GDAA 1001 - Fundamentals of Spatial Data Analytics Final Project - Exploratory Data Analysis & Predictive Modeling

Ayoub Gouriba

2022-12-11

Contents

List of Figures

1. Introduction

This report provides an exploratory data analysis of a webscraped database from the BoardGameGeek.com website. The focus of the analysis is to build a predictive model using machine learning algorithms to predict the difficulty of games. The dataset contains various game attributes, such as rating, game length, and game mechanics. Through the analysis, we seek to identify patterns in the data and to build an effective model to accurately predict the difficulty of a game.

2. Data Preparation

a. Loading Data

```
bg <- read.csv(file = 'BGG_Data_Set.csv')
as_tibble(bg)
```

```
## # A tibble: 20,343 x 14
##        ID Name   Year.~1 Min.P~2 Max.P~3 Play.~4 Min.Age Users~5 Ratin~6 BGG.R~7
##     <int> <chr>    <int>   <int>   <int>   <int>   <int>   <int>   <dbl>   <int>
##  1 174430 Gloom~    2017       1       4     120      14   42055    8.79       1
##  2 161936 Pande~    2015       2       4      60      13   41643    8.61       2
##  3 224517 Brass~    2018       2       4     120      14   19217    8.66       3
##  4 167791 Terra~    2016       1       5     120      12   64864    8.43       4
##  5 233078 Twili~    2017       3       6     480      14   13468    8.7        5
##  6 291457 Gloom~    2020       1       4     120      14    8392    8.87       6
##  7 182028 Throu~    2015       2       4     120      14   23061    8.43       7
##  8 220308 Gaia ~    2017       1       4     150      12   16352    8.49       8
##  9 187645 Star ~    2016       2       4     240      14   23081    8.42       9
## 10  12333 Twili~    2005       2       2     180      13   40814    8.29      10
## # ... with 20,333 more rows, 4 more variables: Complexity.Average <dbl>,
## #   Owned.Users <int>, Mechanics <chr>, Domains <chr>, and abbreviated variable
## #   names 1: Year.Published, 2: Min.Players, 3: Max.Players, 4: Play.Time,
## #   5: Users.Rated, 6: Rating.Average, 7: BGG.Rank
```

```
nrow(bg)
```

```
## [1] 20343
```

There are 20,343 data points.

Attributes in the dataset:

- **ID**: Unique BoardGamesGeek ID
- **Name**: Board game name
- **Year Published**: Year published
- **Min Players**: Minimum suggested players

- **Max Players**: Maximum suggested players
- **Play Time**: Average play time suggested by game creators [Numeric Minutes]
- **Min Age**: Age rating
- **Users Rated**: Amount of BGG users who reviewed the game
- **Rating Average**: Average BGG player rating
- **BGG Rank**: BoardGamesGeek ranking
- **Complexity Average**: Average BGG community complexity rating [Numeric 1-5]
- **Owned Users**: Amount of BGG users who said they own the game
- **Mechanics**: BGG Game Mechanics
- **Domains**: BGG community voted game subgenre

Change the column names

```
bg <- rename(bg,id = ID,name = Name,year = Year.Published,minP = Min.Players,maxP = Max.Players,time = 
```

The dataset contains 20,343 rows. To make it easier to work with, we are going to subest it to leave only games that have been published between 2015 and 2020.

```
bg <- bg %>% filter(between(year, 2015, 2020))
nrow(bg)
```

```
## [1] 6734
```

We end up with 6734 data-points.

b. Data Cleaning

```
colSums(is.na(bg))
```

```
##         id       name       year       minP       maxP       time        age
##          3          0          0          0          0          0          0
## usersRated  ratingAvg       rank complexity      users  mechanics    domains
##          0          0          0          0          6          0          0
```

Column "id" is missing 3 values and "users" have 6 missing values.

Let's remove the records with missing values and check for duplicate .

```
bg <- na.omit(bg)
nrow(bg)
```

```
## [1] 6728
```

The new row number is 6728

Check if the the dataset is unique

```
bg <- unique(bg)
nrow(bg)
```

```
## [1] 6728
```

There are no duplicates in the dataset since the total number didn't change.

c. Data Transforming

-Create new column "audience" based on minimum age -Create new column "complexity" to categorize complexity scores -Create new column "ratingPercent", percentage of users of the game who wrote a review -Create new column "mechanicsCount" that includes the count of mechanics of the game -Convert "complexity" column to factor -Select the columns: rank, name, year, minP, maxP, time, audience, ratingPercent, ratingAvg, complexity and mechanicsCount

```
bg <- bg %>% mutate(audience = case_when(
                    age < 12 ~"Everyone",
                    age >= 12 ~"Teen+")) %>%
             mutate(complexity = case_when(
                    complexity <= 2.00 ~"01 - Easy",
                    complexity > 2.00 & complexity <= 3.00 ~"02 - Average",
                    complexity > 3.00 ~"03 - Difficult")) %>%
             mutate(ratingPercent = usersRated*100/users) %>%
             mutate(mechanicsCount = sapply(strsplit(bg$mechanics, ","), length)) %>%
            mutate(genresCount = as.character(sapply(strsplit(bg$domains, ","), length))) %>%
             mutate(complexity = as.factor(complexity)) %>%
             select(rank, name, year, minP, maxP, time, audience, ratingPercent, ratingAvg, complexity,
as_tibble(bg)
```

```
## # A tibble: 6,728 x 12
##     rank name     year  minP  maxP  time audie~1 ratin~2 ratin~3 compl~4 mecha~5
##    <int> <chr>   <int> <int> <int> <int> <chr>     <dbl>   <dbl> <fct>     <int>
##  1     1 Gloomh~  2017     1     4   120 Teen+      61.6    8.79 03 - D~      19
##  2     2 Pandem~  2015     2     4    60 Teen+      63.8    8.61 02 - A~       8
##  3     3 Brass:~  2018     2     4   120 Teen+      66.8    8.66 03 - D~       9
##  4     4 Terraf~  2016     1     5   120 Teen+      74.5    8.43 03 - D~      12
##  5     5 Twilig~  2017     3     6   480 Teen+      80.0    8.7  03 - D~      12
##  6     6 Gloomh~  2020     1     4   120 Teen+      38.8    8.87 03 - D~      16
##  7     7 Throug~  2015     2     4   120 Teen+      85.5    8.43 03 - D~       7
##  8     8 Gaia P~  2017     1     4   150 Teen+      80.5    8.49 03 - D~      11
##  9     9 Star W~  2016     2     4   240 Teen+      66.2    8.42 03 - D~       8
## 10    11 Great ~  2016     2     4   150 Teen+      82.3    8.3  03 - D~       8
## # ... with 6,718 more rows, 1 more variable: genresCount <chr>, and abbreviated
## #   variable names 1: audience, 2: ratingPercent, 3: ratingAvg, 4: complexity,
## #   5: mechanicsCount
```

The data is now clean, transformed and ready to be analyzed.

3. Exploratory Data Analysis

Show a summary of the new transformed dataset

```r
bg_summary <- summary(bg)

knitr::kable(bg_summary)
```

	rank	name	year	minP	maxP	time	audience	ratingPercent	ratingAvg	complexity	mechanicsCount	genresCount
	Min. : 1	Length:6728	Min. :2015	Min. : 0.000	Min. : 0.000	Min. : 0.00	Length:6728	Min. : 7.534	Min. :1.100	01 - Easy :3985	Min. : 0.000	Length:6728
	1st Qu.: 3930	Class :character	1st Qu.:2016	1st Qu.: 1.000	1st Qu.: 4.000	1st Qu.: 30.00	Class :character	1st Qu.: 31.895	1st Qu.:6.370	02 - Average :2028	1st Qu.: 2.000	Class :character
	Median : 7937	Mode :character	Median :2017	Median : 2.000	Median : 4.000	Median : 45.00	Mode :character	Median : 42.318	Median :6.870	03 - Difficult: 715	Median : 3.000	Mode :character
	Mean : 8290	NA	Mean :2017	Mean : 1.918	Mean : 5.908	Mean : 80.64	NA	Mean : 46.868	Mean :6.864	NA	Mean : 3.281	NA
	3rd Qu.:12286	NA	3rd Qu.:2019	3rd Qu.: 2.000	3rd Qu.: 6.000	3rd Qu.: 86.25	NA	3rd Qu.: 54.023	3rd Qu.:7.380	NA	3rd Qu.: 4.000	NA
	Max. :20327	NA	Max. :2020	Max. :10.000	Max. :999.000	Max. :10000.00	NA	Max. :2191.667	Max. :9.430	NA	Max. :19.000	NA

```r
plot1 <- ggplot(bg, aes(x=ratingAvg, y=maxP, color=complexity)) +
    geom_point(size=1.5, alpha=0.5)+
  scale_color_viridis(discrete=TRUE)+
  theme(legend.position = "none")

plot2 <- ggplot(bg, aes(x=ratingAvg, y=time, color=complexity)) +
    geom_point(size=1.5, alpha=0.5)+
  scale_color_viridis(discrete=TRUE)

grid.arrange(plot1, plot2, ncol = 2)
```

