IOS App Rating Modeling & Analysis

Group 4B Yuhong Lu, Xiaohan Mei, Ziyan Pei,Peng Yuan, Mengqing Zhang, Jiayuan Zou

Oct 15th 2019

Load packages and environment

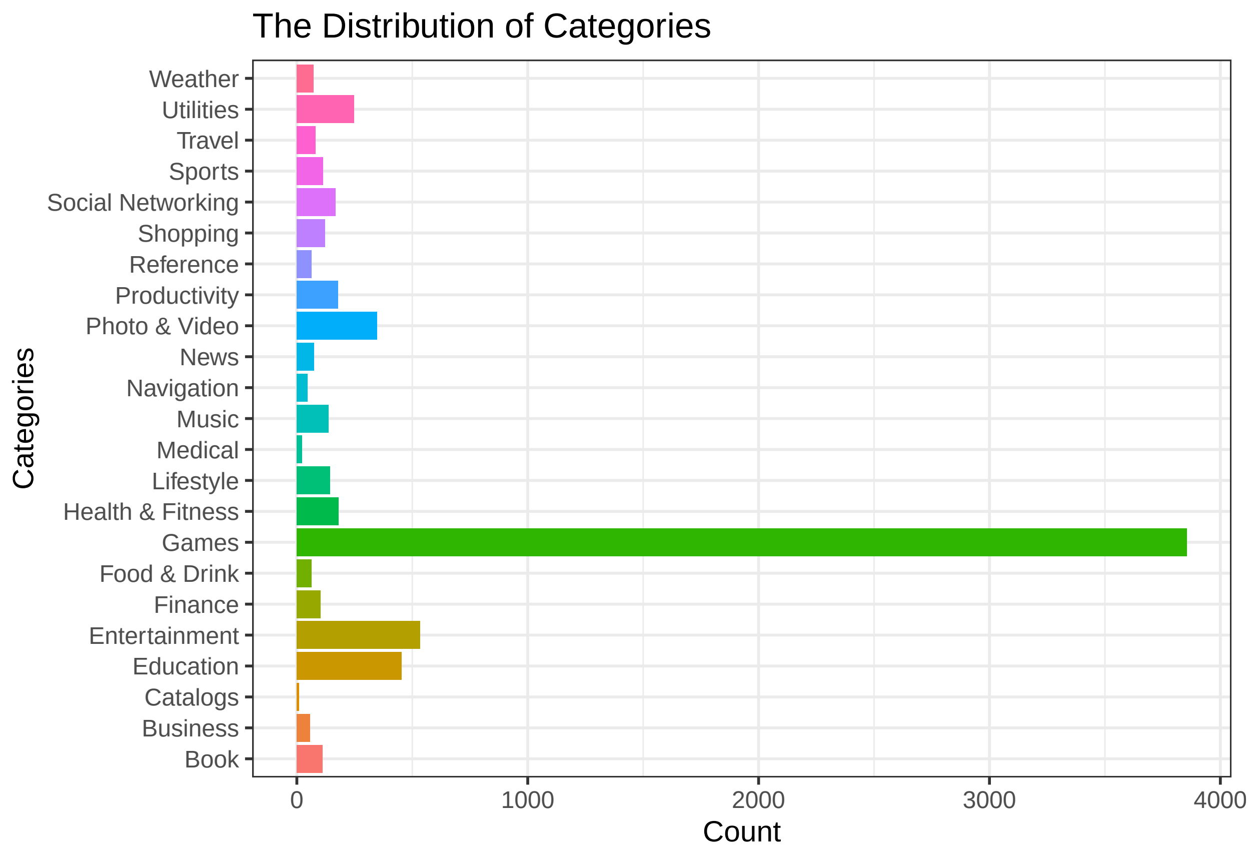
Data Cleaning

```
ap_omit <- read_csv('AP_omit.csv')
# Remove the '+' in `cont_rating` column so that the data
# could be read by r
ap_omit$cont_rating <- str_replace(ap_omit$cont_rating, "[+]", "")
# change the class of variables in order to generate matrix and model for the further use
ap_omit$vpp_lic <- as.factor(ap_omit$vpp_lic)
ap_omit$cont_rating <- as.numeric(ap_omit$cont_rating)
ap_omit$prime_genre <- as.factor(ap_omit$prime_genre)
```

Descriptive Statistics

Distribution of Categories

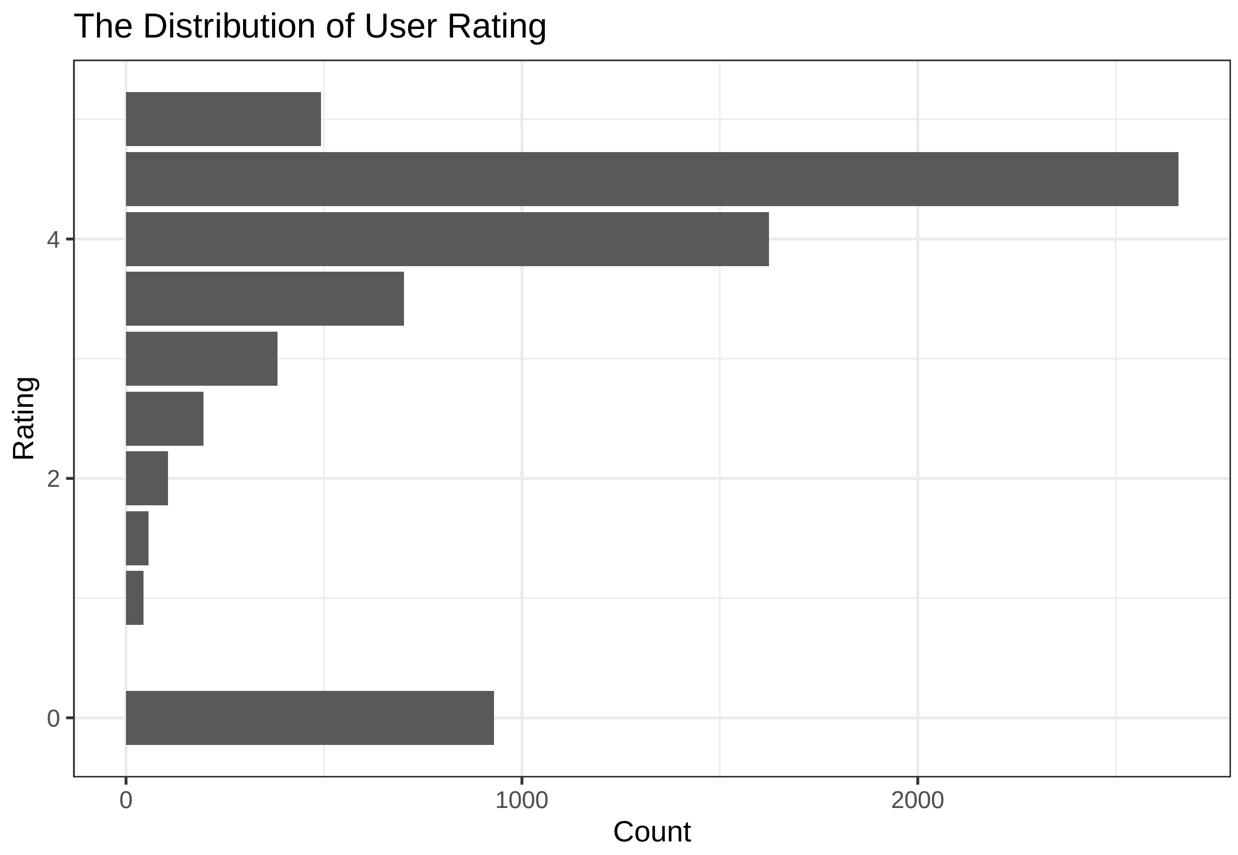
```
ggplot(data=ap_omit) +
  geom_bar(aes(x=prime_genre, fill=prime_genre),show.legend = FALSE)+
  labs(x = "Categories", y = "Count",
        title = "The Distribution of Categories")+
  coord_flip() + theme_bw()
```



This bar chart shows the number of the apps are stored in the apple store and what categories that they belong to. It can tell us what's the majority category of apps that people are used right now, and which categories of apps will attract people to download most.

