MovieLens Project - HarvardX:PH125.9x Data Science Capstone

Silvane Paixao (silpai)

1/2/2021

```
Time for the code to run - please be patience
   R script: Total estimated time is 80 min (1h40m)
```

From beginning to predictions: 40min Matrix factorization: 40min

```
Rmarkdown: Total estimated time is 2h40 min
```

1. Introduction

Recommendation systems rely on the notion of similarity between objects. Nowadays it has been applied to recommend movies, books, hotels, restaurants, doctors, etc. According to Forte and Miller (2017) customers can be considered like each other if they share most of the products that they have purchased, and items can be considered similar to each other if they share a large number of customers who purchased them.

In the real-world, Recommendation systems have been used by e-commerce websites such as Amazon.com, Netflix, iTunes to improve revenue. Customers are guided to the right product and receive recommendations on products with similar characteristics.

This project is a requirement for the HarvardX Professional Certificate Data Science Program which aims to predict movies ratings using *"MovieLens 10M Dataset"*.

2. Methodology and Analysis

a) Several data transformations were implemented to create new categories for:

- movie rating decade - transform timestamp into year and after categorizing the ratings in the 1990s and 2000s
- release decade: split the title and retrieve the release year, then categorize into decades from 1910s to 2000s
- rating group: if the star rating was half or whole star
- single genre: remove all the genres combination
- date: timestamp was also transformed into YYYY-MM-DD
- Validations were preformed to verify if the data wrangling was implemented correctly

b) Exploratory Analysis (EDA) were developed to get more insights for the recommendation system and have a better understanding of the data

c) Modeling was used to create the predictions. Models were built from the training data (edx further split into edx_train and edx_test), and then assessed to the test data (validation):

- Random prediction,

- Linear Models: Linear regression using edx, followed by predictions using train_set to create the model for Only mean of the ratings, Adding Movie effects (bi), Adding Movie + User Effect (bu), Adding Movie + User + time Effect (bt) . Both test_set and validation were used to predict and calculate the RMSE
- Matrix factorization (recosystem)

Final metric is the root mean square estimate (RMSE). A lower RMSE means a better, more accurate prediction. According to Perrier (2017) RMSE is defined as the sum of the squares of the difference between the real values and the predicted values. The closer the predictions are to the real values, the lower the RMSE is. A lower RMSE means a better, more accurate prediction. The aim for successful metrics is to reach RMSE Capstone target < 0.8649.

2.1 Data Transformation

During the process of data transformation, a series of validations occurred to ensure retrieving the correct data. For example, after splitting the title to get the release year, it was found that the generated premier_date retrieve values were between 1000 and 9000. It was clear that the split numbers related to the title were retrieved as release year. Re-coding was necessary to fix this issue.

```
#1.Transformed edx
edx_transf<-edx %>%
  mutate(year_rated = year(as_datetime(timestamp)),           # transform timestamp to year
         premier_date=as.numeric(stri_extract(title, regex = "(\\d{4})", comments = TRUE))) # extract d

#2. Validated transformation (split of the title into premier_date)
start_premier_date=min(edx_transf$premier_date)             # movies release start year
end_premier_date=max(edx_transf$premier_date)               # movies release end year

#3.Identified release year issues
premier_year_tofix <- edx_transf %>% group_by(movieId) %>%
  filter (premier_date<1910|premier_date>2009) %>%
    select(movieId, title, premier_date)
unique(premier_year_tofix)
```

```
## # A tibble: 17 x 3
## # Groups:   movieId [17]
##    movieId title                                                premier_date
##      <dbl> <chr>                                                       <dbl>
##  1    6290 House of 1000 Corpses (2003)                                1000
##  2    1422 Murder at 1600 (1997)                                       1600
##  3     671 Mystery Science Theater 3000: The Movie (1996)              3000
##  4    8198 1000 Eyes of Dr. Mabuse, The (Tausend Augen des Dr. Mab~    1000
##  5    2311 2010: The Year We Make Contact (1984)                       2010
##  6    6645 THX 1138 (1971)                                             1138
##  7    2691 Legend of 1900, The (a.k.a. The Legend of the Pianist o~    1900
##  8    5310 Transylvania 6-5000 (1985)                                  5000
##  9    4159 3000 Miles to Graceland (2001)                              3000
## 10    8905 1492: Conquest of Paradise (1992)                           1492
## 11   27266 2046 (2004)                                                2046
## 12    8864 Mr. 3000 (2004)                                            3000
## 13   53953 1408 (2007)                                                1408
## 14    5472 1776 (1972)                                                1776
```

```
## 15    2308 Detroit 9000 (1973)                                          9000
## 16   26359 1900 (Novecento) (1976)                                      1900
## 17    4311 Bloody Angels (1732 HÃ¸tten: Marerittet Har et Postnumm~      1732
```

```
#4. Recoded the 17 release years between 1000 and 9000
edx_transf[edx_transf$movieId == "2308", "premier_date"] <- 1973
edx_transf[edx_transf$movieId == "5310", "premier_date"] <- 1985
edx_transf[edx_transf$movieId == "671", "premier_date"] <- 1996
edx_transf[edx_transf$movieId == "4159", "premier_date"] <- 2001
edx_transf[edx_transf$movieId == "27266", "premier_date"] <- 2004
edx_transf[edx_transf$movieId == "8864", "premier_date"] <- 2004
edx_transf[edx_transf$movieId == "2311", "premier_date"] <- 1984
edx_transf[edx_transf$movieId == "5472", "premier_date"] <- 1972
edx_transf[edx_transf$movieId == "4311", "premier_date"] <- 1998
edx_transf[edx_transf$movieId == "1422", "premier_date"] <- 1997
edx_transf[edx_transf$movieId == "8905", "premier_date"] <- 1992
edx_transf[edx_transf$movieId == "53953", "premier_date"] <- 2007
edx_transf[edx_transf$movieId == "6645", "premier_date"] <- 1971
edx_transf[edx_transf$movieId == "6290", "premier_date"] <- 2003
edx_transf[edx_transf$movieId == "8198", "premier_date"] <- 1960
edx_transf[edx_transf$movieId == "2691", "premier_date"] <- 1998
edx_transf[edx_transf$movieId == "26359", "premier_date"] <- 1976


#5. Implement final transformations
edx <-edx_transf %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "month"),                                    # tr
         movie_rated_era=ifelse (year_rated<=1999, "Ratings during 1990s","Ratings during 2000s"),     # d
         premier_year_era=ifelse (premier_date >=1910 & premier_date <=1919, "1910s",                  # d
                           ifelse(premier_date >=1920 & premier_date <=1929, "1920s",
                            ifelse(premier_date >=1930 & premier_date <=1939, "1930s",
                             ifelse(premier_date >=1940 & premier_date <=1949, "1940s",
                              ifelse(premier_date >=1950 & premier_date <=1959, "1950s",
                               ifelse(premier_date >=1960 & premier_date <=1969, "1960s",
                                ifelse(premier_date >=1970 & premier_date <=1979, "1970s",
                                 ifelse(premier_date >=1980 & premier_date <=1989, "1980s",
                                  ifelse(premier_date >=1990 & premier_date <=1999, "1990s",
                                   ifelse(premier_date >=2000 & premier_date <=2009, "2000s","2010s"))))))))),
         rating_group= ifelse((rating == 1 |rating == 2 | rating == 3 | rating == 4 | rating == 5), "whol
         Orig_genres=genres)  %>%                                                                     # ke
         separate(genres,c("single_genres"),sep = "\\|")                                              # sp
```

```
## Warning: Expected 1 pieces. Additional pieces discarded in 7259486 rows [1, 2,
## 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 16, 17, 18, 19, 20, 21, ...].
```

```
head(edx)
```

```
##   userId movieId rating timestamp                          title single_genres
## 1      1     122      5 838985046              Boomerang (1992)        Comedy
## 2      1     185      5 838983525               Net, The (1995)        Action
## 3      1     292      5 838983421               Outbreak (1995)        Action
## 4      1     316      5 838983392               Stargate (1994)        Action
## 5      1     329      5 838983392 Star Trek: Generations (1994)        Action
```

```
## 6      1     355      5 838984474       Flintstones, The (1994)      Children
##   year_rated premier_date       date      movie_rated_era premier_year_era
## 1       1996         1992 1996-08-01 Ratings during 1990s            1990s
## 2       1996         1995 1996-08-01 Ratings during 1990s            1990s
## 3       1996         1995 1996-08-01 Ratings during 1990s            1990s
## 4       1996         1994 1996-08-01 Ratings during 1990s            1990s
## 5       1996         1994 1996-08-01 Ratings during 1990s            1990s
## 6       1996         1994 1996-08-01 Ratings during 1990s            1990s
##   rating_group                  Orig_genres
## 1   whole star               Comedy|Romance
## 2   whole star         Action|Crime|Thriller
## 3   whole star  Action|Drama|Sci-Fi|Thriller
## 4   whole star       Action|Adventure|Sci-Fi
## 5   whole star Action|Adventure|Drama|Sci-Fi
## 6   whole star       Children|Comedy|Fantasy
```

```
#str(edx)
```