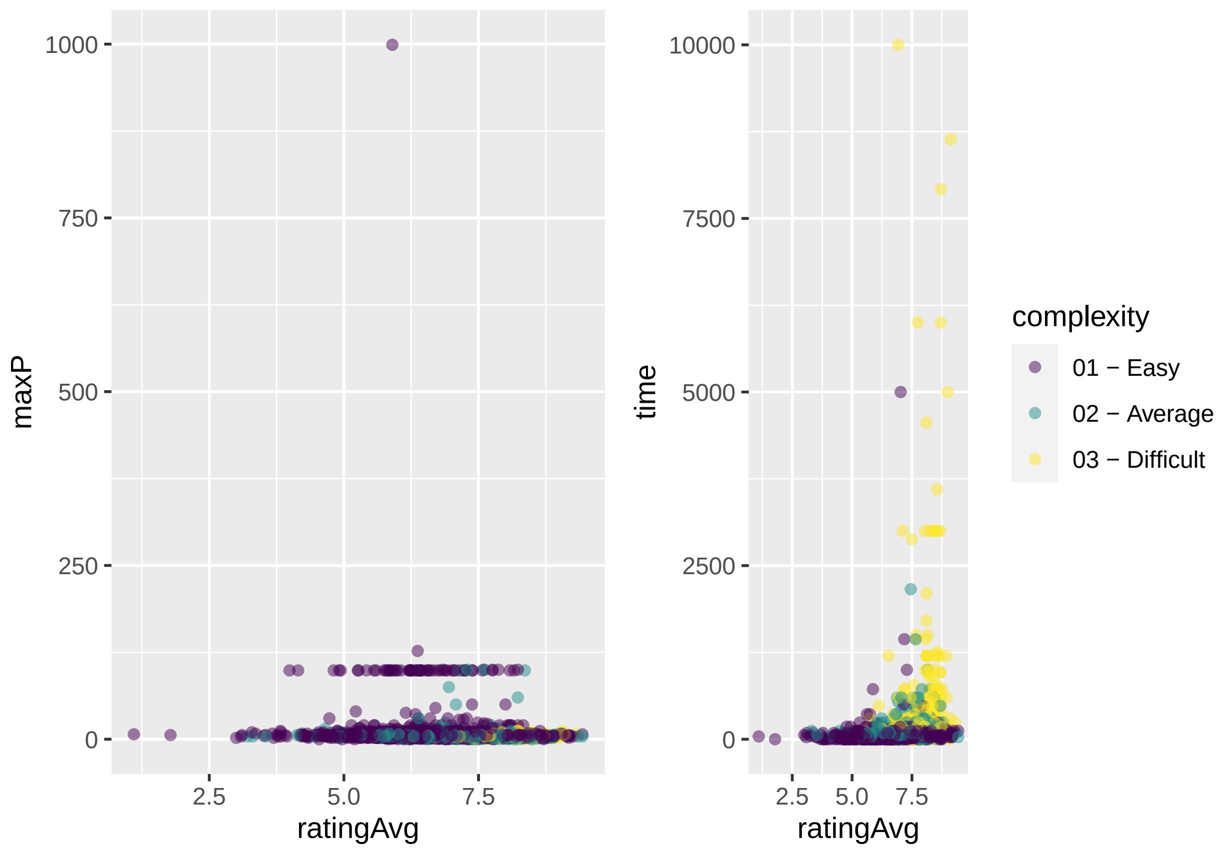

Figure 1: Scatterplots of (1) Rating Average and Max Players and (2) Rating Average and Play Time

Let's filter out Max Players values of over 100 and Play Time values of over 5000 minutes.

```
bg <- bg %>% filter(maxP<100, time<5000)
nrow(bg)
```

```
## [1] 6712
```

That leaves us with 6712 datapoints.

```
ggplot(bg) +
    geom_bar(aes(x = audience, fill = complexity)) +
    scale_fill_viridis(discrete = TRUE) +
    labs(x = "Audience Type", y = "Number of Games")
```

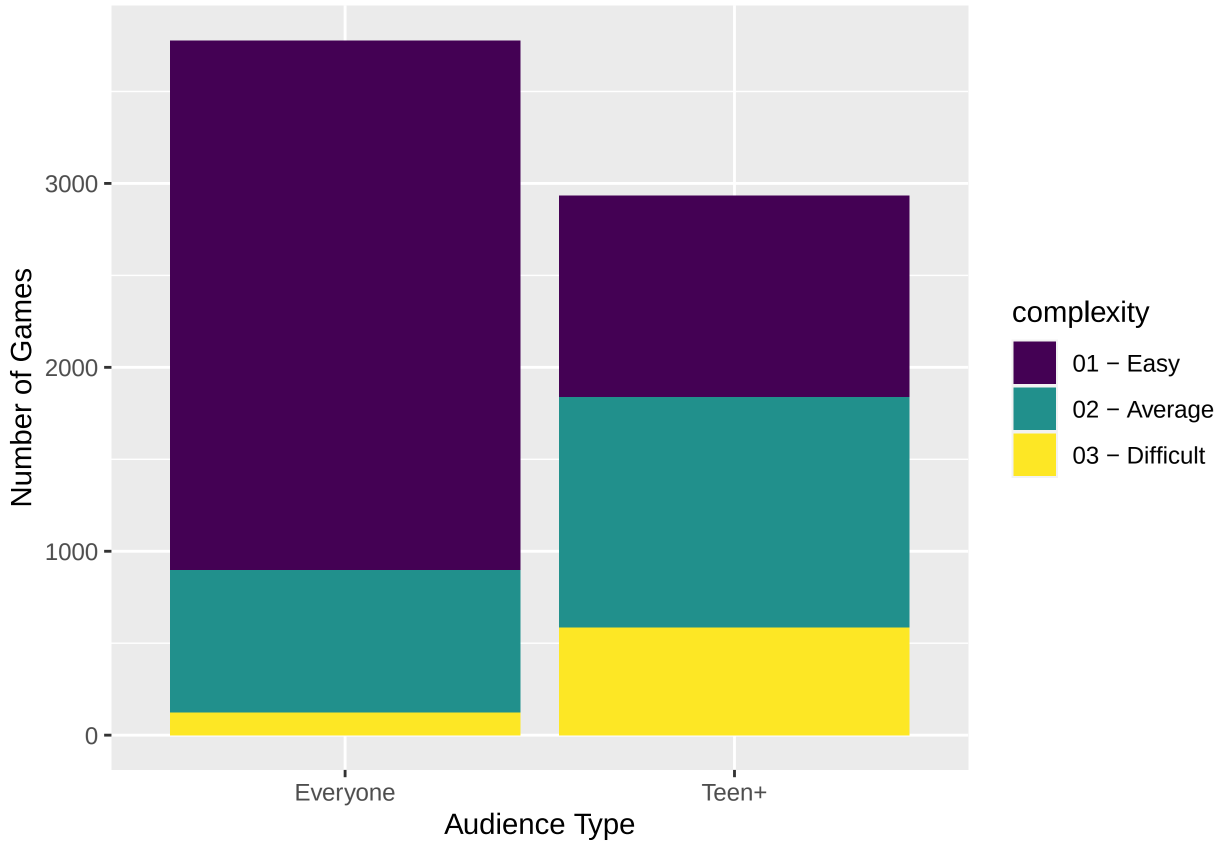

Figure 2: Bar chart of Number of Games by Audience and Complexity

Let's cap percentages in ratingPercent at 100.

Replace all percentages above 100 in ratingPercent with 100.

```
bg <- bg %>%
  mutate(ratingPercent = ifelse(ratingPercent > 100, 100, ratingPercent))
```

```
ggplot(bg,
  aes(x=ratingAvg, y=ratingPercent, color=complexity)) +
  geom_point(size=2, alpha=0.5)+
  scale_color_viridis(discrete=TRUE)+
  labs(x="Average Rating", y="Rating Percentage")
```

Figure 3: Scatterplot of Rating Average vs Rating Percentage by Complexity

```
ggplot(bg, aes(x=ratingAvg, y = factor(year), fill = factor(year))) +
  geom_boxplot() +
  xlim(0, 10) +
  scale_x_continuous(breaks = seq(0, 10, 2)) +
  theme(legend.position = "none")
```

```
## Scale for x is already present.
## Adding another scale for x, which will replace the existing scale.
```

