IMDb Machine Learning Project

2024-03-22

Introduction

I have always had a great interest in movies and television shows. I do my best to keep up with new releases and I try to stay up to date with what the critics and audience members are saying. With a site like IMDb that complies all essentials information about a movie or TV show, it is naturally a hit with movie-lovers all over. IMDb lists a movie or TV show's cast, crew, ratings from audience members and critics alike, and much more. Considering IMDb has been around since 1990 and is still an immensely popular website, it's apparent the impact it has had on how media is consumed.

With this mind, I explore the data provided within IMDb further. From my project, I chose a data set that contains various different information about thousands of movies that are listed on IMDb. I wanted to see which variables affect the IMDb audience ratings the most. Going in, I assumed variables such as genre and meta critic rating would influence the audience rating the most. However, by doing this project, I was able to get insight on nuance I would not have not known about otherwise.

Data Source

Name: IMDb Movie Dataset

Author: The author's name is not provided, but the owner of the dataset is listed as kianindeed on Kaggle

Source: Kaggle

Date: December 15, 2023

Link: https://www.kaggle.com/datasets/kianindeed/imdb-movie-dataset-dec-2023 (https://www.kaggle.com/datasets/kianindeed/imdb-movie-dataset-dec-2023)

Variables: Movie Name, Audience Rating, Votes, Meta Score, Genre, PG-Rating,Year, Duration, Cast, Directors (I will delve into the variables more in depth later on)

Codebook

movie_name: the name of the movie (data type = character)

rating: the rating on IMDb by the users, out of 10 (data type = numerical)

votes: the number of people voting on the ratings (data type = numerical)

meta_score: score from a different rating website called metacritic, out of 100 (data type = numerical)

genre: the genre of the movie, such as comedy, drama, romance, etc (data type = factor, originally character)

pg_rating: what the movie is rated from a scale of G (family friendly) to R (not suitable for minors). Also contains other rating types such as 13+ or "Unrated" for movies that did not receive a standard pg-rating. (data type = factor, originally character)

year: what year the movie was released

duration: how long the movie is

cast: the actors in the movie

director: who directed the movie

Exploratory Data Analysis (EDA)

Tidying the Data

```
imdb_data <- read_csv("imdb_data.csv", show_col_types = FALSE) %>%
  clean_names() %>%
  dplyr::select(-c(x1)) %>%
  rename(movie_name = moive_name) %>%
  na.omit(imdb_data)


imdb_data$genre <- str_replace(word(imdb_data$genre, 1), ",", "")


imdb_data <- imdb_data %>%
  mutate(pg_rating = factor(pg_rating)) %>%
  mutate(genre = factor(genre)) %>%
  mutate(genre = fct_lump_lowfreq(genre)) %>%
  mutate(pg_rating = fct_lump_lowfreq(pg_rating))

imdb_data
```

```
## # A tibble: 1,784 × 10
##    movie_name      rating  votes meta_score genre pg_rating  year duration cast
##    <chr>            <dbl>  <dbl>      <dbl> <fct> <fct>     <dbl> <chr>    <chr>
##  1 Leave the Worl…    6.5  90000         67 Drama R          2023 2h 18m   Juli…
##  2 Wonka              7.4  24000         66 Adve… Other      2023 1h 56m   Timo…
##  3 Poor Things        8.5   6700         86 Come… R          2023 2h 21m   Emma…
##  4 Killers of the…    7.8 128000         89 Crime R          2023 3h 26m   Leon…
##  5 May December       7    21000         85 Come… R          2023 1h 57m   Nata…
##  6 The Hunger Gam…    7.1  56000         54 Acti… PG-13      2023 2h 37m   Rach…
##  7 Napoleon           6.6  66000         64 Acti… R          2023 2h 38m   Joaq…
##  8 Oppenheimer        8.4 553000         89 Biog… R          2023 3h       Cill…
##  9 Love Actually      7.6 517000         55 Come… R          2003 2h 15m   Hugh…
## 10 Candy Cane Lane    5.6  13000         47 Come… Other      2023 1h 57m   Eddi…
## # ℹ 1,774 more rows
## # ℹ 1 more variable: director <chr>
```

Missing Data

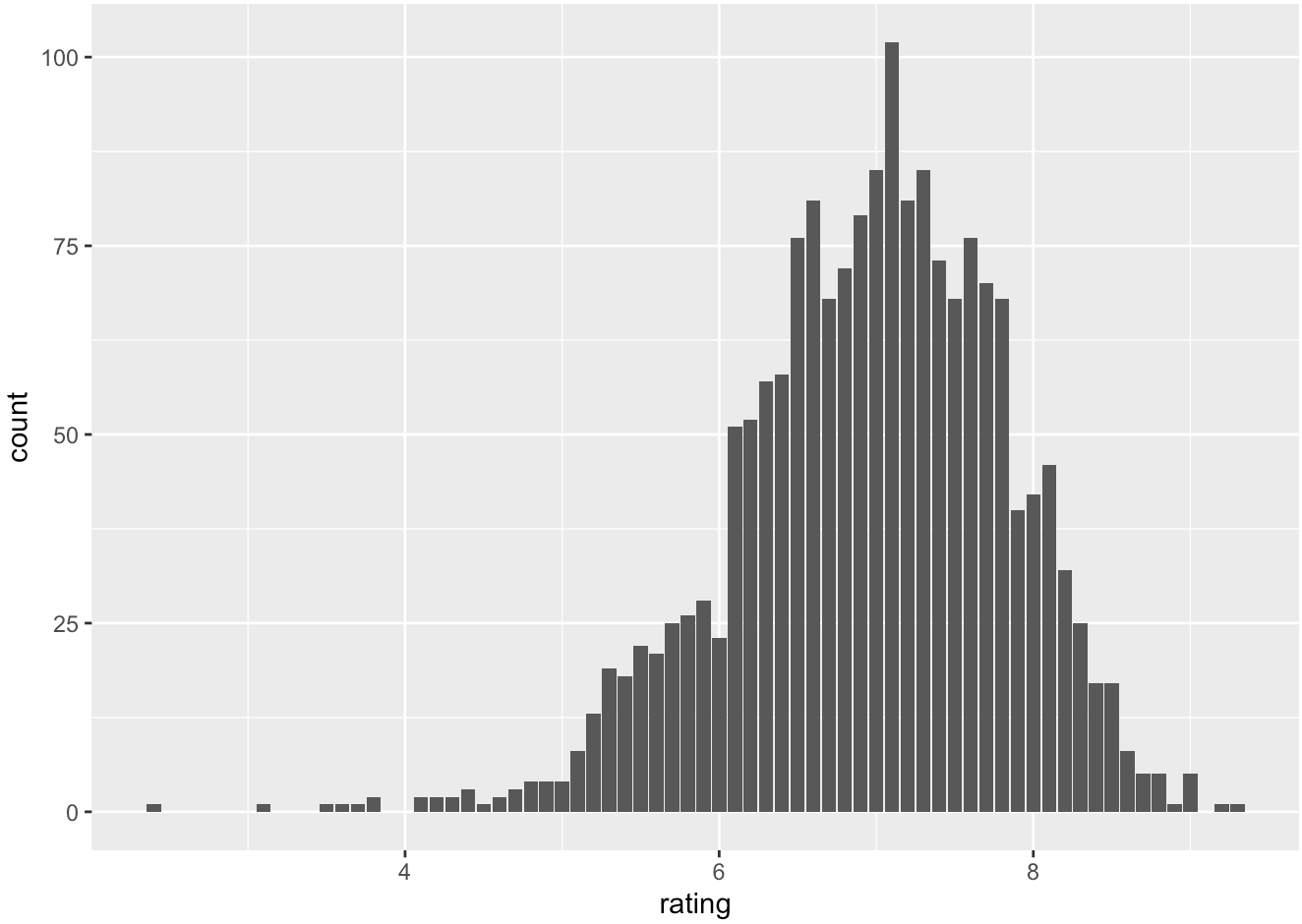
On an initial look through of the data set, I noticed there was data missing on a few of the variables, particularly for the ratings, meta score, and pg-rating. After testing the data on some machine learning models such as k-nearest neighbors, I realized that the missing data will prove to be an issue. Missing data consisted of a small

