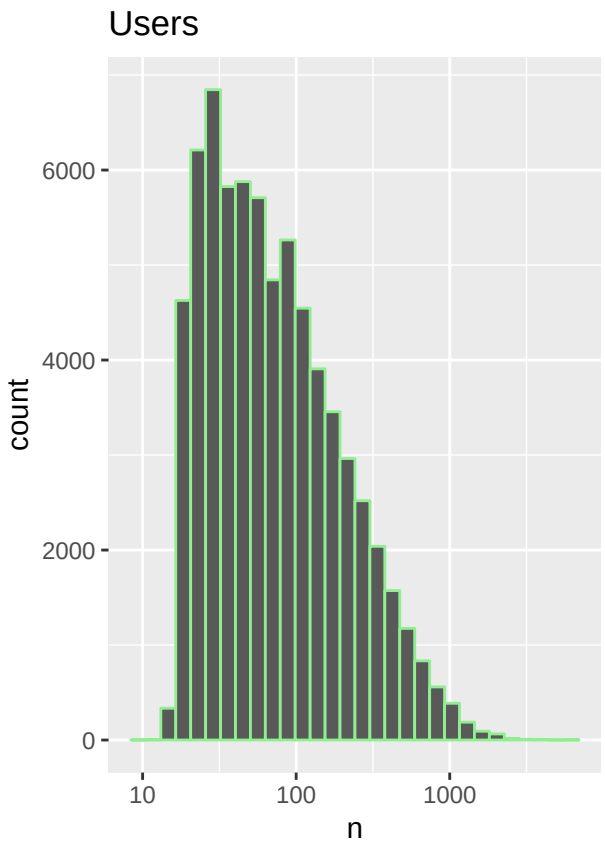
The below charts show variation between the amount of ratings and the average rating for each movie (movieId). Some movies have more ratings than others and some movies have higher average rating scores than others.

