Apple iOS App Store Ratings Project Report

BUAN 6340.002 PROGRAMMING FOR DATA SCIENCE

GROUP 2

GROUP MEMBERS

Qianyu Lin

Fan Zhang

Yiyuan Guo

Jianheng Li

Wei Feng

Liyuan Cao

Table of Contents

Executive Summary 2

Project Background 2

Data Description 2

Exploratory Data Analysis (Text Analysis) 13

Models and Analysis 17

Findings and Managerial Implications 25

Conclusions 25

Appendix 26

1. Executive Summary

In this project, we explored the relationships between multiple variables and ratings to find out how to increase the rating of an App and predict the trend of a popular App. We found free RPG games whose size is between 100MB and 250MB are more likely to be successful. Also, the model we created could help predict the performance of the new version of an app.

2. Project Background

The ever-changing mobile landscape is a challenging space to navigate. The percentage of mobile over desktop is increasing. Android holds about 53.2% of the smartphone market, while iOS is 43%. To get more people to download your app, you need to make sure they can easily find your app. Mobile app analytics is a great way to understand the existing strategy to drive growth and retention of future user.

3. Data Description

The data we are going to use is Mobile App Store (7200 Apps). This dataset contains more than 7000 Apple iOS mobile application details and their ratings.

Data collection date: July 2017

Dimension of the data set: 7197 rows and 19 columns

Variables:

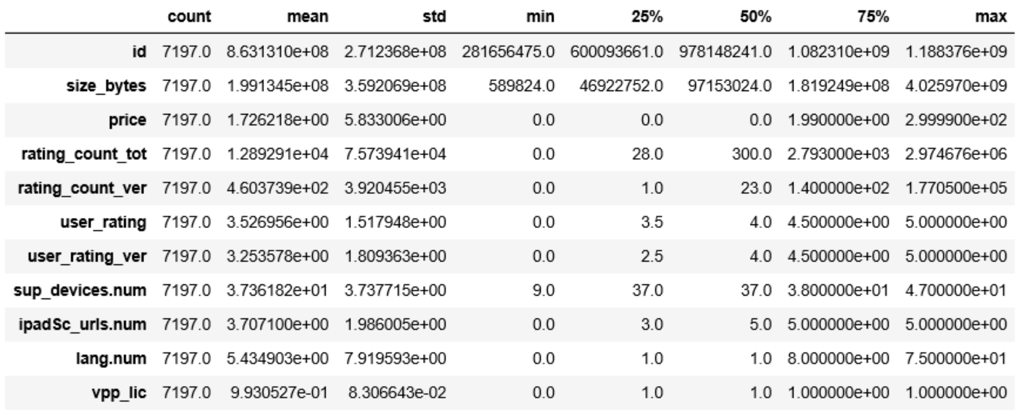
1. "id": App ID
2. "track_name": App Name
3. "size_bytes": Size (in Bytes)
4. "currency": Currency Type
5. "price": Price amount
6. "rating_count_tot": User Rating counts (for all version)
7. "rating_count_ver": User Rating counts (for current version)
8. "user_rating": Average User Rating value (for all version)
9. "user_rating_ver": Average User Rating value (for current version)
10. "ver": Latest version code
11. "cont_rating": Content Rating
12. "prime_genre": Primary Genre
13. "sup_devices.num": Number of supporting devices
14. "ipadSc_urls.num": Number of screenshots shown for display

15. "lang.num": Number of supported languages
16. "vpp_lic": Vpp Device Based Licensing Enabled
17. track_name: Application name
18. size_bytes: Memory size (in Bytes)
19. app_desc: Application description

3.1 General Analysis Based on Data Visualization

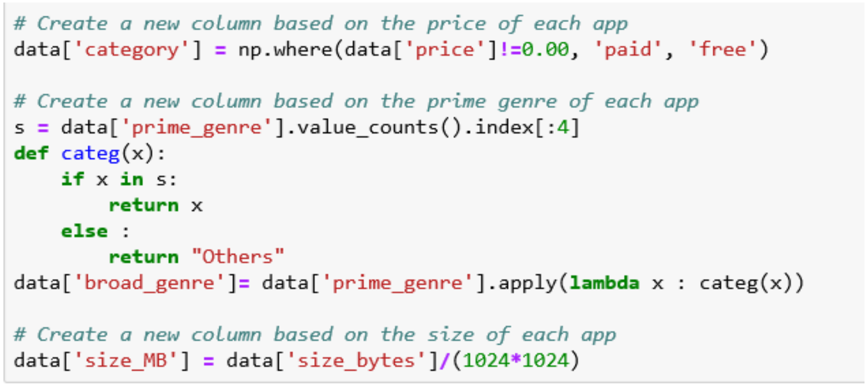
The following figure 1 shows the descriptive statistics of the variables.

Figure 1: Descriptive Statistics of the Variables



	count	mean	std	min	25%	50%	75%	max
id	7197.0	8.631310e+08	2.712368e+08	281656475.0	600093661.0	978148241.0	1.082310e+09	1.188376e+09
size_bytes	7197.0	1.991345e+08	3.592069e+08	589824.0	46922752.0	97153024.0	1.819249e+08	4.025970e+09
price	7197.0	1.726218e+00	5.833006e+00	0.0	0.0	0.0	1.990000e+00	2.999900e+02
rating_count_tot	7197.0	1.289291e+04	7.573941e+04	0.0	28.0	300.0	2.793000e+03	2.974676e+06
rating_count_ver	7197.0	4.603739e+02	3.920455e+03	0.0	1.0	23.0	1.400000e+02	1.770500e+05
user_rating	7197.0	3.526956e+00	1.517948e+00	0.0	3.5	4.0	4.500000e+00	5.000000e+00
user_rating_ver	7197.0	3.253578e+00	1.809363e+00	0.0	2.5	4.0	4.500000e+00	5.000000e+00
sup_devices.num	7197.0	3.736182e+01	3.737715e+00	9.0	37.0	37.0	3.800000e+01	4.700000e+01
ipadSc_urls.num	7197.0	3.707100e+00	1.986005e+00	0.0	3.0	5.0	5.000000e+00	5.000000e+00
lang.num	7197.0	5.434903e+00	7.919593e+00	0.0	1.0	1.0	8.000000e+00	7.500000e+01
vpp_lic	7197.0	9.930527e-01	8.306643e-02	0.0	1.0	1.0	1.000000e+00	1.000000e+00

We created below new variables to describe the data better.



```python
# Create a new column based on the price of each app
data['category'] = np.where(data['price']!=0.00, 'paid', 'free')

# Create a new column based on the prime genre of each app
s = data['prime_genre'].value_counts().index[:4]
def categ(x):
    if x in s:
        return x
    else :
        return "Others"
data['broad_genre']= data['prime_genre'].apply(lambda x : categ(x))

# Create a new column based on the size of each app
data['size_MB'] = data['size_bytes']/(1024*1024)
```

The following pie charts (chart 1 and chart 2) demonstrate the percentage of each category.