Distribution of User Rating

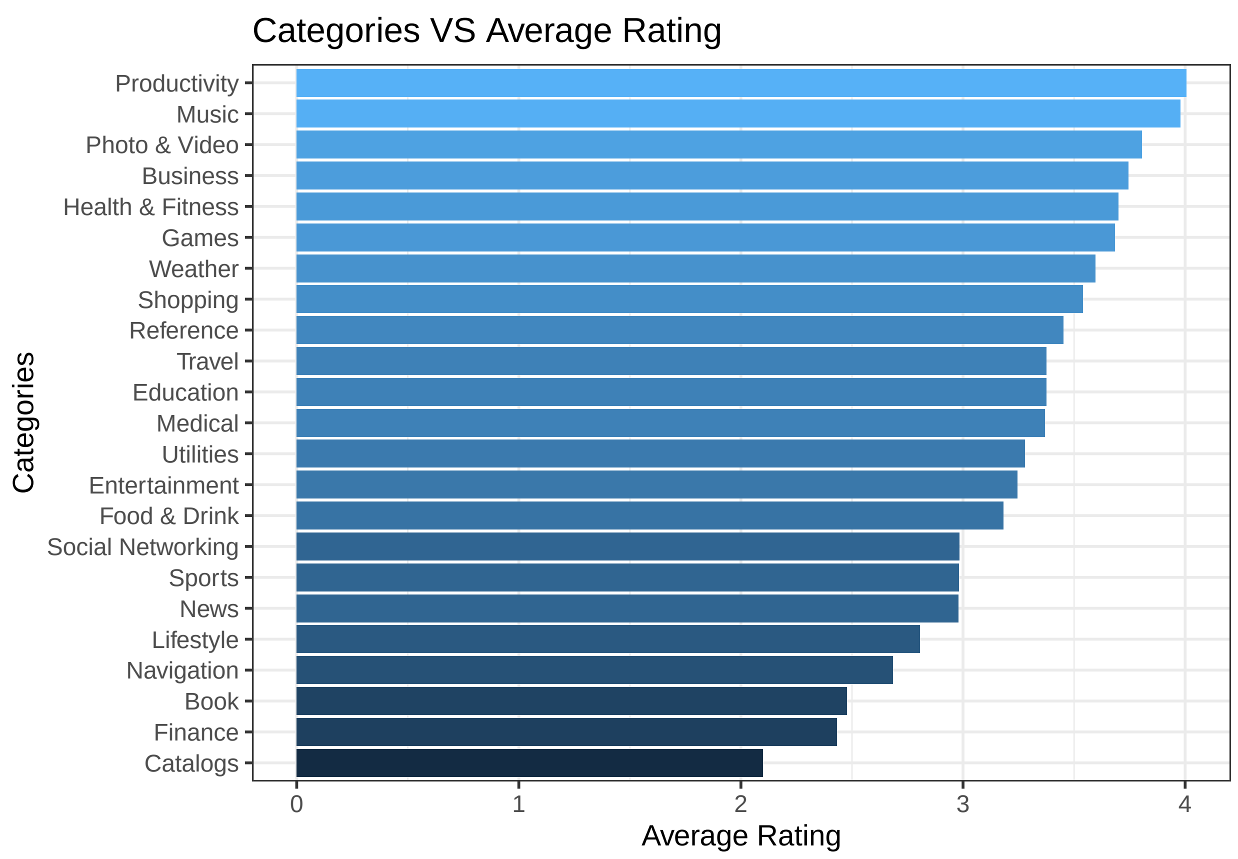
```r
ggplot(data=ap_omit) +
  geom_bar(aes(x=user_rating),show.legend = FALSE)+
  labs(x = "Rating", y = "Count",
        title = "The Distribution of User Rating")+
  coord_flip() + theme_bw()
```



This bar chart shows the distribution of user rating. When we are using ios apps, based on whether we enjoy its function, we can give ratings from one star to five star, and in this chart, we can tell, the number of comments for each level of rating. As 4.5 star is the most common rating, many people also give 0 star.

Categories VS Average Rating(descending)

```
avg_categ <- ap_omit%>%
  group_by(prime_genre)%>%
  summarise(average_rating = mean(user_rating))%>%
  arrange(desc(average_rating))

ggplot(data = avg_categ, aes(reorder(prime_genre,
                                     average_rating),
                             y = average_rating, fill = average_rating))+
  geom_bar(stat = "identity", show.legend = FALSE)+
  labs (x = "Categories", y = "Average Rating",
        title = "Categories VS Average Rating") +
  coord_flip()
```



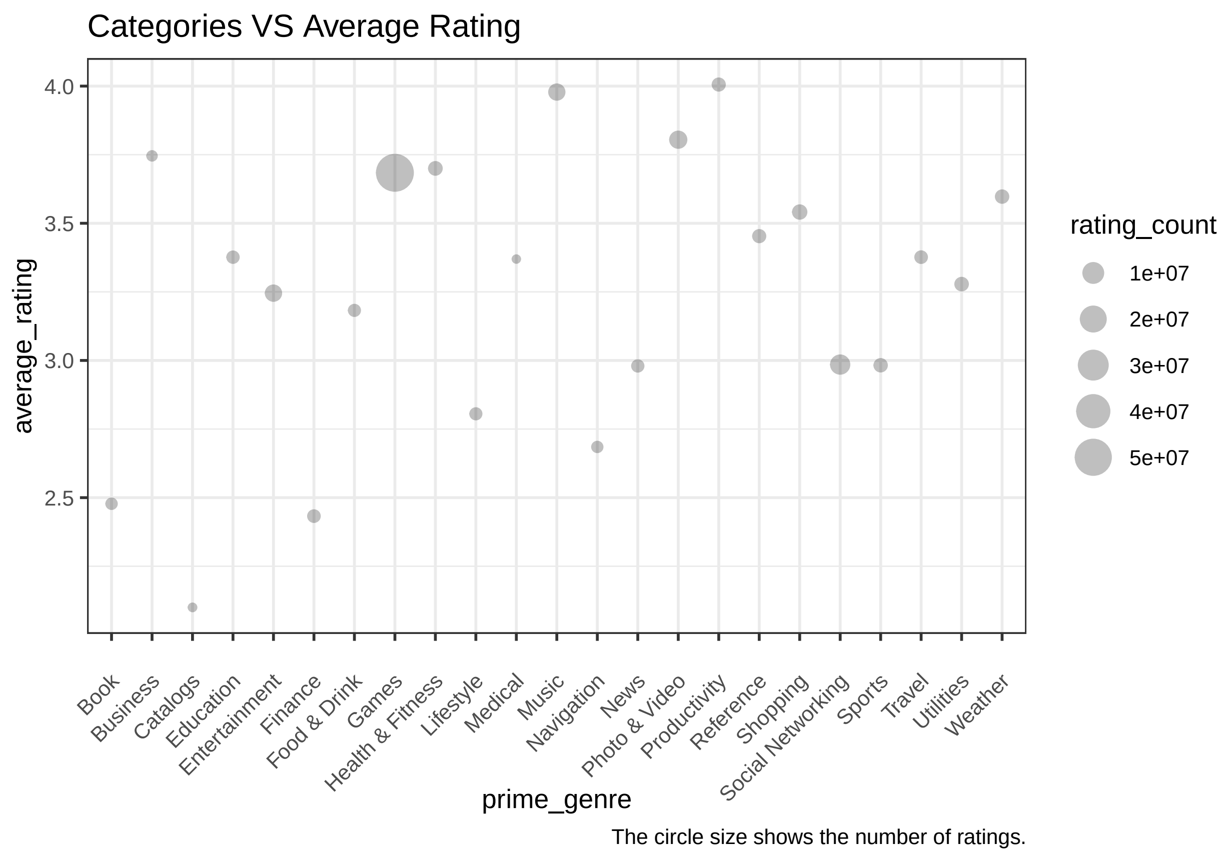
The Categories VS Average Rating shows that in each specific category, what rate people usually give out. For example, the productivty and music category are the top two categories which get almost four out of five stars. And catelogs are the lowest category that only get a little bit over two points.