2.2 Exploratory Data Analysis (EDA) & Visualizations

MovieLens is comprised by 2 datasets: *movies.dat* containing Movie ID, title and genres and *ratings.dat* containing User ID, Movie ID, rating, and timestamp.

EDA indicated that edx contains 9,000,055 ratings applied to 106,770 movies from 797 genres combinations. Movies were rated by 69,878 users from 1995 to 2009. Movie's premier took place between 1915 and 2008.

```
##   ratings_n unique_movies unique_genres users_n start_year_rate end_year_rate
## 1   9000055         10677           797   69878            1995          2009
##   start_premier_date end_premier_date
## 1               1915             2008
```

Genres After splitting the genre combinations (such as Action from "Action|Crime|Thriller" or Comedy from "Comedy|Romance") into single genres, 20 unique single genres were found.

```
top_genres <- edx %>% group_by(single_genres) %>%
  summarize(count = n()) %>%
  arrange(desc(count))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
head(top_genres)
```

```
## # A tibble: 6 x 2
##   single_genres   count
##   <chr>           <int>
## 1 Action        2560545
## 2 Comedy        2437260
## 3 Drama         1741668
## 4 Adventure      753650
## 5 Crime          529521
## 6 Horror         233074
```

```
#layout(matrix(c(1,2), nrow =2) , heights = c(1,4))   # create wordcloud based on the counts of the 20
#par(mar=rep(0,4))
#plot.new()
#text(x=0.5,y=0.5, "Top Single Genres by number of ratings")
#wordcloud(words=top_genres$single_genres,freq=top_genres$count,min.freq=50,
         # max.words = 19,random.order=FALSE,random.color=FALSE,
         # rot.per=0.50,colors = brewer.pal(8,"Dark2"),scale=c(5,.2),
         # family="plain",font=2,
         # main = "Top single genres by number of ratings")
```

It was possible to identify that the top 3 single genres were Action, Comedy and Drama. Having Action and Comedy similar ratings. The lowest rated single genres were Romance, War and IMAX.

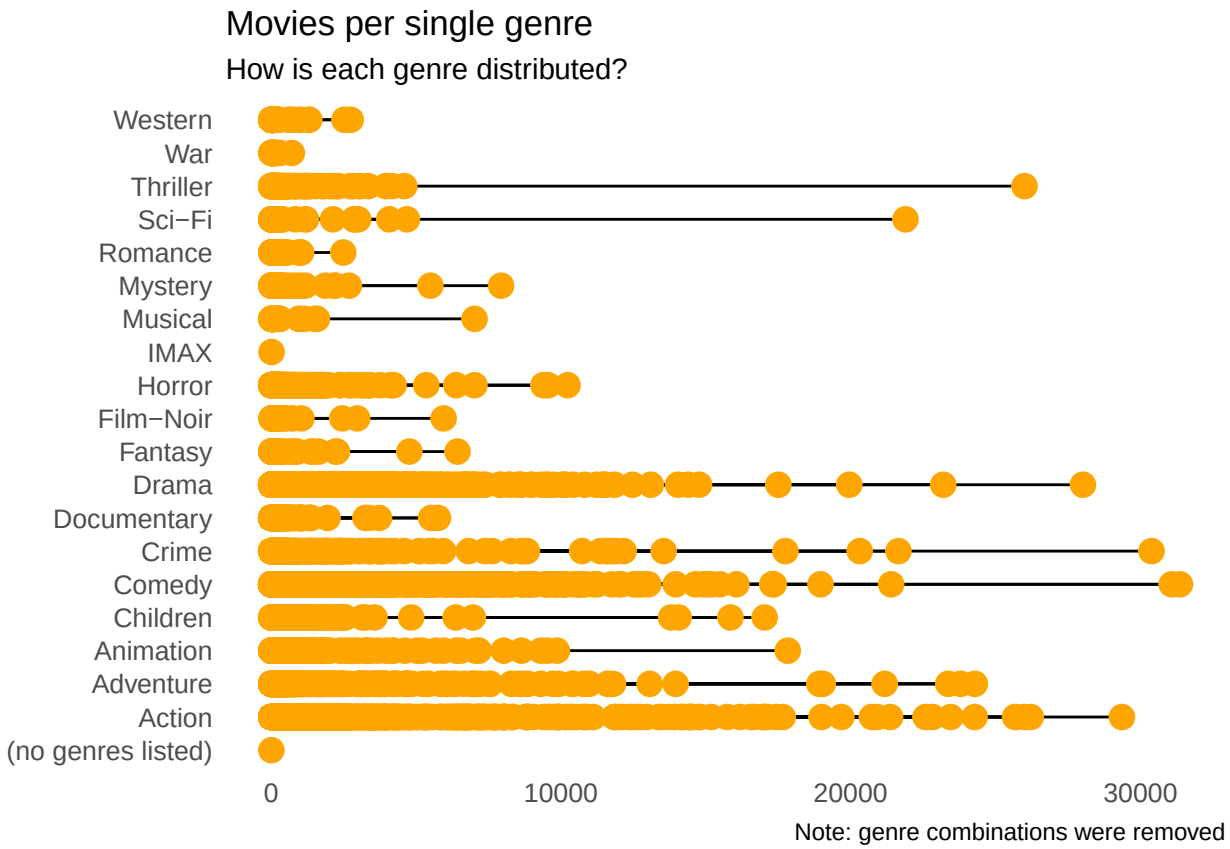
```
single_genres_sum<-edx %>% group_by(single_genres,movieId) %>%
  summarize(n_rating_of_movie = n(),
            mu_movie = mean(rating),
            sd_movie = sd(rating))
```

```
## `summarise()` regrouping output by 'single_genres' (override with `.groups` argument)
```

```
head(single_genres_sum)
```

```
## # A tibble: 6 x 5
## # Groups:   single_genres [2]
##   single_genres      movieId n_rating_of_movie mu_movie sd_movie
##   <chr>                <dbl>             <int>    <dbl>    <dbl>
## 1 (no genres listed)    8606                 7     3.64    1.11
## 2 Action                   6             12346     3.82    0.885
## 3 Action                   9              2278     3.00    0.968
## 4 Action                  10             15187     3.43    0.864
## 5 Action                  15              1654     2.72    1.06
## 6 Action                  20              2212     2.87    0.932
```

```
single_genres_sum %>%
  ggplot( aes(x=single_genres, y=n_rating_of_movie)) +
  geom_segment( aes(xend=single_genres, yend=0)) +
  geom_point( size=4, color="orange") +
  coord_flip() +   ggtitle("Movies per single genre") +
  labs(fill = "number of ratings",
       subtitle="How is each genre distributed?",
       caption ="Note: genre combinations were removed") +
  mytheme + theme(axis.text.x = element_text(size = 10),
                  axis.text.y = element_text(size = 10),
                  legend.position = "none")
```