percentage of the data set. So, I came to the conclusion that it would be most suitable for me to remove the rows containing missing data. Doing so allowed for a much smoother process during the rest of the project.

Plots and Tables

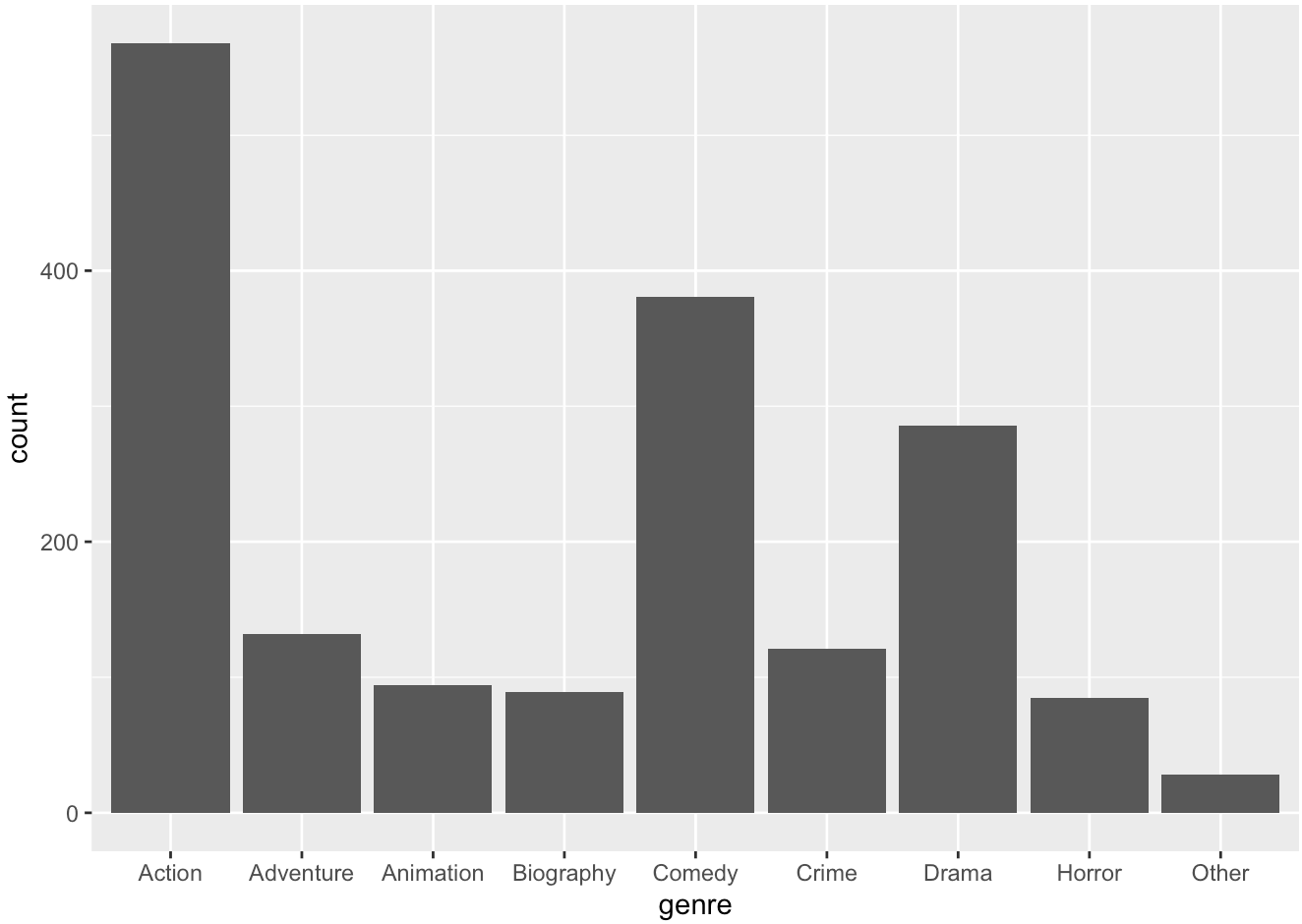
Bar Chart

```
rating_bar <- ggplot(imdb_data, aes(rating)) +
  geom_bar()
rating_bar
```