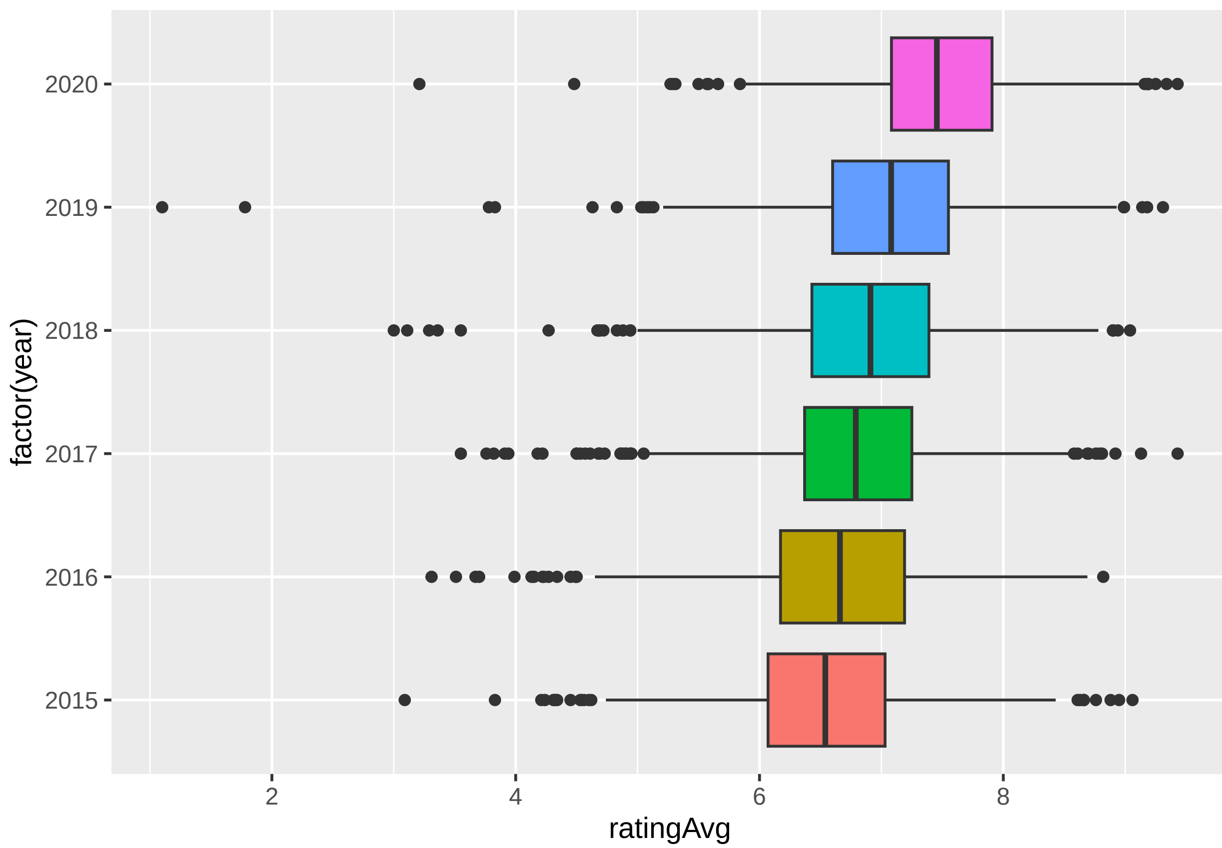

Figure 4: Boxplot of Rating Average by Year

```
ggplot(data = bg, aes(x = ratingAvg))+
  geom_density(aes(fill = complexity), alpha = 0.7) +
  geom_histogram(aes(y = ..density..),binwidth = 0.5, alpha = 0.7, color = "black") +
  scale_fill_viridis(discrete = TRUE) +
  xlim(0,10) +
  labs(x = "Rating Average", y = "Density")
```

```
## Warning: The dot-dot notation (`..density..`) was deprecated in ggplot2 3.4.0.
## i Please use `after_stat(density)` instead.
```

```
## Warning: Removed 2 rows containing missing values (`geom_bar()`).
```

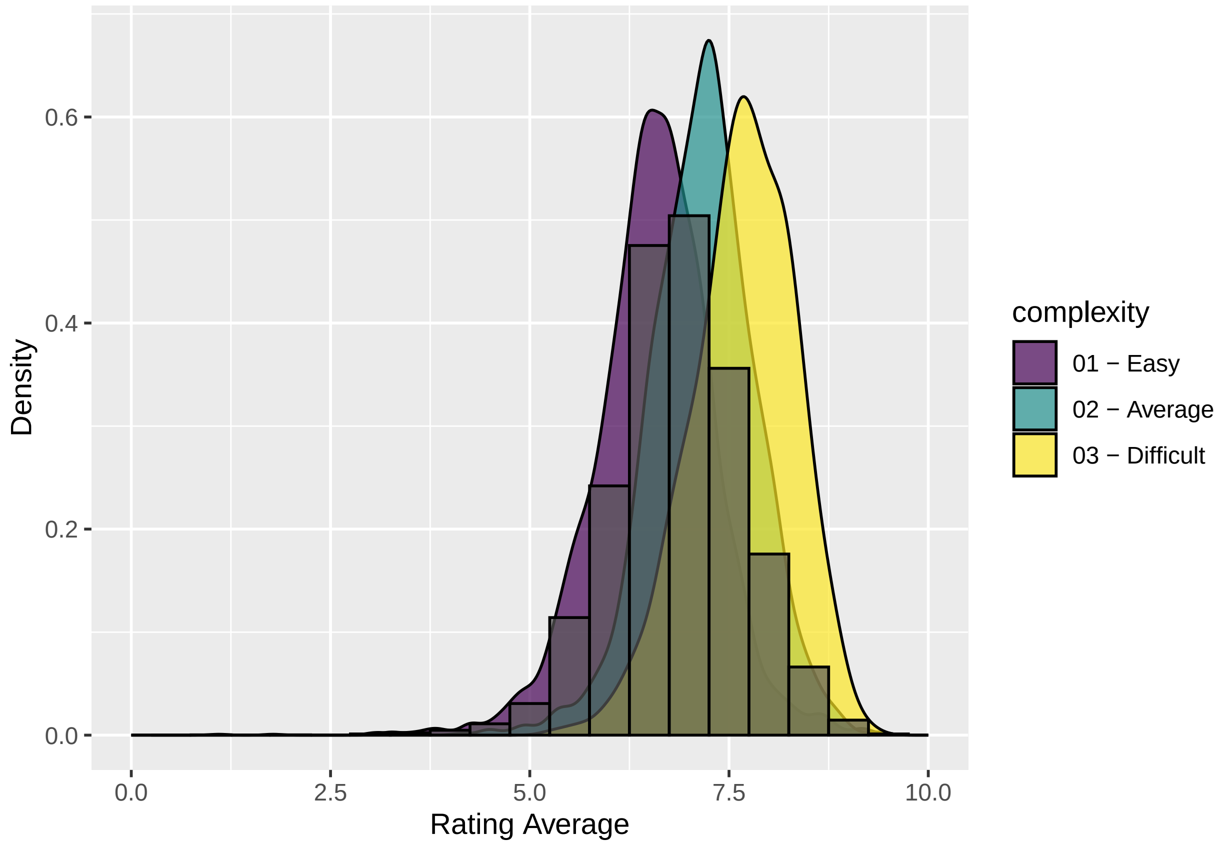

Figure 5: Density Histogram of Rating Average density by Game Complexity

```
ggplot(bg) +
    geom_bar(aes(x = mechanicsCount, fill = complexity)) +
    scale_fill_viridis(discrete = TRUE) +
    labs(x = "Number of Mechanics", y = "Frequency")
```

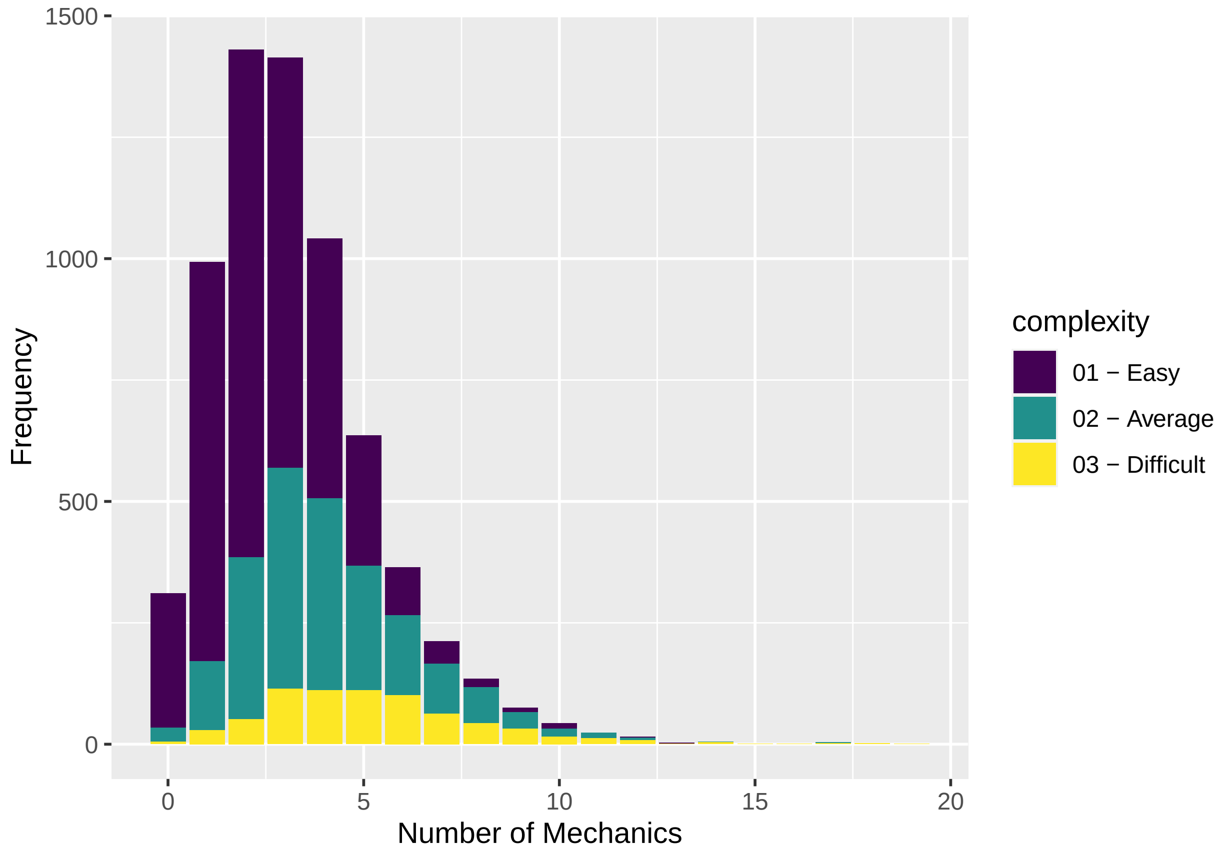

Figure 6: Histogram of Mechanics Count by Complexity

```
boxplot(mechanicsCount ~ complexity, data = bg, xlab = "Complexity", ylab = "mechanicsCount", col = vir
```