Categories VS Average Rating(number of)

```
rating_cate <- ap_omit%>%
   group_by(prime_genre)%>%
  summarize(average_rating = mean(user_rating, na.rm = TRUE),
            rating_count = sum(rating_count_tot, na.rm = TRUE)) %>%
   print
```

```
## # A tibble: 23 x 3
##    prime_genre      average_rating rating_count
##    <fct>                     <dbl>        <dbl>
##  1 Book                       2.48       574049
##  2 Business                   3.75       272921
##  3 Catalogs                   2.1         17325
##  4 Education                  3.38      1014371
##  5 Entertainment              3.25      4030339
##  6 Finance                    2.43      1148956
##  7 Food & Drink               3.18       878133
##  8 Games                      3.68     52872546
##  9 Health & Fitness           3.7       1784371
## 10 Lifestyle                  2.81       887294
## # ... with 13 more rows
```

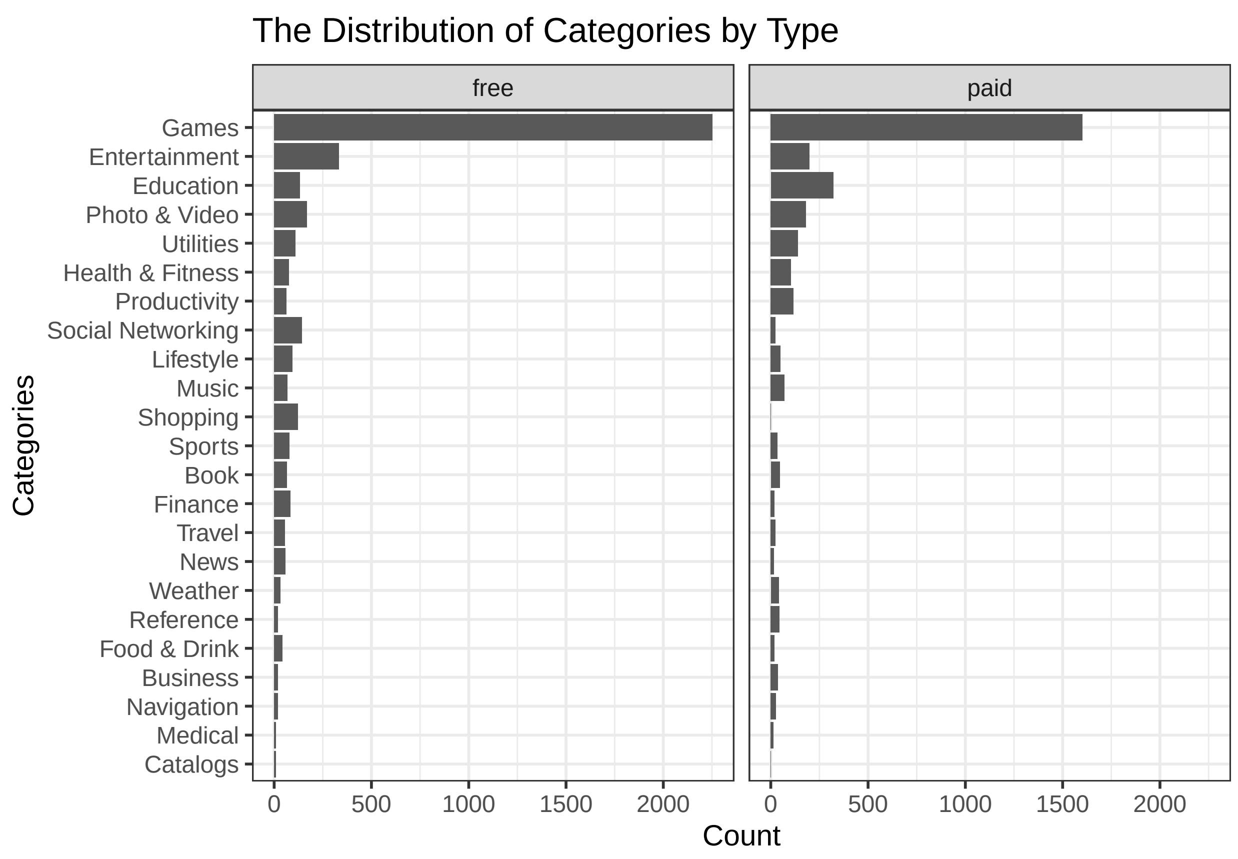
```r
ggplot(data = rating_cate, mapping = aes(x = prime_genre, y = average_rating)) +
  geom_point(aes(size = rating_count), alpha = 1/4) +
  labs(title = "Categories VS Average Rating",
       caption = "The circle size shows the number of ratings.") +
  theme(text = element_text(size = 10),
      axis.text.x = element_text(angle = 45, hjust = 1, vjust = 0.8))
```



This chart also shows the relationship between categories and user rating but in a more visual way. The circle size shows the number of ratings and if the size is large and up high means this category gets lots of good ratings.

The Distribution of Categories by Type(Free and Paid)

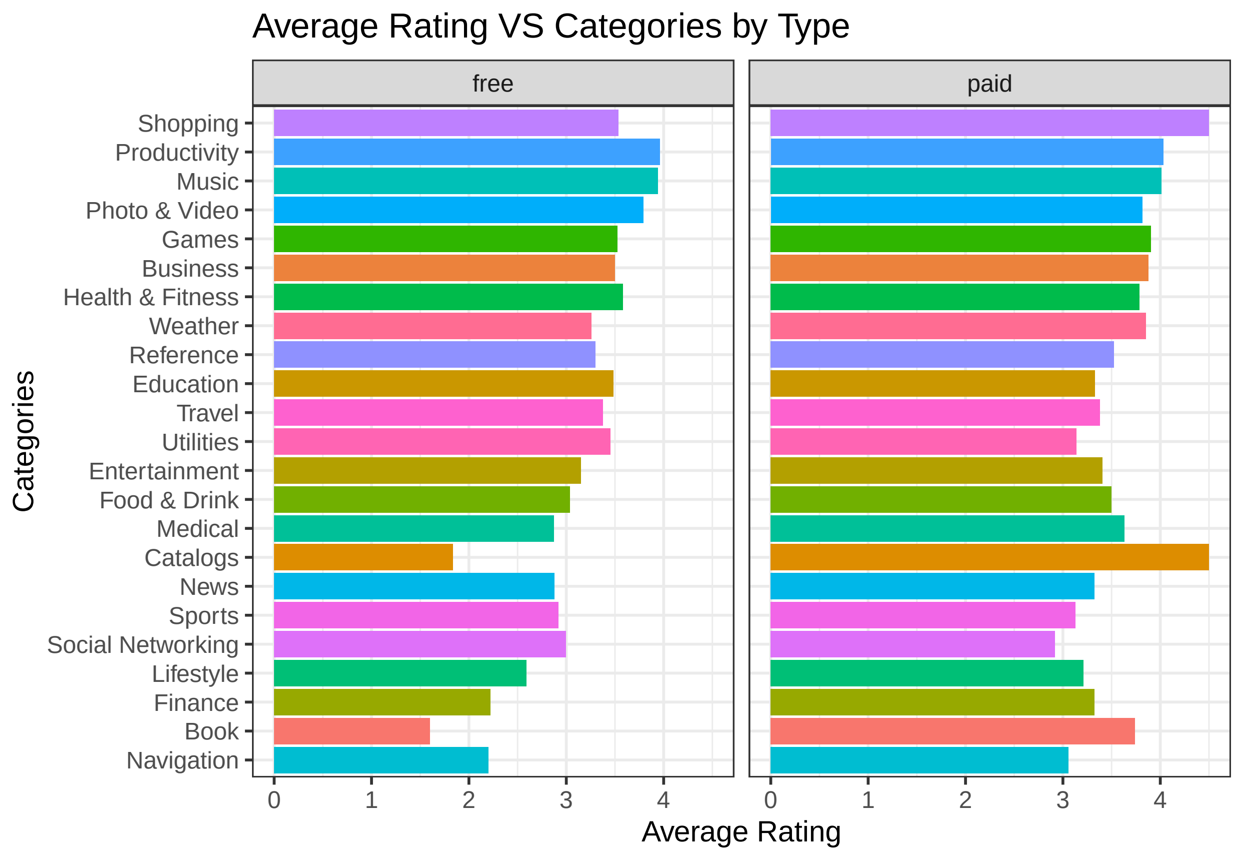
```r
type_count_cate <- ap_omit%>%
  mutate(type=ifelse(price==0,"free","paid"))%>%
  group_by(type,prime_genre)%>%
  summarise(count=n())

ggplot(data=type_count_cate,aes(reorder(prime_genre,count),y=count)) +
  geom_bar(stat = "identity") +
  labs (x="Categories", y="Count",
        title="The Distribution of Categories by Type") +
  coord_flip() +
  facet_wrap(~type)
```