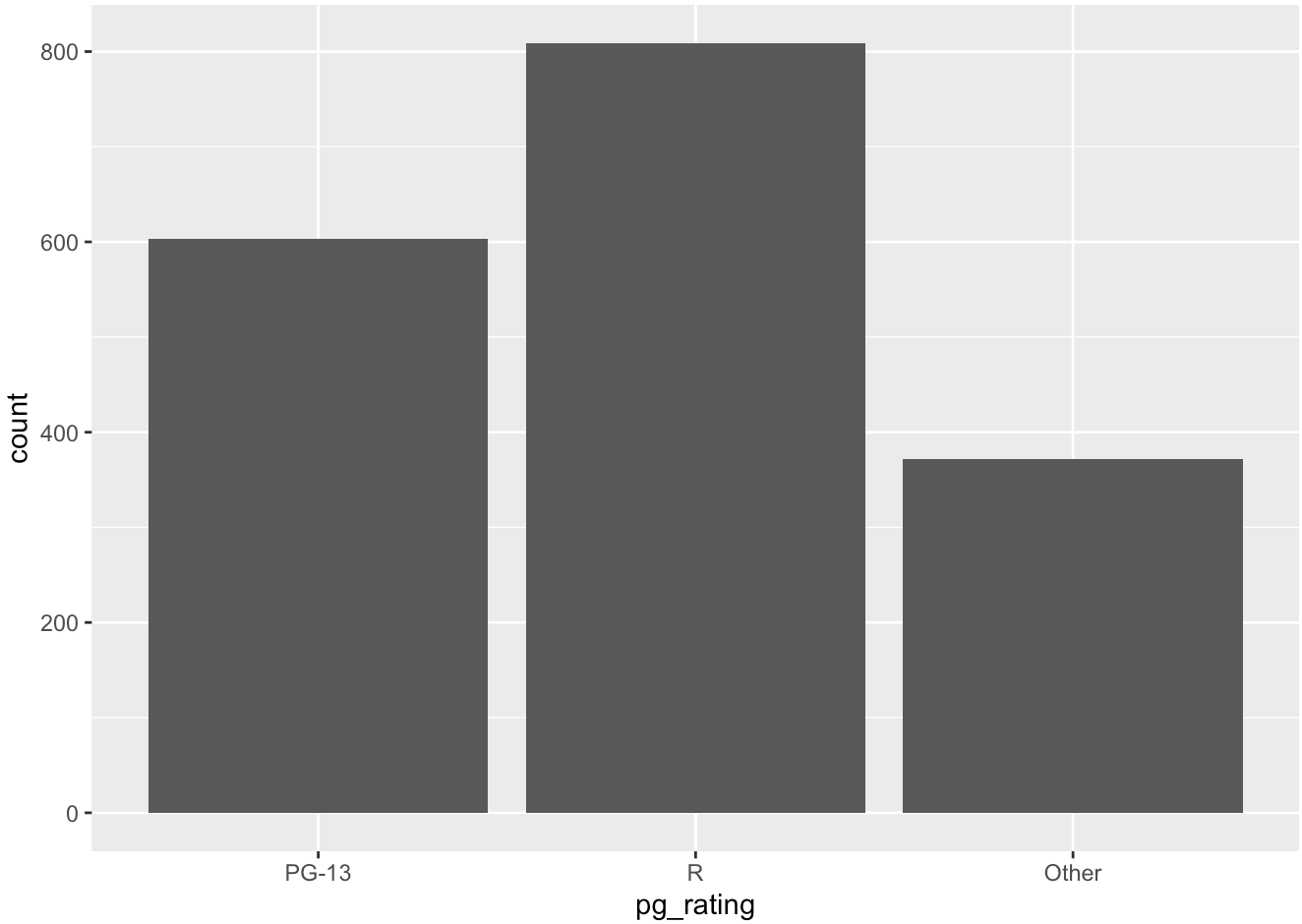
The graph above is a bar chart showing the count of the various ratings in the data set. From the graph, it is pretty clear that the majority of ratings are between 6 and 8 (out of 10). With this in mind, I wonder if this will affect the rest of the project.

```
genre_bar <- ggplot(imdb_data, aes(genre)) +
  geom_bar()
genre_bar
```



The chart above shows how many movies of each genre is represented in the data. Most of the movies are of the action genre, while there are also quite a few comedy and drama movies.

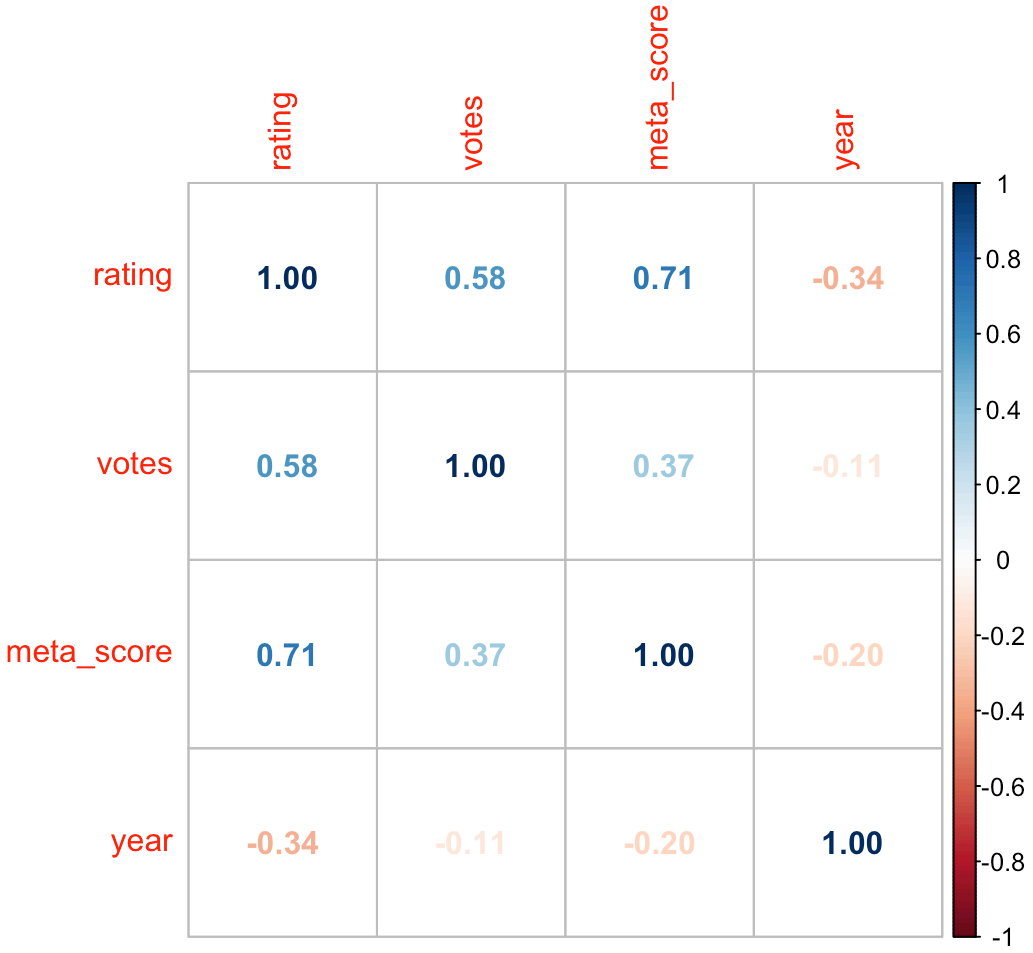
```
pg_rating_bar <- ggplot(imdb_data, aes(pg_rating)) +
  geom_bar()
pg_rating_bar
```



The chart above shows that the majority of the movies represented within the data set are R-rated. The next most popular category is PG-13. The least popular category is “Other” that consists of all other pg-ratings such as Unrated and NC-17.