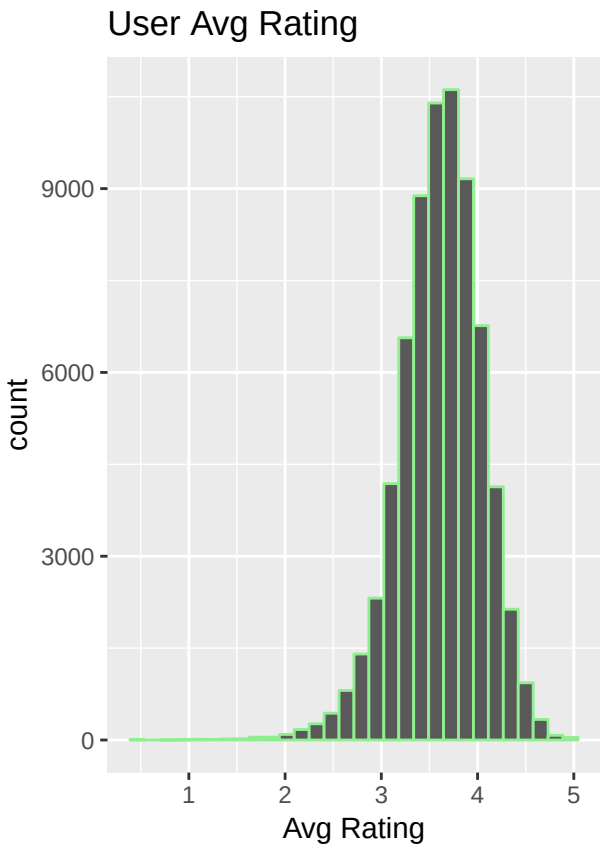



## 3.2 Ratings by UserId

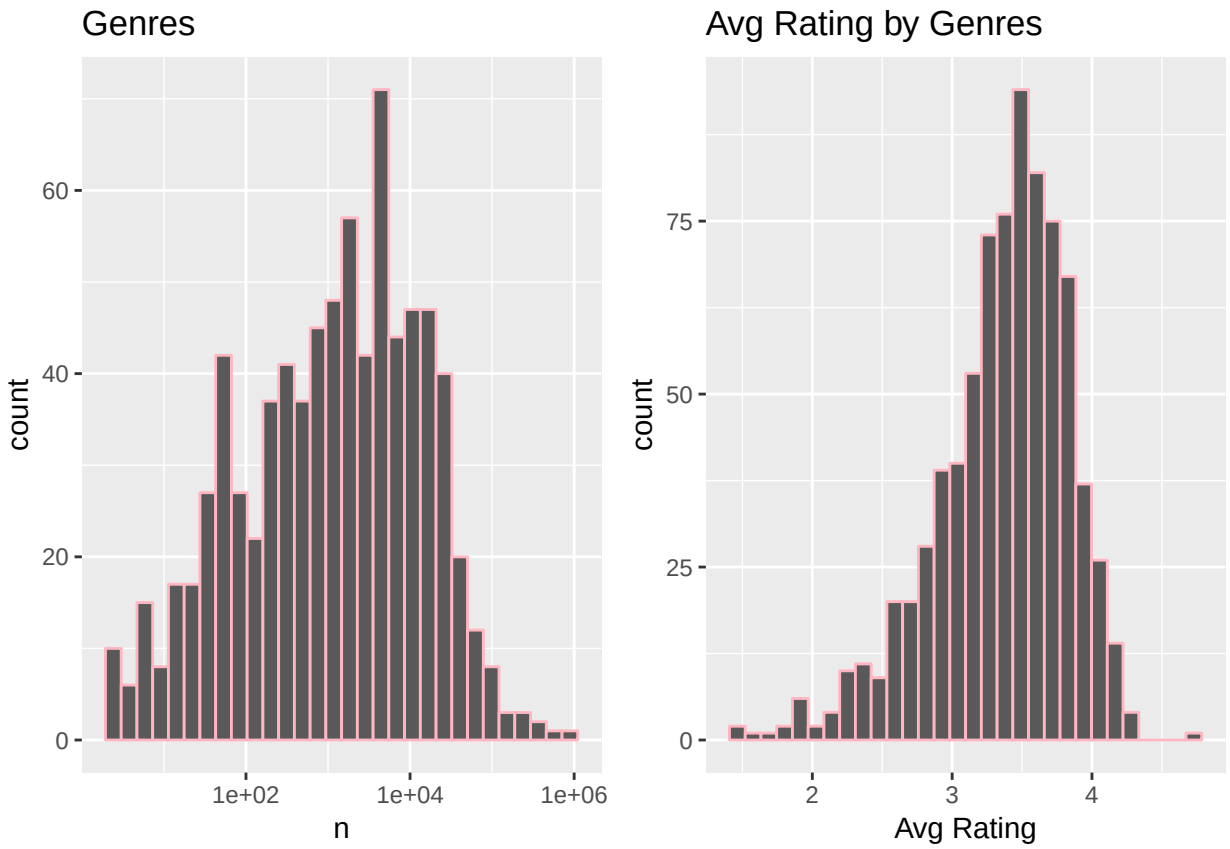
The below charts show variation between the amount of ratings each user has submitted, and the users average rating. Some Users are on average harsher critiques than others when rating movies and some like to rate more movies than others