There are free/paid, two different type of apps in the apple store, and this chart analyze the number of free and paid apps in different categories. We can tell game related apps occupied a huge proportion of the overall market share and the second top category will be entertainment apps while the least is catalog apps.

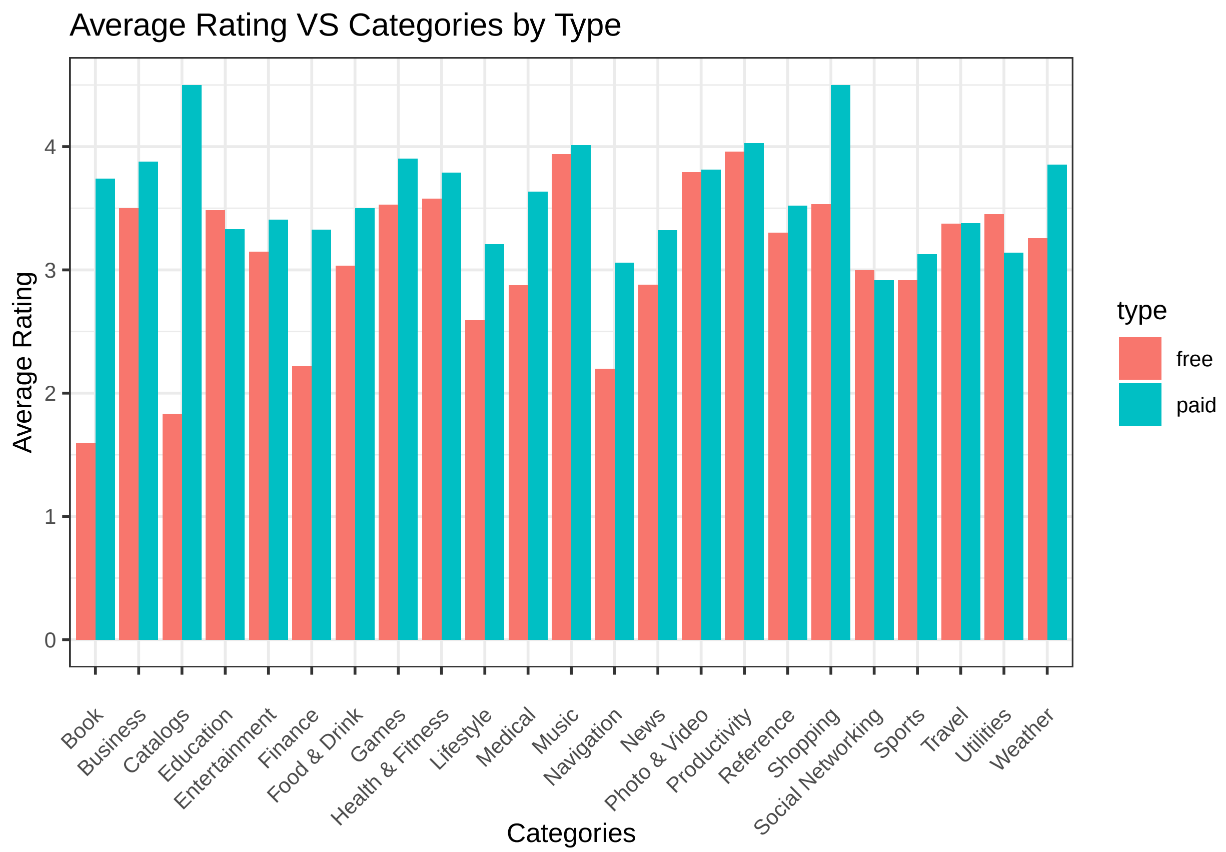
The Distribution of User Rating VS Categories by Type(Free and Paid)

```
type_rating_cate <- ap_omit%>%
  mutate(type = ifelse(price == 0, "free", "paid"))%>%
  group_by(type, prime_genre)%>%
  summarise(average_rating = mean(user_rating))

ggplot(data = type_rating_cate, aes(reorder(prime_genre, average_rating), y = average_rating)) +
  geom_bar(stat = "identity",aes(fill=prime_genre),show.legend = F) +
  labs (x = "Categories", y = "Average Rating",
        title = "Average Rating VS Categories by Type") +
  coord_flip() +
  facet_wrap(~type)
```



In this chart, we added average rating as the third variable into comparision. It is more obvious for us to tell the difference of the rating between free and paid apps in the same category. The top three paid categories that get high ratings are shopping, productivity, and music. For these apps there aren't a significant difference between free and paid apps. And for catalogs, the free apps only gets around two out of five, while the paid apps gets almost four point five out of five.

```
ggplot(data = type_rating_cate, aes(x = prime_genre, y = average_rating, fill=type))+
  geom_bar(stat = "identity", position = position_dodge())+
  labs (x = "Categories", y = "Average Rating",
        title = "Average Rating VS Categories by Type")+
  theme(text = element_text(size=10),
        axis.text.x = element_text(angle=45,hjust=1,vjust=0.8))
```



This chart shows the same relationship as the previous one but in a different version. It put the free and paid apps of the same category side by side and mark the free and paid apps into two different colors. It would be more obvious to analyze the relationship different variables.

Test and Train data split

```r
# this is needed to create reproducible results,
# otherwise your random split will be different each time you run the code
set.seed(12345)
# This will split into train and test 70-30
ap_omit$train <- sample(c(0, 1), nrow(ap_omit), replace = TRUE, prob = c(.3, .7))
ap_test <- ap_omit %>% filter(train == 0)
ap_train <- ap_omit %>% filter(train == 1)
y_train <- ap_train$user_rating
y_test <- ap_test$user_rating
```

Create the variable that we use

In this project, we want to predict how the rating for a specific APP would be given 9 independent variables. Therefore, we set up `user_rating` as our dependent variable; 9 independent variables that we adopted are `price`, `rating_count_tot` (the number of ratings for an App), `cont_rating` (content rating), `sup_devices.num` (number of supporting devices), `size_bytes` (the size of app in bytes), `prime_genre` (the category of the app), `ipadSc_urls.num`(Number of screenshots showed for display), `lang.num` (Number of supported languages), and `vpp_lic`(Vpp Device Based Licensing Enabled,1 means APP uses device-based liscence, 0 means APP uses apple-id based liscence).

```
f1 <- as.formula(user_rating~ price + rating_count_tot +
                   cont_rating +sup_devices.num +size_byte+
                  prime_genre+ipadSc_urls.num+lang.num+vpp_lic)
```