Correlation Matrix

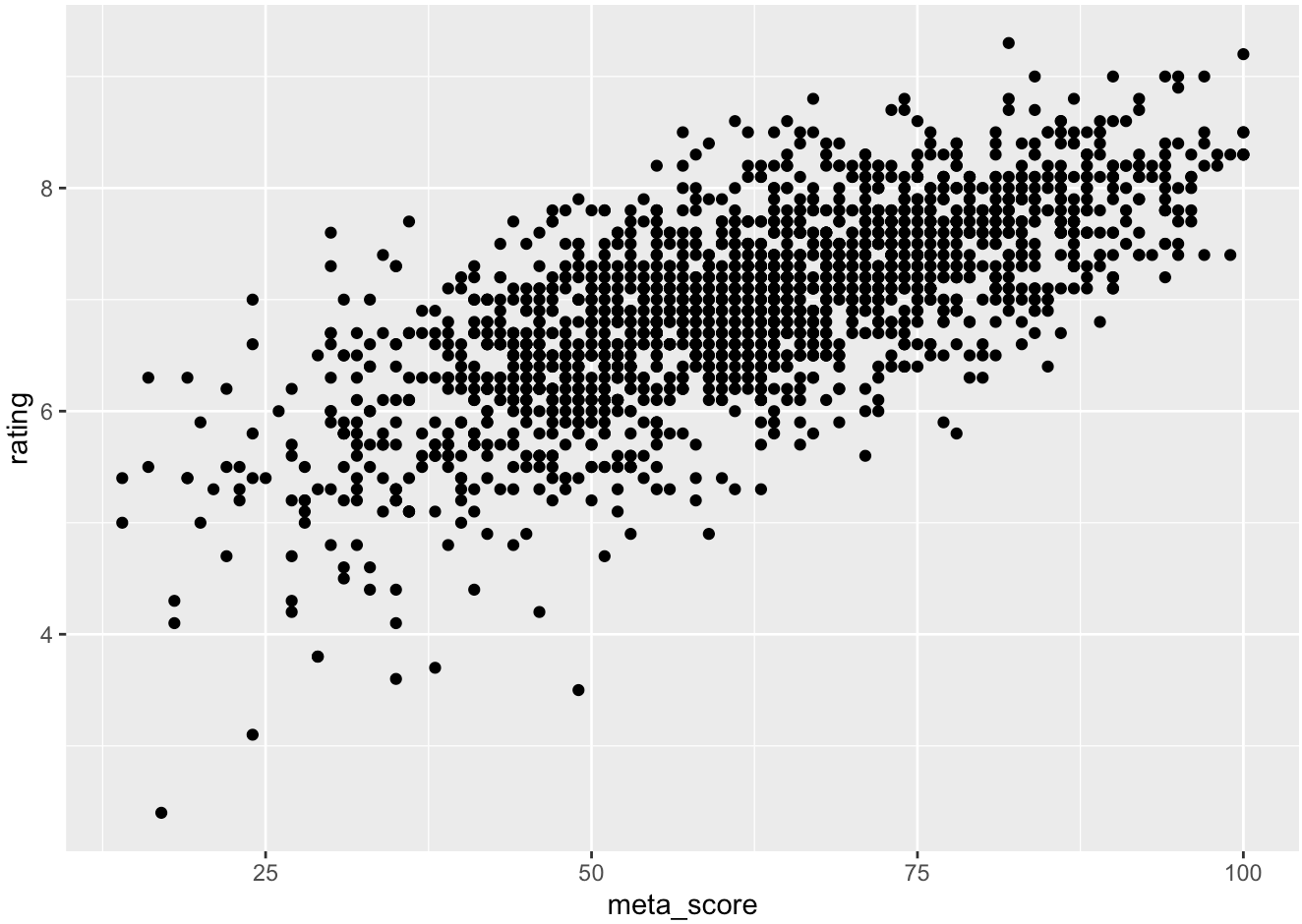
```
imdb_data1 <- subset(imdb_data, select = -c(1,5,6,8,9,10))
M <- cor(imdb_data1)
corrplot(M, method = "number")
```



The correlation matrix above shows that there is a moderately strong positive correlation between meta_score and rating with a coefficient of 0.71. The next strongest positive correlation is between votes and rating with a coefficient of 0.58. There is a slight negative correlation between year and rating with a coefficient of -0.34.

Scatter Plot

```
ggplot(imdb_data, aes(x=meta_score, y=rating)) + geom_point()
```



The scatter plot above showcases the relationship between meta_score and rating. From seeing the correlation matrix and realizing there is strong positive correlation between the two, I wanted to see what the scatter plot would look like. As seen above, there is a positive relationship between the two meaning, in general, the higher the meta score is the higher the rating is.

EDA Conclusion

For my exploratory data analysis, I first started off by tidying up my data. This allowed for the data set to be much easier to read through and work with. I also dealt with the missing data found within the original data set. Lastly I plotted various different graphs and charts to find out more about the variables. With what I found, I am most interested in exploring the relationships between ratings, meta score, pg-rating, genre, and votes. More specifically, I am curious to see how the latter four predictor variables affect the outcome variable rating.

Data Split

```
set.seed(3435)

imdb_split <- initial_split(imdb_data, prop = 0.80,
                                strata = rating)
imdb_train <- training(imdb_split)
imdb_test <- testing(imdb_split)
```

In this step, I split the data into training and testing data. Before I could split the data, I set the seed to ensure that my results were reproducible. I split the data so that 80% of the original data set would be used in the training data set and 20% of the data set would be used in the testing data set. I also stratified the data set on my outcome variable, rating. Stratifying sampling ensures that the training and testing data have a similar proportion of the outcome variable.

Cross-Validation

```
imdb_folds <- vfold_cv(imdb_train, v = 5, strata = rating)
```

K-fold cross-validation is a technique for evaluating predictive models where there are k subsets or folds. The model is then trained and evaluated k times one for each of the different folds. Then the metrics are averaged together to give a general estimate of the model's performance. It helps reduce the risk of over fitting the data and gives a more accurate image for the entire data. I also stratified the data on the outcome variable, rating. Similarly to stratifying the data when splitting, this allows for there to be an equal proportion of rating in each fold. Later on in the project, I use imdb_folds in order to fit and tune my models.