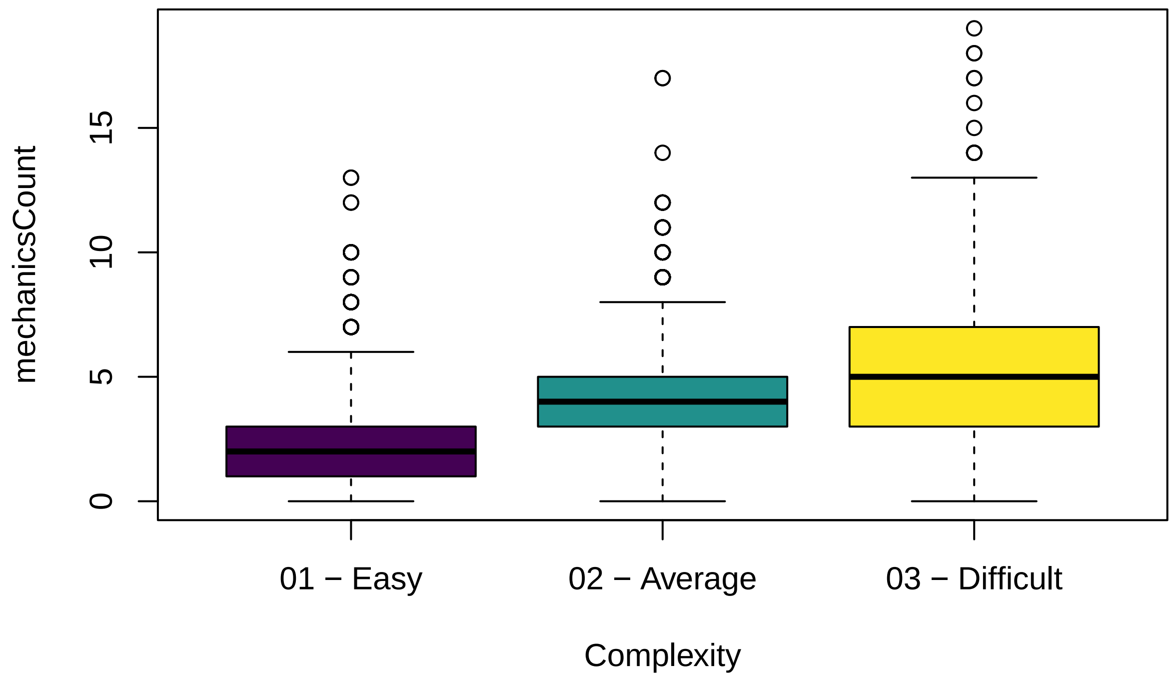

Figure 7: Boxplot of Mechanics Count by Complexity

```
bg %>%
  ggplot(aes(ratingAvg, ratingPercent))+
  geom_hex()+
  scale_fill_viridis() +
  labs(x = "Rating Average", y = "Rating Percentage")
```

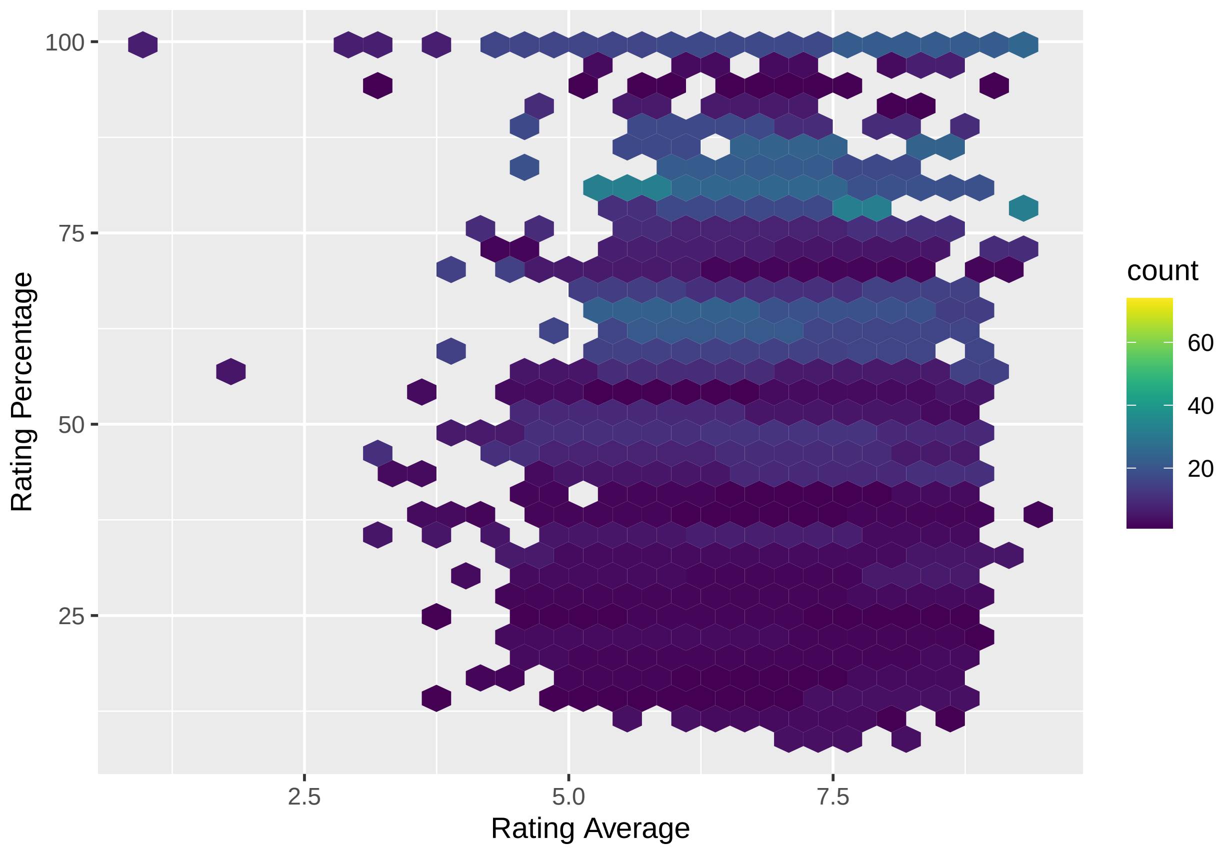

Figure 8: Hexagonal heatmap of Rating Average and Rating Percentage

```
ggplot(bg, aes(x = complexity, y = year)) +
  geom_tile(aes(fill = ratingAvg)) +         scale_fill_viridis_c()+
  labs(x = "Complexity", y = "Year")
```

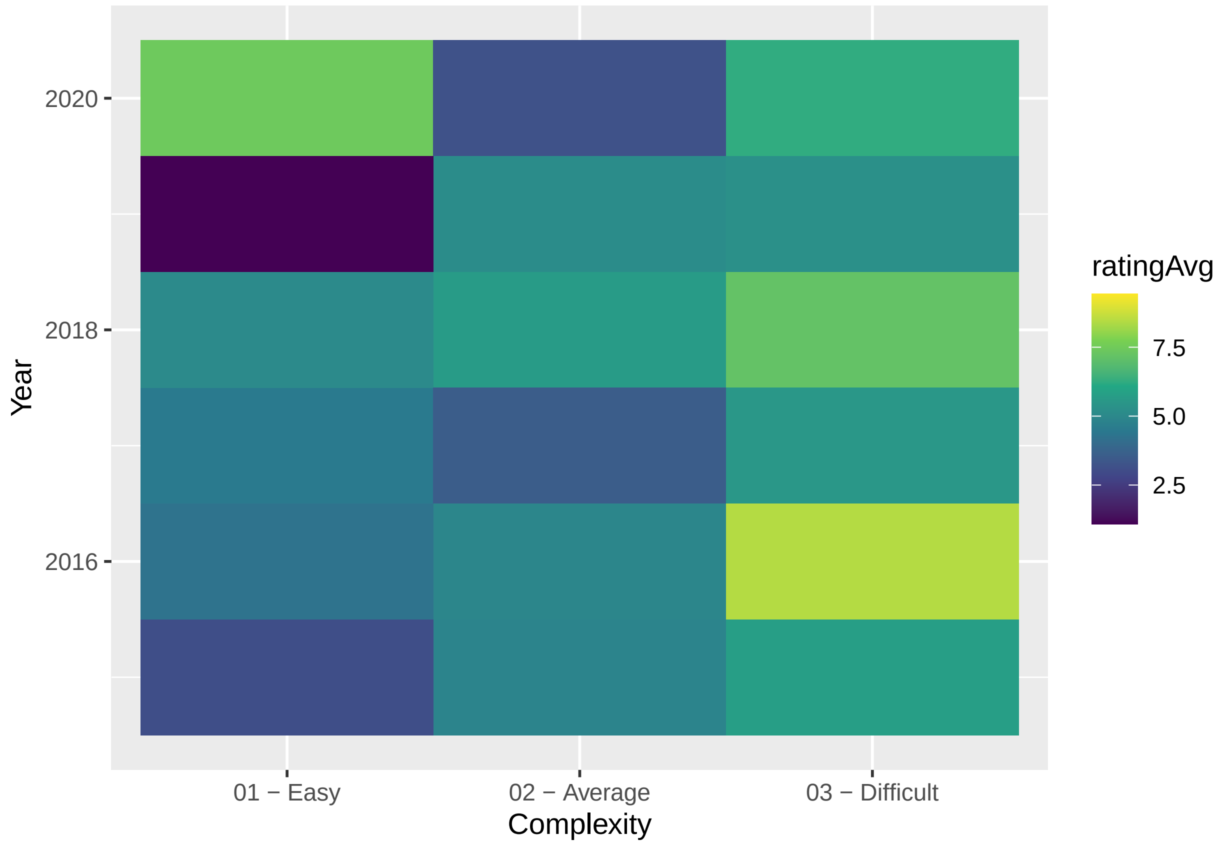

Figure 9: Heatmap of Game Complexity by Year

```
bg %>% ggpairs(.,
               legend = 1,
               columns = c("ratingAvg","mechanicsCount","genresCount", "complexity"),
               mapping = ggplot2::aes(colour=complexity),
               upper = list(combo = wrap("box_no_facet", alpha=0.5), continuous = wrap("cor", size=3)),
               diag = list(discrete="barDiag",
  continuous = wrap("densityDiag", alpha=0.5 )),
               lower = list(continuous = wrap("smooth", alpha = 0.3, size=1),
  discrete = "barDiag")) +
theme(legend.position = "bottom")
```

