## Install Packages

install.packages('plyr', repos = "http://cran.us.r-project.org") install.packages('dplyr', repos = "http://cran.us.r-project.org") install.packages('tidyverse', repos = "http://cran.us.r-project.org") install.packages('caret', repos = "http://cran.us.r-project.org") install.packages('data.table', repos = "http://cran.us.r-project.org") install.packages('splitstackshape', repos = "http://cran.us.r-project.org")

## Load necessary packages

library(plyr) library(dplyr) library(tidyverse) library(caret) library(data.table) library(splitstackshape)

## MovieLens 10M dataset:

dl <- tempfile() download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings <- fread(text = gsub("::", ", readLines(unzip(dl,"ml-10M100K/ratings.dat"))), col.names = c("userId","movieId","rating","timestamp"))

movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\::", 3) colnames(movies) <- c("movieId", "title", "genres") movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId), title = as.character(title), genres = as.character(genres))

movielens <- left_join(ratings, movies, by = "movieId")

## Validation set will be 10% of MovieLens data

set.seed(1) # if using R 3.5 or earlier, use `set.seed(1)` instead test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE) edx <- movielens[-test_index,] temp <- movielens[test_index,]

## Make sure userId and movieId in validation set are also in edx set

validation <- temp %>% semi_join(edx, by = "movieId") %>% semi_join(edx, by = "userId")

## Add rows removed from validation set back into edx set

removed <- anti_join(temp, validation) edx <- rbind(edx, removed)

rm(dl, ratings, movies, test_index, temp, movielens, removed)

## INTRODUCTION

**The MovieLens 10M dataset looks at 10 million movie ratings applied to 10,000 movies.**

**Our goal for this project will be to predict the movie rating to the nearest .5 rating from the 6 features in this dataset.**

**In order to accurately predict the rating, we will first look at some of the data and then begin some feature engineering.**

## METHODS/ANALYSIS

**In this section, we will perform data exploration, feature engineering and some clean up.**

**To understand what our dataset looks like, we will first look at the existing structure of the overall dataset and distribution of some possible features.**

## Data Exploration

## Checkout data structure

head(edx) # unable to interpret and analyze timestamp,title and genres data as is. We will need to do some feature engineering.

## Check for any NA values

sapply(edx, function(x) sum(is.na(x))) # None

## Check Distribution of y variable

plot(table(edx$rating)) # most values at 4,3,5. This is a left skewed dataset

## Check Distribution of some possible features

summary(edx %>% group_by(userId) %>% summarize(ct=n())) # number of ratings per user summary(edx %>% group_by(movieId) %>% summarize(ct=n())) # number of ratings per movie edx %>% mutate(year = year(as.Date(as.POSIXct(edx$timestamp,origin="1970-01-01"))))%>% group_by(year) %>% summarize(ct=n()) %>% arrange(desc(ct)) # number of ratings by year edx %>% mutate(month = month(as.Date(as.POSIXct(edx$timestamp,origin="1970-01-01"))))%>% group_by(month) %>% summarize(ct=n()) %>% arrange(desc(ct)) # number of ratings by month genres_split <- cSplit(edx,"genres","|",direction="long") # split out movie genres genres_split %>% group_by(genres_split$genres) %>% summarize(ct=n()) %>% arrange(desc(ct)) # look at number of ratings by movie genre summary(edx %>% group_by(title) %>% summarize(ct = n()) %>% arrange(desc(ct))) # look at distribution of number of ratings per movie title

## Feature Engineering

## Change timestamp from unix to normal timestamp and create date features

data <- edx %>% mutate(rating_dt = as.Date(as.POSIXct(edx$timestamp,origin="1970-01-01"))) # convert timestamp to standard date format data <- data %>% mutate(year = year(rating_dt)) # create year feature data <- data %>% mutate(month = month(rating_dt)) # create month feature data <- data %>% mutate(day = as.numeric(format(as.Date(data$rating_dt, format='%Y-%m-%d'), format="%d"))) # create day feature

test_data <- validation %>% mutate(rating_dt = as.Date(as.POSIXct(validation$timestamp,origin="1970-01-01"))) # convert timestamp to standard date format in test set test_data <- test_data %>% mutate(year = year(rating_dt)) # create year feature in test set test_data <- test_data %>% mutate(month = month(rating_dt)) # create month feature in test set test_data <- test_data %>% mutate(day = as.numeric(format(as.Date(test_data$rating_dt, format='%Y-%m-%d'), format="%d"))) # create day feature in test set

## Add two features: an average rating and number of ratings by user

avg_user_rating <- data %>% group_by(userId) %>% summarize(avg_user_rt=mean(rating)) # avg_user_rating feature data <- data %>% inner_join(avg_user_rating) # join avg user rating to original data num_user_rt <- data %>% group_by(userId) %>% summarize(user_rt=n()) # number of ratings per user feature data <- data %>% inner_join(num_user_rt) # join number of ratings per user with original data

avg_user_rating_test <- test_data %>% group_by(userId) %>% summarize(avg_user_rt=mean(rating)) # avg user rating feature in test set test_data <- test_data %>% inner_join(avg_user_rating_test) # join avg user rating to original data in test set num_user_rt_test <- test_data %>% group_by(userId) %>% summarize(user_rt=n()) # number of ratings per user feature in test set test_data <- test_data %>% inner_join(num_user_rt_test) # join number of ratings per user with the original data