Recipe

```
imdb_recipe <- recipe(rating ~ votes + meta_score + pg_rating + genre, data = imdb_trai
n) %>%
  #step_unknown(all_nominal_predictors(), new_level = "NA") %>%
  step_dummy(all_nominal_predictors())  %>%
  step_zv() %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors())

prep(imdb_recipe) %>%
  bake(new_data = imdb_train) %>%
  kable() %>%
  kable_styling(full_width = F) %>%
  scroll_box(width = "100%", height = "200px")
```

votes	meta_score	rating	pg_rating_R	pg_rating_Other	genre_Adventure	genre_Animation
-0.8647908	-0.9243154	5.6	-0.9046012	1.9699843	-0.2745212	-0.2355324
-0.8987738	-1.8276832	6.3	-0.9046012	-0.5072623	-0.2745212	-0.2355324
-0.8556062	-1.5265606	5.6	-0.9046012	1.9699843	-0.2745212	-0.2355324
-0.0473629	-0.9845400	6.3	-0.9046012	1.9699843	-0.2745212	-0.2355324

In this step, I set up the recipe for the rest of the project. Since the purpose of the project is to see if certain variables can predict audience ratings on IMDb, that is my outcome variable. I chose votes, meta score, genre, and pg-rating to as my predictor variables. I have noticed over time that ratings sometimes can depend on the

genre and pg-rating the movie is given so I thought they would be good variables to test on. I chose votes because I wanted to test whether a higher number of votes leads to a higher rating. Lastly, I chose meta score because that, at first glance, seems like it would be the best predictor for audience rating.

Models and Tuning

Models

```
# linear regression model
lm_mod_imdb <- linear_reg() %>%
  set_engine("lm")

# k-nearest neighbors model
knn_mod_imdb <- nearest_neighbor(neighbors = tune()) %>%
  set_mode("regression") %>%
  set_engine("kknn")

# random forest model
rf_mod_imdb <- rand_forest(mtry = tune(),
                           trees = tune(),
                           min_n = tune()) %>%
  set_engine("ranger", importance = "impurity") %>%
  set_mode("regression")

# gradient boosted tree model
gb_mod_imdb <- decision_tree(cost_complexity = tune()) %>%
  set_engine("rpart") %>%
  set_mode("regression")
```

The four models that I ended up using are linear regression, k-nearest neighbors, random forest, and gradient boosted trees. Linear regression is the simplest model of the four and it describes the relationship between the outcome and predicted variables. K-nearest neighbors is a non-parametric supervised learning model. It calculates the distance between the number of neighbors provided and the input data. Then it calculates the average between the distances to predict the input value. The random forest model takes into consideration the outcome of the various tree to make prediction. It contains the hyperarameters mtry, trees, and min_n. The hyperparameter mtry represents an integer for the number of predictors that will be randomly sampled at each split when creating the tree models. Tree represents an integer for the number of trees contained in the ensemble. Min_n represents an integer for the minimum number of data points in a node that are required for the node to be split further. The gradient boosted tree model combines several simple models to make a stronger and more accurate final prediction.

Workflows

```
# linear regression workflow
lm_wkflow_imdb <- workflow() %>%
  add_model(lm_mod_imdb) %>%
  add_recipe(imdb_recipe)

# k-nearest neighbors workflow
knn_wkflow_imdb <- workflow() %>%
  add_model(knn_mod_imdb) %>%
  add_recipe(imdb_recipe)

# random forest workflow
rf_wkflow_imdb <- workflow() %>%
  add_model(rf_mod_imdb) %>%
  add_recipe(imdb_recipe)

# gradient boosted workflow
gb_wkflow_imdb <- workflow() %>%
  add_model(gb_mod_imdb) %>%
  add_recipe(imdb_recipe)
```

In this step, I have set up workflows for the linear regression, k-nearest neighbors, random forest, and gradient boosted tree models. I added each of the models to their respective workflows and added the imdb_recipe to each of them

Grids

```
# no grids for linear regression

# k-nearest neighbors grid
neighbors_grid <- grid_regular(neighbors(range = c(1, 5)), levels = 3)

# random forest grid
rf_grid <- grid_regular(mtry(range = c(1, 4)),
                        trees(range = c(200, 600)),
                        min_n(range = c(10, 20)),
                        levels = 5)

# gradient boosted grid
gb_grid <- grid_regular(cost_complexity(range = c(-3, -1)), levels = 10)
```

In this step, I set up grids for three out of four of my models. The linear regression model does not require a grid as it does not need to be tuned. For k-nearest neighbors, the grid consists of neighbors ranging from 1 through 5 and 3 levels. The random forest grid consists of the mtry value of 1-4, tree value of 200-600, and min_n of 10-20, with 5 levels each. The gradient boosted grid consists of a range of -3 to -1 with 10 levels.

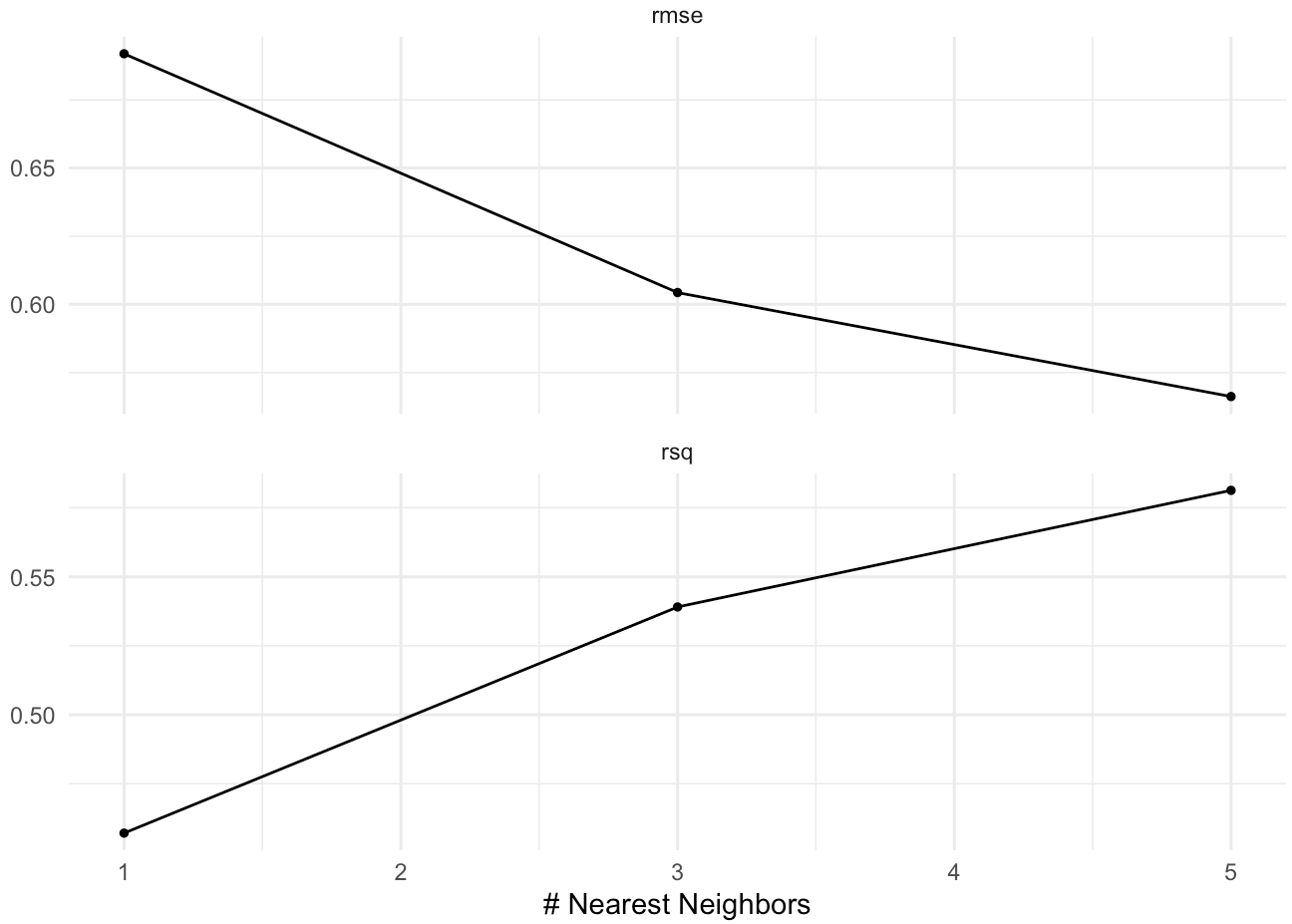
Tuning

```
# linear regression tuning
fit_lm_imdb <- lm_wkflow_imdb %>%
  fit_resamples(resamples = imdb_folds)

# k-nearest neighbors tuning
tune_knn_imdb <- tune_grid(
  object = knn_wkflow_imdb,
  resamples = imdb_folds,
  grid = neighbors_grid
)

# random forest tuning
tune_rf_imdb <- tune_grid(
  rf_wkflow_imdb,
  resamples = imdb_folds,
  grid = rf_grid
)

# gradient boosted tuning
tune_gb_imdb <- tune_grid(
  gb_wkflow_imdb,
  resamples = imdb_folds,
  grid = gb_grid,
  control = control_grid(verbose = TRUE)
)
```