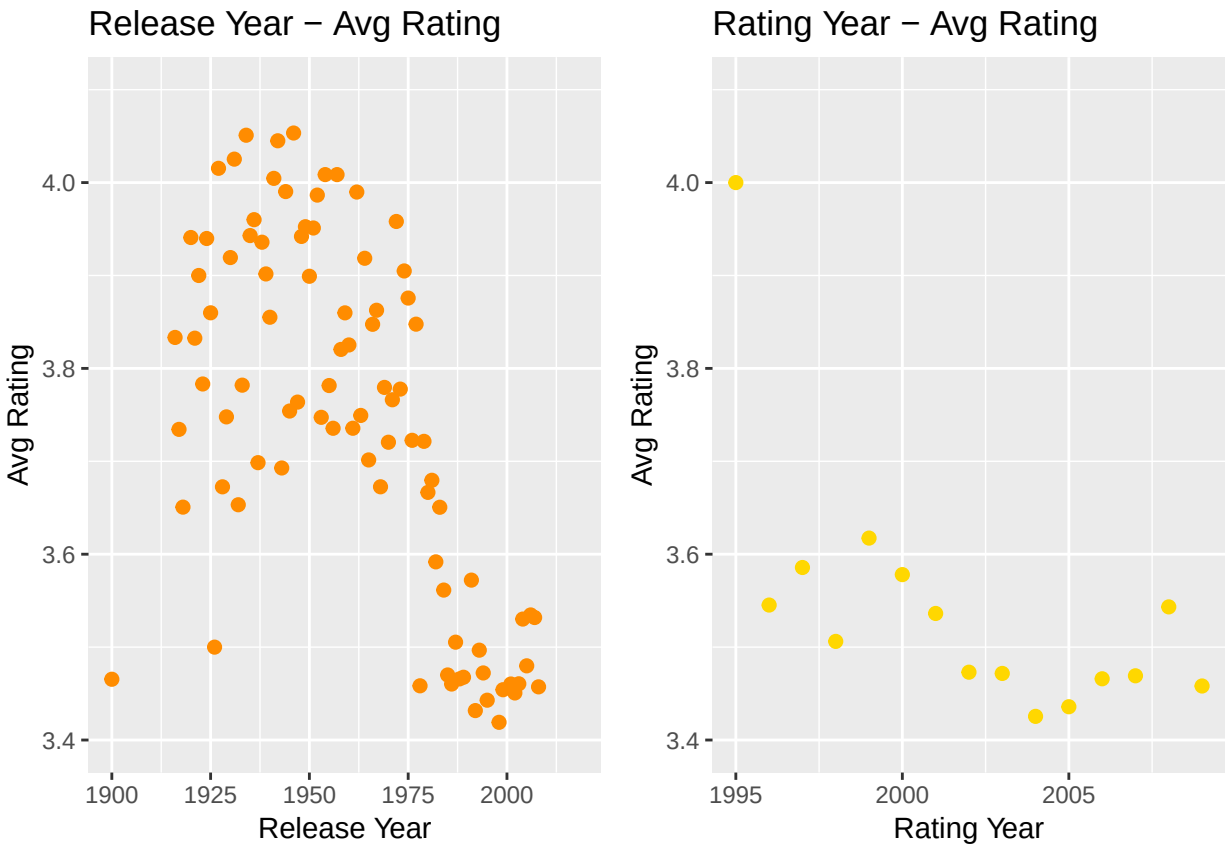

### 3.3 Ratings by Genres

The below charts show variation between the amount of ratings each genres combination has (movies can have multiple genres), and the average rating of each




### 3.4 Ratings by Year of Movie Release and Year of Rating

The below charts show the additional attributes, "movie_year" and "rating_year" and the variation in average ratings.
**Movie Year:** Old movies released prior to the 80's appear to have higher average reviews than more recent movies.
**Rating Year:** Ratings that were made in the late 90's appear more generous than ratings in the early 00's. There is also an outlier in 1995 where the average was 4.




# 4 The Models

## 4.1 Model 1 - Simple Average

The first model is just using the average movie rating as the prediction across all movies and users. In this case all differences explained by random variation. The equation for the rating of movie i by user u is:

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

$\varepsilon_{i,u}$ independent errors are sampled from the same distribution centred at 0 and $\mu$ the true rating for all movies. estimate that minimizes the RMSE is the least squares estimate of $\mu$ which is the average of all ratings:

The mean of the train_set $\hat{\mu}$ used for prediction below

```
# Model 1 take the mean of the train_set ratings.
# Note: the _t is used to tag variables from the train_set/test_set

mu_hat_t <- mean(train_set$rating)
# Use the RMSE formula to
model_1_rmse_t <- RMSE(test_set$rating, mu_hat_t)
model_1_rmse_t
```

```
## [1] 1.060054
```

## 4.2 Model 2 - Add Average Movie effect

Not all movies have the same rating as seen in the MovieId charts above.
Model 2 is an augmented version of Model 1 with the term: **beta_i** added to represent the average movie rating for movie i

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

The least square estimate of beta_i_hat is the average of $Y_{u,i} - \hat{\mu}$ for each movie i

```
# Calculate beta_i using the train_set
movie_avg_t <- train_set %>%
  group_by(movieId) %>%
  summarize(beta_i = mean(rating - mu_hat_t))

# Join on to the test_set and make the predictions
predicted_ratings_2_t <- test_set %>%
  left_join(movie_avg_t, by = "movieId") %>%
  mutate(pred = mu_hat_t + beta_i) %>%
  pull(pred)

# Pull the RMSE
model_2_rmse_t <- RMSE(predicted_ratings_2_t, test_set$rating)
model_2_rmse_t
```

```
## [1] 0.9429615
```

## 4.3 Model R3 - Add Average User effect

"_R" denotes Regularized

The userId charts above showed there is variability among the average ratings of individual users.
Model R3 introduces a term: **beta_u** user-specific effect.