Linear Regression without cross validation

```
ap_lm <- lm(f1, ap_train)
yhat_train_lm <- predict(ap_lm)
mse_train_lm <- mean((y_train - yhat_train_lm)^2)
yhat_test_lm <- predict(ap_lm, ap_test)
mse_test_lm <- mean((y_test - yhat_test_lm)^2)
coef(ap_lm)
```

```
##                  (Intercept)                        price
##                 1.497873e+00                 7.241523e-03
##             rating_count_tot                  cont_rating
##                 1.341094e-06                -1.348505e-02
##              sup_devices.num                    size_byte
##                -1.064893e-02                 1.199909e-10
##          prime_genreBusiness          prime_genreCatalogs
##                 8.506596e-01                -6.184893e-01
##         prime_genreEducation     prime_genreEntertainment
##                 5.356894e-01                 8.071965e-01
##           prime_genreFinance      prime_genreFood & Drink
##                 3.263427e-01                 9.553925e-01
##             prime_genreGames prime_genreHealth & Fitness
##                 1.029728e+00                 1.393644e+00
##         prime_genreLifestyle           prime_genreMedical
##                 5.647699e-01                 6.147764e-01
##             prime_genreMusic        prime_genreNavigation
##                 1.236721e+00                 3.924685e-01
##              prime_genreNews     prime_genrePhoto & Video
##                 8.150043e-01                 1.242851e+00
##      prime_genreProductivity         prime_genreReference
##                 1.216735e+00                 8.102167e-01
##          prime_genreShopping prime_genreSocial Networking
##                 1.359421e+00                 7.622821e-01
##            prime_genreSports             prime_genreTravel
##                 7.559700e-01                 8.160684e-01
##         prime_genreUtilities           prime_genreWeather
##                 8.437295e-01                 8.707539e-01
##              ipadSc_urls.num                     lang.num
##                 1.743684e-01                 2.265787e-02
##                     vpp_lic1
##                 7.543220e-01
```

```
mse_train_lm
```

```
## [1] 2.02616
```

```r
mse_test_lm
```

```
## [1] 1.986448
```

We first tried to use the linear regression without cross validation, and get the train and test mse, but we found that the test mse is less then the train mse, then we try to use the cross validation to get a better result.

Linear regression with cross validation

```r
set.seed(12345)

ap_folds <- createFolds(ap_train$user_rating, k = 10)
k <- length(ap_folds)
lmmse <- data.frame()
model <- list()
for(i in 1 : k){
  train.data <- ap_train[-ap_folds[[i]], ]
  valid.data <- ap_train[ap_folds[[i]], ]
  train_y <- y_train[-ap_folds[[i]]]
  valid_y <- y_train[ap_folds[[i]]]
  model[[i]] <- lm(f1, train.data)
  y_train_hat <- predict(model[[i]], train.data)
  y_valid_hat <- predict(model[[i]], valid.data)
  lmmse[i, 1] <- mean((y_train_hat - train_y) ^ 2)
  lmmse[i, 2] <- mean((y_valid_hat - valid_y) ^ 2)
}
names(lmmse) <- c('Train.MSEs', 'Valid.MSEs')
lmmse.min.index <- which.min(lmmse$Valid.MSEs)
lmmse[lmmse.min.index, ]
```

```
##   Train.MSEs Valid.MSEs
## 5   2.050166   1.823457
```

```r
mse_test <- mean((predict(model[[lmmse.min.index]], ap_test) - ap_test$user_rating) ^ 2)
mse_test
```

```
## [1] 1.983673
```

```r
summary(model[[lmmse.min.index]])
```

```
##
## Call:
## lm(formula = f1, data = train.data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.3128 -0.2746  0.4001  0.8386  3.4453
##
```

```
## Coefficients:
##                              Estimate Std. Error t value Pr(>|t|)    
## (Intercept)                 1.742e+00  4.133e-01   4.215 2.55e-05 ***
## price                       7.312e-03  3.913e-03   1.869 0.061713 .  
## rating_count_tot            1.333e-06  3.006e-07   4.433 9.51e-06 ***
## cont_rating                -1.508e-02  5.186e-03  -2.908 0.003650 ** 
## sup_devices.num            -1.109e-02  6.057e-03  -1.832 0.067088 .  
## size_byte                   9.416e-11  6.680e-11   1.410 0.158719    
## prime_genreBusiness         8.931e-01  3.103e-01   2.878 0.004024 ** 
## prime_genreCatalogs        -9.860e-01  6.121e-01  -1.611 0.107286    
## prime_genreEducation        5.437e-01  1.960e-01   2.775 0.005547 ** 
## prime_genreEntertainment    7.790e-01  1.916e-01   4.065 4.89e-05 ***
## prime_genreFinance          2.418e-01  2.432e-01   0.994 0.320082    
## prime_genreFood & Drink     9.779e-01  3.024e-01   3.234 0.001231 ** 
## prime_genreGames            1.026e+00  1.775e-01   5.781 7.91e-09 ***
## prime_genreHealth & Fitness  1.388e+00  2.185e-01   6.354 2.31e-10 ***
## prime_genreLifestyle        5.321e-01  2.282e-01   2.331 0.019775 *  
## prime_genreMedical          7.380e-01  4.363e-01   1.691 0.090822 .  
## prime_genreMusic            1.251e+00  2.283e-01   5.478 4.53e-08 ***
## prime_genreNavigation       3.874e-01  3.121e-01   1.241 0.214605    
## prime_genreNews             7.944e-01  2.701e-01   2.942 0.003281 ** 
## prime_genrePhoto & Video    1.232e+00  2.017e-01   6.105 1.11e-09 ***
## prime_genreProductivity     1.265e+00  2.285e-01   5.535 3.30e-08 ***
## prime_genreReference        8.793e-01  2.805e-01   3.134 0.001735 ** 
## prime_genreShopping         1.323e+00  2.383e-01   5.552 2.99e-08 ***
## prime_genreSocial Networking 7.390e-01  2.252e-01   3.282 0.001037 ** 
## prime_genreSports           6.619e-01  2.454e-01   2.697 0.007021 ** 
## prime_genreTravel           8.593e-01  2.789e-01   3.081 0.002077 ** 
## prime_genreUtilities        8.751e-01  2.103e-01   4.160 3.24e-05 ***
## prime_genreWeather          9.958e-01  2.753e-01   3.617 0.000301 ***
## ipadSc_urls.num             1.729e-01  1.219e-02  14.184  < 2e-16 ***
## lang.num                    2.184e-02  2.745e-03   7.955 2.25e-15 ***
## vpp_lic1                    5.501e-01  2.782e-01   1.977 0.048050 *  
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 1.437 on 4507 degrees of freedom
## Multiple R-squared:  0.1266, Adjusted R-squared:  0.1208 
## F-statistic: 21.78 on 30 and 4507 DF,  p-value: < 2.2e-16
```

In this section, we use Cross Validation to train the better Linear Regression Model. By using the For Loop, we can split the training data into 10 folds and using each of the fold for training the model. As the result, we can got a model with the lowest validation MSE = 1.823. Then we got the train MSE = 2.05. Though the train MSE is higher than our validation MSE, which indicates that our model might not be strong, our linear regression model with cv is not overfitting. Also, its MSE are lower than linear regression without cv.