## Add two features: an average rating and number of ratings by movie

avg_movie_rating <- data %>% group_by(movieId) %>% summarize(avg_movie_rt=mean(rating)) # avg movie rating data <- data %>% inner_join(avg_movie_rating) # join avg movie rating with original data avg_movie_rating_test <- test_data %>% group_by(movieId) %>% summarize(avg_movie_rt=mean(rating)) # avg movie rating in test set test_data <- test_data %>% inner_join(avg_movie_rating_test) # join avg movie rating with original data in test set

num_movie_rt <- data %>% group_by(movieId) %>% summarize(movie_rt=n()) # number of ratings per movie data <- data %>% inner_join(num_movie_rt) # join number of ratings per movie with original data num_movie_rt_test <- test_data %>% group_by(movieId) %>% summarize(movie_rt=n()) # number of ratings per movie in test set test_data <- test_data %>% inner_join(num_movie_rt_test) # join number of ratings per movie with original data in test set

## Add Movie Year as a feature

data <- data %>% mutate(movie_yr = as.numeric(str_extract(data$title,"(\\d\\d\\d\\d)"))) # extract movie year # data <- data %>% mutate(diff = year - movie_yr) look at difference between movie year and rating year test_data <- test_data %>% mutate(movie_yr = as.numeric(str_extract(test_data$title,"(\d\d\d\d)"))) # extract movie year in test set # test_data <- test_data %>% mutate(diff = year - movie_yr) look at difference between movie year and rating year in test set

## Remove the old timestamp, rating_dt and dummy columns from the dataset

data <- subset(data, select = -c(timestamp,rating_dt,title,day)) test_data <- subset(test_data, select = -c(timestamp,rating_dt,title,day))

## Determine how many unique movie genres there are

```
genres_split <- cSplit(data,"genres","|",direction="long") unique_genres <-
unique(genres_split$genres) # 20 unique movie genres
```

## Add features that show a correlation >= the absolute value of .05 with the rating variable (originally ran for every movie genre)

```
data <- data %>% mutate(Action = ifelse(str_extract(data$genres,"Action")=="Action",1,0))
data <- data %>% mutate(Crime = ifelse(str_extract(data$genres,"Crime")=="Crime",1,0))
data <- data %>% mutate(Drama =
ifelse(str_extract(data$genres,"Drama")=="Drama",1,0)) data <- data %>%
mutate(Adventure = ifelse(str_extract(data$genres,"Adventure")=="Adventure",1,0)) data
<- data %>% mutate(FilmNoir = ifelse(str_extract(data$genres,"Film-Noir")=="Film-
Noir",1,0)) data <- data %>% mutate(Horror =
ifelse(str_extract(data$genres,"Horror")=="Horror",1,0))
```

```
test_data <- test_data %>% mutate(Action =
ifelse(str_extract(test_data$genres,"Action")=="Action",1,0)) test_data <- test_data %>%
mutate(Crime = ifelse(str_extract(test_data$genres,"Crime")=="Crime",1,0)) test_data <-
test_data %>% mutate(Drama =
ifelse(str_extract(test_data$genres,"Drama")=="Drama",1,0)) test_data <- test_data %>%
mutate(Adventure = ifelse(str_extract(test_data$genres,"Adventure")=="Adventure",1,0))
test_data <- test_data %>% mutate(FilmNoir = ifelse(str_extract(test_data$genres,"Film-
Noir")=="Film-Noir",1,0)) test_data <- test_data %>% mutate(Horror =
ifelse(str_extract(test_data$genres,"Horror")=="Horror",1,0))
```

### Clean Up the training data

```
data <- subset(data, select=-c(genres)) # Remove the genres column data[is.na(data)] <- 0
# Replace NA values with 0 sapply(data, function(x) sum(is.na(x))) # double check all NA
values are gone
```

```
test_data <- subset(test_data, select=-c(genres)) # Remove the genres column
test_data[is.na(test_data)] <- 0 # Replace NA values with 0 sapply(test_data, function(x)
sum(is.na(x))) # double check all NA values are gone
```

## View Correlation Matrix to determine correlation of variables

```
cor_matrix <- round(cor(data),2) # Originally had year of movie, year of rating and
difference in years between the 2. Ended up remove the difference between years due to
multicollinearity between the movie_yr and diff. cor_matrix
```

## Create Train and Test Set

test_index <- createDataPartition(y = data$rating, times=1, p=0.1, list=FALSE) # 90% train set, 10% test set test_set <- data[test_index,] train_set <- data[-test_index,]

## Seeing that we are trying to predict a continuous variable (i.e. not 1 or 0), we will use a linear regression model.

### Fit the Model

## Fit a Linear Regression model, make predictions based off the validation data and calculate RMSE on the model

lr_model <- lm(rating~.,data=train_set) # fit the model summary(lr_model) # .326 Adjusted R squared. Determine statistical significane of each variable p2 <- round_any(predict(lr_model,test_set),.5) # Make predictions on the movie rating to the nearest .5 RMSE(p2,test_set$rating) # Print rmse between the predicted value and actual values

### Test the model on the validation set

p2 <- round_any(predict(lr_model,test_data),.5) # Make predictions on the movie rating to the nearest .5 RMSE(p2,test_data$rating) # Print rmse between the predicted value and actual values

## Results

Our Final Model has a RMSE = .857. This means that our standard deviation relative to our linear regression line is .857.

Given that we are predicting ratings to the nearest .5, this is a good RMSE for this dataset.

## Conclusion

We were able to predict movie rating with a RMSE of .857 on our test data.

If we were going to improve this in the future, we could have looked at doing more data engineering to better predict.

Some possible techniques we could have looked into would be: multiplying features together to create a new feature, completing stepwise (forward and backward) regression and extracted more insights from the movie title column.