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

beta_u is estimated by $y_{u,i} - \hat{\mu} - \hat{b}_i$

In addition the charts above showed that some movies and some users have a low count of ratings. The lower the count of ratings a user has the greater the uncertainty will be. Therefore Regularization is used to penalize large estimates that are formed using small counts of ratings.

This is done for both the movie and user bias introducing Lambda_1 representing the penalization parameter for movieId and Lambda_2 as the penalization parameter for userId

```
# Select the Lambda values
# create a grid of potential values for lambda_1 and Lambda_2.
lambdas <- seq(0, 6, 0.5)
lamdas_matrix <- expand.grid(Lam_1 = lambdas , Lam_2 = lambdas)

# create a function to select the optimal values for lambda
# Note: using the test_set and train_sets as the validation_set
# is reserved for the final testing only)
```

```
cross_val_R3  <- function(L1,L2){
  mu_hat <- mean(train_set$rating)

  movie_avg <- train_set %>%
    group_by(movieId) %>%
    summarize(beta_i = sum(rating - mu_hat)/(n()+L1))

  user_bias <- train_set %>%
    left_join(movie_avg, by = "movieId") %>%
    group_by(userId) %>%
    summarize(beta_u = sum(rating - beta_i - mu_hat)/(n()+L2))

  predicted_ratings <- test_set %>%
    left_join(movie_avg, by = "movieId") %>%
    left_join(user_bias , by= "userId") %>%
    mutate(pred = mu_hat + beta_i + beta_u) %>%
    pull(pred)
  rmse <-  RMSE(predicted_ratings, test_set$rating)
  return(rmse)
}
# Use "mapply" to apply the cross_val_R3 function to the potential lambdas
rmses_R3  <- mapply(cross_val_R3 ,lamdas_matrix[,1],lamdas_matrix[,2])

# Get the optimal lambdas
Lambda_R3  <- lamdas_matrix[which.min(rmses_R3 ),]
Lambda_R3
```

```
##     Lam_1 Lam_2
## 140   4.5     5
```

Now generate the RMSE for model 3 using the penalization parameters chosen

```
# Calculate beta_i using the train_set and penalization parameters
#"_R" denotes regularized
movie_avg_t_R <- train_set %>%
  group_by(movieId) %>%
  summarize(beta_i = sum(rating - mu_hat_t)/(n()+Lambda_R3 [,1]))

# Calculate beta_u using the train_set and regularization
user_bias_t_R <- train_set %>%
  left_join(movie_avg_t_R, by = "movieId") %>%
  group_by(userId) %>%
  summarize(beta_u = sum(rating - beta_i - mu_hat_t)/(n()+Lambda_R3 [,2]))

# make prediction using the test_set
predicted_ratings_R3_t <- test_set %>%
  left_join(movie_avg_t_R, by = "movieId") %>%
  left_join(user_bias_t_R , by= "userId") %>%
  mutate(pred = mu_hat_t + beta_i + beta_u) %>%
  pull(pred)

# get the RMSE of model 3
```

```
model_R3_rmse_t <- RMSE(predicted_ratings_R3_t, test_set$rating)
model_R3_rmse_t
```

```
## [1] 0.8641359
```

### 4.4 Model R4 - Add Average Genres effect

The Genres charts above showed there is variability among the average ratings of each combination of genres. Model R4 introduces a term: **beta_g** as a genre specific effect