- Games, entertainment, education, photo & video account for more than 70% of all the apps
- Free apps account for more than paid apps by 12.8%

Chart1: Percentage of Each Category

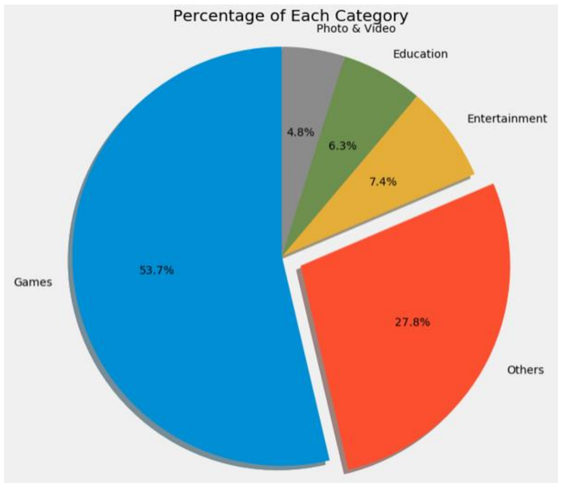
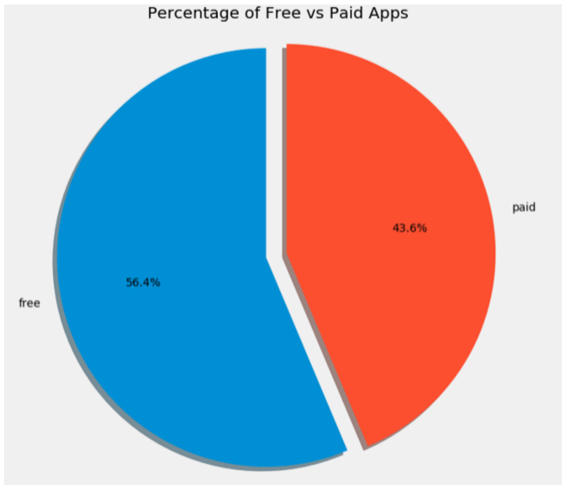


Chart2: Percentage of Free vs Paid Apps



We used the *heatmap* (figure 2) and *lmplot* (figure 3) from seaborn package to explore the correlations between variables.

- There is no strong relationship between the variables, except 'user_rating' and 'user_rating_ver'
- Correlation between 'user_rating_ver' and 'user_rating' is 0.7 which leads to a strong positive linear relationship as the above plot shows

Figure 2: Correlation among variables

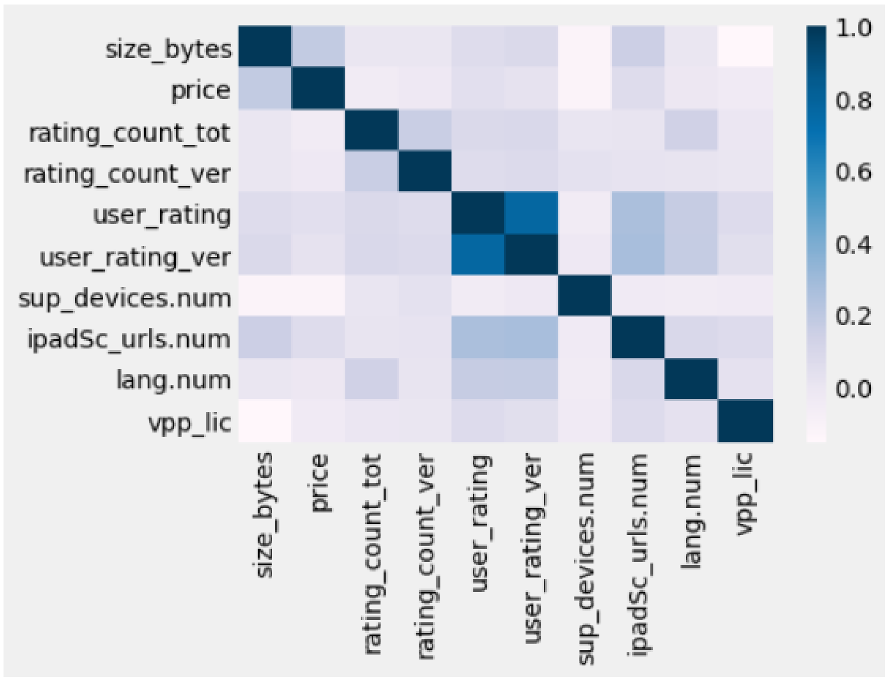
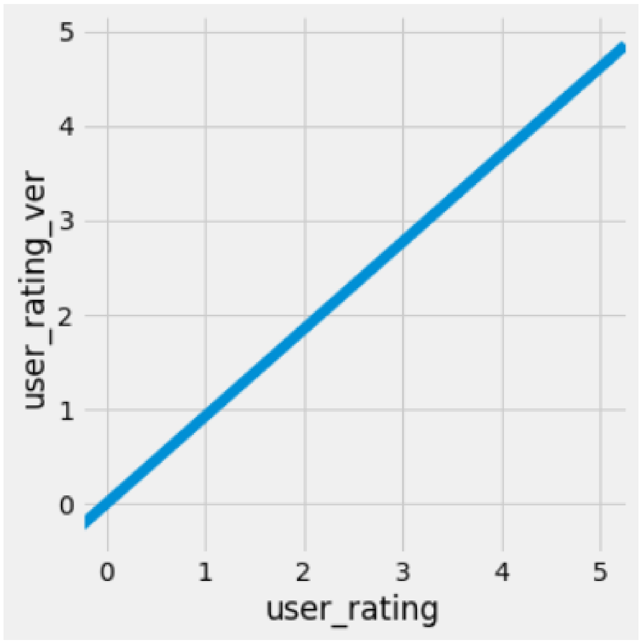
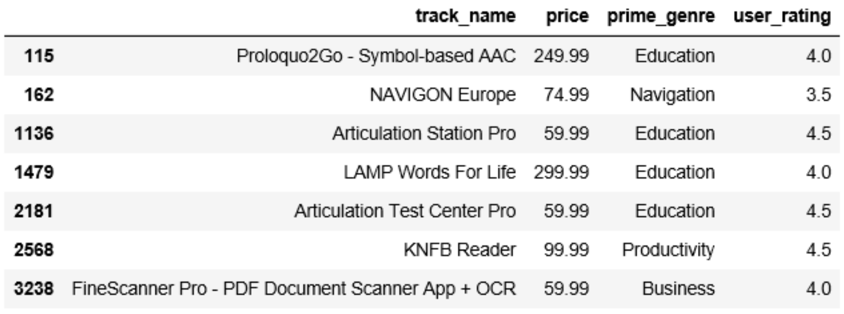


Figure 3: Realtionship between user_rating and user_rating_ver



3.2 Price descriptive analysis

To explore the general price distribution of all the apps, we first removed some outliers and then plot a histogram to show the distribution.