Release Decade The highest ratings were given to the 1980s and 1990s movies (3 and 4 rating stars). Movies from the 1910s to 1970s also received good ratings, the majority were above 3 stars, having their highest number of ratings with 4 stars.

```
edx %>% mutate(premier_year_era=premier_year_era)%>% group_by (rating,premier_year_era) %>%
  count(premier_year_era) %>%
  ggplot(aes(x = premier_year_era, y = n, fill= n)) +  # Plot with values on top
  geom_bar(stat = "identity") +
  ggtitle("Release Decade") +
  labs(fill = "number of ratings", subtitle="How are movie release decades distributed over the star ra
  mytheme +
  theme(axis.text.y = element_blank(), axis.text.x =element_text(size = rel(0.75),angle = 90), legend.p
  guides(fill = guide_colorbar(title = "number of ratings",                                          # e
                               label.position = "bottom", label.hjust = 1,
                               title.position = "left", title.vjust = 0.75,
                               frame.colour = "black",                                               # d
                               barwidth = 13,
                               barheight = 0.75)) +
  scale_fill_distiller(palette = "Spectral") +
  facet_wrap(~ rating,ncol=3)
```

Release Decade

How are movie release decades distributed over the star ratings?

Note: release decade category derived from the split of title

The analysis of the trend: "average ratings vs. date" shows that there is some evidence of a time effect. The movies from the 1970s and earlier received on average higher star rate score than movies which premier took place after the 1980s.

It can be noticed a decrease on the average star rating after year 2000 for the 1900s and 2000s movies (from an average of about 4 to 3.5 stars), whereas the 1980s movies maintained the 3.5 star rating over the period of time.

```
release_decade<-edx %>%
  mutate(premier_year_era_group=ifelse (premier_date >=1910 & premier_date <=1979, "1970s or earlier",
                                  ifelse(premier_date >=1980 & premier_date <=1989, "1980s",
                                   ifelse(premier_date >=1990 & premier_date <=1999, "1990s",
                                    ifelse(premier_date >=2000 & premier_date <=2009, "2000s","2010s")))
  group_by(date, premier_year_era_group) %>%
  summarize(avgrating = mean(rating))
```

```
## `summarise()` regrouping output by 'date' (override with `.groups` argument)
```

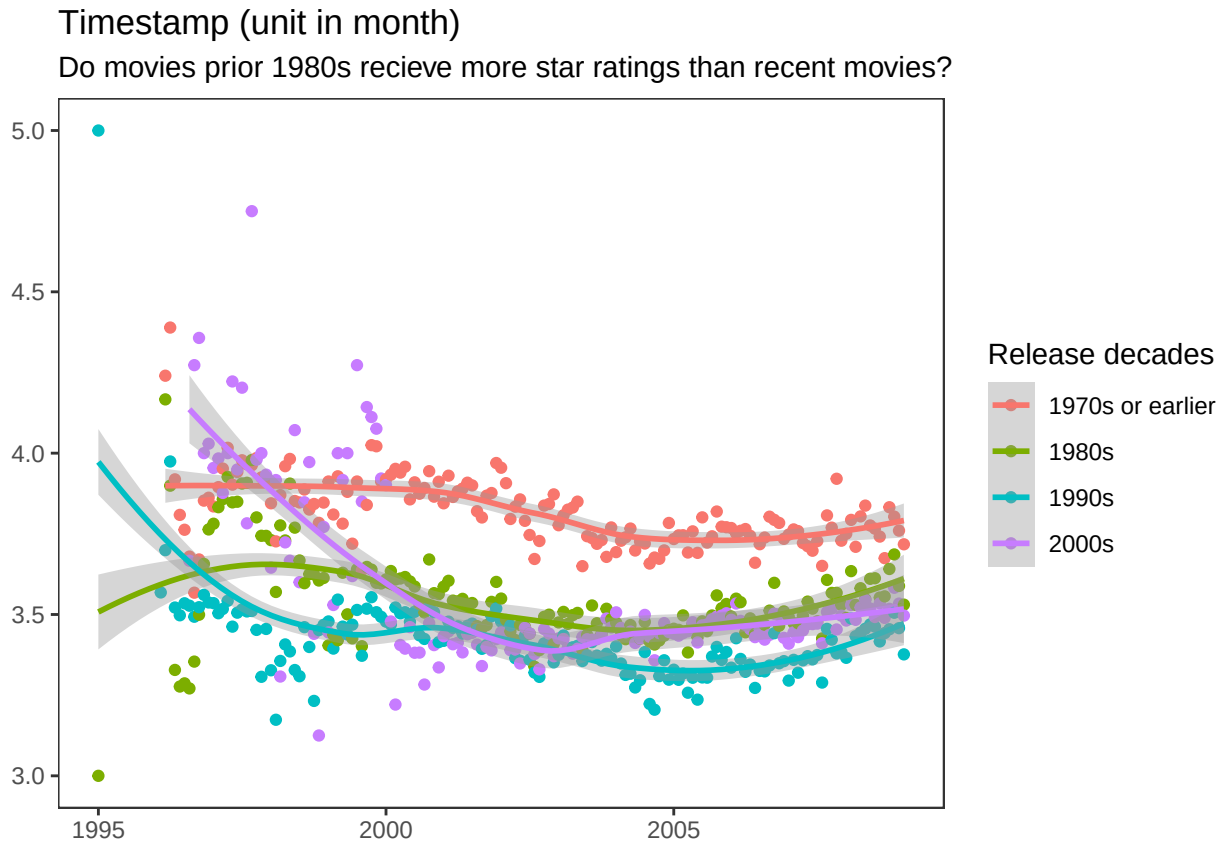
```
head(release_decade)
```

```
## # A tibble: 6 x 3
## # Groups:   date [3]
##   date                premier_year_era_group avgrating
##   <dttm>              <chr>                       <dbl>
## 1 1995-01-01 00:00:00 1980s                        3
## 2 1995-01-01 00:00:00 1990s                        5
```

```
## 3 1996-02-01 00:00:00 1990s                       3.57
## 4 1996-03-01 00:00:00 1970s or earlier            4.24
## 5 1996-03-01 00:00:00 1980s                       4.17
## 6 1996-03-01 00:00:00 1990s                       3.70
```

```
release_decade%>%
  ggplot(aes(date, avgrating, colour = premier_year_era_group)) +
  geom_point() +
  geom_smooth() +
 theme_bw() + theme(panel.grid = element_blank(),axis.title = element_blank()) +
  labs(colour = "Release decades",
       title="Timestamp (unit in month)",
       subtitle="Do movies prior 1980s recieve more star ratings than recent movies?")
```

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



2.3 Modelling

In this step edx was split into train_set (90% of the data) and test_set (10% of the data). Using the original code as template. Dimensions were: edx train_set = 8100048 x 12; edx test_set = 899990 X 12.