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

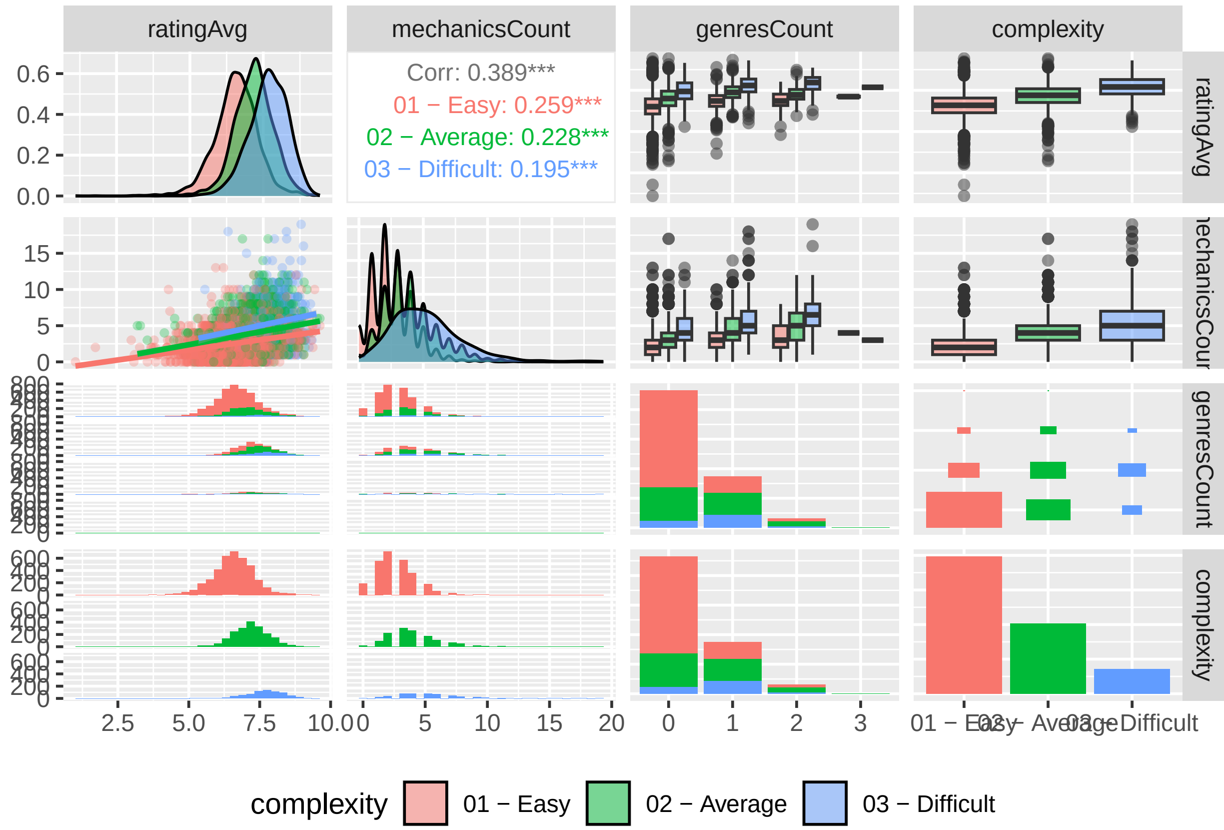

Figure 10: Scatterplot matrix of ratingAvg, mechanicsCount, genresCount and complexity

```
#change genresCount back to numeric so we can use it for the prediction models
bg <- bg %>% mutate(genresCount = as.numeric(genresCount))
```

4. Predictive Modelling

We will be running four predictive models on board game data – a decision tree, random forest (rf), k-nearest neighbors (KNN), and support vector machine (SVM).

The target variable is complexity, and the predictors we will be using include year, minP, maxP, time, ratingPercent, ratingAvg, mechanicsCount, and genresCount. We will be analyzing the data to determine which model is the most accurate at predicting complexity.

a. Decision Tree

```
bg_model <- rpart(complexity ~ year+minP+maxP+time+ ratingPercent+ratingAvg+mechanicsCount+genresCount,

#Visualize as Tree of Decisions using rpart.plot()
rpart.plot(bg_model)
```

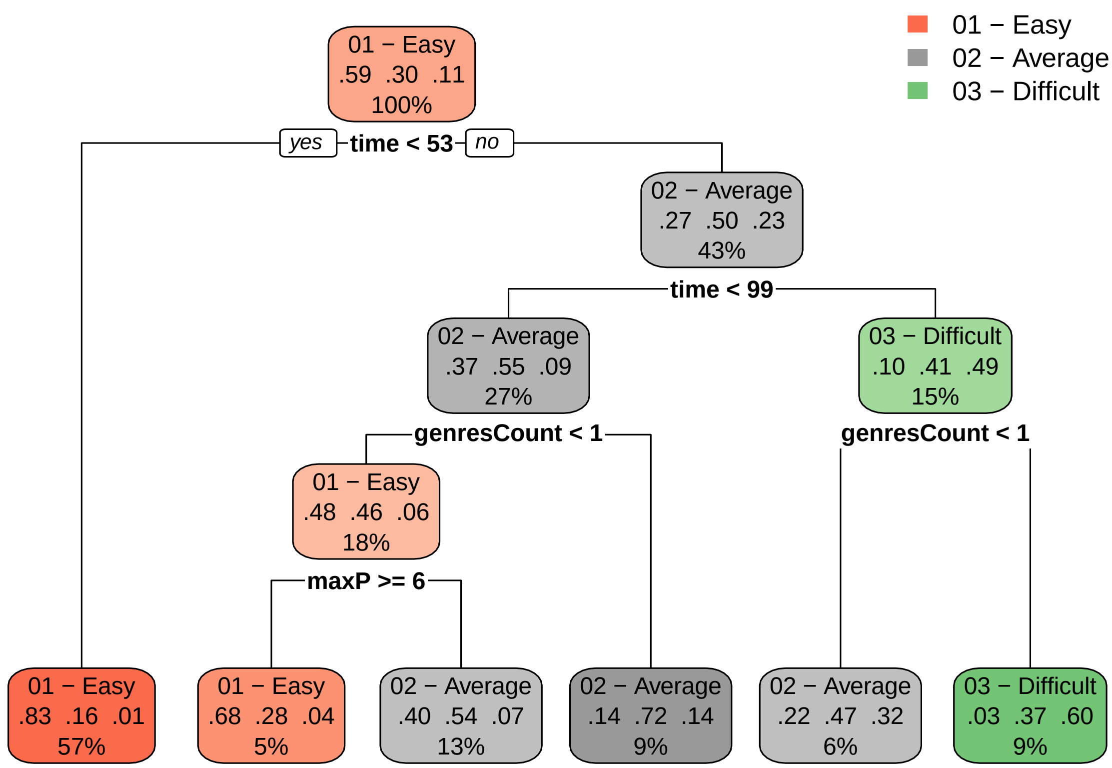

Figure 11: Decision Tree of the Decision Tree Model

Now for the following models (rf, knn and svm) we need to create training and validation data as inputs

```
# create training and validation data
inTraining <- createDataPartition(bg$complexity, p=0.80, list=FALSE)
training <- bg[inTraining,]
validation <- bg[-inTraining,]

# run algorithms (10-fold cross validation)
control <- trainControl(method="cv", number=10)
metric <- "Accuracy"
```

b. Random Forest

Train and predict the model using random forest.

Print summary results using confusion matrix.

```
set.seed(123)
# Train the model
fit.rf <- train(complexity ~ year+minP+maxP+time+ ratingPercent+ratingAvg+mechanicsCount+genresCount, da

# Predict the model
predictions <- predict(fit.rf, validation)
```

```r
# Confusion matrix
cm1 <- confusionMatrix(predictions, as.factor(validation$complexity))
cm1
```

```
## Confusion Matrix and Statistics
##
##                  Reference
## Prediction        01 - Easy 02 - Average 03 - Difficult
##   01 - Easy             714          128              8
##   02 - Average           76          251             56
##   03 - Difficult          5           26             77
##
## Overall Statistics
##
##                Accuracy : 0.777
##                  95% CI : (0.7538, 0.7991)
##     No Information Rate : 0.5928
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5789
##
##  Mcnemar's Test P-Value : 1.602e-05
##
## Statistics by Class:
##
##                      Class: 01 - Easy Class: 02 - Average Class: 03 - Difficult
## Sensitivity                    0.8981              0.6198               0.54610
## Specificity                    0.7509              0.8590               0.97417
## Pos Pred Value                 0.8400              0.6554               0.71296
## Neg Pred Value                 0.8350              0.8392               0.94809
## Prevalence                     0.5928              0.3020               0.10515
## Detection Rate                 0.5324              0.1872               0.05742
## Detection Prevalence           0.6339              0.2856               0.08054
## Balanced Accuracy              0.8245              0.7394               0.76013
```

c. KNN

Train and predict the model using k-nearest neighbor.