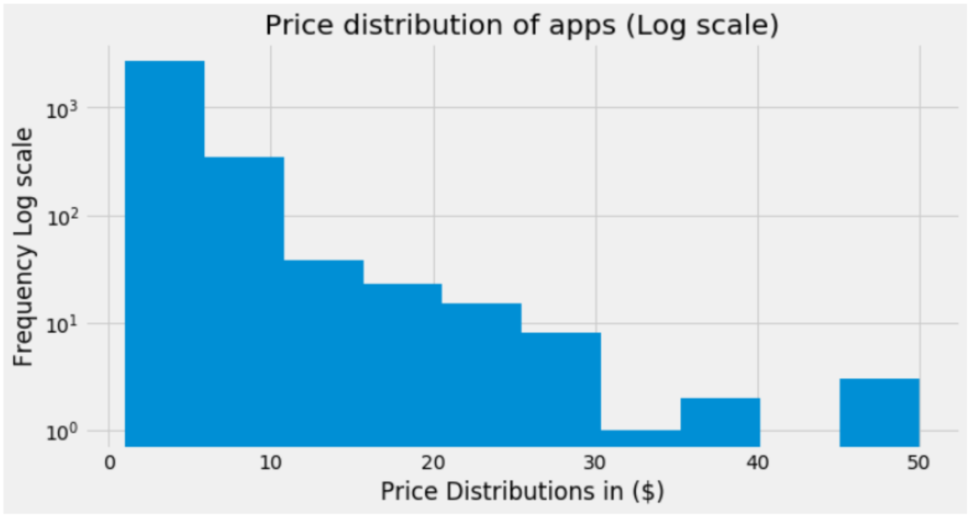


	track_name	price	prime_genre	user_rating
115	Proloquo2Go - Symbol-based AAC	249.99	Education	4.0
162	NAVIGON Europe	74.99	Navigation	3.5
1136	Articulation Station Pro	59.99	Education	4.5
1479	LAMP Words For Life	299.99	Education	4.0
2181	Articulation Test Center Pro	59.99	Education	4.5
2568	KNFB Reader	99.99	Productivity	4.5
3238	FineScanner Pro - PDF Document Scanner App + OCR	59.99	Business	4.0

As figure 4 shows, we conclude below:

- Count of paid apps exponentially decreases as the price increases
- Very few apps are priced above $30

Figure 4: Price Distribution of Apps (log scale)

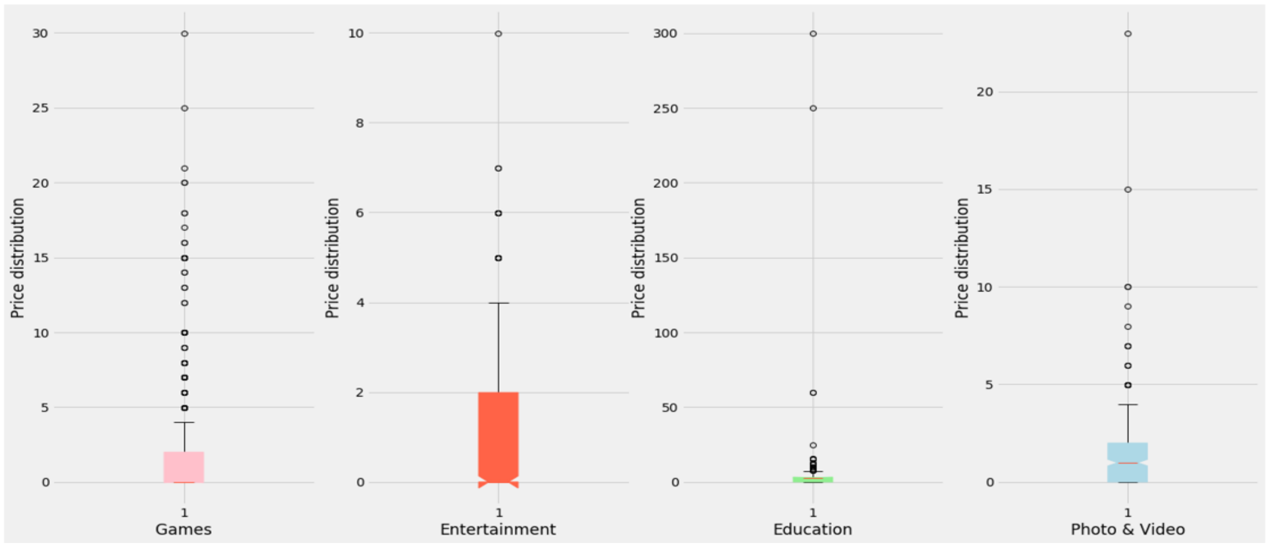


Secondly, we analyzed the price distribution of the top 4 most popular categories, which are Games, Entertainment, Education, and Photo & Video.

As per figure 5, we described it as below:

- Paid gaming apps are highly priced, and the distribution extends till $30
- Paid entertainment apps have a lower price range
- There are some high-price apps that exceed $50 in education

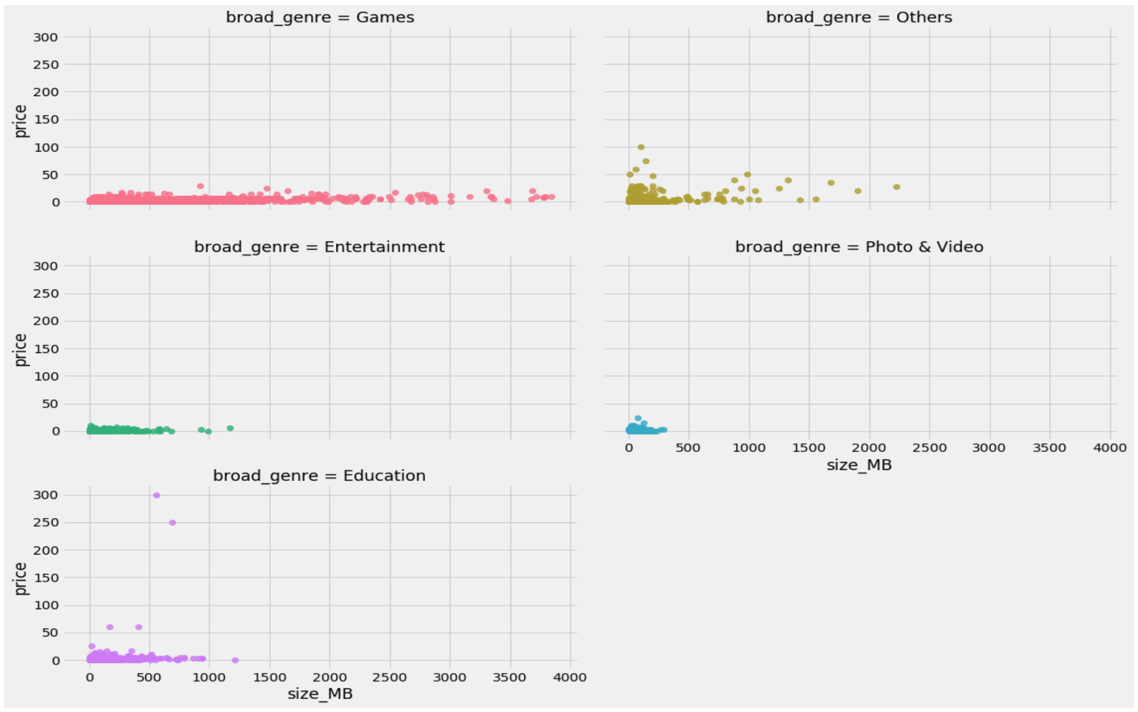
Figure 5: Box Plot of Price Distribution of Top 4 Categories