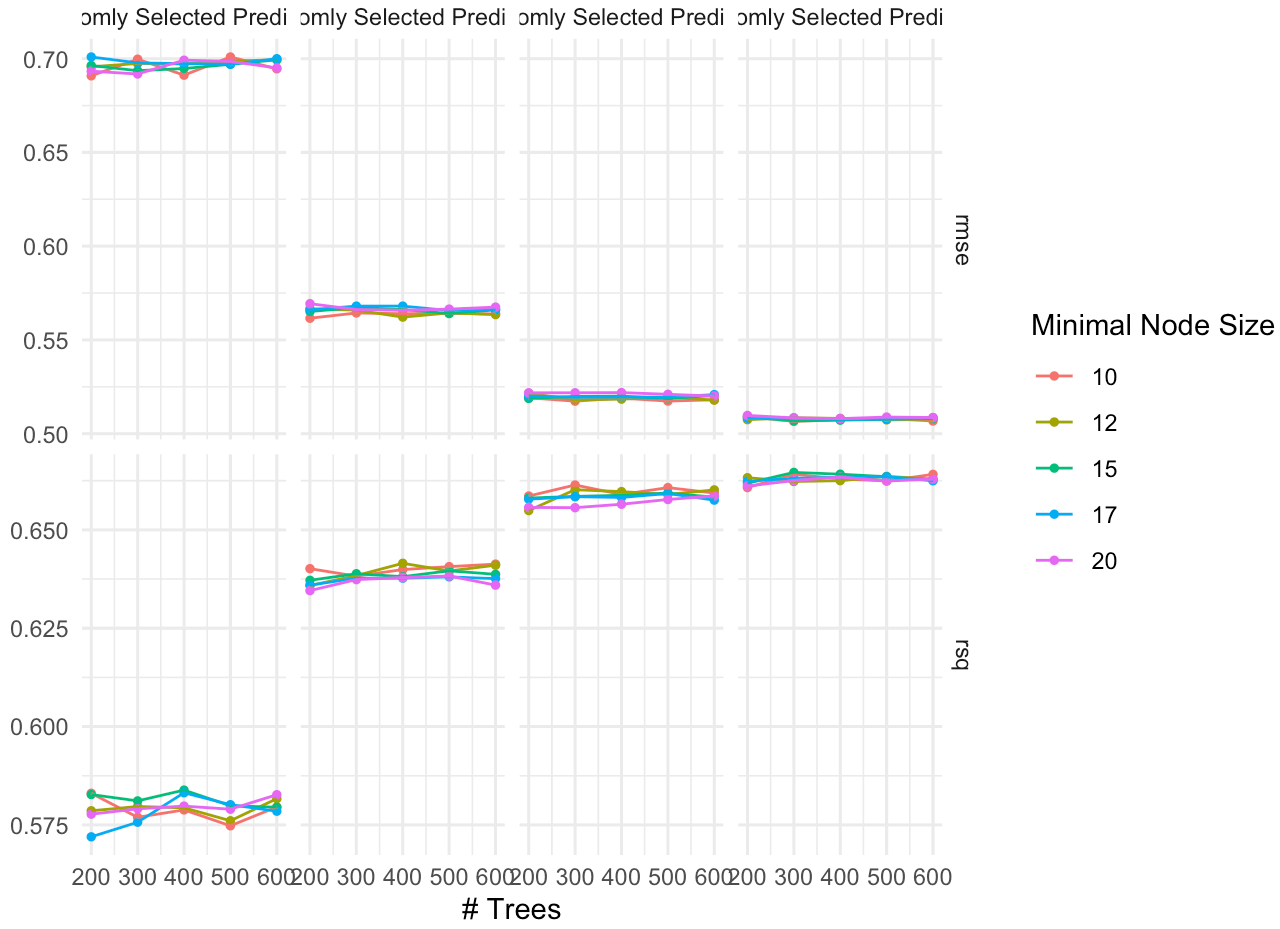
In this step, I have fitted the linear regression model and tuned the k-nearest neighbors, random forest, and gradient boosted tree models to imdb_folds.