Print summary results using confusion matrix.

```r
set.seed(123)
# Train the model
fit.knn <- train(complexity ~ year+minP+maxP+time+ ratingPercent+ratingAvg+mechanicsCount+genresCount, 

# Predict the model
predictions <- predict(fit.knn, validation)

# Confusion matrix
cm2 <- confusionMatrix(predictions, as.factor(validation$complexity))
cm2
```

```
## Confusion Matrix and Statistics
```

```
##
##                  Reference
## Prediction        01 - Easy 02 - Average 03 - Difficult
##   01 - Easy             683          145              8
##   02 - Average          106          211             64
##   03 - Difficult          6           49             69
##
## Overall Statistics
##
##                Accuracy : 0.7181
##                  95% CI : (0.6932, 0.7421)
##     No Information Rate : 0.5928
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.473
##
##  Mcnemar's Test P-Value : 0.03954
##
## Statistics by Class:
##
##                      Class: 01 - Easy Class: 02 - Average Class: 03 - Difficult
## Sensitivity                    0.8591              0.5210               0.48936
## Specificity                    0.7198              0.8184               0.95417
## Pos Pred Value                 0.8170              0.5538               0.55645
## Neg Pred Value                 0.7782              0.7979               0.94084
## Prevalence                     0.5928              0.3020               0.10515
## Detection Rate                 0.5093              0.1573               0.05145
## Detection Prevalence           0.6234              0.2841               0.09247
## Balanced Accuracy              0.7894              0.6697               0.72176
```

d. SVM

Train and predict the model using support vector machine.

Print summary results using confusion matrix.

```
set.seed(123)
# Train the model
fit.svm <- train(complexity ~ year+minP+maxP+time+ ratingPercent+ratingAvg+mechanicsCount+genresCount, 

# Predict the model
predictions <- predict(fit.svm, validation)

# Confusion matrix
cm3 <- confusionMatrix(predictions, as.factor(validation$complexity))
cm3
```

```
## Confusion Matrix and Statistics
##
##                  Reference
## Prediction        01 - Easy 02 - Average 03 - Difficult
##   01 - Easy             741          165             10
##   02 - Average           53          227             77
##   03 - Difficult          1           13             54
```

```
##
## Overall Statistics
##
##                Accuracy : 0.7621
##                  95% CI : (0.7384, 0.7847)
##     No Information Rate : 0.5928
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5329
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: 01 - Easy Class: 02 - Average Class: 03 - Difficult
## Sensitivity                    0.9321             0.5605               0.38298
## Specificity                    0.6795             0.8611               0.98833
## Pos Pred Value                 0.8090             0.6359               0.79412
## Neg Pred Value                 0.8729             0.8191               0.93166
## Prevalence                     0.5928             0.3020               0.10515
## Detection Rate                 0.5526             0.1693               0.04027
## Detection Prevalence           0.6831             0.2662               0.05071
## Balanced Accuracy              0.8058             0.7108               0.68566
```

5. Evaluation of Results

a. Confusion Matrices

Let's plot the confusion matrices of the models used.

```
cm1_d <- as.data.frame(cm1$table)
cm1_d$diag <- cm1_d$Prediction == cm1_d$Reference # Get the Diagonal
cm1_d$ndiag <- cm1_d$Prediction != cm1_d$Reference # Off Diagonal
cm1_d[cm1_d == 0] <- NA # Replace 0 with NA for white tiles
cm1_d$Reference <-  reverse.levels(cm1_d$Reference) # diagonal starts at top left
cm1_d$ref_freq <- cm1_d$Freq * ifelse(is.na(cm1_d$diag),-1,1)

plt1 <-  ggplot(data = cm1_d, aes(x = Prediction , y =  Reference, fill = Freq))+
  scale_x_discrete(position = "top") +
  geom_tile( data = cm1_d,aes(fill = ref_freq)) +
  scale_fill_gradient2(guide = FALSE ,low="red3",high="orchid4", midpoint = 0,na.value = 'white') +
  geom_text(aes(label = Freq), color = 'black', size = 3)+
  theme_bw() +
  theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank(),
        legend.position = "none",
        panel.border = element_blank(),
        plot.background = element_blank(),
        axis.line = element_blank(),
  )
plt1
```

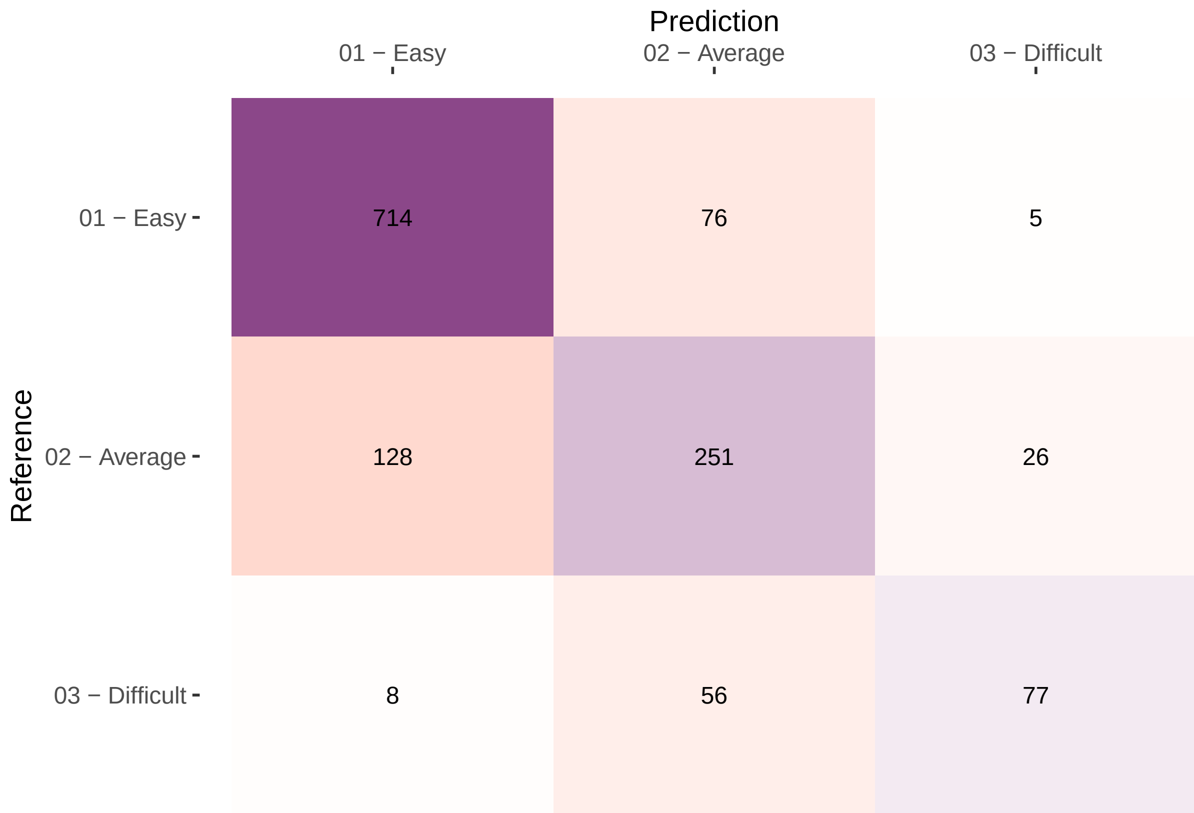

Figure 12: Random Forest Confusion Matrix

```
cm2_d <- as.data.frame(cm2$table)
cm2_d$diag <- cm2_d$Prediction == cm2_d$Reference # Get the Diagonal
cm2_d$ndiag <- cm2_d$Prediction != cm2_d$Reference # Off Diagonal
cm2_d[cm2_d == 0] <- NA # Replace 0 with NA for white tiles
cm2_d$Reference <-  reverse.levels(cm2_d$Reference) # diagonal starts at top left
cm2_d$ref_freq <- cm2_d$Freq * ifelse(is.na(cm2_d$diag),-1,1)

plt2 <-  ggplot(data = cm2_d, aes(x = Prediction , y =  Reference, fill = Freq))+
  scale_x_discrete(position = "top") +
  geom_tile( data = cm2_d,aes(fill = ref_freq)) +
  scale_fill_gradient2(guide = FALSE ,low="red3",high="orchid4", midpoint = 0,na.value = 'white') +
  geom_text(aes(label = Freq), color = 'black', size = 3)+
  theme_bw() +
  theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank(),
        legend.position = "none",
        panel.border = element_blank(),
        plot.background = element_blank(),
        axis.line = element_blank(),
  )
plt2
```