Thirdly, we thought some factors that may affect the price of apps like size and number of supported devices and explored using the plot method.
As figure 6 shows:

- As the size of apps increases the apps do not get more expensive
- There is no significant relationship between price and size

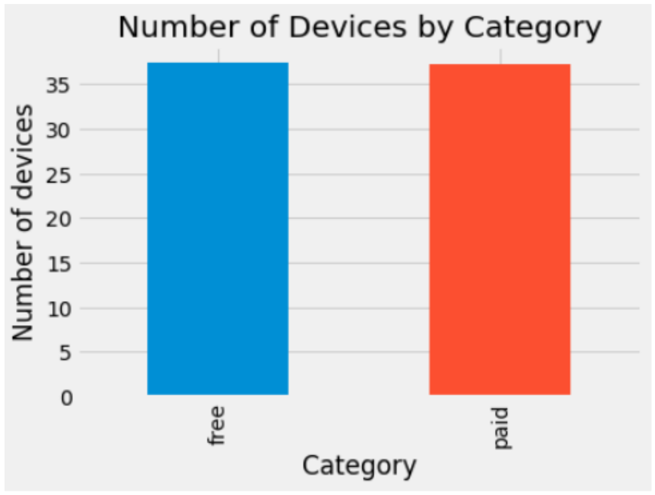
Figure 6: Relationship between Price and Size



As Figure 7 describes:

- Free and paid apps have similar supported number of devices
- There is no significant relationship between price and number of supported devices

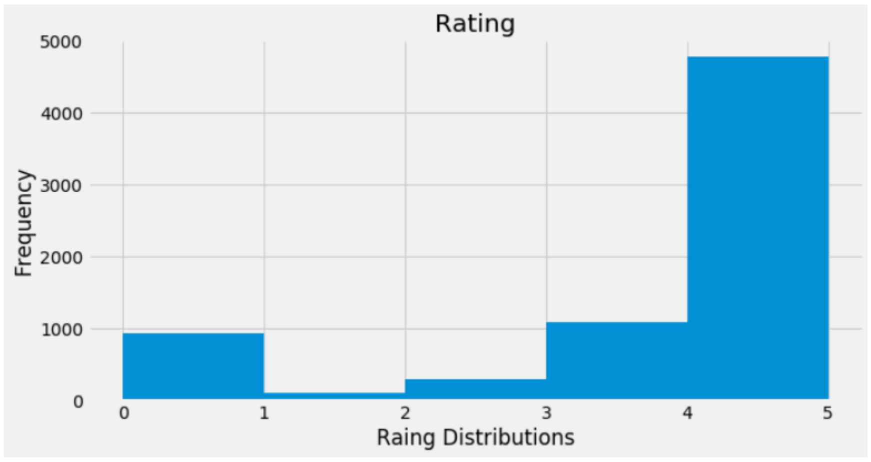
Figure 7: Number of Devices by Category



3.3 Rating descriptive analysis

Firstly, we used the histogram (figure 8) to have a general overview of the rating distribution. Most of apps are rated 4.5 and very few are rated as average 5

Figure 8: Rating Distributions



Secondly, we explored the rating amount and average rating of each category. We summarized below key points from figure 9, 10 ,11, and 12:

- Games gets the most ratings. The number is 3862, which is much larger than other categories
- The 4 most popular categories are Games, Entertainment, Education, and Photo & Video
- Free apps usually get more ratings than paid apps

- Productivity get the highest average rating. The number is 4.005618
- Paid apps usually get higher average ratings than free apps