Tuning (And Fitting) Results

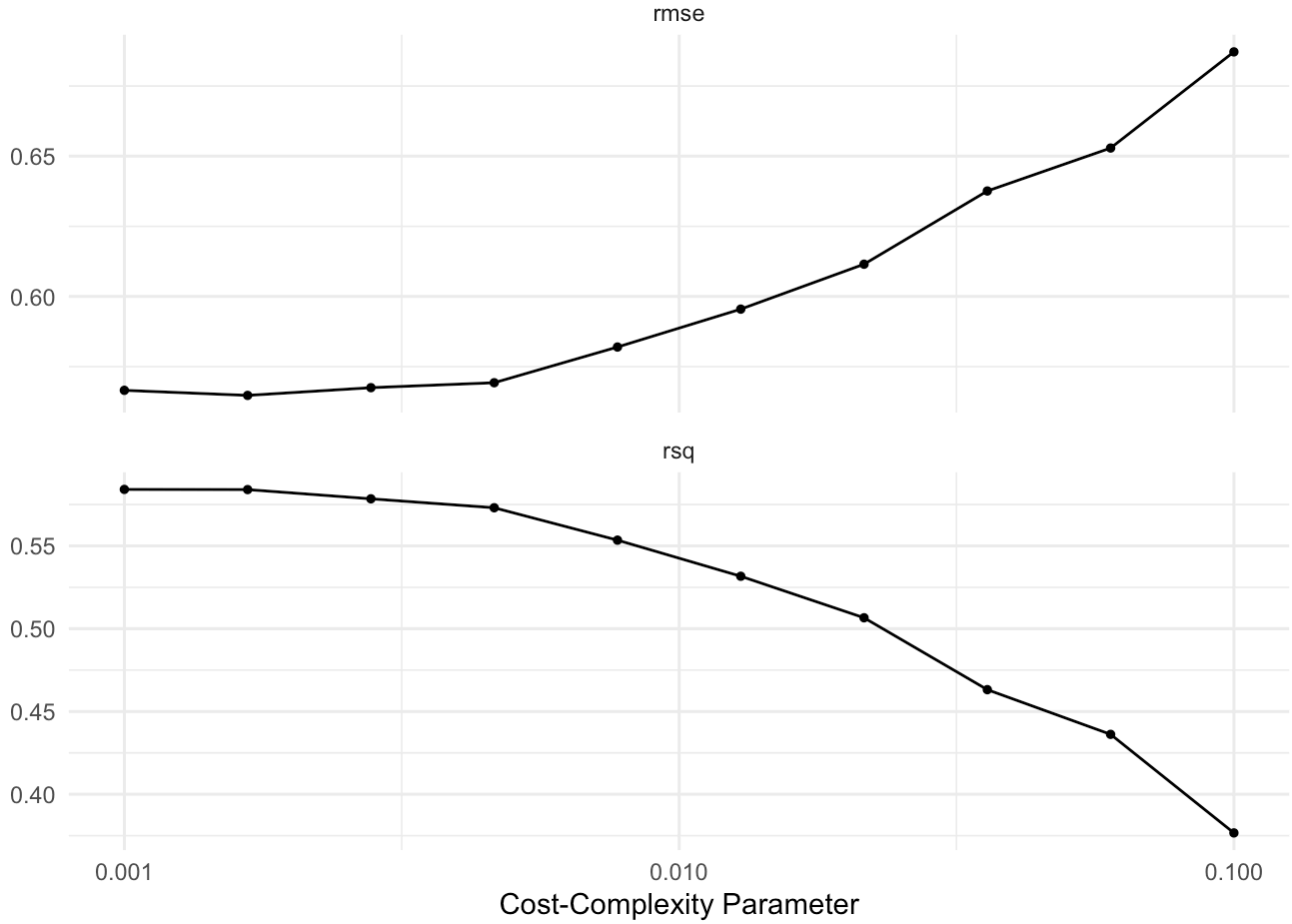
```
autoplot(tune_knn_imdb) + theme_minimal()
```




```
autoplot(tune_rf_imdb) + theme_minimal()
```




```
autoplot(tune_gb_imdb) + theme_minimal()
```




```
# table of best RMSE values of linear regression model
show_best(fit_lm_imdb)
```

```
## # A tibble: 1 × 6
##   .metric .estimator  mean     n std_err .config
##   <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 rmse    standard   0.525     5  0.0115 Preprocessor1_Model1
```

```
# table of best k-nearest neighbors model
show_best(tune_knn_imdb)
```

```
## # A tibble: 3 × 7
##   neighbors .metric .estimator  mean     n std_err .config
##       <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1         5 rmse    standard   0.566     5 0.00796 Preprocessor1_Model3
## 2         3 rmse    standard   0.604     5 0.00839 Preprocessor1_Model2
## 3         1 rmse    standard   0.692     5 0.00741 Preprocessor1_Model1
```

```
# table of best random forest model
show_best(tune_rf_imdb)
```

```
## # A tibble: 5 × 9
##    mtry trees min_n .metric .estimator  mean     n std_err .config
##   <int> <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1     4   300    10 rmse    standard   0.507     5  0.0115 Preprocessor1_Model0…
## 2     4   600    10 rmse    standard   0.507     5  0.0114 Preprocessor1_Model0…
## 3     4   300    15 rmse    standard   0.507     5  0.0115 Preprocessor1_Model0…
## 4     4   400    15 rmse    standard   0.507     5  0.0109 Preprocessor1_Model0…
## 5     4   500    12 rmse    standard   0.507     5  0.0108 Preprocessor1_Model0…
```

```
# table of best gradient boosted model
show_best(tune_gb_imdb)
```

```
## # A tibble: 5 × 7
##   cost_complexity .metric .estimator  mean     n std_err .config
##             <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1         0.00167 rmse    standard   0.565     5 0.00734 Preprocessor1_Model02
## 2         0.001   rmse    standard   0.567     5 0.0106  Preprocessor1_Model01
## 3         0.00278 rmse    standard   0.567     5 0.0117  Preprocessor1_Model03
## 4         0.00464 rmse    standard   0.569     5 0.00942 Preprocessor1_Model04
## 5         0.00774 rmse    standard   0.582     5 0.00985 Preprocessor1_Model05
```

```
# best model
best_model_rf_imdb <- select_best(tune_rf_imdb)
```

Above are the results of which of all the models performed the best. There graphs above show the results of the tuning of k-nearest neighbors, random forest, and gradient boosted trees. The tables above show the best performing models within each model. The best model ended up being model 64 of random forest. It had an RMSE value of 0.50699. This is pretty small RMSE value which leads me to believe that the data fits well within this model and it is fairly accurate. The other models also ended up performing quite well. Linear regression resulted in an RMSE value of 0.52543, k-nearest neighbors resulted in an RMSE value of 0.566186, and gradient boosted resulted in an RMSE value of 0.5647113. These values are also relatively small as well and could have been used to to predict the testing set. However, since random forest performed the best, that is the model I will be using.