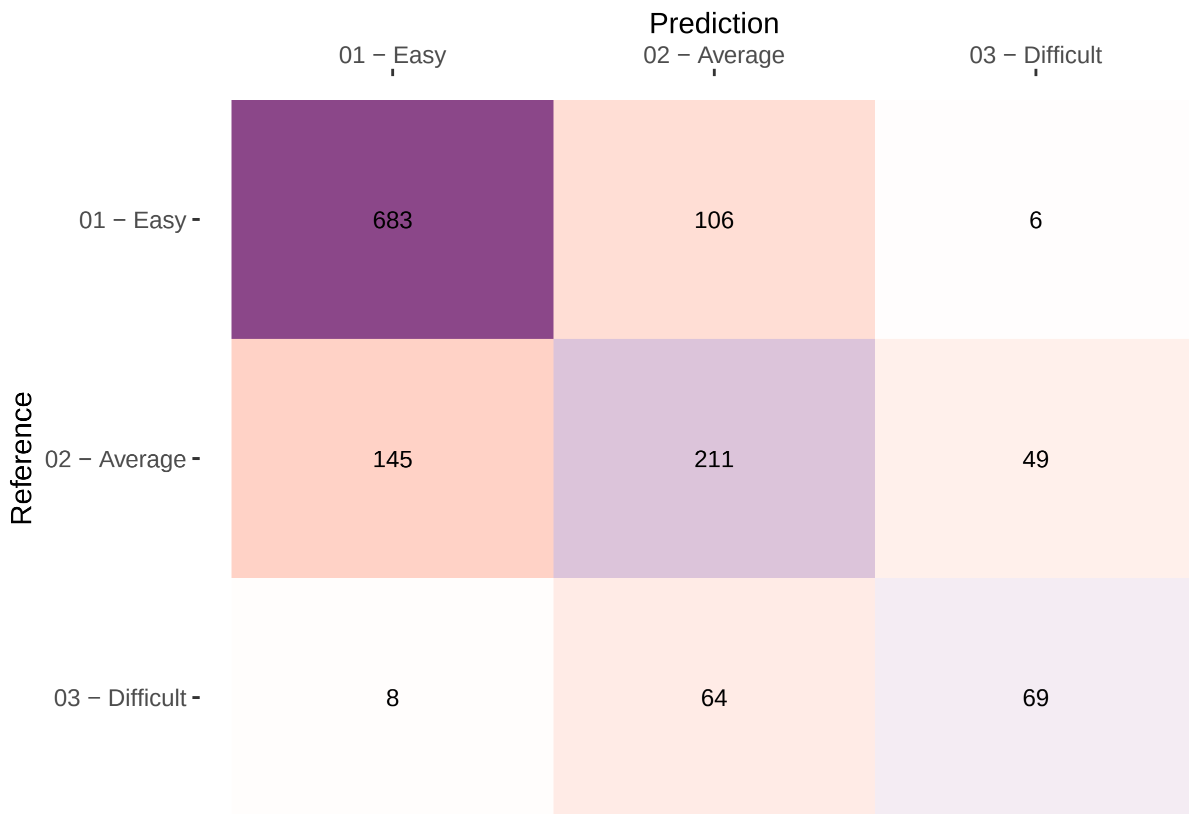

Figure 13: K-Nearest Neighbors Confusion Matrix

```
cm3_d <- as.data.frame(cm3$table)
cm3_d$diag <- cm3_d$Prediction == cm3_d$Reference # Get the Diagonal
cm3_d$ndiag <- cm3_d$Prediction != cm3_d$Reference # Off Diagonal
cm3_d[cm3_d == 0] <- NA # Replace 0 with NA for white tiles
cm3_d$Reference <-  reverse.levels(cm3_d$Reference) # diagonal starts at top left
cm3_d$ref_freq <- cm3_d$Freq * ifelse(is.na(cm3_d$diag),-1,1)

plt3 <-  ggplot(data = cm3_d, aes(x = Prediction , y =  Reference, fill = Freq))+
  scale_x_discrete(position = "top") +
  geom_tile( data = cm3_d,aes(fill = ref_freq)) +
  scale_fill_gradient2(guide = FALSE ,low="red3",high="orchid4", midpoint = 0,na.value = 'white') +
  geom_text(aes(label = Freq), color = 'black', size = 3)+
  theme_bw() +
  theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank(),
        legend.position = "none",
        panel.border = element_blank(),
        plot.background = element_blank(),
        axis.line = element_blank(),
  )
plt3
```

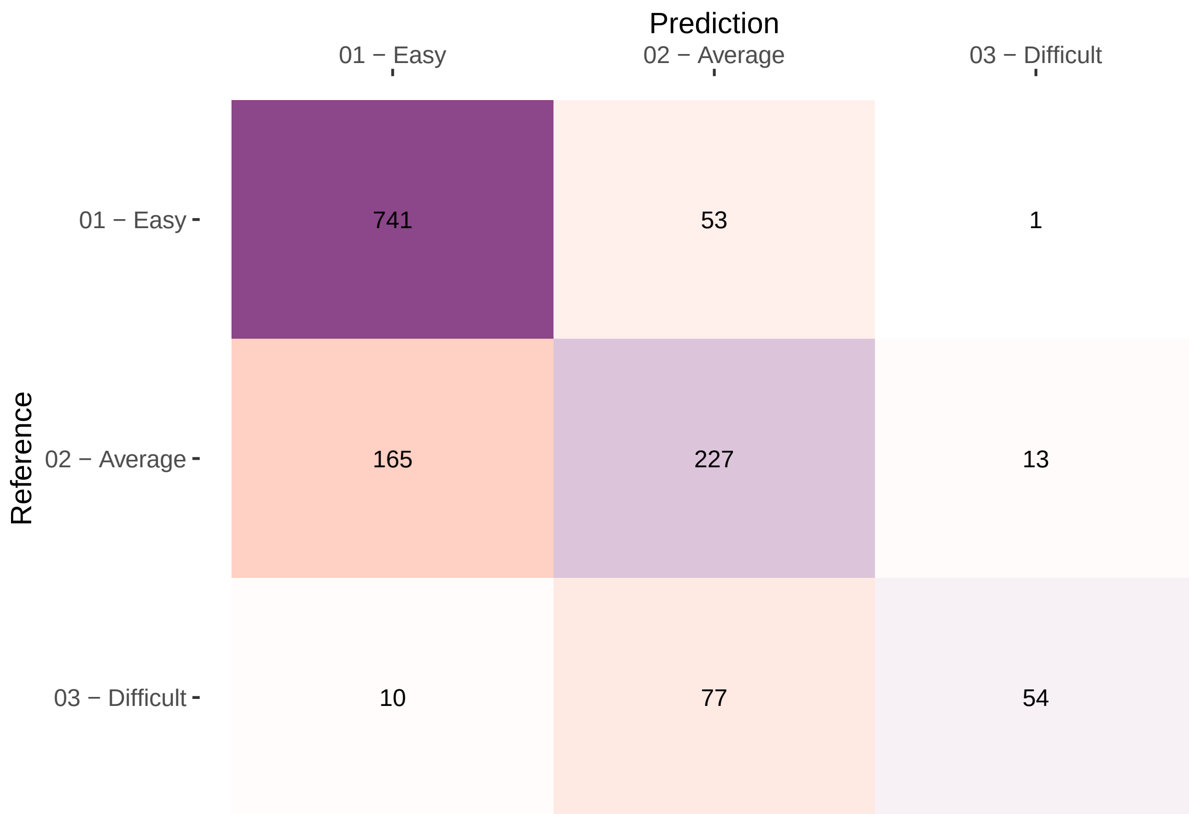

Figure 14: Support Vector Machine Confusion Matrix

b. Model Comapraison

As shown in the section above, all models had little difficulty predicting the "Easy" complexity, but struggled to accurately predict the "Difficult" complexity. Let's now compare the models side by side.

```
#summary results
results <- resamples(list(rf=fit.rf, knn=fit.knn, svm=fit.svm))
summary(results)
```

```
##
## Call:
## summary.resamples(object = results)
##
## Models: rf, knn, svm
## Number of resamples: 10
##
## Accuracy
##          Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## rf  0.7145522 0.7524402 0.7614164 0.7553412 0.7652247 0.7709497    0
## knn 0.7033582 0.7166815 0.7225326 0.7246201 0.7383613 0.7472119    0
## svm 0.7126866 0.7238505 0.7418459 0.7363537 0.7458101 0.7639405    0
##
## Kappa
```

```
##          Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## rf  0.4635372 0.5349393 0.5502041 0.5367327 0.5567105 0.5601416    0
## knn 0.4496636 0.4720554 0.4813925 0.4854257 0.5093189 0.5236569    0
## svm 0.4292352 0.4519308 0.4923957 0.4812971 0.5023558 0.5358446    0
```

```
#plot results
results_df <- as.data.frame(results)

results_tidy <- results_df %>%
  pivot_longer(names_to = "Model", values_to = "Accuracy", -Resample) %>%
  group_by(Model) %>%
  summarise(Mean_Accuracy = mean(Accuracy))

mean_acc <- results_tidy %>%
  ggplot(aes(x=fct_reorder(Model, Mean_Accuracy), y=Mean_Accuracy))+
  geom_bar(stat = "identity", fill ="orangered")+
  coord_flip()+
  xlab("Model")+
  ylab("Mean Accuracy")+
  theme(text = element_text(size = 20))

mean_acc
```