Figure 9: Rating Amount by Category

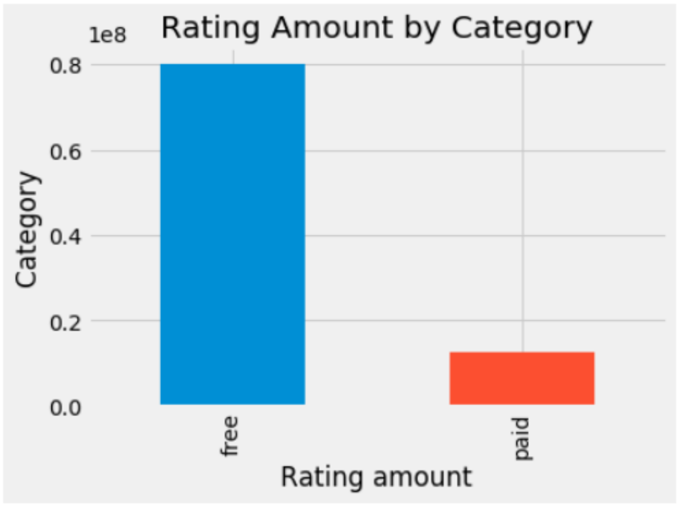


Figure 10: Average Rating by Category

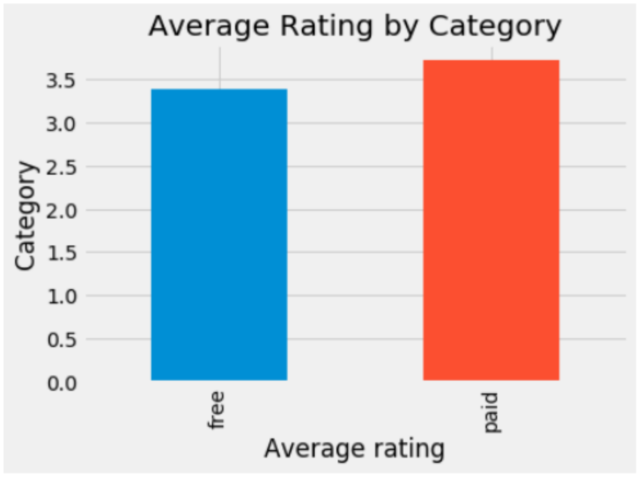


Figure 11: Rating Amount by Category

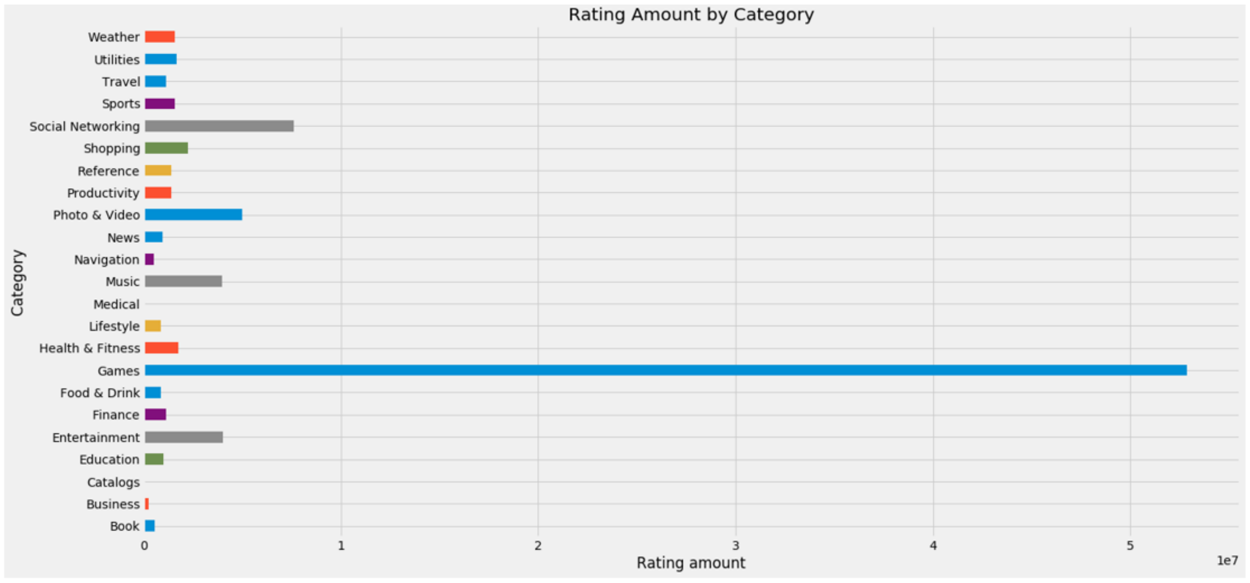
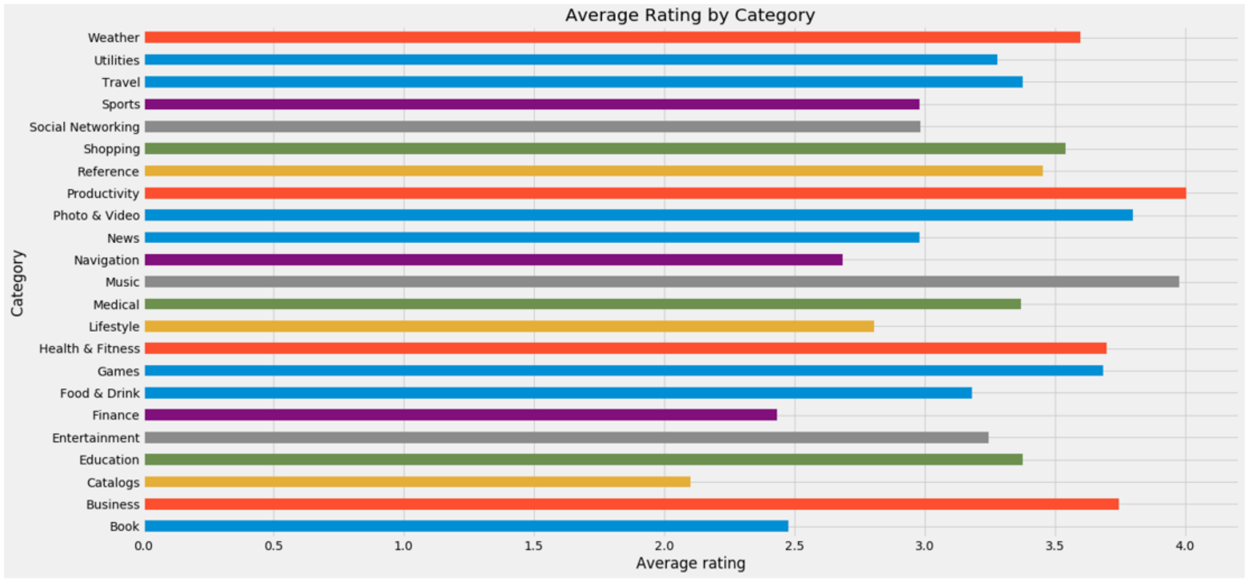
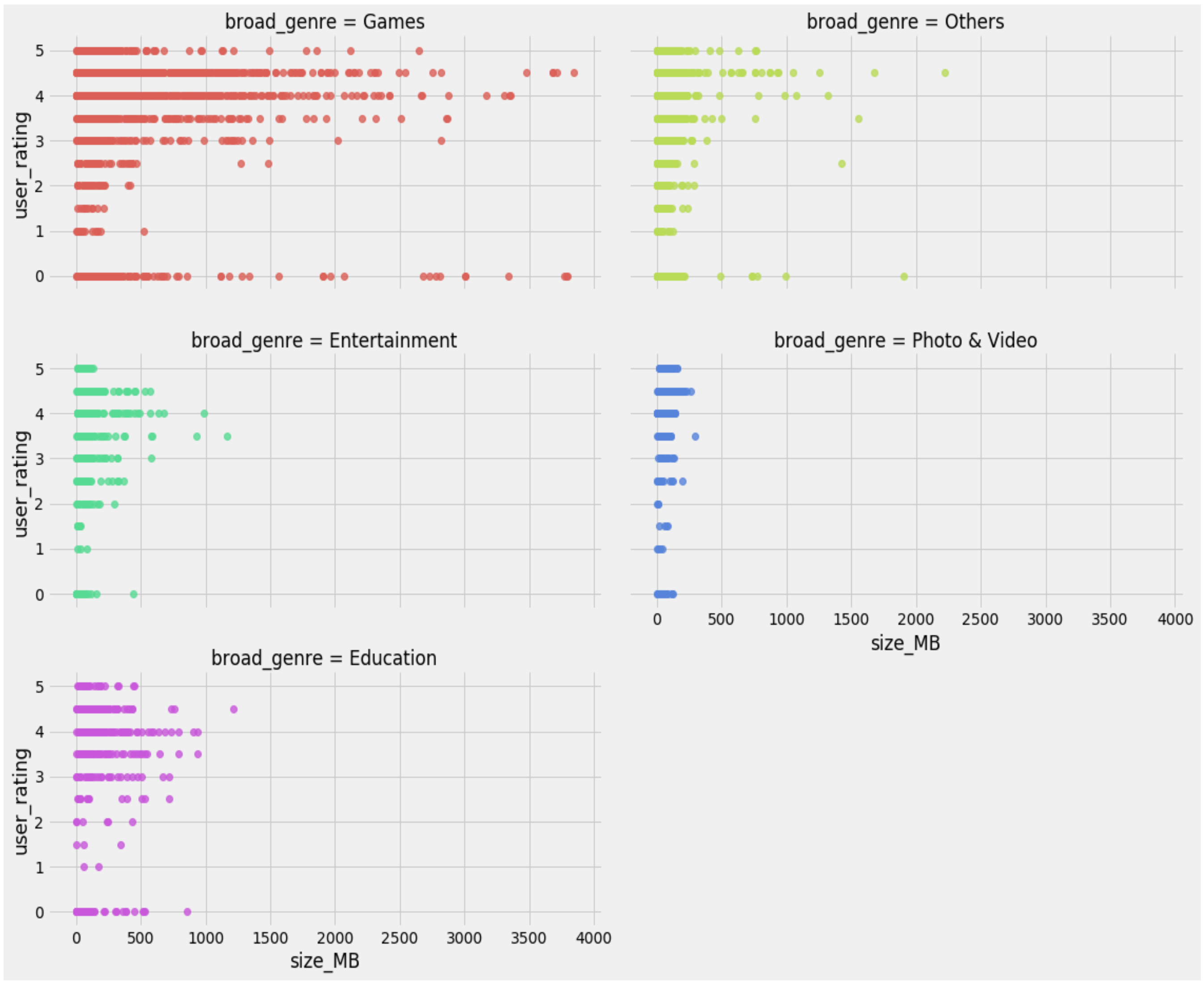


Figure 12: Average Rating by Category



Thirdly, we thought some factors that may affect the rating of apps like size and number of supported devices and explored using the plot method.
As figure 13 shows, there is no significant relationship between rating and size.