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

beta_g is estimated by $y_{u,i} - \hat{\mu} - \hat{b}_i - \hat{b}_u$

```
# create a grid of potential values for the penalization parameter for beta_g
lambdas <- seq(0, 6, 0.5)
# create a function to select the optimal values for lambda
# Note:  using the test_set and train_sets as the validation_set
#is reserved for the final testing only)

cross_val_R4 <- function(L4){

  mu_hat <- mean(train_set$rating)

  movie_avg <- train_set %>%
    group_by(movieId) %>%
    summarize(beta_i = sum(rating - mu_hat)/(n()+4.5))

  user_bias <- train_set %>%
    left_join(movie_avg_t, by = "movieId") %>%
    group_by(userId) %>%
    summarize(beta_u = sum(rating - beta_i - mu_hat)/(n()+5))

  genres_bias <- train_set %>%
    left_join(movie_avg , by = "movieId") %>%
    left_join(user_bias, by = "userId") %>%
    group_by(genres) %>%
    summarize(beta_g = sum(rating - mu_hat - beta_i - beta_u)/(n()+L4))

  predicted_ratings <- test_set %>%
    left_join(movie_avg, by= "movieId") %>%
    left_join(user_bias, by= "userId") %>%
    left_join(genres_bias, by= "genres") %>%
    mutate(pred = mu_hat + beta_i + beta_u + beta_g) %>%
    pull(pred)

  rmse <-  RMSE(predicted_ratings, test_set$rating)
  return(rmse)
}
# Use "sapply" to apply the cross_val_R4 function to the lambda vector
rmses_R4  <- sapply(lambdas,cross_val_R4 )
# Get the optimal lambda for the beta_g
```

```
Lambda_R4  <- lambdas[which.min(rmses_R4 )]
Lambda_R4
```

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

In this case the optimal penalization parameter for beta_g is zero.

```
# Calculate beta_g using the train_set
genres_bias_t_R <- train_set %>%
  left_join(movie_avg_t_R , by = "movieId") %>%
  left_join(user_bias_t_R, by = "userId") %>%
  group_by(genres) %>%
  summarize(beta_g = sum(rating - mu_hat_t - beta_i - beta_u)/(n()+Lambda_R4))

# make prediction using the test_set
predicted_ratings_R4_t <- test_set %>%
  left_join(movie_avg_t_R, by= "movieId") %>%
  left_join(user_bias_t_R, by= "userId") %>%
  left_join(genres_bias_t_R, by= "genres") %>%
  mutate(pred = mu_hat_t + beta_i + beta_u + beta_g) %>%
  pull(pred)

# get the RMSE of model R4
model_R4_rmse_t <- RMSE(predicted_ratings_R4_t, test_set$rating)
model_R4_rmse_t
```

```
## [1] 0.8638141
```

### 4.5 Model R5 - Add Movie Release Year effect

The charts for avg rating by movie release year shows variation in average rating by year.

Therefore an additional parameter will be added to the model to account for this 'beta_d'

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

beta_d is estimated by $y_{u,i} - \hat{\mu} - \hat{b}_i - \hat{b}_u - \hat{b}_g$

```
# create a grid of potential values for the lambda associated with the movie release year
lambdas <- seq(0, 6, 0.5)
# create a function to select the optimal values for lambda
cross_val_R5 <- function(L5){

  mu_hat <- mean(train_set$rating)

  movie_avg <- train_set %>%
    group_by(movieId) %>%
    summarize(beta_i = sum(rating - mu_hat)/(n()+4.5))

  user_bias <- train_set %>%
    left_join(movie_avg, by = "movieId") %>%
    group_by(userId) %>%
```

```
    summarize(beta_u = sum(rating - beta_i - mu_hat)/(n()+5))

  genres_bias <- train_set %>%
    left_join(movie_avg , by = "movieId") %>%
    left_join(user_bias, by = "userId") %>%
    group_by(genres) %>%
    summarize(beta_g = sum(rating - mu_hat - beta_i - beta_u)/(n()+0))

  movie_year_bias <- train_set %>%
    left_join(movie_avg, by= "movieId") %>%
    left_join(user_bias, by= "userId") %>%
    left_join(genres_bias, by= "genres") %>%
    group_by(movie_year) %>%
    summarize(beta_d = sum(rating - mu_hat - beta_i - beta_u - beta_g)/(n()+ L5))

  predicted_ratings <- test_set %>%
    left_join(movie_avg, by= "movieId") %>%
    left_join(user_bias, by= "userId") %>%
    left_join(genres_bias, by= "genres")%>%
    left_join(movie_year_bias, by= "movie_year") %>%
    mutate(pred = mu_hat + beta_i + beta_u + beta_g + beta_d) %>%
    pull(pred)

  rmse <-  RMSE(predicted_ratings, test_set$rating)
  return(rmse)
}
# Use "sapply" to apply the cross_val_R5 function to the potential lambdas
rmses_R5  <- sapply(lambdas,cross_val_R5 )
# Get the optimal lambda for the beta_d
Lambda_R5  <- lambdas[which.min(rmses_R5 )]
Lambda_R5
```

```
## [1] 6
```