Shrinkage Analysis

Use ridge and lasso model get mse, then we compare the models by mse. ### Ridge

```
x1_train <- model.matrix(f1, ap_train)[, -1]
x1_test <- model.matrix(f1, ap_test)[, -1]
```

```r
ap_ridge <- cv.glmnet(x1_train, y_train, alpha = 0, nfolds = 10)
coef(ap_ridge)
```

```
## 31 x 1 sparse Matrix of class "dgCMatrix"
##                                         1
## (Intercept)                  2.982447e+00
## price                        3.964596e-03
## rating_count_tot             7.300481e-07
## cont_rating                 -8.711897e-03
## sup_devices.num             -6.860408e-03
## size_byte                    8.869838e-11
## prime_genreBusiness         -6.322355e-03
## prime_genreCatalogs         -6.846062e-01
## prime_genreEducation        -1.198654e-01
## prime_genreEntertainment    -8.813053e-02
## prime_genreFinance          -3.617269e-01
## prime_genreFood & Drink     -6.265676e-02
## prime_genreGames             1.102607e-01
## prime_genreHealth & Fitness  1.881990e-01
## prime_genreLifestyle        -2.419812e-01
## prime_genreMedical          -1.657166e-01
## prime_genreMusic             1.729305e-01
## prime_genreNavigation       -2.679581e-01
## prime_genreNews             -1.044540e-01
## prime_genrePhoto & Video     1.528310e-01
## prime_genreProductivity      1.832259e-01
## prime_genreReference        -1.527038e-02
## prime_genreShopping          1.326129e-01
## prime_genreSocial Networking -1.447516e-01
## prime_genreSports           -1.223468e-01
## prime_genreTravel           -7.085027e-02
## prime_genreUtilities        -6.608569e-02
## prime_genreWeather          -1.895399e-02
## ipadSc_urls.num              8.119122e-02
## lang.num                     1.210451e-02
## vpp_lic1                     4.173206e-01
```

```r
yhat_train_ridge <- predict(ap_ridge, x1_train, s = ap_ridge$lambda.min)
mse_train_ridge <- mean((y_train - yhat_train_ridge)^2)
yhat_test_ridge <- predict(ap_ridge,x1_test,s=ap_ridge$lambda.min)
mse_test_ridge <- mean((y_test-yhat_test_ridge)^2)
mse_train_ridge
```

```
## [1] 2.030358
```

```r
mse_test_ridge
```

```
## [1] 1.984174
```

Lasso

```
ap_lasso <- cv.glmnet(x1_train, y_train, alpha = 1, nfolds = 10)
yhat_train_lasso <- predict(ap_lasso, x1_train, s = ap_lasso$lambda.min)
yhat_test_lasso <- predict(ap_lasso,x1_test,s=ap_lasso$lambda.min)
mse_train_lasso <- mean((y_train - yhat_train_lasso)^2)
mse_test_lasso <- mean((y_test-yhat_test_lasso)^2)
coef(ap_lasso)
```

```
## 31 x 1 sparse Matrix of class "dgCMatrix"
##                                         1
## (Intercept)                  2.788989e+00
## price                        .
## rating_count_tot             5.321534e-07
## cont_rating                 -1.925995e-03
## sup_devices.num              .
## size_byte                    .
## prime_genreBusiness          .
## prime_genreCatalogs          .
## prime_genreEducation        -6.028150e-02
## prime_genreEntertainment     .
## prime_genreFinance          -1.147490e-01
## prime_genreFood & Drink      .
## prime_genreGames             5.578546e-02
## prime_genreHealth & Fitness  1.098305e-01
## prime_genreLifestyle         .
## prime_genreMedical           .
## prime_genreMusic             .
## prime_genreNavigation        .
## prime_genreNews              .
## prime_genrePhoto & Video     6.647378e-02
## prime_genreProductivity      .
## prime_genreReference         .
## prime_genreShopping          .
## prime_genreSocial Networking .
## prime_genreSports            .
## prime_genreTravel            .
## prime_genreUtilities         .
## prime_genreWeather           .
## ipadSc_urls.num              1.643147e-01
## lang.num                     1.829998e-02
## vpp_lic1                     .
```

```
mse_train_lasso
```

```
## [1] 2.026193
```

```
mse_test_lasso
```

```
## [1] 1.985956
```

While MSE for validation set of Ridge is lower than that of Lasso, Lasso is more interpretable by eliminating most of coefficients, and only keeps 9 variables (including dummy variables that generated by `prime_genre`).

Tree - Regression Tree

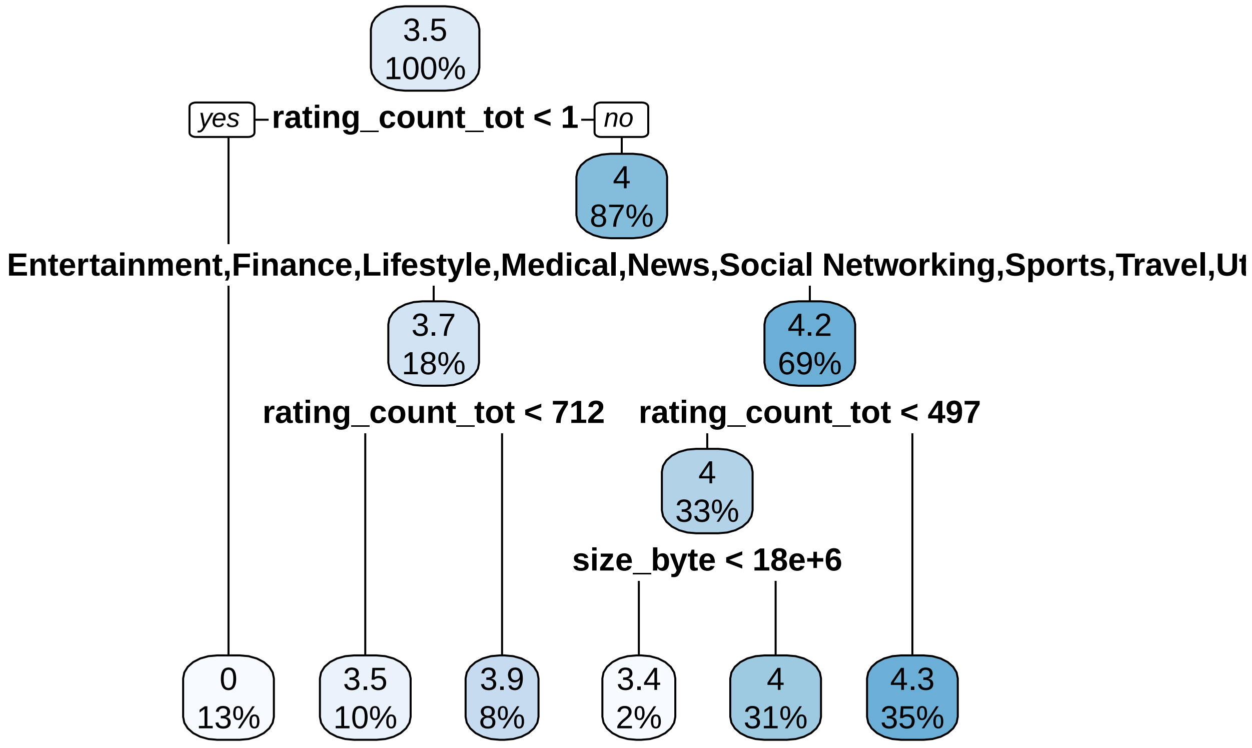
```
#Regression tree
 fit.tree <- rpart(f1,
 ap_train,
 control = rpart.control(cp = 0.001))
 par(xpd = TRUE)

## Printcp will tell you what the cp of spliting into
## diffrent number layer and the xerror and xstd of each cp.
printcp(fit.tree)
```

```
##
## Regression tree:
## rpart(formula = f1, data = ap_train, control = rpart.control(cp = 0.001))
##
## Variables actually used in tree construction:
## [1] ipadSc_urls.num  prime_genre      rating_count_tot size_byte
##
## Root node error: 11709/5042 = 2.3222
##
## n= 5042
##
##          CP nsplit rel error  xerror      xstd
## 1 0.7992540      0   1.00000 1.00048 0.0241363
## 2 0.0143084      1   0.20075 0.20088 0.0067002
## 3 0.0096168      2   0.18644 0.18954 0.0063237
## 4 0.0036879      3   0.17682 0.17799 0.0058148
## 5 0.0027723      4   0.17313 0.17559 0.0057384
## 6 0.0015818      5   0.17036 0.17362 0.0056416
## 7 0.0011615      6   0.16878 0.17457 0.0057007
## 8 0.0011338      9   0.16529 0.17490 0.0057321
## 9 0.0010000     10   0.16416 0.17496 0.0057327
```

```
## use the following method to choose the cp with the smallest xerror
fit.tree$cptable[which.min(fit.tree$cptable[,"xerror"]),"CP"]
```

```
## [1] 0.001581791
```

```
## Build the tree model with the cp which
# has smallest xerror
tree2 <- prune(fit.tree, cp= fit.tree$cptable[which.min(fit.tree$cptable[,"xerror"]),"CP"])
## Make the visuallization of regreesion tree
rpart.plot(tree2)
```



```
## MSE of train
tree.pred.train = predict(tree2,ap_train)
mean((tree.pred.train-ap_train$user_rating)^2)
```

```
## [1] 0.3956153
```

```
## MSE of test
tree.pred.test = predict(tree2,ap_test)
mean((tree.pred.test-ap_test$user_rating)^2)
```

```
## [1] 0.39381
```

```
printcp(tree2)
```