Figure 13: Relationship between Size and Rating



From figure 14 and 15, we can know that Apps with 37 supported devices get the most ratings and a relatively high average rating, so it may be the optimal number.

Figure 14: Relationship between Number of Devices Supported and Average Rating

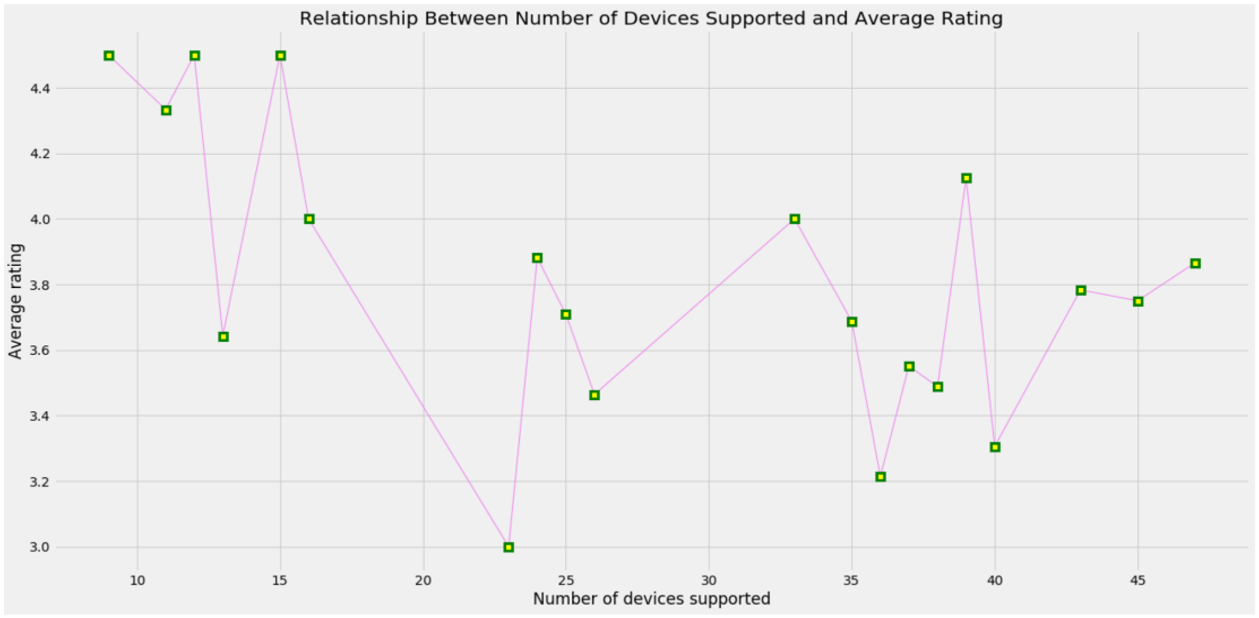
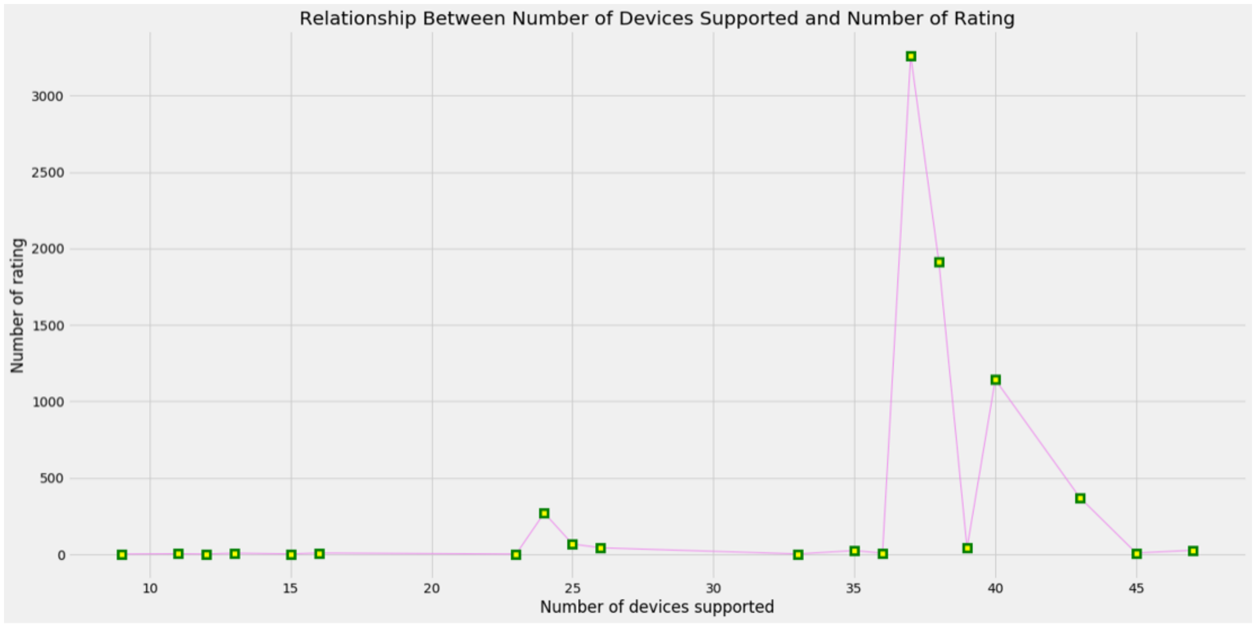