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "single_genres", "year_rated",
```

```
##   userId movieId rating timestamp
## 1      1     231      5 838983392
## 2      1     480      5 838983653
## 3      1     586      5 838984068
## 4      2     151      3 868246450
## 5      2     858      2 868245645
## 6      2    1544      3 868245920
##                                                  title
## 1                                  Dumb & Dumber (1994)
## 2                                  Jurassic Park (1993)
## 3                                     Home Alone (1990)
## 4                                        Rob Roy (1995)
## 5                                  Godfather, The (1972)
## 6 Lost World: Jurassic Park, The (Jurassic Park 2) (1997)
##                                  genres       date
## 1                                Comedy 1996-08-01
## 2        Action|Adventure|Sci-Fi|Thriller 1996-08-01
## 3                         Children|Comedy 1996-08-01
## 4               Action|Drama|Romance|War 1997-07-01
## 5                           Crime|Drama 1997-07-01
## 6 Action|Adventure|Horror|Sci-Fi|Thriller 1997-07-01
```

2.3.1 - Random Prediction As the initial insight benchmark, probability distribution of the variables was calculated. This means that any model should be better than the values found. EDA analysis identified that there were more movies being scored with 3- and 4-star ratings (23% and 28% respectively). This represents an overall 51% of the ratings given to just two score stars.

With the random prediction, ratings use the observed probabilities. Initially the probability of each individual rating is calculated on the train_set. Then the test_set is used to predict the rating and compare with the actual rating. The final table results show the probabilities of the edx, train_set, monte carlo using train_set, test_set and RMSE using the test_set.

```
prop_edx_ratings<- (prop.table(table(edx$rating)))         # Proportion of each edx ratings
prop_train_ratings<- (prop.table(table(train_set$rating))) # Proportion of each train_set ratings
prop_test_ratings<- (prop.table(table(test_set$rating)))   # Proportion of each test_set ratings

set.seed(1, sample.kind = "Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
# Create the probability of each rating - seq (min,max,interval)
p <- function(x, y) mean(y == x)
rating <- seq(0.5,5,0.5)

# Estimate the probability of each rating with Monte Carlo simulation
B <- 10000
M <- replicate(B, {
  s <- sample(train_set$rating, 100, replace = TRUE)
  sapply(rating, p, y= s)
})
```

```
prob <- sapply(1:nrow(M), function(x) mean(M[x,]))          # Proportion of each train_set ratings with
prob
```

```
##  [1] 0.009592 0.038171 0.011777 0.078600 0.036984 0.236294 0.088360 0.287167
##  [9] 0.058362 0.154693
```

```
# Predict random ratings
y_hat_random <- sample(rating, size = nrow(test_set),
                       replace = TRUE, prob = prob)

RMSE_rating<- RMSE(test_set$rating, y_hat_random)
RMSE_rating
```

```
## [1] 1.499797
```

```
#calculation of the RMSE for each rating.
#It is the same as sqrt(mean((true_ratings - predicted_ratings)^2))
individual_RMSE_rating <- data.frame (c(RMSE(test_set$rating==0.5, y_hat_random==0.5),   # Proportion o
                                RMSE(test_set$rating==1, y_hat_random==1),
                                RMSE(test_set$rating==1.5, y_hat_random==1.5),
                                RMSE(test_set$rating==2, y_hat_random==2),
                                RMSE(test_set$rating==2.5, y_hat_random==2.5),
                                RMSE(test_set$rating==3, y_hat_random==3),
                                RMSE(test_set$rating==3.5, y_hat_random==3.5),
                                RMSE(test_set$rating==4, y_hat_random==4),
                                RMSE(test_set$rating==4.5, y_hat_random==4.5),
                                RMSE(test_set$rating==5, y_hat_random==5)))
                  # validated by  sqrt(mean(((test_set$rating==5) - (y_hat_random==5))^2))



### create each rating proportion & RMSE final table
rating_prop<-data.frame (rating= c(0.5,1,1.5,2,2.5,3,3.5,4,4.5,5),
                         prop_edx_ratings, prop_train_ratings, prob, prop_test_ratings, individual_RMSE_
select (rating, Freq, Freq.1, prob, Freq.2, c.RMSE.test_set.rating....0.5..y_hat_random....0.5...RMSE.te
names(rating_prop)<-c("rating","prob_edx", "prob_train_set", "prob_MonteCarlo_train_set", "prob_test_se


kable(rating_prop) %>%
  kable_styling(bootstrap_options = "striped" , full_width = F , position = "center") %>%
  kable_styling(bootstrap_options = "bordered", full_width = F , position ="center") %>%
  column_spec(1,bold = T ) %>%
  column_spec(6,bold =T ,color = "white" , background ="#D7261E")
```

rating	prob_edx	prob_train_set	prob_MonteCarlo_train_set	prob_test_set	RMSE test_set
0.5	0.0094859	0.0094877	0.009592	0.0094701	**0.1372839**
1.0	0.0384085	0.0384012	0.038171	0.0384749	**0.2711021**
1.5	0.0118250	0.0118326	0.011777	0.0117568	**0.1523601**
2.0	0.0790464	0.0790855	0.078600	0.0786942	**0.3808550**
2.5	0.0370009	0.0369727	0.036984	0.0372549	**0.2671822**
3.0	0.2356919	0.2356793	0.236294	0.2358048	**0.6009064**
3.5	0.0879577	0.0879240	0.088360	0.0882610	**0.4011090**
4.0	0.2876016	0.2876351	0.287167	0.2872999	**0.6399185**
4.5	0.0585259	0.0585256	0.058362	0.0585284	**0.3319356**
5.0	0.1544562	0.1544563	0.154693	0.1544550	**0.5112290**

Results of the RMSE test_set for individual star rating using the Random prediction model were below the RMSE target.

```
### create RMSE for rating
rating_RMSE_1 <- data.frame (Method = c("RMSE Target", " ", "Random prediction"), RMSE = c(0.8649," ",RI
#rating_RMSE_1

kable(rating_RMSE_1) %>%
  kable_styling(bootstrap_options = "striped" , full_width = F , position = "center") %>%
  kable_styling(bootstrap_options = "bordered", full_width = F , position ="center") %>%
  row_spec(1,bold = T , color = "blue" ) %>%
  column_spec(2,bold =T) %>%
  row_spec(2,bold =T ,color = "white" , background ="grey") %>%
 row_spec(3,bold =T ,color = "white" , background ="#D7261E")
```

Method	RMSE
RMSE Target	**0.8649**
Random prediction	**1.49979655801755**

The RMSE of random prediction for the overall rating was very high (1.499), above the RMSE Capstone Target of 0.8649. RMSE of each individual rating met the RMSE Capstone Target.

2.3.2 Linear Models P-value of the F-statistic is < 2.2e-16, which is highly significant y=mx+b. The only predictor that was significantly related to the rating was single_genresIMAX. All the other genres had $Pr(>|t|) > 0.05$. This means that changes in single_genresIMAX ratings are significantly associated to changes in ratings, while changes in all remaining individual single genres seem not to be significantly associated with ratings.

single_genresIMAX coefficient suggests that for every 1 rating increase, it can be expected a decrease of -1.3214286 *1 = 1.32 star units, on average. Meaning the maximum score would be 5 stars - 1.32 = 3.68 stars (since the slope is negative, the decrease ratio is 1.32 per unit)

R-squared represents the correlation coefficient. Strong correlation means that R-squared is close to 1, positively or negatively. In the rating ~ single_genres linear regression model, the multiple R-squared was 0.0221, demonstrating that there was no correlation. Only 2.21% of the variance in the measure of rating

can be predicted by single genres. Residual standard error (RSE) is the measure of error of prediction. The lower the RSE is, the more accurate is the model.

rating ~ single_genres

```
set.seed(1, sample.kind = "Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
fit_genres<-lm(rating ~ single_genres, data = edx)
#fit_genres
summary(fit_genres)
```

```
##
## Call:
## lm(formula = rating ~ single_genres, data = edx)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.6459 -0.5492  0.1288  0.5786  1.9665
##
## Coefficients:
##                            Estimate Std. Error t value Pr(>|t|)
## (Intercept)               3.6428572  0.3963158   9.192  < 2e-16 ***
## single_genresAction      -0.2214526  0.3963163  -0.559  0.57631
## single_genresAdventure   -0.0782987  0.3963176  -0.198  0.84338
## single_genresAnimation   -0.0936693  0.3963221  -0.236  0.81316
## single_genresChildren    -0.3953004  0.3963234  -0.997  0.31856
## single_genresComedy      -0.1890340  0.3963164  -0.477  0.63338
## single_genresCrime        0.2282971  0.3963184   0.576  0.56458
## single_genresDocumentary  0.1443992  0.3963329   0.364  0.71561
## single_genresDrama        0.0236177  0.3963166   0.060  0.95248
## single_genresFantasy     -0.3328879  0.3963690  -0.840  0.40100
## single_genresFilm-Noir    0.5030855  0.3964035   1.269  0.20440
## single_genresHorror      -0.6093571  0.3963217  -1.538  0.12416
## single_genresIMAX        -1.3214286  0.4853857  -2.722  0.00648 **
## single_genresMusical     -0.0008257  0.3964011  -0.002  0.99834
## single_genresMystery      0.0579222  0.3963612   0.146  0.88381
## single_genresRomance     -0.3624834  0.3964247  -0.914  0.36052
## single_genresSci-Fi      -0.2092698  0.3963434  -0.528  0.59750
## single_genresThriller    -0.1128840  0.3963304  -0.285  0.77578
## single_genresWar          0.0295715  0.3969148   0.075  0.94061
## single_genresWestern     -0.1069422  0.3964064  -0.270  0.78733
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.049 on 9000035 degrees of freedom
## Multiple R-squared:  0.0221, Adjusted R-squared:  0.02209
## F-statistic: 1.07e+04 on 19 and 9000035 DF,  p-value: < 2.2e-16
```

```r
#confidence intervals
confint(fit_genres)
```

```
##                                2.5 %     97.5 %
## (Intercept)                2.8660924  4.4196220
## single_genresAction       -0.9982184  0.5553133
## single_genresAdventure    -0.8550671  0.6984697
## single_genresAnimation    -0.8704466  0.6831079
## single_genresChildren     -1.1720802  0.3814794
## single_genresComedy       -0.9657999  0.5877318
## single_genresCrime        -0.5484728  1.0050670
## single_genresDocumentary -0.6323992  0.9211975
## single_genresDrama        -0.7531486  0.8003840
## single_genresFantasy      -1.1097569  0.4439811
## single_genresFilm-Noir    -0.2738512  1.2800222
## single_genresHorror       -1.3861335  0.1674193
## single_genresIMAX         -2.2727673 -0.3700900
## single_genresMusical      -0.7777576  0.7761062
## single_genresMystery      -0.7189316  0.8347760
## single_genresRomance      -1.1394616  0.4144949
## single_genresSci-Fi       -0.9860887  0.5675490
## single_genresThriller     -0.8896775  0.6639095
## single_genresWar          -0.7483673  0.8075103
## single_genresWestern      -0.8838846  0.6700003
```

rating ~ movie_rated_era + premier_year_era There is no stronger correlation between Release decades (1980s, 1900s and 2000s) and related average ratings, although p-value was significant. P-value of the F-statistic is < 2.2e-16, which is also highly significant. The only predictors that were not significantly related to the rating were release decades 1980s, 1990s and 2000s. All the other decades prior 1980s had highly significant association with ratings.

The coefficients suggesting a decrease in units were for the release decade 1990s and movies rated during 2000s (with-0.0587 and -0.1184530 star units respectively, on average). As a result, the maximum score would be 4.95 and 4.89 stars respectively. In the rating ~ movie_rated_era + premier_year_era Multiple linear regression (MLR), the multiple R-squared was 0.01827, demonstrating that there was also no correlation. Only 1.8% of the variance in the measurement of rating can be predicted by release decade and rating during 2000.

```r
set.seed(1, sample.kind = "Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```r
fit_decade<-lm(rating ~ movie_rated_era + premier_year_era, data = edx)
#fit_decade
summary(fit_decade)
```

```
##
## Call:
## lm(formula = rating ~ movie_rated_era + premier_year_era, data = edx)
##
## Residuals:
```

```
##     Min      1Q  Median      3Q     Max
## -3.4194 -0.5098  0.1087  0.6087  1.6087
##
## Coefficients:
##                                    Estimate Std. Error  t value Pr(>|t|)
## (Intercept)                       3.5684925  0.0457729   77.961  < 2e-16 ***
## movie_rated_eraRatings during 2000s -0.1184530  0.0008601 -137.721  < 2e-16 ***
## premier_year_era1920s              0.4212917  0.0467081    9.020  < 2e-16 ***
## premier_year_era1930s              0.4019297  0.0459054    8.756  < 2e-16 ***
## premier_year_era1940s              0.4694075  0.0458557   10.237  < 2e-16 ***
## premier_year_era1950s              0.4092166  0.0458212    8.931  < 2e-16 ***
## premier_year_era1960s              0.3425602  0.0458048    7.479 7.51e-14 ***
## premier_year_era1970s              0.2983833  0.0457878    6.517 7.19e-11 ***
## premier_year_era1980s              0.0601428  0.0457735    1.314    0.189
## premier_year_era1990s             -0.0587000  0.0457688   -1.283    0.200
## premier_year_era2000s              0.0096627  0.0457720    0.211    0.833
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.051 on 9000044 degrees of freedom
## Multiple R-squared:  0.01827,    Adjusted R-squared:  0.01826
## F-statistic: 1.674e+04 on 10 and 9000044 DF,  p-value: < 2.2e-16
```

```
#confidence intervals
confint(fit_decade)
```

```
##                                        2.5 %      97.5 %
## (Intercept)                       3.47877931  3.65820575
## movie_rated_eraRatings during 2000s -0.12013876 -0.11676725
## premier_year_era1920s              0.32974555  0.51283782
## premier_year_era1930s              0.31195677  0.49190257
## premier_year_era1940s              0.37953198  0.55928304
## premier_year_era1950s              0.31940880  0.49902446
## premier_year_era1960s              0.25278448  0.43233595
## premier_year_era1970s              0.20864092  0.38812563
## premier_year_era1980s             -0.02957157  0.14985711
## premier_year_era1990s             -0.14840518  0.03100517
## premier_year_era2000s             -0.08004882  0.09937413
```

a) Prediction 1 - *Only mean of the ratings* The initial prediction is test that all users will give the same rating to all movies and each rating is randomly distributed. formula is **y_hat=mu+error**

```
mu <- mean(train_set$rating)             # Mean of observed values
RMSE_mu <- RMSE(test_set$rating, mu)
RMSE_mu
```

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

```
v_RMSE_mu <- RMSE(validation$rating, mu)
v_RMSE_mu
```

```
## [1] 1.061202
```

b) Prediction 2 - *Adding Movie Effect (bi)*

Movie variability may be related to the fact that different movies will have different rating distribution. Some maybe bias by the producer, cast, topic this movie bias is expressed as bi in this formula: *y_hat=mu+bi+error* (mean of the difference between the observed rating y and the mean mu).

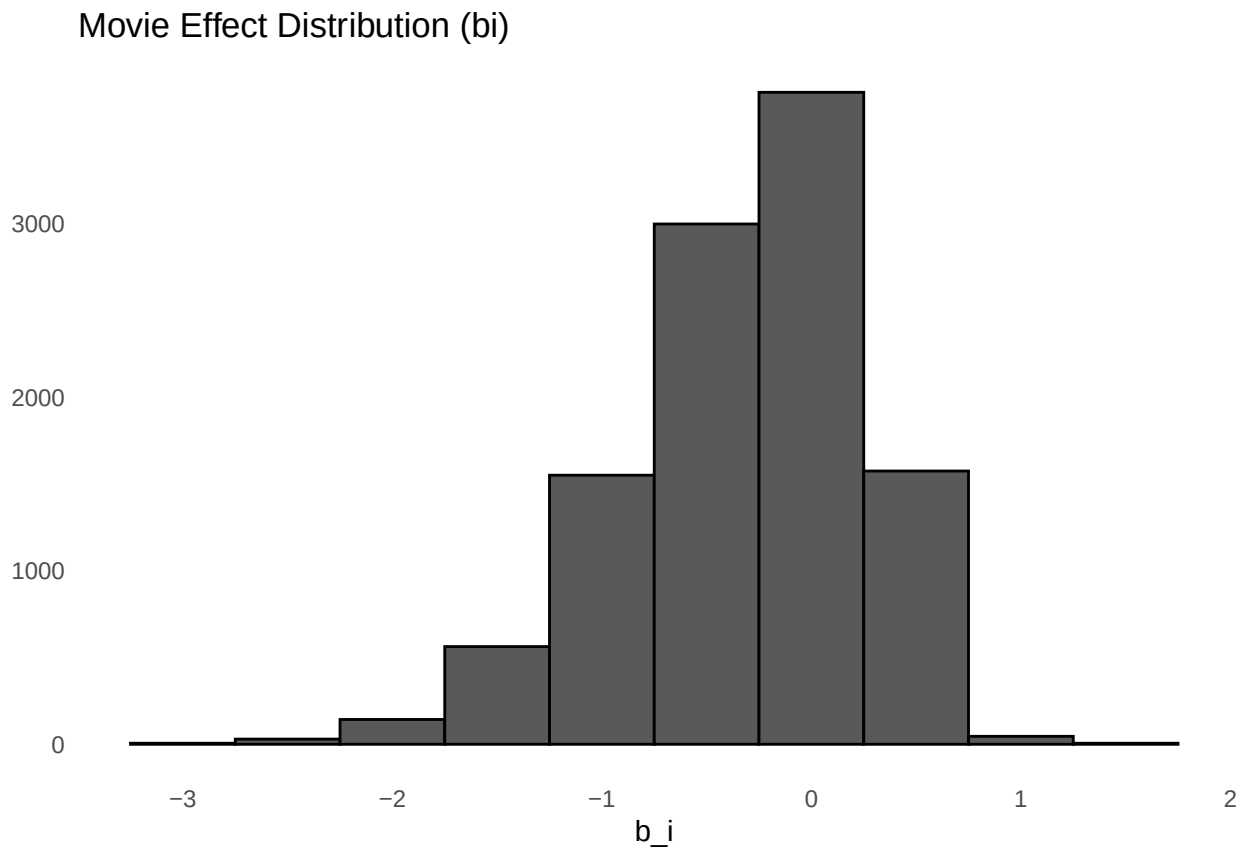
```
# Movie effects (bi)
bi <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
head(bi)
```

```
## # A tibble: 6 x 2
##   movieId    b_i
##     <dbl>  <dbl>
## 1       1  0.415
## 2       2 -0.306
## 3       3 -0.361
## 4       4 -0.637
## 5       5 -0.442
## 6       6  0.302
```

```
#bi distribution
bi %>% ggplot(aes(x = b_i)) +
  geom_histogram(bins=10, col = I("black")) +
  ggtitle("Movie Effect Distribution (bi) ") +
  theme_minimal()+ theme(panel.grid = element_blank(),axis.title.y = element_blank())
```




```
# Predict the rating with mean + bi with test_set
y_hat_bi <- mu + test_set %>%
  left_join(bi, by = "movieId") %>%
  .$b_i
RMSE_bi<- RMSE(test_set$rating, y_hat_bi)
RMSE_bi
```

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

```
# Predict the rating with mean + bi with validation
v_y_hat_bi <- mu + validation %>%
  left_join(bi, by = "movieId") %>%
  .$b_i
v_RMSE_bi<- RMSE(validation$rating, v_y_hat_bi)
v_RMSE_bi
```

```
## [1] 0.9439729
```

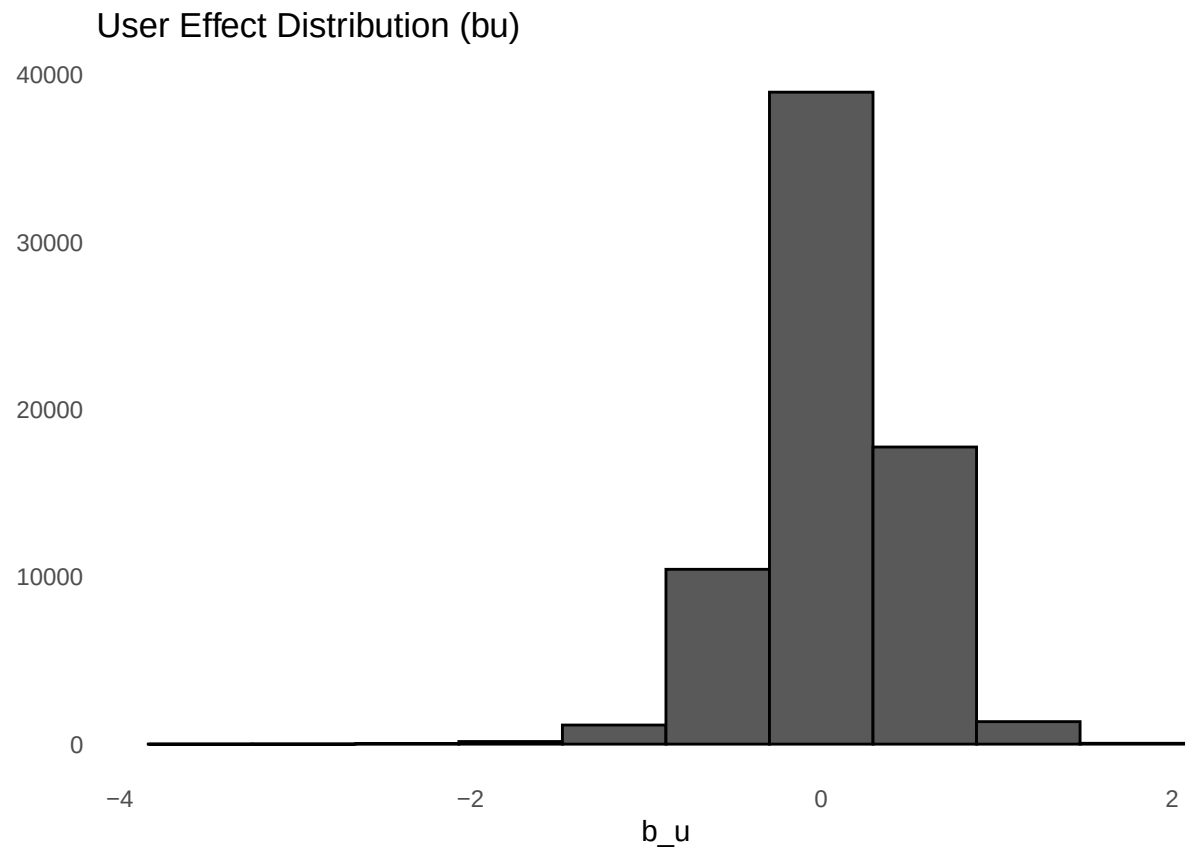
c) Prediction 3 - Adding *User Effect (bu)*

Different users have different rating distribution, this can also be biased by their own movie preference, physiological effect when they were watching and rating the movies. Users can also be influenced by the movie rating itself. The formula is *y_hat=mu+bi+bu+error*

```
# User effect (bu)
bu <- train_set %>%
  left_join(bi, by = 'movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
#User effect distribution shows to be more normally distributed
bu %>% ggplot(aes(x = b_u)) +
  geom_histogram(bins=10, col = I("black")) +
  ggtitle("User Effect Distribution (bu)") +
  theme_minimal()+ theme(panel.grid = element_blank(),axis.title.y = element_blank())
```




```
# Predict the rating with mean + bi+ bu with test_set
y_hat_bi_bu <- test_set %>%
  left_join(bi, by='movieId') %>%
  left_join(bu, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred
RMSE_bi_bu <- RMSE(test_set$rating, y_hat_bi_bu)
RMSE_bi_bu
```

```
## [1] 0.8646843
```

```
# Predict the rating with mean + bi+ bu with validation
v_y_hat_bi_bu <- validation %>%
  left_join(bi, by='movieId') %>%
  left_join(bu, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred
v_RMSE_bi_bu <- RMSE(validation$rating, v_y_hat_bi_bu)
v_RMSE_bi_bu
```

```
## [1] 0.8658528
```

d) Prediction 4 - Adding *Time Effect (t), the year of the rating*

Time can also be biased by the season. For example during valentine's day and fall season people may watch more romance movies, whereas during Halloween more horror movies. Also people can be influenced by critical events that occurred during a given period of time. The formula is *y_hat=mu+bi+bu+error*

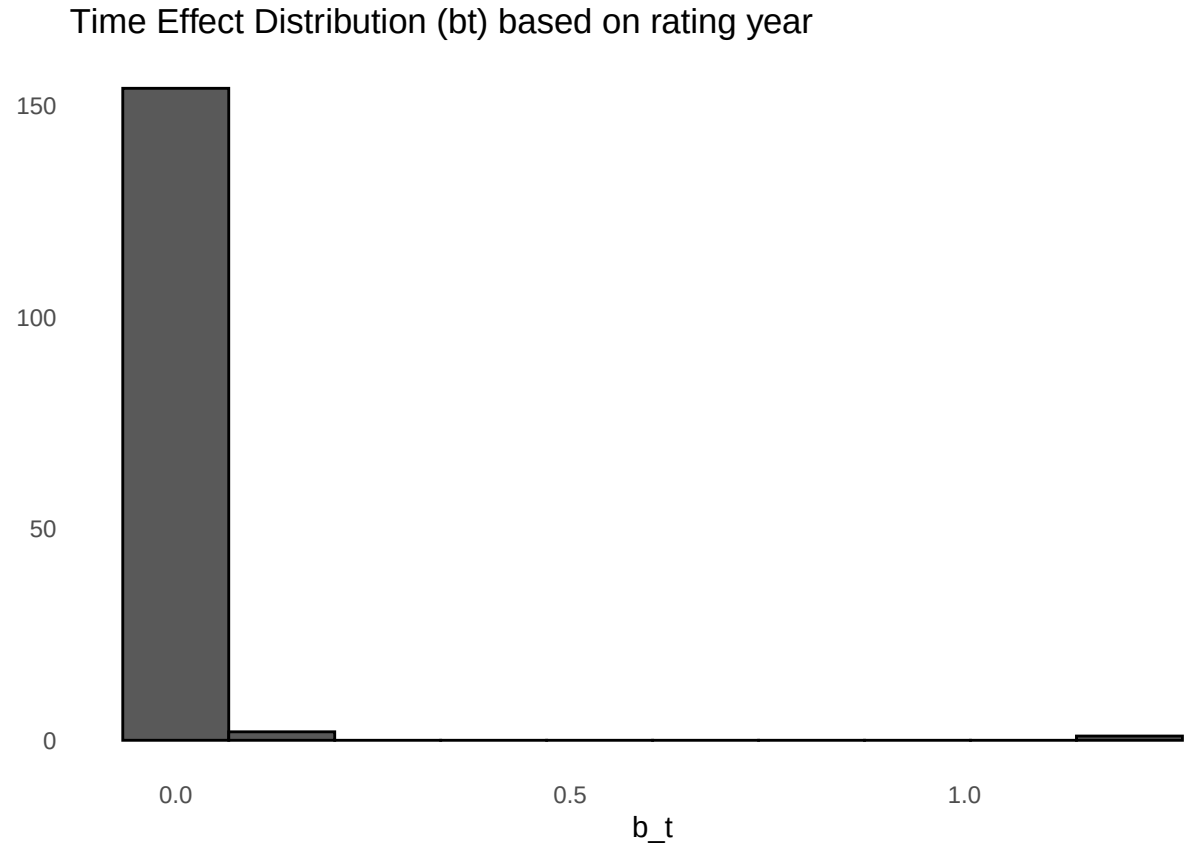
```
# Time effect (t)
t <- train_set %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "month")) %>%               #Timestamp in wee
  left_join(bi, by = 'movieId') %>%
  left_join(bu, by = 'userId') %>%
  group_by(date) %>%
  summarize(b_t = mean(rating - mu - b_i - b_u))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
t
```

```
## # A tibble: 157 x 2
##    date                  b_t
##    <dttm>              <dbl>
##  1 1995-01-01 00:00:00  1.15
##  2 1996-02-01 00:00:00  0.127
##  3 1996-03-01 00:00:00  0.0826
##  4 1996-04-01 00:00:00  0.0480
##  5 1996-05-01 00:00:00  0.000482
##  6 1996-06-01 00:00:00 -0.00524
##  7 1996-07-01 00:00:00 -0.00196
##  8 1996-08-01 00:00:00 -0.00250
##  9 1996-09-01 00:00:00  0.000370
## 10 1996-10-01 00:00:00 -0.00109
## # ... with 147 more rows
```

```
#User effect distribution shows to be more normally distributed
t %>% ggplot(aes(x = b_t)) +
  geom_histogram(bins=10, col = I("black")) +
  ggtitle("Time Effect Distribution (bt) based on rating year") +
  theme_minimal()+ theme(panel.grid = element_blank(),axis.title.y = element_blank())
```




```
# Predict the rating with mean + bi+ bu = bt with test_set
y_hat_bi_bu_bt <- test_set %>%
  left_join(bi, by='movieId') %>%
  left_join(bu, by='userId') %>%
  left_join(t, by='date') %>%
  mutate(pred = mu + b_i + b_u + b_t) %>%
  .$pred

RMSE_bi_bu_bt <- RMSE(test_set$rating, y_hat_bi_bu_bt)
RMSE_bi_bu_bt
```

```
## [1] 0.8646637
```

```
# Predict the rating with mean + bi+ bu = bt with validation
v_y_hat_bi_bu_bt <- validation %>%
  left_join(bi, by='movieId') %>%
  left_join(bu, by='userId') %>%
  left_join(t, by='date') %>%
  mutate(pred = mu + b_i + b_u +b_t) %>%
  .$pred

v_RMSE_bi_bu_bt <- RMSE(validation$rating, v_y_hat_bi_bu_bt)
v_RMSE_bi_bu_bt
```

```
## [1] 0.8658189
```

RMSE comparison table

```
### create RMSE comparison table
rating_RMSE2 <- data.frame (Method = c("RMSE Target"," ", "Random prediction", "Only mean of the ratings
#rating_RMSE2

kable(rating_RMSE2) %>%
  kable_styling(bootstrap_options = "striped" , full_width = F , position = "center") %>%
  kable_styling(bootstrap_options = "bordered", full_width = F , position ="center") %>%
  column_spec(1,bold = T ) %>%
  row_spec(1,bold =T ,color = "blue") %>%
  row_spec(2,bold =T ,color = "white" , background ="gray") %>%
  row_spec(7,bold =T ,color = "white" , background ="#D7261E")
```

Method	RMSE_test_set	RMSE_validation
RMSE Target	**0.8649**	**0.8649**
Random prediction	1.49979655801755	
Only mean of the ratings	1.06005370222409	1.0612018064374
Adding Movie effects (bi)	0.942961498004501	0.943972915845845
Adding Movie + User Effect (bu)	0.86468429490229	0.865852823367063
Adding Movie + User + time Effect (bt)	**0.864663712815486**	**0.865818869174341**

The time effect seems not make much difference at the RMSE validation results. It went from RMSE of 0.86585(adding movie and user effect) to RMSE of 0.86581 (adding movie + user + time effect).

Note: code will take about 40min to run the matrix factorization - please be patience

2.3.3 Matrix factorization Trying to achieve the RMSE Target < 0.8649 and tired to fight the memory limit issues due to the large dataset. I decide to try the recosystem package to create matrix factorization. According to the package documentation https://cran.r-project.org/web/packages/recosystem/recosystem.pdf the default parameters are: - costp_l2 Tuning parameter, the L2 regularization cost for user factors. Can be specified as a numeric vector, with default value c(0.01,0.1). - costq_l2 Tuning parameter, the L2 regularization cost for item factors. Can be specified as a numeric vector, with default value c(0.01,0.1). - lrate Tuning parameter, the learning rate, which can be thought of as the step size in gradient descent. Can be specified as a numeric vector, with default value c(0.01,0.1). - niter Integer, the number of iterations. Default is 20. - nthread Integer, the number of threads for parallel computing. Default is 1.