Finalizing and Fit Workflow

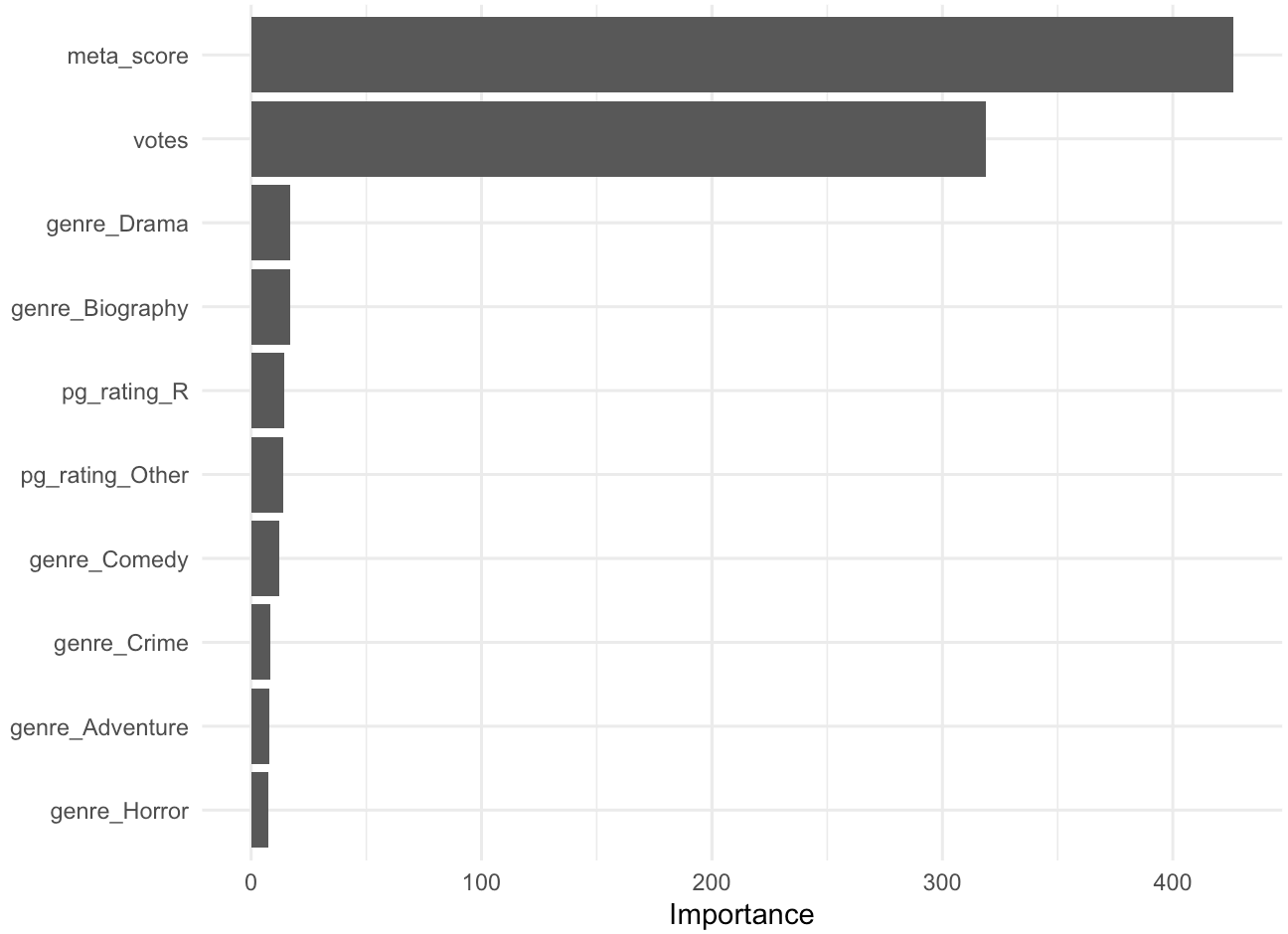
```
final_wkflow_rf_imdb <- finalize_workflow(rf_wkflow_imdb, best_model_rf_imdb)

final_fit_rf_imdb <- final_wkflow_rf_imdb %>%
  fit(data = imdb_train)

final_fit_rf_imdb
```

```
## ══ Workflow [trained] ══════════════════════════════════════════════════════
## Preprocessor: Recipe
## Model: rand_forest()
##
## ── Preprocessor ────────────────────────────────────────────────────────────
## 4 Recipe Steps
##
## • step_dummy()
## • step_zv()
## • step_center()
## • step_scale()
##
## ── Model ───────────────────────────────────────────────────────────────────
## Ranger result
##
## Call:
##  ranger::ranger(x = maybe_data_frame(x), y = y, mtry = min_cols(~4L,      x), num.tre
es = ~300L, min.node.size = min_rows(~10L, x),      importance = ~"impurity", num.thread
s = 1, verbose = FALSE,      seed = sample.int(10^5, 1))
##
## Type:                             Regression
## Number of trees:                  300
## Sample size:                      1426
## Number of independent variables:  12
## Mtry:                             4
## Target node size:                 10
## Variable importance mode:         impurity
## Splitrule:                        variance
## OOB prediction error (MSE):       0.2592563
## R squared (OOB):                  0.6564072
```

In this step, I set up the final workflow for the training data set by adding the random forest workflow and the best performing model. Then I fit that work flow to the training data set.

Variable Importance Plot

```
final_fit_rf_imdb %>% extract_fit_parsnip() %>%
  vip(importance = "impurity") +
  theme_minimal()
```



With the plot above, I can clearly see which variables were most important in predicting the audience score. Like I originally predicted, meta score was the best predictor variable, with votes also being quite high as well. However, specific genres and pg-ratings were not quite as useful in predicted the ratings.

Augmenting to the Testing Set

```
final_augment_test_imdb <- augment(final_fit_rf_imdb, imdb_test)

final_augment_test_imdb %>%
  rmse(truth = rating, estimate = .pred)
```

```
## # A tibble: 1 × 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard       0.487
```

In this step I augmented the results found in the training set to the testing set. From that, I found the RMSE value of the testing set and got a value of 0.4850. This value is slightly lower than the best performing training model, number 64 of random forest, which had a value of 0.50699.

Conclusion

For this project, I sought out to find whether variables such as Meta Critic Score, PG-Ratings, Votes, and the Genre of a movie can predict how people on IMDb would rate the movie. While there were several other variables in the data set, after some exploratory data analysis I realized that these variables would be the best in order to predict audience IMDb ratings. After doing exploratory data analysis, I split the data and set up k-fold cross-validation. Once I did that, I set up the recipe for the rest of the project. From there I set up four models, linear regression, k-nearest neighbors, random forest, and gradient boosted. I then set up the workflow for each and fit/tuned the models. From there, I evaluated which of the models performed the best by comparing their RMSE, or root mean squared error, values. From there, I augmented the findings from the training data set to the testing data set.

From this project, I realized that the random forest model performed the best for my selected data set. Out the many models that were run through the four main models, model 64 of random forest had the smallest RMSE value. I then used the results from this model to augment to the testing set. The testing RMSE value is 0.4850. Since this value is quite small, it tells me that the predictions made by the models fit the data well and are fairly accurate.

I am glad I got to explore IMDb data further in this project. It made me realize how well audience ratings can be predicted for movies. In the future, I would want to test even more aspects of the IMDb website. Specifically, I think it would be interesting to see how the cast and crew of a movie can affect audience ratings. Also, I would like to approach the data set with a different outcome variable. For example, I would want to explore if different aspects of a movie can help predict who the director is.