Figure 15: Relationship between Number of Devices Supported and Number of Rating

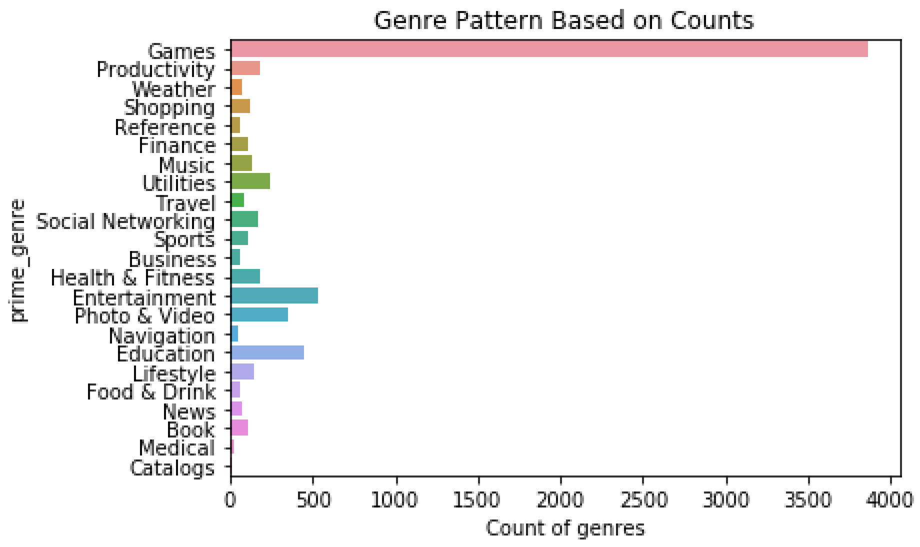


4. Exploratory data analysis (Text Analysis)

This part is trying to analyze what kind of app is more likely to be successful in present iOS App market. iOS platform is a large market for app investors. Launching an App is a business with highly time and money cost. Learning about the data about ratings of apps in a more detailed way is needful for investors of App. This part will show what is the popular iOS App now step by step.

Step 1: Create App Genre Pattern of iOS Apps
This figure 16 illustrates the share of present iOS Apps based on App genre. Game App are the most in present iOS Apps market.

Figure 16: App Genre Pattern of iOS Apps



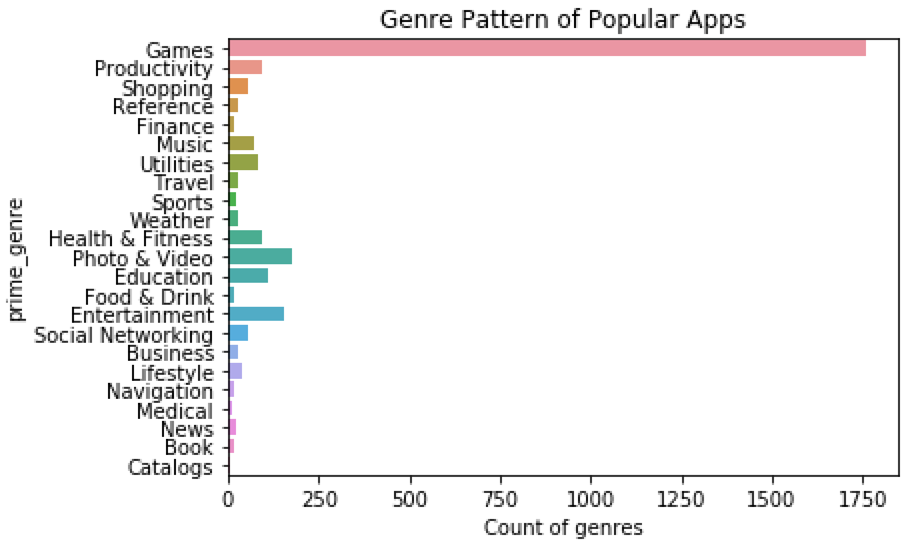
However, is the app type with the largest market share the most popular iOS App type? In order to know it, we need to define the “popularity”.

Step 2: Define the Weighted Rating
Since the simple average ratings of an app could be significantly affected by the quantity of ratings it received, it is impossible for us to believe the prediction based on such a variable. Thus, we would define a new weighted variable related to ratings, with the quantity of ratings considered. The method we are using is called shrinkage estimator, which is famous for being used by IMDB to rank their top 250 movies list. We would define our variable as following: weighted rating (WR) = (Q / (Q+M)) × R + (M / (Q+M)) × C, where:

R = average of ratings for the apps (mean) = (Ratings)
Q = quantity of ratings for the app = (Quantity)
M = minimum ratings required to be considered into prediction (we set it to be 500)
C = the mean rating across the whole report (3.526956 for the dataset)

Step 3: Re-create App Genre Pattern of iOS Apps
Below figure 17 represents the app genre pattern of iOS Apps after considering weighted rating. If an App both has many downloads and over-average rating, which refers to high weighted rating, this app will be considered as a popular app. Figure 2 proves that game app is not only popular among vendors but also popular among customers.

Figure 17: App Genre Pattern of Popular iOS Apps



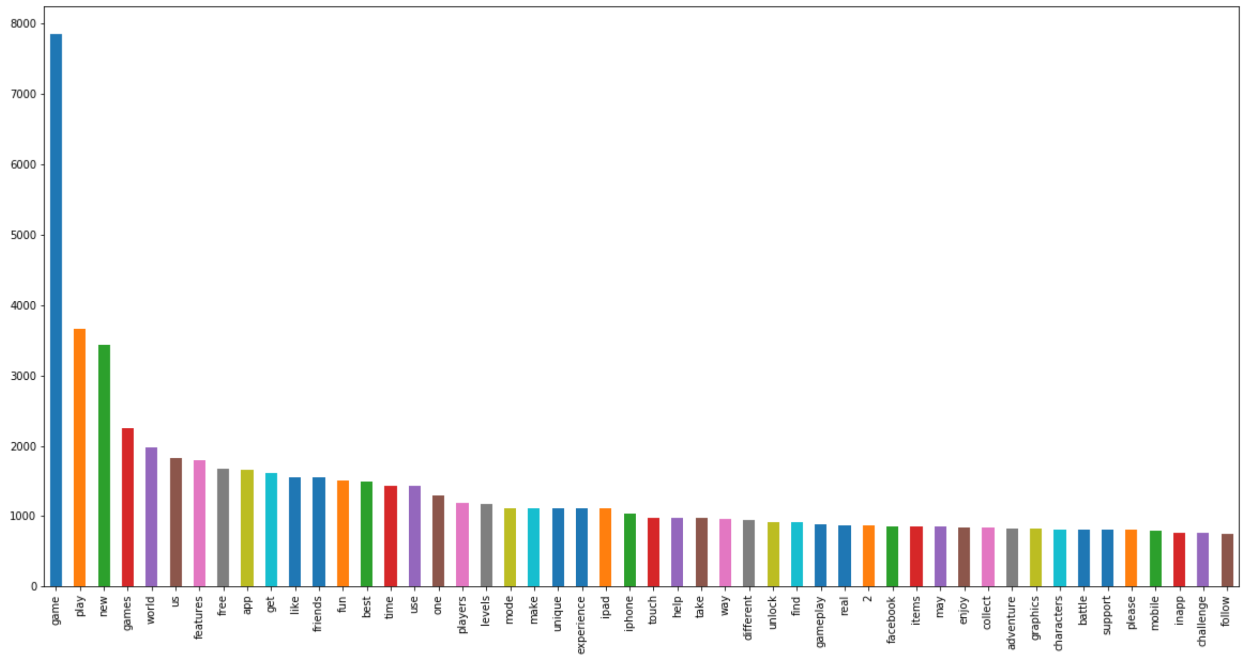
Step 4: Filter data
We had already known the most popular type of apps is game. There are 2923 Apps which are above average. 1759 of them are game apps. Since over-average game apps are a large group, it's more comparable among this group. The average rating of game apps is higher than overall rating. We continued to analyze game apps which are over the average weighted rating of game. There are 1612 game apps that above the average rating of game apps.