Now generate the RMSE for Model 5

```
# Calculate beta_d using the train_set
movie_year_bias_t_R <- train_set %>%
  left_join(movie_avg_t_R, by= "movieId") %>%
  left_join(user_bias_t_R, by= "userId") %>%
  left_join(genres_bias_t_R, by= "genres") %>%
  group_by(movie_year) %>%
  summarize(beta_d = sum(rating - mu_hat_t - beta_i - beta_u - beta_g)/(n()+ Lambda_R5))

# make prediction using the test_set
predicted_ratings_R5_t <- test_set %>%
  left_join(movie_avg_t_R, by= "movieId") %>%
  left_join(user_bias_t_R, by= "userId") %>%
  left_join(genres_bias_t_R, by= "genres")%>%
  left_join(movie_year_bias_t_R, by= "movie_year") %>%
  mutate(pred = mu_hat_t + beta_i + beta_u + beta_g + beta_d) %>%
  pull(pred)
```

```
# get the RMSE of model R5
model_R5_rmse_t <- RMSE(predicted_ratings_R5_t, test_set$rating)
model_R5_rmse_t
```

```
## [1] 0.8636465
```

### 4.6 Model R6 - Add Year of Rating Bias

The charts for avg rating by the year the rating was added (timestamp) shows variation across Therefore an additional parameter will be added to the model to account for this 'beta_r'

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

beta_r is estimated by $y_{u,i} - \hat{\mu} - \hat{b}_i - \hat{b}_u - \hat{b}_g - \hat{b}_d$

```
# create a grid of potential values for the lambda associated with the beta_r
lambdas <- seq(0, 6, 0.5)

# create a function to select the optimal values for lambda
cross_val_R6 <- function(L6){

  mu_hat <- mean(train_set$rating)

  movie_avg <- train_set %>%
    group_by(movieId) %>%
    summarize(beta_i = sum(rating - mu_hat)/(n()+4.5))

  user_bias <- train_set %>%
    left_join(movie_avg, by = "movieId") %>%
    group_by(userId) %>%
    summarize(beta_u = sum(rating - beta_i - mu_hat)/(n()+5))

  genres_bias <- train_set %>%
    left_join(movie_avg , by = "movieId") %>%
    left_join(user_bias, by = "userId") %>%
    group_by(genres) %>%
    summarize(beta_g = sum(rating - mu_hat - beta_i - beta_u)/(n()+0))

  movie_year_bias <- train_set %>%
    left_join(movie_avg, by= "movieId") %>%
    left_join(user_bias, by= "userId") %>%
    left_join(genres_bias, by= "genres") %>%
    group_by(movie_year) %>%
    summarize(beta_d = sum(rating - mu_hat - beta_i - beta_u - beta_g)/(n()+ 6))

  movie_rating_year <- train_set %>%
    left_join(movie_avg , by= "movieId") %>%
    left_join(user_bias , by= "userId") %>%
    left_join(genres_bias , by= "genres") %>%
    left_join(movie_year_bias , by= "movie_year") %>%
    group_by(rating_year) %>%
    summarize(beta_r = sum(rating - mu_hat - beta_i - beta_u - beta_g - beta_d)/(n()+L6))
```

```
  predicted_ratings  <- test_set %>%
    left_join(movie_avg , by= "movieId") %>%
    left_join(user_bias , by= "userId") %>%
    left_join(genres_bias , by= "genres")%>%
    left_join(movie_year_bias , by= "movie_year") %>%
    left_join(movie_rating_year, by= "rating_year") %>%
    mutate(pred = mu_hat  + beta_i + beta_u + beta_g + beta_d + beta_r) %>%
    pull(pred)


  rmse <-  RMSE(predicted_ratings, test_set$rating)
  return(rmse)
}
# Use "sapply" to apply the cross_val_R6 function to the potential lambdas
rmses_R6  <- sapply(lambdas,cross_val_R6)
# Get the optimal lambda for the beta_r
Lambda_R6  <- lambdas[which.min(rmses_R6)]
Lambda_R6
```

```
## [1] 6
```