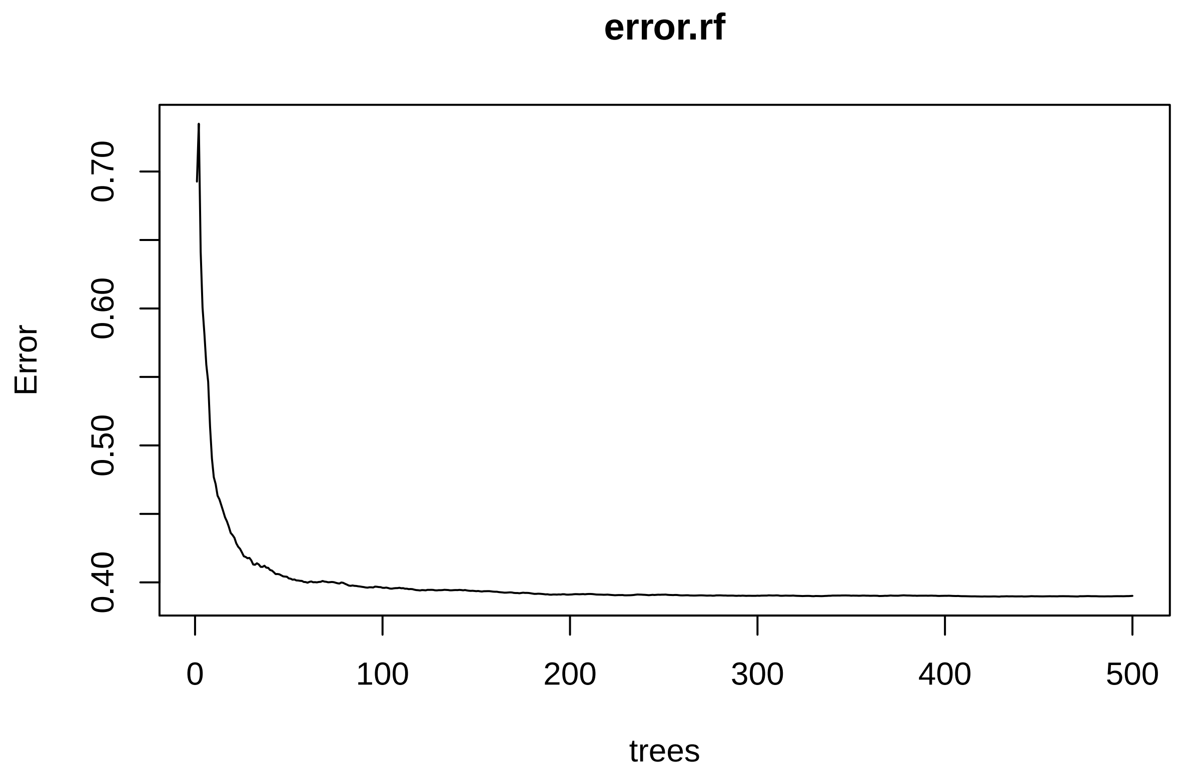
```
##
## Regression tree:
## rpart(formula = f1, data = ap_train, control = rpart.control(cp = 0.001))
##
## Variables actually used in tree construction:
## [1] prime_genre      rating_count_tot size_byte
##
## Root node error: 11709/5042 = 2.3222
##
## n= 5042
```

```
##
##          CP nsplit rel error  xerror      xstd
## 1 0.7992540      0   1.00000 1.00048 0.0241363
## 2 0.0143084      1   0.20075 0.20088 0.0067002
## 3 0.0096168      2   0.18644 0.18954 0.0063237
## 4 0.0036879      3   0.17682 0.17799 0.0058148
## 5 0.0027723      4   0.17313 0.17559 0.0057384
## 6 0.0015818      5   0.17036 0.17362 0.0056416
```

After prune useing the best parameter cp with smallest xerror, we can find that the variables acatually used in tree construction is `prime_genre`,`rating_count_tot` and `size_byte`. The MSE of train is 0.3956153 and the MSE of test is 0.39381.

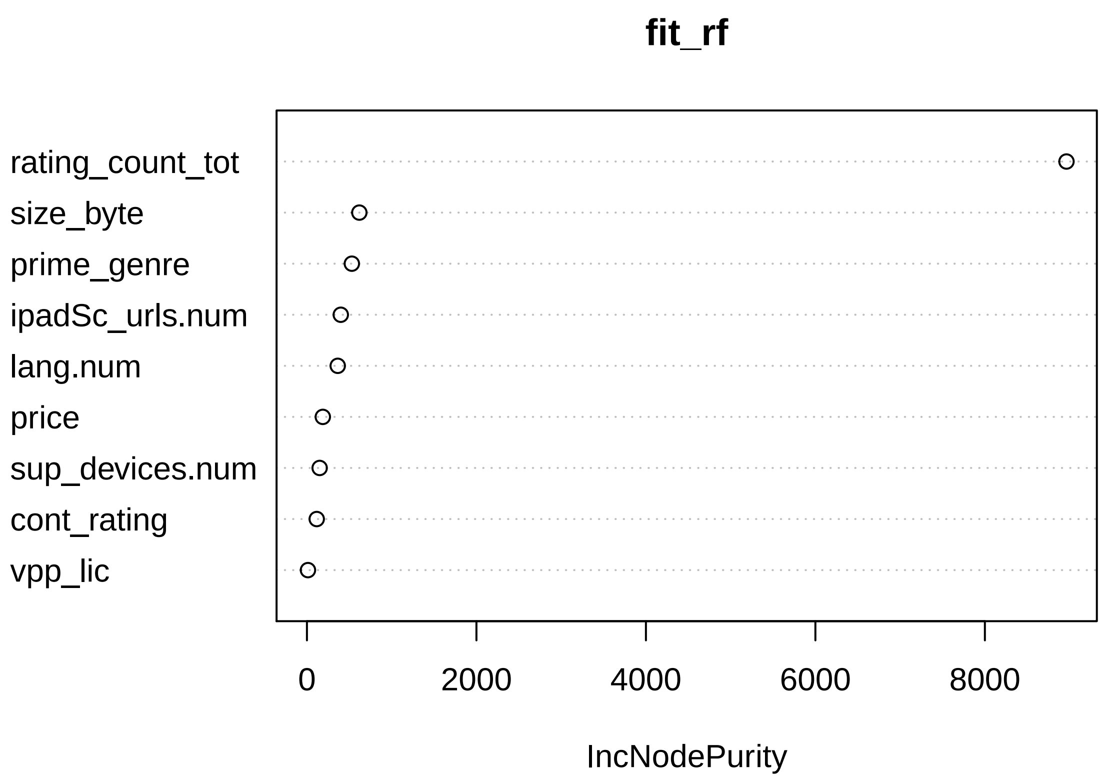
Random Forest

```
#decide ntree by the plot of error vs ntree
error.rf <- randomForest(f1, data = ap_omit , subset = (1:nrow(ap_omit))[ap_omit$train==1])
plot(error.rf)
```