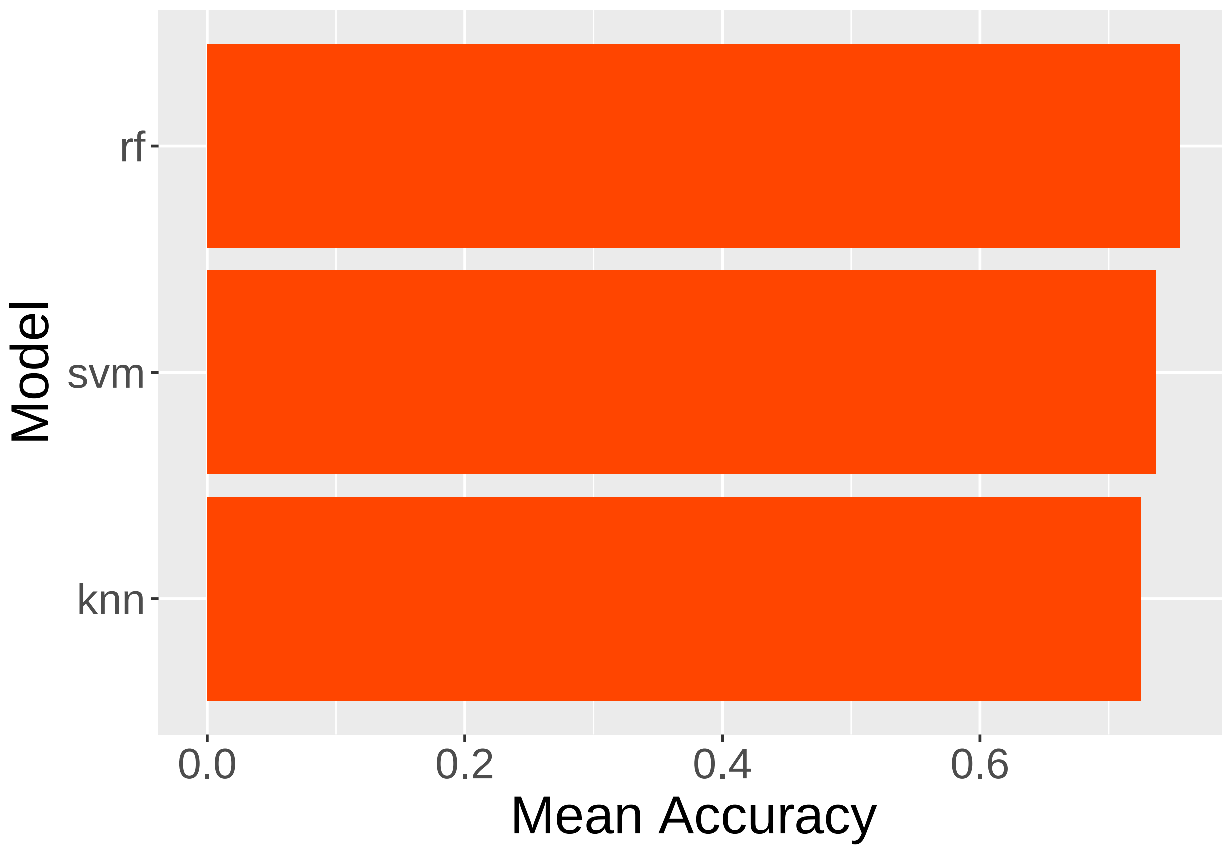

Figure 15: Predictive Models Mean Accuracy Chart

The results of the 10 resamples of the rf, knn, and svm models show that the rf model obtained the highest accuracy score. The knn and svm models had slightly lower accuracy scores. The kappa scores also followed a similar trend. Overall, the rf model appears to have the highest accuracy and kappa scores, making it the best option for the given data set.

c. Variable Importance

```
# determining variable importance
importance1 <- varImp(fit.rf)
importance2 <- varImp(fit.knn)
importance3 <- varImp(fit.svm)

imp1 <- importance1$importance
imp2 <- importance2$importance
imp3 <- importance3$importance
```

```
p1 <- imp1 %>%
  mutate(Predictor = rownames(imp1)) %>%
  pivot_longer(names_to = "Complexity", values_to = "Importance", -Predictor) %>%
  ggplot(aes(x=Predictor, y=Importance))+
  geom_segment(aes(x=Predictor, xend=Predictor, y=0, yend=Importance), color="violet") +
  geom_point(color="purple", size=4, alpha=0.6) +
  theme_light() +
  coord_flip() +
  theme(
    panel.grid.major.y = element_blank(),
    panel.border = element_blank(),
    axis.ticks.y = element_blank())+
  ylab("Random Forest")+
  xlab("")
```

```
p2 <- imp2 %>%
  mutate(Predictor = rownames(imp2)) %>%
  pivot_longer(names_to = "Complexity", values_to = "Importance", -Predictor) %>%
  ggplot(aes(x=Predictor, y=Importance))+
  geom_segment(aes(x=Predictor, xend=Predictor, y=0, yend=Importance), color="violet") +
  geom_point(color="purple", size=4, alpha=0.6) +
  theme_light() +
  coord_flip() +
  theme(
    panel.grid.major.y = element_blank(),
    panel.border = element_blank(),
    axis.ticks.y = element_blank())+
  ylab("KNN")+
  xlab("")
```

```
p3 <- imp3 %>%
  mutate(Predictor = rownames(imp3)) %>%
  pivot_longer(names_to = "Complexity", values_to = "Importance", -Predictor) %>%
  ggplot(aes(x=Predictor, y=Importance))+
  geom_segment(aes(x=Predictor, xend=Predictor, y=0, yend=Importance), color="violet") +
```

```
  geom_point(color="purple", size=4, alpha=0.6) +
  theme_light() +
  coord_flip() +
  theme(
    panel.grid.major.y = element_blank(),
    panel.border = element_blank(),
    axis.ticks.y = element_blank())+
  ylab("SVM")+
  xlab("")
```

```
plot_importance <- ggarrange(p1, p2, p3, ncol=1)
plot_importance
```

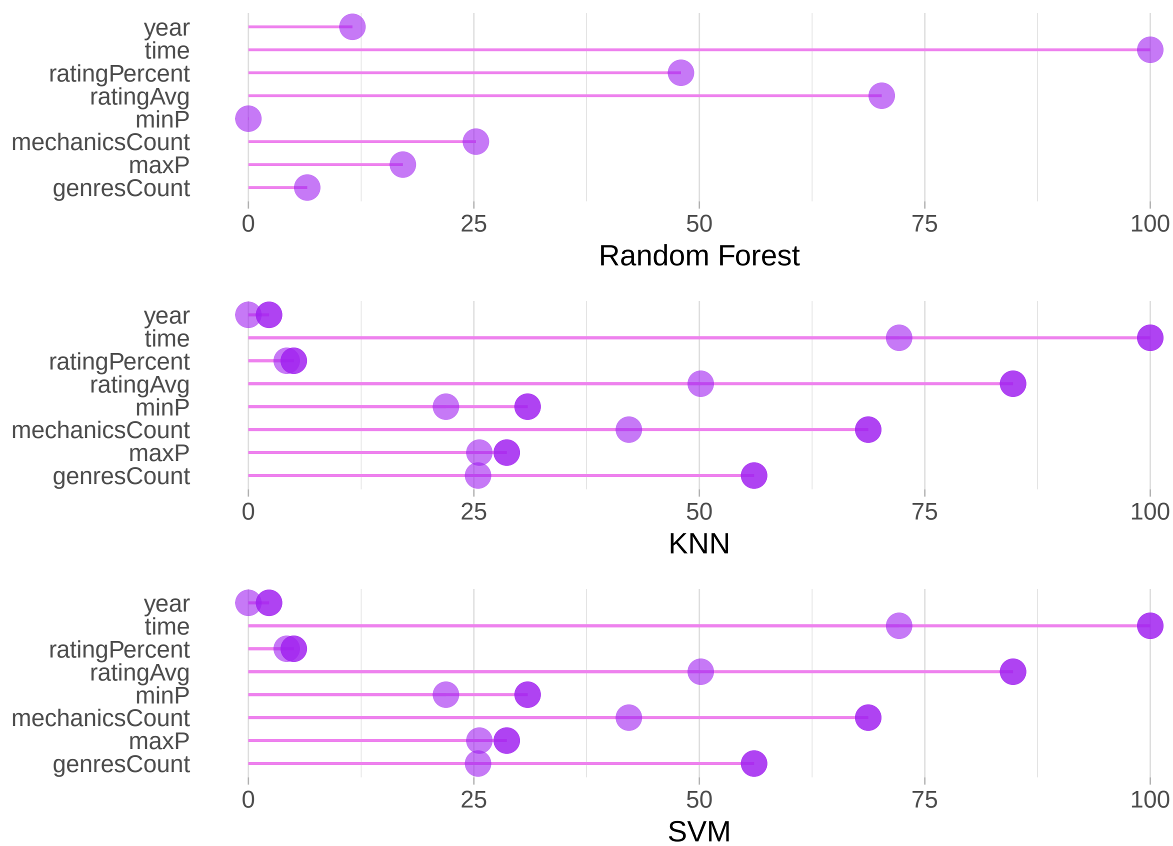

Figure 16: Predictors Importance by Predictive Model

The results from the three predictive models indicate that time, ratingAvg, and mechanicsCount are the most important variables for predicting the complexity of board games. This suggests that longer games, games with higher average ratings, and games with more mechanics tend to be more complex.

The three least important variables appear to be year, maxP, and minP, which suggests that the age of the game, the maximum number of players, and the minimum number of players are not significant factors in predicting the complexity of a game.

6. Conclusion

In conclusion, the Random Forest model was found to be the most accurate model for predicting game complexity in the board game dataset. Although the Support Vector Machine model was not far behind, the K-Nearest Neighbors model was found to be the least accurate.

This analysis shows that when predicting game complexity from board game datasets, Random Forest is the most suitable model.

7. References

Samarasinghe, D. (2022, May 18). *BoardGameGeek dataset on board games.* IEEE DataPort. Retrieved December 12, 2022, from https://ieee-dataport.org/open-access/boardgamegeek-dataset-board-games

A grammar of data manipulation. A Grammar of Data Manipulation • . (n.d.). Retrieved December 12, 2022, from https://dplyr.tidyverse.org/

Create elegant data visualisations using the grammar of graphics. Create Elegant Data Visualisations Using the Grammar of Graphics • . (n.d.). Retrieved December 12, 2022, from https://ggplot2.tidyverse.org/