Now generate the RMSE for Model 6

```
# Calculate beta_r using the train_set
movie_rating_year_t_R <- train_set %>%
  left_join(movie_avg_t_R , by= "movieId") %>%
  left_join(user_bias_t_R , by= "userId") %>%
  left_join(genres_bias_t_R , by= "genres") %>%
  left_join(movie_year_bias_t_R , by= "movie_year") %>%
  group_by(rating_year) %>%
  summarize(beta_r = sum(rating - mu_hat_t - beta_i - beta_u - beta_g - beta_d)/(n()+ Lambda_R6))

# make prediction using the test_set
predicted_ratings_R6_t  <- test_set %>%
  left_join(movie_avg_t_R , by= "movieId") %>%
  left_join(user_bias_t_R , by= "userId") %>%
  left_join(genres_bias_t_R , by= "genres")%>%
  left_join(movie_year_bias_t_R , by= "movie_year") %>%
  left_join(movie_rating_year_t_R, by= "rating_year") %>%
  mutate(pred = mu_hat_t  + beta_i + beta_u + beta_g + beta_d + beta_r) %>%
  pull(pred)

# get the rmse of model 6
model_R6_rmse_t  <- RMSE(predicted_ratings_R6_t,test_set$rating)
model_R6_rmse_t
```

```
## [1] 0.8635238
```

# 5 Interim Results

Models R6 has the best RMSE so far, but is very close to the RMSE of Model R5 and R4

```
#Model results from the test_set so far
model_results <- as.data.frame(rbind(
  c(model = "model 1", RMSE =    round(model_1_rmse_t,4))
  ,c(model = "model 2", RMSE =   round(model_2_rmse_t,4))
  ,c(model = "model R3", RMSE =  round(model_R3_rmse_t,4))
  ,c(model = "model R4", RMSE =  round(model_R4_rmse_t,4))
  ,c(model = "model R5", RMSE =  round(model_R5_rmse_t,4))
  ,c(model = "model R6", RMSE =  round(model_R6_rmse_t,4))))
model_results
```

```
##      model   RMSE
## 1  model 1 1.0601
## 2  model 2  0.943
## 3 model R3 0.8641
## 4 model R4 0.8638
## 5 model R5 0.8636
## 6 model R6 0.8635
```

# 6 Final Model - Ensemble

In order to generate the final model, combine the best scoring models: R3,R4,R5 and R6 via a weighted average of the predictions of each. The weighting of each model need to be calculated.
The below code will select the optimal weighting of each model to use.

```
# Combine the predictions of the 4 models
prediction_matrix <- cbind(predicted_ratings_R3_t
                           ,predicted_ratings_R4_t
                           ,predicted_ratings_R5_t
                           ,predicted_ratings_R6_t)

# to determine the weightings, create a matrix of possible proportions such that each row adds to 1.
# Use the restrictedparts function from the partitions package

weights <- as.matrix(t(restrictedparts(100,ncol(prediction_matrix)
                                       ,include.zero = TRUE
                                       , decreasing = FALSE)/100))

weights <- rbind(
  weights
  ,cbind(weights[,2],weights[,3],weights[,4],weights[,1])
  ,cbind(weights[,3],weights[,4],weights[,1],weights[,2])
  ,cbind(weights[,4],weights[,1],weights[,2],weights[,3]))

# create a function "ensemble weights" to evaluate the RMSE of the predictions
ensemble_weights <- function(w1,w2,w3,w4){

  ensemble_weights <- as.matrix(c(w1,w2,w3,w4))
  ensemble_prediction <- cbind(prediction_matrix %*% ensemble_weights,test_set$rating)
  rmse <- RMSE(ensemble_prediction[,1],ensemble_prediction[,2])
}
  #Apply the ensemble weights function to each row of the weights matrix using mapply
ens_rmse <- mapply(ensemble_weights,weights[,1],weights[,2],weights[,3],weights[,4])
```

```
ensemble_weights <- as.matrix(weights[which.min(ens_rmse),])

#optimal ensemble weights are give
ensemble_weights
```

```
##      [,1]
## [1,]  0.0
## [2,]  0.0
## [3,]  0.1
## [4,]  0.9
```