This plot shows the error of the random forest model. As the number of trees increases, the error approaches around 0.3. Thus we choose ntree =300 into the randomForest function.

```r
fit_rf <- randomForest(f1,
                       ap_train,
                       ntree=300,
                       do.trace=F)

varImpPlot(fit_rf)
```



```r
yhat_rf <- predict(fit_rf, ap_train)
train_mse_rf <- mean((yhat_rf - ap_train$user_rating) ^ 2)
print(train_mse_rf)
```

```
## [1] 0.1044315
```

```r
yhat_rf <- predict(fit_rf, ap_test)
test_mse_rf <- mean((yhat_rf - ap_test$user_rating) ^ 2)
print(test_mse_rf)
```

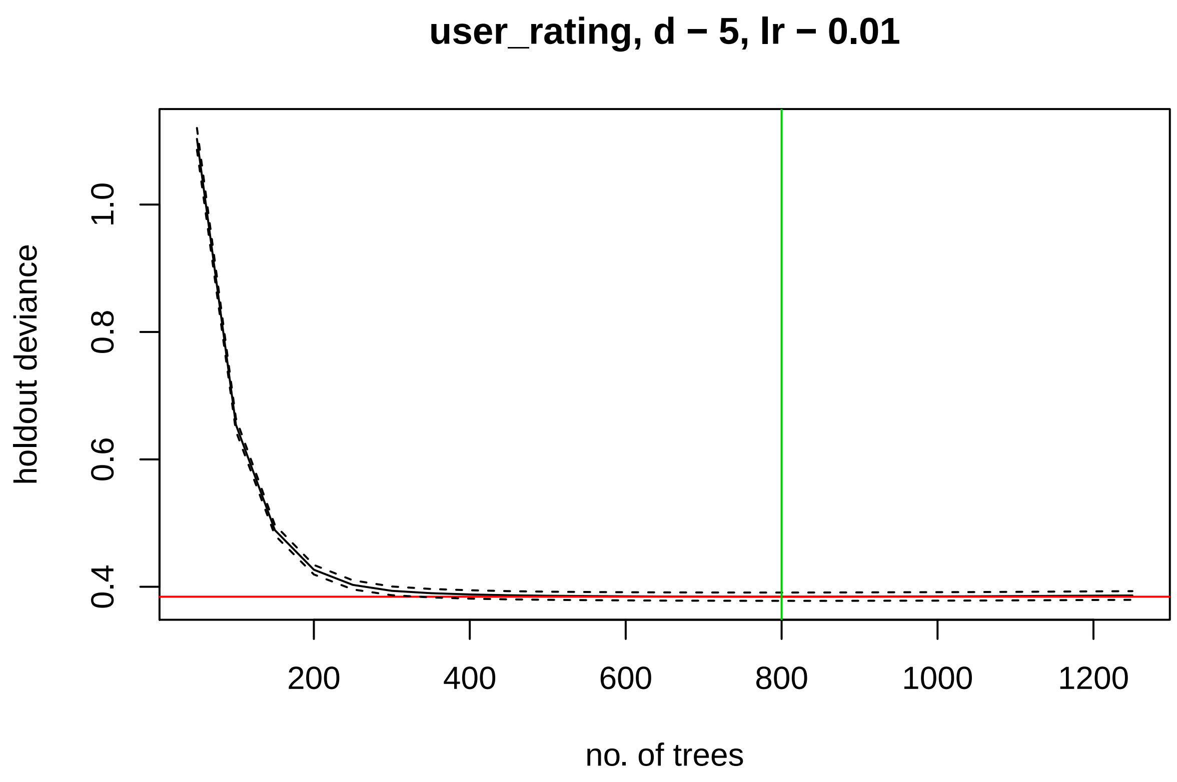
```
## [1] 0.3684317
```

The importance of variables plot shows important variables that are ordered from top to bottom as most-to-least important. Therefore, the most important variables are at the top, which is `rating_count_tot`. Other 3 important variables are `size_byte`, `prime_genre` and `ipadSc_urls.num`.

Boosting

```r
# Use gbm.step function in dismo package to fit the boosting model
library(dismo)
# first, generate a train dataset for use
ap_train2 <- subset(ap_train,select = c('price','rating_count_tot','cont_rating',
                                        'sup_devices.num','size_byte','prime_genre',
                                        'ipadSc_urls.num','lang.num','vpp_lic','user_rating'))
ap_train2 <- as.data.frame(ap_train2)
# train boosting model, with learning.rate =0.01
# use full train dataset
fit_btree <- gbm.step(data=ap_train2, gbm.x =1:9,
                      gbm.y = 10, tree.complexity = 5,
                      family='gaussian',learning.rate = 0.01,
                      bag.fraction = 1)
```

```
##
##
##  GBM STEP - version 2.9
##
## Performing cross-validation optimisation of a boosted regression tree model
## for user_rating and using a family of gaussian
## Using 5042 observations and 9 predictors
## creating 10 initial models of 50 trees
##
##  folds are unstratified
## total mean deviance =  2.3222
## tolerance is fixed at  0.0023
## ntrees resid. dev.
## 50    1.1032
## now adding trees...
## 100   0.6544
## 150   0.4889
## 200   0.4268
## 250   0.403
## 300   0.3936
## 350   0.39
## 400   0.388
## 450   0.3867
## 500   0.3859
## 550   0.3854
## 600   0.385
## 650   0.3847
## 700   0.3846
## 750   0.3845
## 800   0.3845
## 850   0.3845
## 900   0.3846
## 950   0.3848
## 1000   0.3849
## 1050   0.3851
## 1100   0.3854
## 1150   0.3857
## 1200    0.386
## 1250    0.3863
```



```
##
## mean total deviance = 2.322
## mean residual deviance = 0.339
##
## estimated cv deviance = 0.384 ; se = 0.007
##
## training data correlation = 0.924
## cv correlation =  0.913 ; se = 0.002
##
## elapsed time -  0.95 minutes
```

```
#relevant significance of variables
relative.influence(fit_btree,n.trees=750)
```

```
##           price rating_count_tot      cont_rating  sup_devices.num
##        874.6409      481149.1837         313.6160         545.2724
##       size_byte      prime_genre  ipadSc_urls.num         lang.num
##       3740.8882       12835.7813        1139.4292        1315.7101
##         vpp_lic
##          0.0000
```

```r
#calculate train and test mse
yhat_btree <- predict(fit_btree, ap_train, n.trees = 750)
mse_btree <- mean((yhat_btree - y_train) ^ 2)
yhat_btree_test <- predict(fit_btree,ap_test,n.trees=750)
mse_btree_test <- mean((yhat_btree_test-y_test)^2)
print(mse_btree)
```

```
## [1] 0.3412437
```

```r
print(mse_btree_test)
```

```
## [1] 0.3679166
```

Last, we adopted boosting model. To lower our computational cost, we use gbm.step with learning rate of 0.01 to select the best number of trees with the lowest mse, and we can see that from the table, 750 would be the most optimal number of trees within the 0.01 learning rate. Then, we computed the significance level and MSEs for train and validation sets. As other models show, `rating_count_tot` is the most predictable predictor. Then are `prime_genre` and `size_byte`. Our MSE for train set is 0.3412, and for validation set is 0.3679.

Summary

For overall analysis, the model with the lowest MSE for validation/test set is boosting. However, considering the level of interpretability, we may adopt the tree model, whose MSE is slightly higher than boosting (0.3938 for validation set of tree vs. 0.3679 for validation set of boosting) but more interpretable, when we have audience who have no technical background.

The biggest challenge we faced is to create k-fold cross validation for linear regression and choose the number of trees that should be used for boosting by gbm.step.

The lessons that we have learned that could be put into practice in the future is that for machine learning model, the trade-off between interpretability and flexibility is very important. Depends on our audience and the business context/academic context, we should adopt different models - since right now we do not have a perfect model that is both ideal for interpretability and flexibility.