Step 5: Text Analysis

In order to fully understand the content of popular iOS Game App, we performed text analysis of app description. We standardized all the letters and cleaned meaningless words in the App Description column.

Below figure 18 shows the frequency of words that included in Apps.

Figure 18: Frequency of Words in App Description



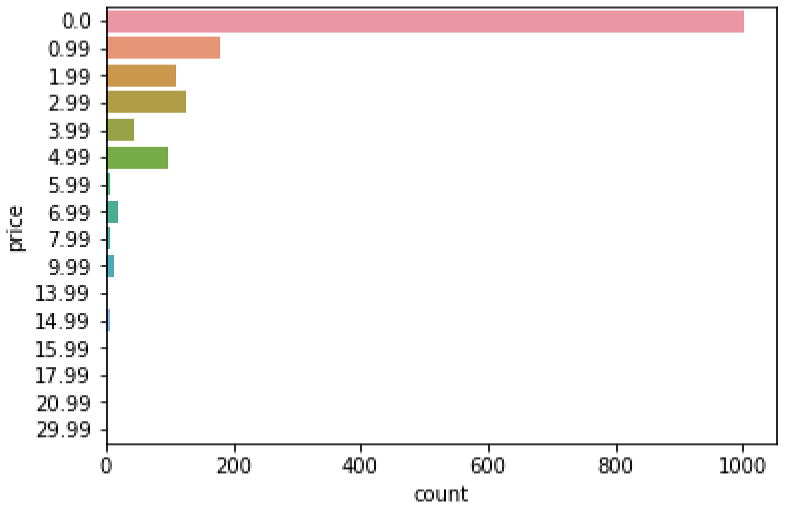
Without considering some useless words, such as "game ", "new". Below words will be helpful for investors to distinguish the features of popular Apps. They include "World", "friends", "players", "levels", "experience", "real", "Facebook", "adventures", "graphics", "battle", "characters".

These words can be matched with role play games or strategy game. Gamers can battle with friends or in-game characters. They can explore the in-game world, create their own character and increase their levels in it. Including social interaction feature in Apps is also necessary.

Step 6: Create App Price Genre Pattern of iOS Popular Game Apps

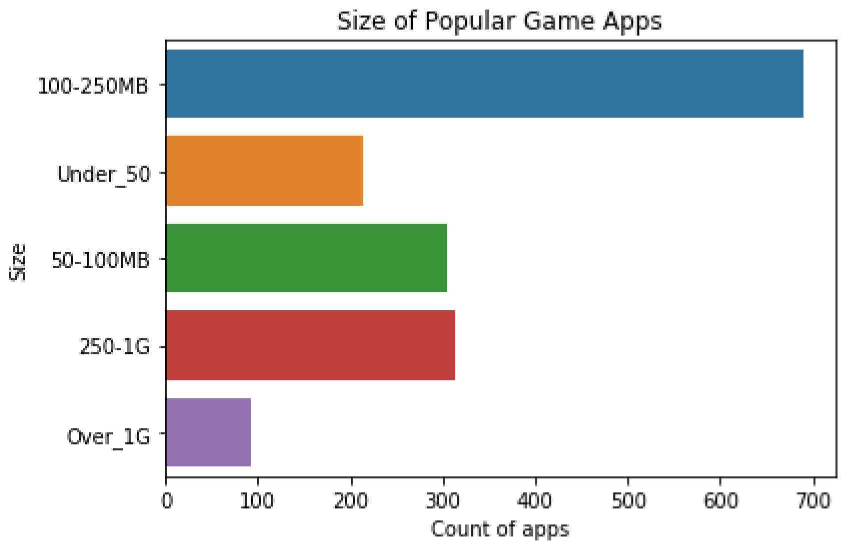
Below figure 19 shows that most of Apps are free. This refers that most Apps now make profits through in-App purchase or advertisements. Customers prefer to download free Apps to try whether it is attractive for them. If they are obsessed into it, they will be willing to pay money on it.

Figure 19: Count of Popular Game Apps based on Price



Step 7: Create App Size Genre Pattern of iOS Popular Game Apps
Most popular Apps are between 100-250 MB, which are medium-size Apps. It's a win-win strategy for both users and vendors. Vendors can maintain the app much more easily and don't need to cost too much. Users can get smooth experience without making their phones slow.

Figure 20: Count of Popular Game Apps based on Size



5. Models and Analysis

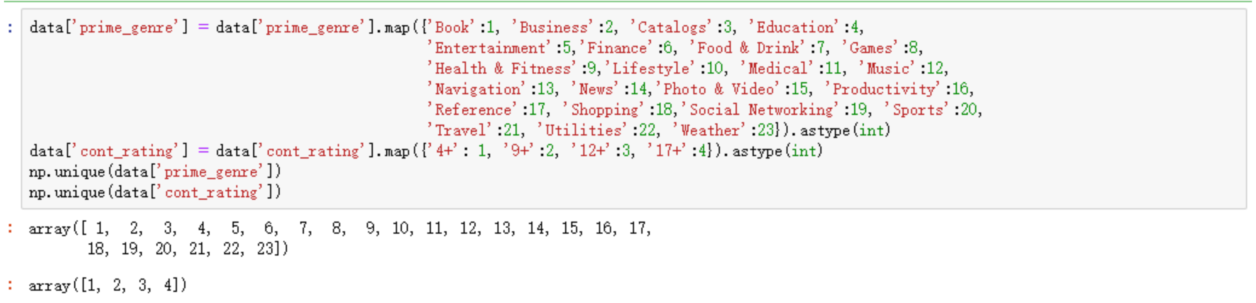
In models and analysis part, we will introduce how we built models and selected the best model to predict ratings for future applications versions. This part consists of three parts: data preparation, regression analysis and classification analysis.

5.1 Data Preparation

Before we started to build models, we made three main data transformations on the original dataset.

Data transformation1: revert key features into numerical values

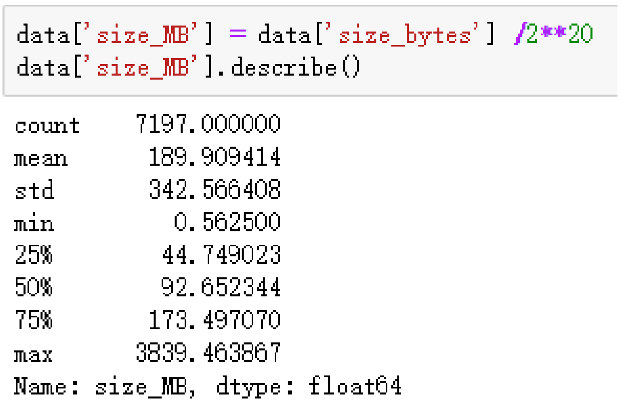
We needed this transformation because some of methods or algorithms we used cannot process text values.



```
: data['prime_genre'] = data['prime_genre'].map({'Book':1, 'Business':2, 'