# HarvardX - Data Science Professional Certificate - Capstone - Choose Your Own Project

Rachel L SH

4/2/2021

## ABSTRACT

This project is the final assignment for the completion of the Professional Certificate in Data Science from Harvard via the EDX online learning platform. It is a "Choose Your Own" Project with the freedom to pick a topic and incorporate lessons from the certification programme into developing machine learning models in R.

For my project, I have decided to create a recommendation engine of video games for Steam users to play. The *Steam Store Games (Clean Dataset)* and *Steam Video Games ("Recommend video games from 200k interactions user interactions.")* data files used for the project were made available on Kaggle by Nik Davis and Tamber respectively.

The idea is to recommend video games a particular Steam user would likely play based on predicted hours played from a machine learning algorithm. Several techniques were employed over the course of this project and the best performing one was the Parallel Matrix Factorization from Recosystem package.

## INTRODUCTION

As a final step on the Capstone module from the Data Science Professional Certificate from HarvardX, this "Choose Your Own" project is with the freedom to pick a topic and incorporate lessons from the certification programme into developing machine learning models in R. I have decided to work on a recommendation engine for video games on Steam.

According to Wikipedia, Steam is a video game digital distribution service by Valve. ['https://en.wikipedia.org/wiki/Steam_(service)']. For users to play video games from Steam, digital copies of these games have to be purchased beforehand. ['https://store.steampowered.com/about/'].

The quality of the predictions were scored against the true grades in terms of root mean squared error (RMSE).

For the scope of this project, we will gather, explore, visualize, analyze and make predictions over the data from the *MovieLens* data set with 10,000,000 ratings provided by GroupLens, a research lab in the Department of Computer Science and Engineering at the University of Minnesota, United States.

Recommendations can be done using the users own past ratings, but also using *collaborative filtering* techniques to filter out movies that the user might like based on ratings from e.g. from similar users.

## LOAD THE DATA

## Setup R environment, load and install packages

During this analysis we will use the following libraries. The code below checks if these are installed, if not, installs the necessary packages.

```r
# Note: this process could take a couple of minutes

if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")
if(!require(recommenderlab)) install.packages("recommenderlab", repos = "http://cran.us.r-project.org")

library(tidyverse)
library(caret)
library(data.table)
library(recommenderlab)
```

## Download source files

The 2 datasets utilised are available from the following URLs: * Steam Store Games (Clean dataset), gathered around May 2019: - For full folder: https://www.kaggle.com/nikdavis/steam-store-games - For specific "steam.csv" file that would be used for this project: https://www.kaggle.com/nikdavis/steam-store-games?select=steam.csv * Steam Video Games dataset ("Recommend video games from 200k interactions user interactions.", gathered around 2017): https://www.kaggle.com/tamber/steam-video-games/download

For ease of access to the two data files "steam.csv" and "steam-200k.csv", we have attached them in the same folder as this report.

The "steam.csv" file is essentially a catalogue of game titles on Steam and has columns divided by "," and each line will be split into the following 18 columns: "appid", "name", "release_date", "english", "developer", "publisher", "platforms", "required_age", "categories", "genres", "steamspy_tags", "achievements", "positive_ratings", "negative_ratings", "average_playtime", "median_playtime", "owners", and "price".

```r
# `catalogue` dataset derived from `steam.csv` from the Steam Store Games (Clean dataset) folder
catalogue <- fread("steam.csv",
                  col.names = c("appid", "name", "release_date", "english",
                                "developer", "publisher",    "platforms",
                                "required_age",  "categories",   "genres",
                                "steamspy_tags", "achievements", "positive_ratings",
                                "negative_ratings",  "average_playtime",
                                "median_playtime", "owners", "price"))

# Show the first 5 observations:
head(catalogue, 5)
```

```
##    appid                       name release_date english        developer
## 1:    10             Counter-Strike   2000-11-01       1            Valve
## 2:    20      Team Fortress Classic   1999-04-01       1            Valve
## 3:    30              Day of Defeat   2003-05-01       1            Valve
## 4:    40         Deathmatch Classic   2001-06-01       1            Valve
## 5:    50 Half-Life: Opposing Force   1999-11-01       1 Gearbox Software
##    publisher         platforms required_age
## 1:     Valve windows;mac;linux            0
```

```
## 2:     Valve windows;mac;linux           0
## 3:     Valve windows;mac;linux           0
## 4:     Valve windows;mac;linux           0
## 5:     Valve windows;mac;linux           0
##                                                                    categories
## 1: Multi-player;Online Multi-Player;Local Multi-Player;Valve Anti-Cheat enabled
## 2: Multi-player;Online Multi-Player;Local Multi-Player;Valve Anti-Cheat enabled
## 3:                                      Multi-player;Valve Anti-Cheat enabled
## 4: Multi-player;Online Multi-Player;Local Multi-Player;Valve Anti-Cheat enabled
## 5:                         Single-player;Multi-player;Valve Anti-Cheat enabled
##    genres                 steamspy_tags achievements positive_ratings
## 1: Action       Action;FPS;Multiplayer            0           124534
## 2: Action       Action;FPS;Multiplayer            0             3318
## 3: Action FPS;World War II;Multiplayer            0             3416
## 4: Action       Action;FPS;Multiplayer            0             1273
## 5: Action            FPS;Action;Sci-fi            0             5250
##    negative_ratings average_playtime median_playtime            owners price
## 1:             3339            17612             317 10000000-20000000  7.19
## 2:              633              277              62  5000000-10000000  3.99
## 3:              398              187              34  5000000-10000000  3.99
## 4:              267              258             184  5000000-10000000  3.99
## 5:              288              624             415  5000000-10000000  3.99
```

The “steam-200k.csv” file will be loaded by the function fread(), and adding the names of the 4 columns respectively “userid”, “Name”, “Purchase_play”, “Hours”, “Dummy”.

```
# `interactions` dataset derived from `steam-200k.csv` file i.e. Steam Video Games dataset that display
interactions <- fread("steam-200k.csv",
               col.names = c("userid", "Name", "Purchase_play", "Hours", "Dummy"))

# Show the first 5 observations:
head(interactions,5)
```

```
##       userid                        Name Purchase_play Hours Dummy
## 1: 151603712 The Elder Scrolls V Skyrim      purchase     1     0
## 2: 151603712 The Elder Scrolls V Skyrim          play   273     0
## 3: 151603712                  Fallout 4      purchase     1     0
## 4: 151603712                  Fallout 4          play    87     0
## 5: 151603712                      Spore      purchase     1     0
```

## Data preparation and wrangling

We will prepare and clean the catalogue and interactions datasets so that we can subsequently combine them together for analysis/wrangling.

First, we will observe the “catalogue” dataset:

```
summary(catalogue)
```

```
##      appid              name            release_date           english
##  Min.   :     10   Length:27075       Min.   :1997-06-30   Min.   :0.0000
##  1st Qu.: 401230   Class :character   1st Qu.:2016-04-04   1st Qu.:1.0000
```

```
##  Median : 599070   Mode  :character   Median :2017-08-08   Median :1.0000
##  Mean   : 596204                      Mean   :2016-12-31   Mean   :0.9811
##  3rd Qu.: 798760                      3rd Qu.:2018-06-06   3rd Qu.:1.0000
##  Max.   :1069460                      Max.   :2019-05-01   Max.   :1.0000
##   developer          publisher          platforms          required_age
##  Length:27075       Length:27075       Length:27075       Min.   : 0.0000
##  Class :character   Class :character   Class :character   1st Qu.: 0.0000
##  Mode  :character   Mode  :character   Mode  :character   Median : 0.0000
##                                                            Mean   : 0.3549
##                                                            3rd Qu.: 0.0000
##                                                            Max.   :18.0000
##   categories           genres          steamspy_tags       achievements
##  Length:27075       Length:27075       Length:27075       Min.   :   0.00
##  Class :character   Class :character   Class :character   1st Qu.:   0.00
##  Mode  :character   Mode  :character   Mode  :character   Median :   7.00
##                                                            Mean   :  45.25
##                                                            3rd Qu.:  23.00
##                                                            Max.   :9821.00
##  positive_ratings  negative_ratings average_playtime   median_playtime
##  Min.   :      0   Min.   :     0   Min.   :     0.0   Min.   :     0.0
##  1st Qu.:      6   1st Qu.:     2   1st Qu.:     0.0   1st Qu.:     0.0
##  Median :     24   Median :     9   Median :     0.0   Median :     0.0
##  Mean   :   1001   Mean   :   211   Mean   :   149.8   Mean   :   146.1
##  3rd Qu.:    126   3rd Qu.:    42   3rd Qu.:     0.0   3rd Qu.:     0.0
##  Max.   :2644404   Max.   :487076   Max.   :190625.0   Max.   :190625.0
##     owners              price
##  Length:27075       Min.   :  0.000
##  Class :character   1st Qu.:  1.690
##  Mode  :character   Median :  3.990
##                     Mean   :  6.078
##                     3rd Qu.:  7.190
##                     Max.   :421.990
```

```
# Checking distinct values for columns that matter:
catalogue %>% summarise(n_appid = n_distinct(appid), n_name = n_distinct(name), n_publisher = n_distinct
```

```
##   n_appid n_name n_publisher n_genres n_tags
## 1   27075  27033       14261     1552   6423
```

```
# n_appid n_name n_publisher n_genres n_tags
#   27075  27033       14261     1552   6423

# Since there are less unique name values than appid values,
# we check to see how many games have duplicated names with different ids
n_occur <- data.frame(table(catalogue$name))
# gives you a dataframe with a list of names and the number of times they occurred.

catalogue_duplicates <- n_occur[n_occur$Freq > 1,]
catalogue_duplicates # names which occurred more than once
```

```
##                          Var1 Freq
## 145                      2048    2
## 1041                    Alone    2
```

```
## 1078                Alter Ego    2
## 1668                    Ashes    2
## 2542          Beyond the Wall    2
## 3141                   Bounce    2
## 3823                  Castles    2
## 3992             Chaos Theory    2
## 4274             City Builder    2
## 4528                   Colony    2
## 4778                   Cortex    2
## 5439              Dark Matter    3
## 6459                    Dodge    2
## 7613                   Escape    2
## 7644              Escape Room    2
## 7815                Evolution    2
## 7861                   Exodus    2
## 7884               Experience    2
## 8424                Fireflies    2
## 10615           Hide and Seek    2
## 11515                Invasion    2
## 12127            Killing Time    2
## 13263                    Luna    2
## 13652               Mars 2030    2
## 14849                Mystical    2
## 15181 New York Bus Simulator    2
## 15262    Nightmare Simulator    2
## 18700                  Rumpus    2
## 18764                    RUSH    2
## 18976        Santa's Workshop    2
## 19077                  Scorch    2
## 19901              Slice&Dice    2
## 20155               Solitaire    2
## 20375              Space Maze    2
## 21611                   Surge    2
## 22007                    Taxi    2
## 22692         The Great Escape    2
## 23011                The Mine    2
## 23410               The Tower    2
## 24575          Ultimate Arena    2
## 26543       Zombie Apocalypse    2
```

```
# Since both datasets would be ultimately joined together,
# we will check to see if the duplicated names appear in the Interactions dataset
sum(catalogue_duplicates$Var1 %in% interactions$Name)
```

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

```
# 6 of the duplicated titles are included in the `interactions` dataset

# Hence to avoid confusion, we would not include the `appid` column in the combined dataset.


# Also, we notice that there are multiple terms used in the `genres` column separated by `;`:
all_genres <- catalogue %>% separate_rows(genres, sep = "\\;") %>%
```

```
  select(genres) %>% unique() # get unique genres

# sort in alphabetical order
all_genres <- all_genres[order(all_genres),]

dim(all_genres) # [1] 29  1
```

```
## [1] 29  1
```

```
#There are 29 unique genres in this dataset
```

We notice that the appid and name columns do not have the same number of values. In fact, there are slightly less names than appids. Since both datasets would be ultimately joined together, we checked to see if the duplicated names appear in the Interactions dataset. 6 of the the duplicated titles are included in the `interactions` dataset. Hence, to avoid confusion, we would not include the `appid` column in the combined dataset.

We also notice that the values in the `genres` column are multiple values joined together with semicolons (;). Hence, we create the `all_genres` object to find that there are 29 unique types of genres in this dataset.

Next, we will clean the "catalogue" dataset. For this analysis, we would consider the following columns from Catalogue dataset: appid, name, release_date, developer, publisher, genres, positive_ratings, negative_ratings, average_playtime, median_playtime, price

We will also create a few columns and remove the columns that they are derived from: * `user_rating` derived from the percentage (%) (rounded up to the nearest whole number) of positive ratings over total ratings for the respective video games * `year` derived from the year portion of the `release_date` column

```
# For this analysis, we would consider the following columns from Catalogue dataset:
# name, release_date, developer, publisher, genres,
# positive_ratings, negative_ratings, price

catalogue_clean <- catalogue %>% select(name, release_date, developer, publisher, genres, positive_rati
summary(catalogue_clean)
```

```
##      name            release_date         developer          publisher
##  Length:27075       Min.   :1997-06-30   Length:27075       Length:27075
##  Class :character   1st Qu.:2016-04-04   Class :character   Class :character
##  Mode  :character   Median :2017-08-08   Mode  :character   Mode  :character
##                     Mean   :2016-12-31
##                     3rd Qu.:2018-06-06
##                     Max.   :2019-05-01
##     genres          positive_ratings  negative_ratings     price
##  Length:27075       Min.   :      0   Min.   :     0   Min.   :  0.000
##  Class :character   1st Qu.:      6   1st Qu.:     2   1st Qu.:  1.690
##  Mode  :character   Median :     24   Median :     9   Median :  3.990
##                     Mean   :   1001   Mean   :   211   Mean   :  6.078
##                     3rd Qu.:    126   3rd Qu.:    42   3rd Qu.:  7.190
##                     Max.   :2644404   Max.   :487076   Max.   :421.990
```

```
# We will add another column called `user_ratings`.
# This would show the rounded-up percentage of positive ratings over total ratings (positive ratings + 
catalogue_clean <- catalogue_clean %>%
```

```r
  mutate(user_ratings = ceiling(positive_ratings*100/(positive_ratings + negative_ratings)) ) %>%
  select(-positive_ratings, -negative_ratings) # Remove `positive_ratings` and `negative_ratings`

# We will create a `year` column from the `release_date`, then remove the `release_date` column
catalogue_clean <- catalogue_clean %>%
  mutate(year = as.numeric(year(release_date))) %>% # create `year` column
  select(-release_date) # remove `release_date`

head(catalogue_clean)
```

```
##                          name        developer publisher genres price
## 1:             Counter-Strike            Valve     Valve Action  7.19
## 2:      Team Fortress Classic            Valve     Valve Action  3.99
## 3:              Day of Defeat            Valve     Valve Action  3.99
## 4:         Deathmatch Classic            Valve     Valve Action  3.99
## 5: Half-Life: Opposing Force Gearbox Software     Valve Action  3.99
## 6:                   Ricochet            Valve     Valve Action  3.99
##    user_ratings year
## 1:           98 2000
## 2:           84 1999
## 3:           90 2003
## 4:           83 2001
## 5:           95 1999
## 6:           81 2000
```

Now that we have scrubbed the Catalogue dataset, we will scrub the Interactions dataset to prepare it for joining. First, we will remove the Dummy column since it only contains zeros (0).

```r
# Remove dummy column
interactions <- interactions %>% select(-Dummy)
```

Then we will check the number of purchased games versus played games.

Since games have to be purchased first before playing, and there are 129,511 purchased games compared to 70,489 played games, it means that there are players who purchase games and did not play them.

Also, according to the website's notes for the `steam-200k.csv`, Rows with "purchase" label have 1.0 indicated under the `Hours` column. So we also checked that not all games with the `Hours` value being 1.0 were referring to only purchased games. Some games had 1 hour of playtime as well.

```r
# Since Steam is a digital distribution service for video games, digital copies of games have to be pur

nrow(interactions[interactions$Purchase_play == "play"])
```

```
## [1] 70489
```

```r
# [1] 70489
nrow(interactions[interactions$Purchase_play == "purchase"])
```

```
## [1] 129511
```

```
# [1] 129511

# Notice that there are users who had purchased games and not played them on Steam

count(interactions[interactions$Hours == 1.0])
```

```
##         n
## 1: 130569
```

```
# n: 130569
# Also, based on count alone, not all `Hours` values that are 1.0 refer to purchases
```

Hence, we create an an `interactions_clean` dataset by adding to the `interactions` dataset an `hours_played` column based on `Purchase_play` values and `Hours`, where those with the "purchase" label had the value 0.

Then, we aggregate the `interactions_clean` dataset by summing up Hours played according to User ID and game title (Name). This reduces confusion on the `Hours` column and removes the need for the `Purchase_play` column.

We also change all the column names to lowercase make it easier when we join the datasets later.

```
# Hence, we create an `hours_played` column based on `Purchase_play` values and `Hours`
interactions_clean <- interactions %>% mutate(hours_played = case_when(
  endsWith(Purchase_play, "play") ~ Hours,
  endsWith(Purchase_play, "purchase") ~ 0
))
# Then, we aggregate the `interactions_clean` dataset by summing up Hours played according to User ID a
# This reduces confusion on the `Hours` column and removes the need for the `Purchase_play` column.
interactions_clean <- aggregate(hours_played~userid+Name, data=interactions_clean, FUN=sum)

# We also change all the column names to lowercase make it easier when we join the datasets later.
colnames(interactions_clean) <- c("userid", "name", "hours_played")
```

We will perform an inner_join function on the "catalogue_clean" and "interactions_clean" datasets by the column "name". The inner join ensures that only complete rows in this combined dataset would be used to train, test and validate the model.

The result of the inner_join will be stored in the 'combined' dataset. We can see the summary below:

```
combined <- interactions_clean %>% inner_join(catalogue_clean, by = "name")
```

## Summary of the dataset

The summary() function presents us with a initial assessment of the data quartiles, min, max, mean and median values for each variable. For the character types it displays the vector length, class and mode.

```
##      userid              name            hours_played      developer
##  Min.   :     5250   Length:56497       Min.   :    0.0   Length:56497
##  1st Qu.: 49462664   Class :character   1st Qu.:    0.0   Class :character
##  Median : 99264709   Mode  :character   Median :    0.9   Mode  :character
##  Mean   :112868455                      Mean   :   34.9
```

```
##  3rd Qu.:167815968                      3rd Qu.:    7.8
##  Max.   :309903146                      Max.   :10442.0
##   publisher            genres              price         user_ratings
##  Length:56497       Length:56497       Min.   : 0.000   Min.   :  0.00
##  Class :character   Class :character   1st Qu.: 0.000   1st Qu.: 78.00
##  Mode  :character   Mode  :character   Median : 4.990   Median : 86.00
##                                        Mean   : 6.176   Mean   : 83.93
##                                        3rd Qu.: 7.990   3rd Qu.: 94.00
##                                        Max.   :69.990   Max.   :100.00
##       year
##  Min.   :1997
##  1st Qu.:2010
##  Median :2013
##  Mean   :2012
##  3rd Qu.:2014
##  Max.   :2019
```

Custom summary of the combined data display the number of distinct users and movies, along with minimum and maximum values for rating value, release and rating year.

```
##   n_users n_games n_genres n_publishers n_developers hours_played_min
## 1   10122    2191      310         1250         1690                0
##   hours_played_max release_min release_max price_min price_max
## 1            10442        1997        2019         0     69.99
```

## Data exploration and vizualisation

For machine learning purposes, data comes in two forms, namely the *outcome* and the *features*. Before we start creating our models, we need to determine what inputs we will use as predictors (features) and what output will be our target variable (outcome). For this project, the hours played will be our target. This means that we will train different models and aim to predict the actual number of hours the user would play to an unknown (not purchased or not played) game to that particular user.

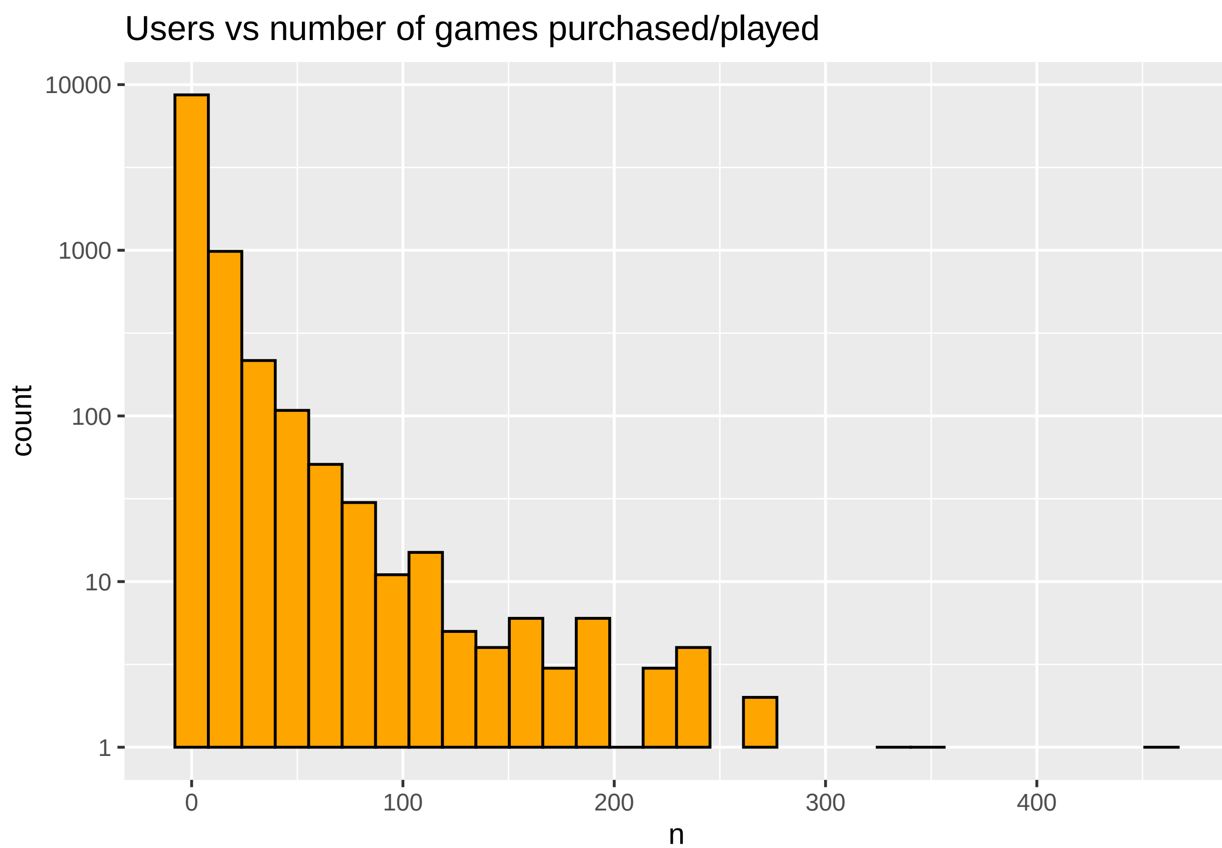
The Combined data set contains 10,122 users, 2,191 games and 56,497 combinations showing hours played plus additional data including genres, developer, publisher, ratings and price.

The distribution of the hours played column versus number of users, as displayed on Figure 1:

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## Warning: Transformation introduced infinite values in continuous y-axis
```

```
## Warning: Removed 10 rows containing missing values (geom_bar).
```




We also want to see the users who had purchased and/or played the most number of video games from Steam. The largest number of video games purchased and/or played by a single user was 460.

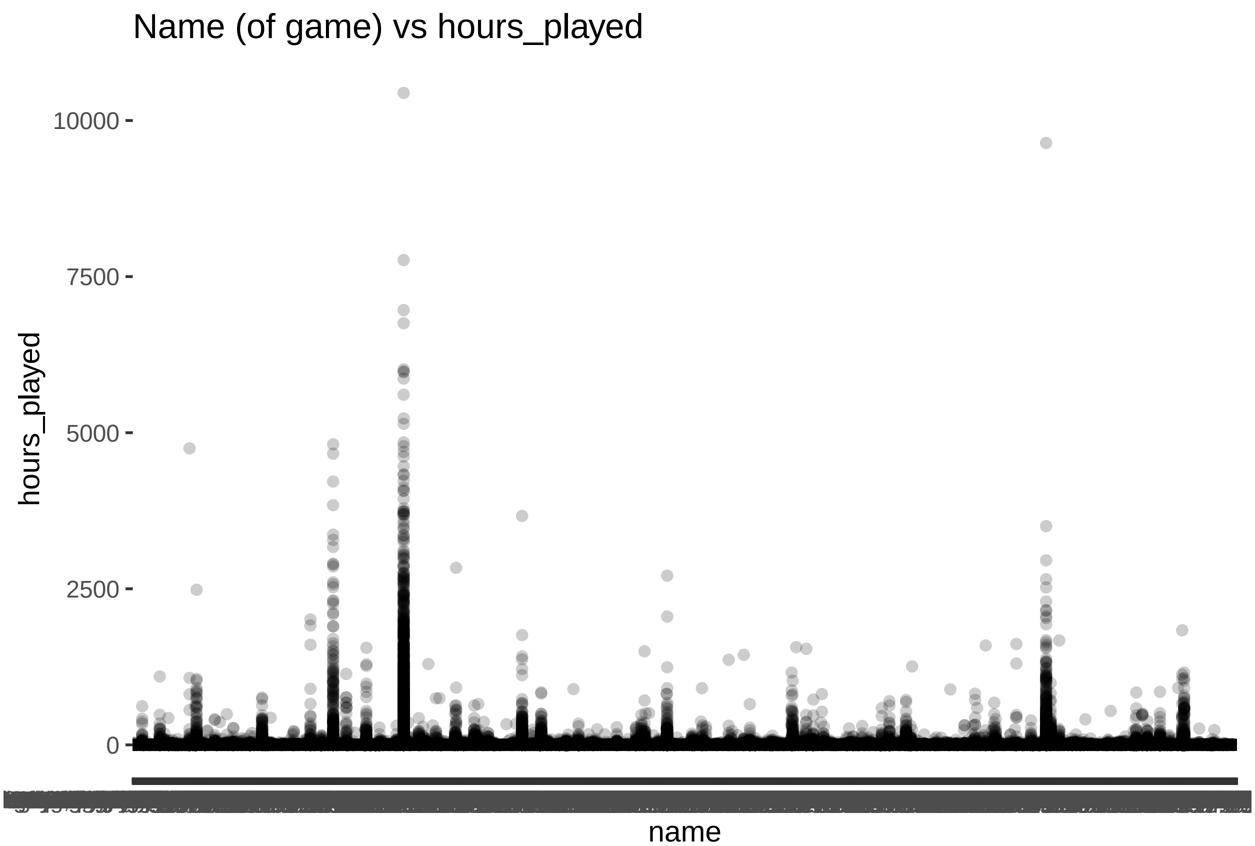
```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## Selecting by count
```

```
## # A tibble: 20 x 2
##       userid count
##        <int> <int>
##  1  62990992   460
##  2  33865373   355
##  3  30246419   338
##  4  11403772   274
##  5  58345543   274
##  6 154230723   243
##  7  64787956   236
##  8  53875128   234
##  9  47457723   232
## 10  22301321   226
## 11 138941587   224
## 12  33013552   220
## 13  11373749   204
## 14  49893565   196
## 15  36557643   193
## 16  20772968   192
```

```
## 17  36546868   192
## 18  59825286   187
## 19  24721232   185
## 20  24469287   179
```

In fact, by looking at the graph below, we can see that relatively few games had users who played for more than 1000 hours.




If we take a look at the summary of hours played (see below), the median (i.e. 50th percentile) was 0.9 hours played, while the 3rd quartile (75th percentile) was 7.8 hours played. However, the mean was 34.9 hours played, which is more than 4 times that of the 3rd quartile. What is even more surprising is that both the minimum and the 1st quartile (25th percentile) were 0.0 hours played (i.e. purchased but not played at all). So let's analyse these observations even further.

```
##   hours_played
##  Min.   :    0.0
##  1st Qu.:    0.0
##  Median :    0.9
##  Mean   :   34.9
##  3rd Qu.:    7.8
##  Max.   :10442.0
```

Hence we will check to see what proportion of games-user combinations had be purchases without playing.

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

```
## Selecting by pct_of_total
```

```
## # A tibble: 20 x 3
##    hours_played count pct_of_total
##           <dbl> <int>        <dbl>
##  1          0   20194       35.7
##  2          0.2  1513        2.68
##  3          0.3  1315        2.33
##  4          0.4  1123        1.99
##  5          0.5   931        1.65
##  6          0.1   893        1.58
##  7          0.6   838        1.48
##  8          0.7   731        1.29
##  9          0.8   683        1.21
## 10          0.9   644        1.14
## 11          1     562        0.995
## 12          1.1   525        0.929
## 13          1.2   494        0.874
## 14          1.3   494        0.874
## 15          1.4   460        0.814
## 16          1.5   443        0.784
## 17          1.6   401        0.710
## 18          1.7   399        0.706
## 19          1.8   374        0.662
## 20          1.9   356        0.630
```

By sorting the number and percentage of occurences of `hours_played` values in descending order, we saw that 35.7% of the time, 0.0 hours were played for games purchased on Steam. This means that games are hoarded and not played about 35% of the time.

```r
# Summarising occurrences of Hours played that are above the mean of 34.9 hours
above_avg_hours_played <- combined %>% group_by(hours_played) %>%
  summarise(count = n(), pct_of_total = 100*(count/nrow(combined)) ) %>% arrange(desc(count)) %>% filte
```

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

```r
above_avg_hours_played
```

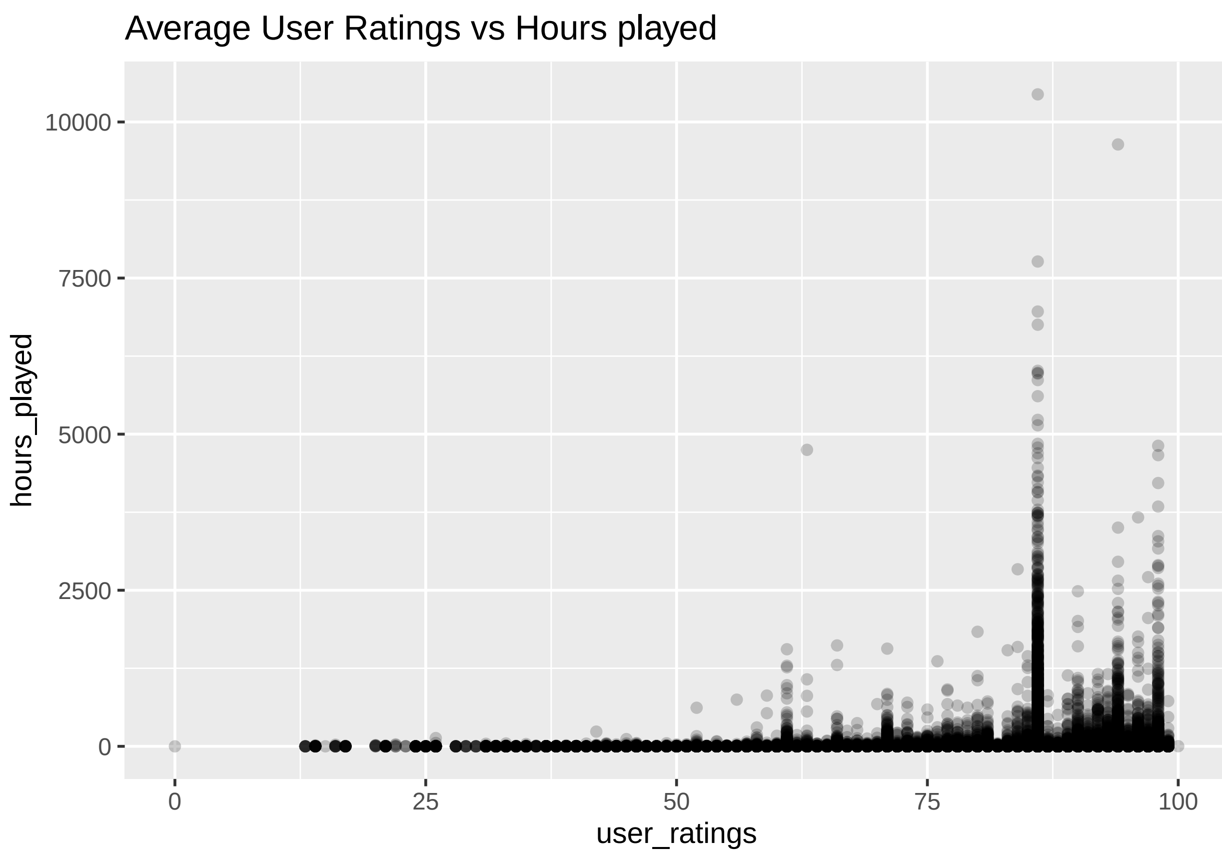
```
## # A tibble: 1,101 x 3
##    hours_played count pct_of_total
##           <dbl> <int>        <dbl>
##  1           36   130        0.230
##  2           35   108        0.191
##  3           38   108        0.191
##  4           37   106        0.188
##  5           39   106        0.188
##  6           40    91        0.161
##  7           41    82        0.145
##  8           43    76        0.135
##  9           44    76        0.135
## 10           47    76        0.135
## # ... with 1,091 more rows
```

```
sum(above_avg_hours_played$pct_of_total)
```

```
## [1] 10.59525
```

Around 10.6% of the time, users in the combined dataset played Steam games above the average number of hours.

From the chart shown below, there is generally a positively skewed trend *between Average User Ratings to Hours played*, with the most number of hours played peaking at around 82.




We can also see from the list below, Dota 2 significantly beat the other video games in terms of total number of hours played by users, with a total of 981684.6 hours played.

```
# Top played games by total number of hours played
combined %>% select(name, hours_played) %>% group_by(name) %>% summarise(total_hours_played = sum(hours_
  top_n(10) %>% arrange(desc(total_hours_played))
```

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

```
## Selecting by total_hours_played
```

```
## # A tibble: 10 x 2
##    name                 total_hours_played
##    <chr>                             <dbl>
##  1 Dota 2                          981685.
```

```
##   2 Team Fortress 2              173673.
##   3 Counter-Strike               134261.
##   4 Garry's Mod                   49725.
##   5 Left 4 Dead 2                 33597.
##   6 Terraria                      29952.
##   7 Warframe                      27075.
##   8 Arma 3                        24056.
##   9 Grand Theft Auto V            22957.
##  10 Borderlands 2                 22668.
```

# METHODOLOGY

After data gathering, cleaning, exploring and visualization, the next step is to look into the methods we would like to implement and compare before analysing its performance on the final hold-out set: validation set. The first 5 methods will be our baseline for comparison with the Reverse Bias and Regularisation Models.

Bellow is a list over the techniques we will compare during this project:

1 - 'Model 0 - Overall average (Naive RMSE)' 2 - 'Model 1 - added Name bias' 3 - 'Model 2 - added User bias' 4 - 'Model 3 - added Genre bias' 5 - 'Model 4 - added User Rating bias' 6 - 'Model 5 - added Price bias to Model 3' 7 - 'Model 6a - Reverse Bias - added User bias' and 'Model 6b - Reverse Bias - added Name bias' 8 - 'Model 7 - Regularised model'

First we will create a data set from the original combined data to be only used on the final model. For that we are creating the validation set (`combined_validate`) at 10% of the Combined data. We will also remove unnecessary objects from memory with the rm() command:

```
## Create data partition to create validation data set:

set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(1)`
validation_index <- createDataPartition(y = combined$hours_played, times = 1, p = 0.1, list = FALSE)

# Data set for training and testing model
combined_model <- combined[-validation_index,]

# Validation data set
combined_validate <- combined[validation_index,]

rm(validation_index)
```

The validation set created will not be used to train or test our algorithms. So, we need to partition the *combined_model* set in train and test sets.

It is important to determine how much data will be used to train versus test. We want enough observations to train but we also want have a decent proportion of *unseen* observations to test with. We also need to ensure that the same "name" and "userid" also appears in the test set, but not the same observations(rows).

The next step after cleaning and exploring the Combined Model data is to create train and test sets. We are going to reserve 10% of the *combined_model* set as *test_set*. To create the these sets we will use the function *createDataPartition()* from the *Caret* package. To replicate the same results set the seed to 1.

```
# Creating Test and Train sets
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(1)`
test_index <- createDataPartition(y = combined_model$hours_played, times = 1, p = 0.1, list = FALSE)
```

```r
# Test set
test_set <- combined_model[test_index,]

# Train Set
train_set <- combined_model[-test_index,]

# To ensure that we are not testing on games and users we have not seen before
test_set <- test_set %>% semi_join(train_set, by='name') %>%
  semi_join(train_set, by='userid')

rm(test_index)
```

Now let's inspect the dimensions of our sets:

```r
dim(train_set)
```

```
## [1] 45761     9
```

```r
dim(test_set)
```

```
## [1] 4484    9
```

To evaluate the performance of the models, we will use the root mean squared error (RMSE) as the default standard for comparison.

By definition RMSE is:

$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{Y}_i - Y_i)^2}$

where $N$ is the sample size, $\hat{y}_i$ are the predicted values and $y_i$ are the corresponding observations.

Let's define our RMSE function in relation to hours:

```r
# Defining the RMSE function used in this project
RMSE <- function(true_hours = NULL, predicted_hours = NULL) {
  sqrt(mean((true_hours - predicted_hours)^2))
}
```

## Model 0: Overall average (Naive RMSE)

We will start with the quickest and most basic way to predict a rating would be to guess the average hours played from the train dataset. Applying the mean function to the hours_played column in the train_set we get 34.5903936. The simplest method would be to predict using the the average of the rating column. We can see the resulting RMSE bellow:

```r
mu <- train_set$hours_played %>% mean()
naive_rmse <- RMSE(test_set$hours_played, mu)

methods <- ('Model 0 - Just the average')
rmses <- (naive_rmse)

model_evaluations <- tibble(method=methods, RMSE=rmses)
```

### Model 1 - Added Name (of Video Game) bias

Some video games are played for longer compared to others. We can include the average hours played for a video game to our model. To analyze this further we will calculate the difference between the game's average hours played and the total hours played for all video games. If result is positive, it means that the game is played more than the mean.

```
name_avgs <- train_set %>% group_by(name) %>% summarise(b_i=mean(hours_played-mu))
```

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

```
name_bias_model <- mu + test_set %>% left_join(name_avgs, by='name') %>% pull(b_i)
name_bias_rmse <- RMSE(test_set$hours_played, name_bias_model)

methods <- c('Model 0 - Just the average', 'Model 1 - added Name bias')
rmses <- c(naive_rmse, name_bias_rmse)
(model_evaluations <- tibble(method=methods, RMSE=rmses))
```

```
## # A tibble: 2 x 2
##   method                      RMSE
##   <chr>                      <dbl>
## 1 Model 0 - Just the average  225.
## 2 Model 1 - added Name bias   213.
```

As we can see from the table above, there is a drop in the RMSE from the naive RMSE. Let us see if we can further improve the RMSE.

### Model 2: Added User bias term

We can improve our predictions by adding a user effect to our model. Some users purchase many games and do not play them, whereas there are others who tend to spend a lot of time playing specific games.

```
# Calculate User bias
user_avgs <- train_set %>% left_join(name_avgs, by='name') %>% group_by(userid) %>% summarise(b_u=mean(

user_name_bias_model <- test_set %>% left_join(name_avgs, by='name') %>% left_join(user_avgs, by='useri
  mutate(pred=mu+b_i+b_u) %>% pull(pred)

user_name_bias_rmse <- RMSE(test_set$hours_played, user_name_bias_model)


methods <- c('Model 0 - Just the average', 'Model 1 - added Name bias',
             'Model 2 - added User bias')
rmses <- c(naive_rmse, name_bias_rmse, user_name_bias_rmse)

(model_evaluations <- tibble(method=methods, RMSE=rmses))
```

```
## # A tibble: 3 x 2
##   method                      RMSE
##   <chr>                      <dbl>
## 1 Model 0 - Just the average  225.
## 2 Model 1 - added Name bias   213.
## 3 Model 2 - added User bias   230.
```

The RMSE achieved by the Name and User bias was 229.9122872.

Notice that the added User bias makes the RMSE even higher than the Naive RMSE (Model 0).

This big discrepancy creates estimates that might not be trusted. We can try to account for this by introducing penalties for these occurrences.

## Model 3: Added Genre Bias Term

Since we have seen that there has been a drop to the RMSE with the added User bias, we will incorporate the Genre bias term to the previous model and see if there is any impact to the RMSE.

```
genre_avgs <- train_set %>% left_join(name_avgs, by='name') %>% left_join(user_avgs, by='userid') %>%
  group_by(genres) %>% summarise(b_g=mean(hours_played-mu-b_i-b_u))
```

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

```
genre_user_name_bias_model <- test_set %>%
  left_join(name_avgs, by='name') %>%
  left_join(user_avgs, by='userid') %>%
  left_join(genre_avgs, by='genres') %>%
  mutate(pred=mu+b_i+b_u+b_g) %>% pull(pred)

(genre_user_name_bias_rmse <- RMSE(test_set$hours_played, genre_user_name_bias_model))
```

```
## [1] 227.9479
```

```
#[1] 235.3351
```

Below are the models that we have explored and their respective RMSEs thus far.

```
methods <- c('Model 0 - Just the average', 'Model 1 - added Name bias', 'Model 2 - added User bias', 'Mo
rmses <- c(naive_rmse, name_bias_rmse, user_name_bias_rmse,
           genre_user_name_bias_rmse)

(model_evaluations <- tibble(method=methods, RMSE=rmses))
```

```
## # A tibble: 4 x 2
##   method                      RMSE
##   <chr>                      <dbl>
## 1 Model 0 - Just the average  225.
## 2 Model 1 - added Name bias   213.
## 3 Model 2 - added User bias   230.
## 4 Model 3 - added Genre bias  228.
```

It is surprising to see that with the added Genre bias term, the RMSE becomes larger. In fact, it is even bigger than the Naive RMSE. This likely means that Genre does not shape users' interactions with Steam video games.

## Model 4: Added User Rating Bias Term

Now we will incorporate the User Rating bias term to the previous model and see if there is any impact to the RMSE.

```
rating_avgs <- train_set %>% left_join(name_avgs, by='name') %>% left_join(user_avgs, by='userid') %>%
  left_join(genre_avgs, by='genres') %>% group_by(user_ratings) %>% summarise(b_r=mean(hours_played-mu-b
```

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

```
rating_genre_user_name_bias_model <- test_set %>% left_join(name_avgs, by='name') %>% left_join(user_avg
  left_join(genre_avgs, by='genres') %>% left_join(rating_avgs, by='user_ratings') %>% mutate(pred=mu+b_

(rating_genre_user_name_bias_rmse <- RMSE(test_set$hours_played, rating_genre_user_name_bias_model))
```

```
## [1] 227.9565
```

```
# [1] 235.3351
```

Below are the models that we have explored and their respective RMSEs thus far.

```
methods <- c('Model 0 - Just the average', 'Model 1 - added Name bias', 'Model 2 - added User bias',
             'Model 3 - added Genre bias', 'Model 4 - added User Rating bias')
rmses <- c(naive_rmse, name_bias_rmse, user_name_bias_rmse,
           genre_user_name_bias_rmse, rating_genre_user_name_bias_rmse)

(model_evaluations <- tibble(method=methods, RMSE=rmses))
```

```
## # A tibble: 5 x 2
##   method                            RMSE
##   <chr>                            <dbl>
## 1 Model 0 - Just the average        225.
## 2 Model 1 - added Name bias         213.
## 3 Model 2 - added User bias         230.
## 4 Model 3 - added Genre bias        228.
## 5 Model 4 - added User Rating bias  228.
```

From the table above, we observe that there has been barely any change to the RMSE with the added User Rating bias term. This is proof that User Ratings has minimal effect on the RMSE.

## Model 5: Added Price Bias Term to Model 3

Since factoring User Rating in the previous model (Model 4) had barely any effect on the RMSE, we will now check to see if factoring Price bias term to Model 3 (Naive with added Name, User and Genre biases) has any impact on the RMSE.

```
price_avgs <- train_set %>% left_join(name_avgs, by='name') %>% left_join(user_avgs, by='userid') %>%
  left_join(genre_avgs, by='genres') %>% group_by(price) %>% summarise(b_p=mean(hours_played-mu-b_i-b_u
```

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

```
price_genre_user_name_bias_model <- test_set %>% left_join(name_avgs, by='name') %>% left_join(user_avgs
  left_join(genre_avgs, by='genres') %>% left_join(price_avgs, by='price') %>% mutate(pred=mu+b_i+b_u+b

(price_genre_user_name_bias_rmse <- RMSE(test_set$hours_played, price_genre_user_name_bias_model))
```

```
## [1] 227.943
```

```
# [1] 235.3351
```

Below are the models that we have explored and their respective RMSEs thus far.

```
methods <- c('Model 0 - Just the average', 'Model 1 - added Name bias', 'Model 2 - added User bias',
             'Model 3 - added Genre bias', 'Model 4 - added User Rating bias', 'Model 5 - added Price bi
rmses <- c(naive_rmse, name_bias_rmse, user_name_bias_rmse,
           genre_user_name_bias_rmse, rating_genre_user_name_bias_rmse,
           price_genre_user_name_bias_rmse)

(model_evaluations <- tibble(method=methods, RMSE=rmses))
```

```
## # A tibble: 6 x 2
##   method                              RMSE
##   <chr>                              <dbl>
## 1 Model 0 - Just the average          225.
## 2 Model 1 - added Name bias           213.
## 3 Model 2 - added User bias           230.
## 4 Model 3 - added Genre bias          228.
## 5 Model 4 - added User Rating bias    228.
## 6 Model 5 - added Price bias to Model 3  228.
```

From observing the RMSEs for Models 4 and 5, it appears that factoring more biases (such as User Ratings and Price) after considering User, Name and Genre bias terms has little to no effect on the RMSE.

Also, adding the Genre bias term in Model 3 only made the RMSE even higher than the naive RMSE (Model 0). Therefore, we would only consider the Name and User bias terms when building and comparing subsequent models.

### Model 6: Reversed Bias Terms

Since we have tested name bias followed by user bias in earlier models, we will now test to see if the reverse order of testing these biases would change the RMSE.

*Part 1*: This model would build on mu by introducing a *User bias* term.

```
# This model builds on mu by introducing a user bias term
user_avgs <- train_set %>% group_by(userid) %>% summarise(b_u=mean(hours_played-mu))
```

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

```
user_bias_model <- mu + test_set %>% left_join(user_avgs, by='userid') %>% pull(b_u)
(rb_user_bias_rmse <- RMSE(test_set$hours_played, user_bias_model))
```

```
## [1] 241.7927
```

*Part 2*: We then introduce the *Name bias* term to the model built in the previous part.

```
# This model builds on previous by introducing a name bias term
name_avgs <- train_set %>% left_join(user_avgs, by='userid') %>% group_by(name) %>% summarise(b_i=mean(h
```

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

```
name_user_bias_model <- test_set %>% left_join(name_avgs, by='name') %>% left_join(user_avgs, by='userid
  mutate(pred=mu+b_i+b_u) %>% pull(pred)
(rb_name_user_bias_rmse <- RMSE(test_set$hours_played, name_user_bias_model))
```

```
## [1] 235.3351
```

Hence these are the models that we have explored and their respective RMSEs thus far.

```
methods <- c('Model 0 - Just the average', 'Model 1 - added Name bias', 'Model 2 - added User bias',
             'Model 3 - added Genre bias', 'Model 4 - added User Rating bias',
             'Model 5 - added Price bias to Model 3', 'Model 6a - Reverse Bias - added User bias',
             'Model 6b - Reverse Bias - added Name bias')
rmses <- c(naive_rmse, name_bias_rmse, user_name_bias_rmse,
           genre_user_name_bias_rmse, rating_genre_user_name_bias_rmse,
           price_genre_user_name_bias_rmse, rb_user_bias_rmse,
           rb_name_user_bias_rmse)

(model_evaluations <- tibble(method=methods, RMSE=rmses))
```

```
## # A tibble: 8 x 2
##   method                                     RMSE
##   <chr>                                     <dbl>
## 1 Model 0 - Just the average                 225.
## 2 Model 1 - added Name bias                  213.
## 3 Model 2 - added User bias                  230.
## 4 Model 3 - added Genre bias                 228.
## 5 Model 4 - added User Rating bias           228.
## 6 Model 5 - added Price bias to Model 3      228.
## 7 Model 6a - Reverse Bias - added User bias  242.
## 8 Model 6b - Reverse Bias - added Name bias  235.
```

## Model 7: Regularised Bias

In order to find a balance for minimizing the our model's expected error, we will include additional information to prevent overfitting (eg. model has 100% accuracy on train set, but 50% accurate on test set). So here we include an lambda value as independent variable.

We also added the "coalesce()" function to replace missing values with 0. This ensures completeness of the data and factor in that not all data in the train set may be present in the test set.

```
lambdas <- seq(0, 70, 1)
rmses <- sapply(lambdas, function(l){
  mu <- mean(train_set$hours_played)
  b_i <- train_set %>% group_by(name) %>% summarise(b_i=sum(hours_played-mu)/(n()+l))

  b_u <- train_set %>% left_join(b_i, by='name') %>%
    group_by(userid) %>% summarise(b_u=sum(hours_played-b_i-mu)/(n()+l))

  predicted_hours <- test_set %>% left_join(b_i, by='name') %>%
    left_join(b_u, by='userid') %>%
    mutate(b_i = coalesce(b_i, 0), b_u = coalesce(b_u, 0), pred=mu+b_i+b_u) %>%
    # coalesce function to replace missing values with 0 for b_i & b_u
    pull(pred)

  return(RMSE(predicted_hours, test_set$hours_played))
})
```

In order to determine the best value for the independent variable lambda, we initially ran calculations using values from 0 to 30. But to better fit the plot increased the size of the lambda vector to 0 to 70.

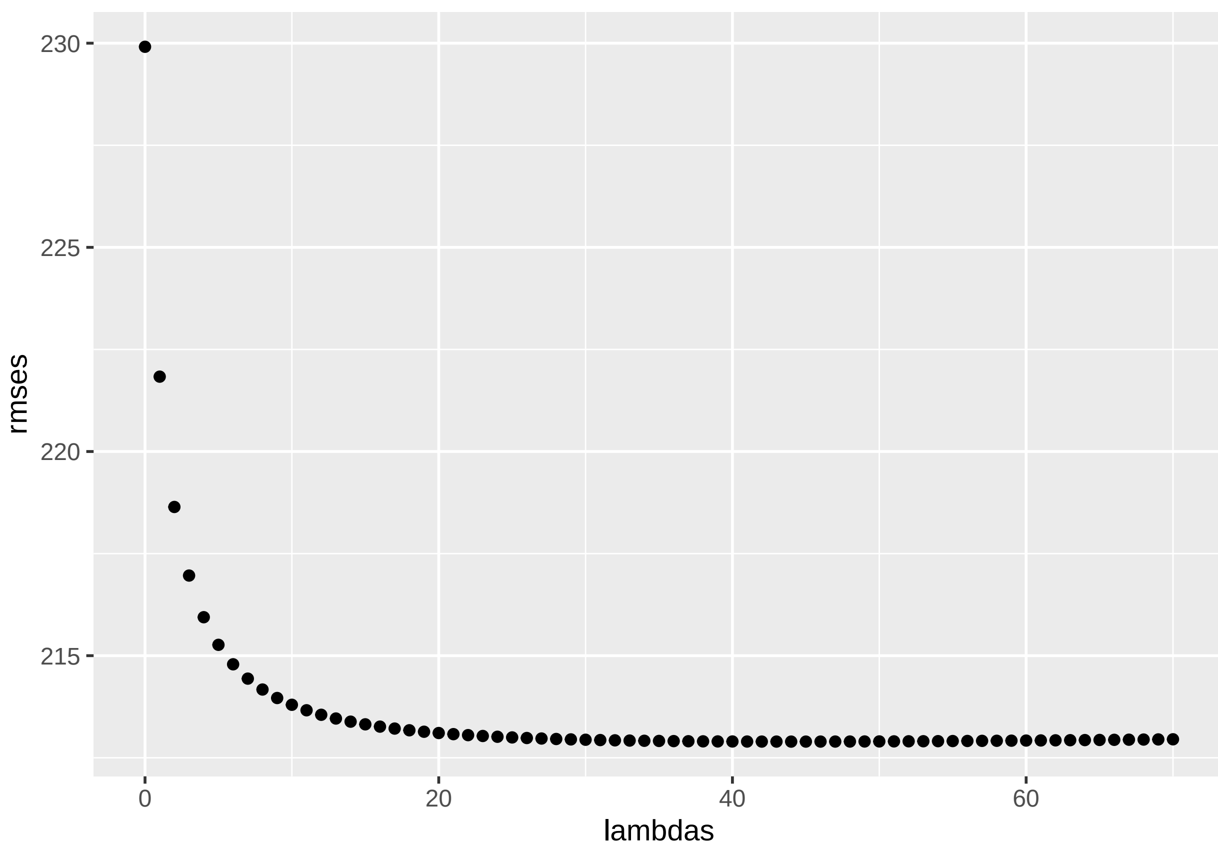



Figure 1: Lambda Versus RMSE plot

The best value for lambda is 44 and the corresponding RMSE is 212.8959952. We will recalculate the regularised model with the new lambda correction.

```r
#Regularised name bias term
b_i <- train_set %>% group_by(name) %>% summarise(b_i=sum(hours_played-mu)/(n()+lambda))
```

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

```r
#Regularised user bias term
b_u <- train_set %>% left_join(b_i, by='name') %>%
  group_by(userid) %>% summarise(b_u=sum(hours_played-b_i-mu)/(n()+lambda))
```

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

```r
regularised_user_name_model <- test_set %>%
  left_join(b_i, by='name') %>%
  left_join(b_u, by='userid') %>%
  mutate(b_i = coalesce(b_i, 0), b_u = coalesce(b_u, 0), pred=mu+b_i+b_u) %>%
  # coalesce function to replace missing values with 0 for b_i & b_u
  pull(pred)

regularised_user_name_rmse <- RMSE(test_set$hours_played, regularised_user_name_model)
```

Hence these are the models that we have explored and their respective RMSEs thus far.

```r
methods <- c('Model 0 - Just the average', 'Model 1 - added Name bias', 'Model 2 - added User bias',
             'Model 3 - added Genre bias', 'Model 4 - added User Rating bias',
             'Model 5 - added Price bias to Model 3', 'Model 6a - Reverse Bias - added User bias',
             'Model 6b - Reverse Bias - added Name bias', 'Model 7 - Regularised model')
rmses <- c(naive_rmse, name_bias_rmse, user_name_bias_rmse,
           genre_user_name_bias_rmse, rating_genre_user_name_bias_rmse,
           price_genre_user_name_bias_rmse, rb_user_bias_rmse,
           rb_name_user_bias_rmse, regularised_user_name_rmse)

(model_evaluations <- tibble(method=methods, RMSE=rmses))
```

```
## # A tibble: 9 x 2
##   method                                     RMSE
##   <chr>                                     <dbl>
## 1 Model 0 - Just the average                 225.
## 2 Model 1 - added Name bias                  213.
## 3 Model 2 - added User bias                  230.
## 4 Model 3 - added Genre bias                 228.
## 5 Model 4 - added User Rating bias           228.
## 6 Model 5 - added Price bias to Model 3      228.
## 7 Model 6a - Reverse Bias - added User bias  242.
## 8 Model 6b - Reverse Bias - added Name bias  235.
## 9 Model 7 - Regularised model                213.
```

## Choosing a model to validate

The table below lists all the models and their performance results when making predictions on the test_set:

```
model_evaluations
```

```
## # A tibble: 9 x 2
##   method                                RMSE
##   <chr>                                <dbl>
## 1 Model 0 - Just the average            225.
## 2 Model 1 - added Name bias             213.
## 3 Model 2 - added User bias             230.
## 4 Model 3 - added Genre bias            228.
## 5 Model 4 - added User Rating bias      228.
## 6 Model 5 - added Price bias to Model 3      228.
## 7 Model 6a - Reverse Bias - added User bias  242.
## 8 Model 6b - Reverse Bias - added Name bias  235.
## 9 Model 7 - Regularised model           213.
```

Based on the table above, the Regularised & Cross-Validation model (Model 7) provided the lowest RMSE among all the models. Hence we would use this to validate the model using the "combined_validate" dataset.

# VALIDATION

As we can observe from the table above, the best performing method was the Regularised & Cross-Validation Model with a 212.8959952 RMSE. This was our best performing model. We will use it for the final test: how it performs on the validation set (unseen/unknown data).

Making predictions on the validation data:

```
# Use cross-validation to search for best lambda term:
lambdas <- seq(0, 70, 1)
rmses <- sapply(lambdas, function(l){
  mu <- mean(combined_model$hours_played)
  b_i <- combined_model %>% group_by(name) %>% summarise(b_i=sum(hours_played-mu)/(n()+l))

  b_u <- combined_model %>% left_join(b_i, by='name') %>%
    group_by(userid) %>% summarise(b_u=sum(hours_played-b_i-mu)/(n()+l))

  predicted_hours <- combined_validate %>% left_join(b_i, by='name') %>%
    left_join(b_u, by='userid') %>%
    mutate(b_i = coalesce(b_i, 0), b_u = coalesce(b_u, 0), pred=mu+b_i+b_u) %>% # replace missing value
    pull(pred)

  return(RMSE(predicted_hours, combined_validate$hours_played))
})
```

In order to determine the best value for the independent variable lambda, we run calculations using values from 0 to 70.

The best value for lambda is 65 and the corresponding RMSE is 178.9138225 on the "combined_validate". We will recalculate the final regularised model with the new lambda correction.

```
# Regularised name bias term
b_i <- combined_model %>% group_by(name) %>% summarise(b_i=sum(hours_played-mu)/(n()+lambda))
```

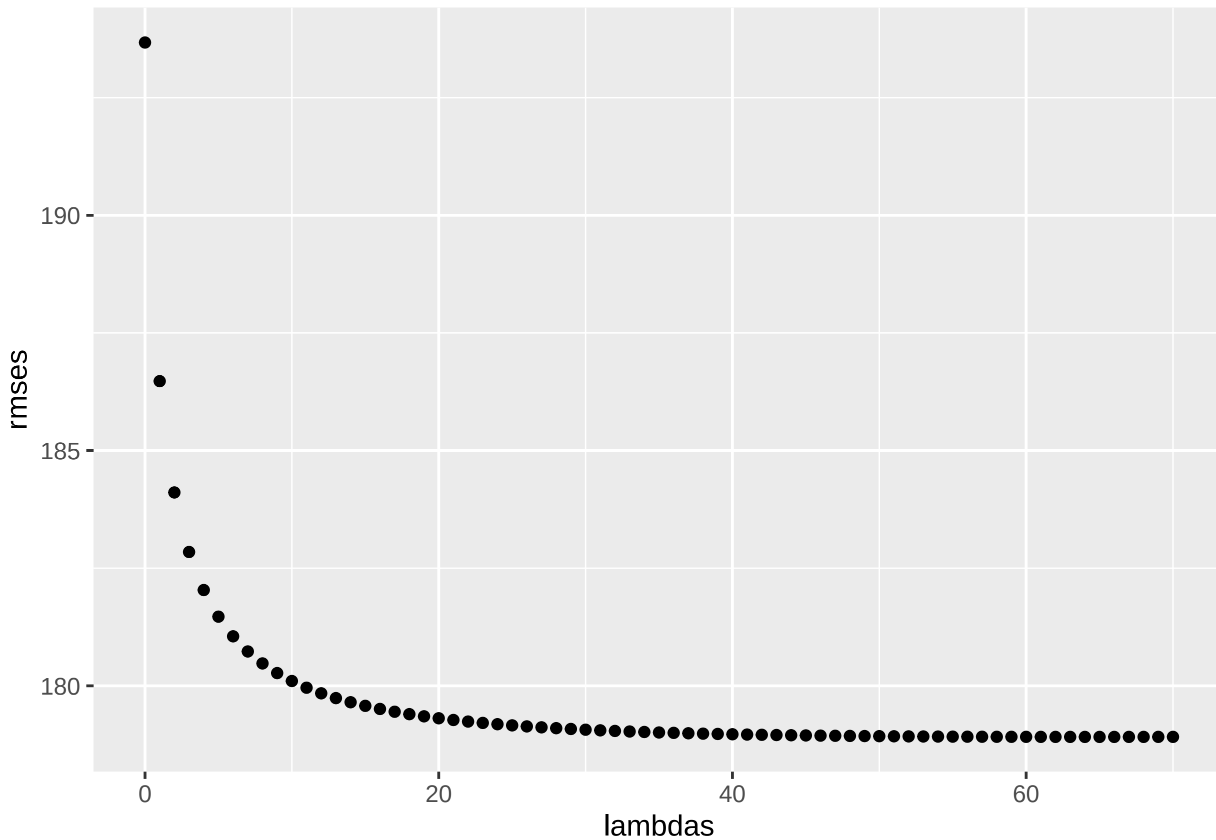



Figure 2: Lambda Versus RMSE plot for validation

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

```r
#Regularised user bias term
b_u <- combined_model %>% left_join(b_i, by='name') %>%
  group_by(userid) %>% summarise(b_u=sum(hours_played-b_i-mu)/(n()+lambda))
```

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

```r
# Calculating final model on validation set:

final_model <- combined_validate %>%
  left_join(b_i, by='name') %>%
  left_join(b_u, by='userid') %>%
  mutate(b_i = coalesce(b_i, 0), b_u = coalesce(b_u, 0), pred=mu+b_i+b_u) %>% # replace missing values
  pull(pred)

(final_rmse <- RMSE(combined_validate$hours_played, final_model))
```

```
## [1] 178.9024
```

Hence these are the models that we have explored and their respective RMSEs thus far.

```r
methods <- c('Model 0 - Just the average', 'Model 1 - added Name bias', 'Model 2 - added User bias',
             'Model 3 - added Genre bias', 'Model 4 - added User Rating bias',
             'Model 5 - added Price bias to Model 3', 'Model 6a - Reverse Bias - added User bias',
             'Model 6b - Reverse Bias - added Name bias', 'Model 7 - Regularised model', 'Final Model -
rmses <- c(naive_rmse, name_bias_rmse, user_name_bias_rmse,
           genre_user_name_bias_rmse, rating_genre_user_name_bias_rmse,
           price_genre_user_name_bias_rmse, rb_user_bias_rmse,
           rb_name_user_bias_rmse, regularised_user_name_rmse, final_rmse)

(model_evaluations <- tibble(method=methods, RMSE=rmses))
```

```
## # A tibble: 10 x 2
##    method                                      RMSE
##    <chr>                                      <dbl>
##  1 Model 0 - Just the average                  225.
##  2 Model 1 - added Name bias                   213.
##  3 Model 2 - added User bias                   230.
##  4 Model 3 - added Genre bias                  228.
##  5 Model 4 - added User Rating bias            228.
##  6 Model 5 - added Price bias to Model 3       228.
##  7 Model 6a - Reverse Bias - added User bias   242.
##  8 Model 6b - Reverse Bias - added Name bias   235.
##  9 Model 7 - Regularised model                 213.
## 10 Final Model - Regularised Model on Validation Set  179.
```

The Regularisation & Cross-Validation method performed with a RMSE of 178.9024033 on the "combined_validate" validation set. This was lower than that produced by training the model.

# CONCLUSION

Exploring the datasets of Steam's video game catalogue and Steam users' purchase and play behaviours has been really interesting. It was really interesting for me to see that a sizeable portion of users analysed bought games without playing them. Also, many factors do not seem to impact the purchase and play behaviour of the users apart from the specific game titles and individual user preferences.

I thoroughly enjoyed analysing the combined dataset. It was really interesting to use visualisations to convey the information. This greatly improves how well we understand the data's variability and relations to other variables in the data.

The final performance of our Regularisation & Cross-Validation model was 212.8959952 (on test set) versus 178.9024033 (on validation set) shows relatively good stability of the prediction precision over unknown data.

I am satisfied with the result and I hope to check other more computationally intensive methods and their performances in the future.

Also, if there are more detailed datasets with additional variables to test, it could potentially make building the models more interesting. too.

# REFERENCES:


- Steam (service) | Wikipedia. Extracted on 02 April 2021: https://en.wikipedia.org/wiki/Steam_(service)
- About | Steam: https://store.steampowered.com/about/
- Steam Store Games (Clean dataset), gathered around May 2019: https://www.kaggle.com/nikdavis/steam-store-games?select=steam.csv
- Steam Video Games dataset ("Recommend video games from 200k interactions user interactions.", gathered around 2017): https://www.kaggle.com/tamber/steam-video-games/download