```
set.seed(123, sample.kind = "Rounding") # This is a randomized algorithm
```

```
## Warning in set.seed(123, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
# Convert the train and test sets into recosystem input format
train_data <-  with(train_set, data_memory(user_index = userId,
                                           item_index = movieId,
```

```
                                          rating     = rating,
                                          date     = date))
test_data  <-  with(test_set,  data_memory(user_index = userId,
                                          item_index = movieId,
                                          rating     = rating,
                                          date     = date))


validation_data  <-  with(validation,  data_memory(user_index = userId,
                                          item_index = movieId,
                                          rating     = rating,
                                          date     = date))

# Create the model object
r <-  recosystem::Reco()

# Select the best tuning parameters. I used the parameters that people has been using wirh Reco() examp
opts <- r$tune(train_data,
               opts = list(dim = c(10, 20, 30),          # dim is number of factors
                      lrate = c(0.1, 0.2),          # learning rate
                      costp_l2 = c(0.01, 0.1),      #regularization for P factors
                      costq_l2 = c(0.01, 0.1),      # regularization for  Q factors
                      nthread  = 4, niter = 10))    #convergence can be controlled by a number of itera

# Train the algorithm
r$train(train_data, opts = c(opts$min, nthread = 4, niter = 20))
```

```
## iter      tr_rmse          obj
##    0       0.9823   1.1049e+07
##    1       0.8759   8.9946e+06
##    2       0.8425   8.3422e+06
##    3       0.8208   7.9614e+06
##    4       0.8047   7.6972e+06
##    5       0.7924   7.5049e+06
##    6       0.7822   7.3624e+06
##    7       0.7735   7.2444e+06
##    8       0.7660   7.1449e+06
##    9       0.7596   7.0689e+06
##   10       0.7539   6.9989e+06
##   11       0.7488   6.9423e+06
##   12       0.7444   6.8953e+06
##   13       0.7403   6.8527e+06
##   14       0.7365   6.8112e+06
##   15       0.7330   6.7766e+06
##   16       0.7298   6.7455e+06
##   17       0.7269   6.7171e+06
##   18       0.7241   6.6910e+06
##   19       0.7216   6.6685e+06
```

```
# Calculate the predicted values using Reco test_data
y_hat_reco <-  r$predict(test_data, out_memory())  #out_memory(): Result should be returned as R object
head(y_hat_reco, 10)
```

```
##  [1] 4.914309 3.720995 3.287802 2.933612 3.519998 4.052493 3.595678 3.378217
##  [9] 4.168875 3.023228
```

```
RMSE_reco <- RMSE(test_set$rating, y_hat_reco)
RMSE_reco
```

```
## [1] 0.7854968
```

```
# Calculate the predicted values using Reco validation_data
v_y_hat_reco <-  r$predict(validation_data, out_memory()) #out_memory(): Result should be returned as R
head(v_y_hat_reco, 10)
```

```
##  [1] 4.365419 5.070285 4.766107 3.102863 3.736580 2.622366 4.114036 4.382323
##  [9] 4.368355 3.301037
```

```
v_RMSE_reco <- RMSE(validation$rating, v_y_hat_reco)
v_RMSE_reco
```

```
## [1] 0.7861042
```

3. Results This the final RMSE comparison table. It contains all the models used to predict ratings. As can been seen from the comparison table, the target RMSE of 0.8649 was almost reached after the Regularization qith Movie, user and time effect (RMSE of 0.8658). The best results of the RMSE were achieved when matrix factorization was implemented **0.78581** (around 9% below the RMSE target of 0.8649).

```
### create RMSE comparison table

rating_RMSE3 <- data.frame (Method = c("RMSE Target"," ", "Random prediction",
                            "Only mean of the ratings", "Adding Movie effects (bi)",
                            "Adding Movie + User Effect (bu)",
                            "Adding Movie + User + time Effect (bt)",
                            "Matrix Factorization (recosystem)"),
                           RMSE_test_set= c(0.8649," ", RMSE_rating,RMSE_mu,RMSE_bi,
                                            RMSE_bi_bu, RMSE_bi_bu_bt, RMSE_reco),      RMSE_validation=
#rating_RMSE3

kable(rating_RMSE3) %>%
  kable_styling(bootstrap_options = "striped" , full_width = F , position = "center") %>%
  kable_styling(bootstrap_options = "bordered", full_width = F , position ="center") %>%
  column_spec(1,bold = T ) %>%
  row_spec(1,bold =T ,color = "blue") %>%
  row_spec(2,bold =T ,color = "white" , background ="gray") %>%
  row_spec(8,bold =T ,color = "white" , background ="#D7261E")
```

Method	RMSE_test_set	RMSE_validation
RMSE Target	**0.8649**	**0.8649**
Random prediction	1.49979655801755	
Only mean of the ratings	1.06005370222409	1.0612018064374
Adding Movie effects (bi)	0.942961498004501	0.943972915845845
Adding Movie + User Effect (bu)	0.86468429490229	0.865852823367063
Adding Movie + User + time Effect (bt)	0.864663712815486	0.865818869174341
Matrix Factorization (recosystem)	**0.785496804746415**	**0.786104155704874**

4. Conclusion This project was a great wrap-up of the certification, it gave us freedom to build our own analysis while we all had in mind a common target. It showcased the importance of validate the data transformation as well gave me different perspectives of data insights.

After running my recommender system algorithms using random prediction, Linear model with regularized effects on movies +users + time effect I reached the *RMSE of* **0.86585** *using the validation* and **0.8646** *using the test_set* which was below the Capstone RMSE Target of *0.8649*. Since I was not fully satisfied with my RMSE results, I decided to try Matrix Factorization method (recosystem), as some of the discussion highly recommended. I was very impressed to reach RMSE of **0.78581** *using the validation* (9% less than the Capstone RMSE Target).

Limitations were mostly related with the dataset size and the memory limit issues that I encounter over several chunks of codes. After some research, I used gc to clean unused memory and I was increased the memory limit. Errors disappears, but the time to run the codes were in between 1h40min and 2h30min. This means that I spent probably 60% of my time waiting for the code to run, instead of been developing/improving my codes.

As future work, I would be curious to see how smaller datasets and different combinations of train_set and test_set would impact the RMSE results.

5.Reference Perrier, A. Effective Amazon Machine Learning . Published by Packt Publishing, 2017.

Forte, R and Miller, J. Mastering Predictive Analytics with R - Second Edition. Published by Packt Publishing, 2017.

Qiu, Y. recosystem: Recommender System Using Parallel Matrix Factorization https://cran.r-project.org/web/packages/recosystem/vignettes/introduction.html, 2020.