Only Models R5 and R6 have non zero weightings therefore the final ensemble will use only these to generate the predictions

```
# Generate the ensemble prediction on the test_set
ensembele_Pred_cv = prediction_matrix %*%  ensemble_weights
ensembele_rmse_cv <- RMSE(ensembele_Pred_cv,test_set$rating)
ensembele_rmse_cv
```

```
## [1] 0.8635222
```

```
#Final Ensemble RMSE

model_results <- as.data.frame(

  rbind(
    c(model = "model 1", RMSE = round(model_1_rmse_t,6))
    ,c(model = "model 2", RMSE =  round(model_2_rmse_t,6))
    ,c(model = "model R3", RMSE =  round(model_R3_rmse_t,6))
    ,c(model = "model R4", RMSE = round(model_R4_rmse_t,6))
    ,c(model = "model R5", RMSE =  round(model_R5_rmse_t,6))
    ,c(model = "model R6", RMSE =  round(model_R6_rmse_t,6))
    ,c(model = "Ensemble", RMSE = round(ensembele_rmse_cv,6))))

model_results
```

```
##      model     RMSE
## 1  model 1 1.060054
## 2  model 2 0.942961
## 3 model R3 0.864136
## 4 model R4 0.863814
## 5 model R5 0.863646
## 6 model R6 0.863524
## 7 Ensemble 0.863522
```

The Ensemble model has the best RMSE score when predicting ratings with the test_subset.

## 6.1 Train the Ensemble model with the Full edx data

Now train Models R5 and R6 for the ensemble with the **full edx data training data** using the penalization lambdas selected earlier

```r
# Model R5 using the EDX Data

mu_hat <- mean(edx$rating)

movie_avg_R <- edx %>%
  group_by(movieId) %>%
  summarize(beta_i = sum(rating - mu_hat)/(n()+Lambda_R3 [,1]))

user_bias_R <- edx  %>%
  left_join(movie_avg_R, by = "movieId") %>%
  group_by(userId) %>%
  summarize(beta_u = sum(rating - beta_i - mu_hat)/(n()+Lambda_R3 [,2]))

genres_bias_R  <- edx   %>%
  left_join(movie_avg_R  , by = "movieId") %>%
  left_join(user_bias_R , by = "userId") %>%
  group_by(genres) %>%
  summarize(beta_g = sum(rating - mu_hat - beta_i - beta_u)/(n()+Lambda_R4))

movie_year_bias_R <- edx %>%
  left_join(movie_avg_R, by= "movieId") %>%
  left_join(user_bias_R, by= "userId") %>%
  left_join(genres_bias_R, by= "genres") %>%
  group_by(movie_year) %>%
  summarize(beta_d = sum(rating - mu_hat - beta_i - beta_u - beta_g)/(n()+ Lambda_R5))

predicted_ratings_R5 <- validation %>%
  left_join(movie_avg_R, by= "movieId") %>%
  left_join(user_bias_R, by= "userId") %>%
  left_join(genres_bias_R, by= "genres")%>%
  left_join(movie_year_bias_R, by= "movie_year") %>%
  mutate(pred = mu_hat + beta_i + beta_u + beta_g + beta_d) %>%
  pull(pred)

# Model R6 using the EDX Data

movie_rating_year_R <- edx %>%
  left_join(movie_avg_R , by= "movieId") %>%
  left_join(user_bias_R , by= "userId") %>%
  left_join(genres_bias_R , by= "genres") %>%
  left_join(movie_year_bias_R , by= "movie_year") %>%
  group_by(rating_year) %>%
  summarize(beta_r = sum(rating - mu_hat - beta_i - beta_u - beta_g - beta_d)/(n()+ Lambda_R6))

predicted_ratings_R6  <- validation %>%
  left_join(movie_avg_R , by= "movieId") %>%
  left_join(user_bias_R , by= "userId") %>%
  left_join(genres_bias_R , by= "genres")%>%
  left_join(movie_year_bias_t_R , by= "movie_year") %>%
  left_join(movie_rating_year_R, by= "rating_year") %>%
  mutate(pred = mu_hat  + beta_i + beta_u + beta_g + beta_d + beta_r) %>%
  pull(pred)
```

### 6.2 Predict with the Ensemble Model

Apply the optimal ensemble weightings to generate the final RMSE

```
model_predictions <- cbind(predicted_ratings_R5,predicted_ratings_R6)
ensembele_final <- model_predictions %*% ensemble_weights[3:4]
ensembele_rmse <- RMSE(ensembele_final,validation$rating)
```

## 7 Conclusion

6 models were trained with the training subset of the edx data. The model_R6 that contained all the attributes examined had the best RMSE. However by combining it with model_R5 in an ensemble model this achieved the best result

The final RMSE of the Ensemble model when making predictions on the Validation set is **0.86416** and has beaten the target of **0.87750**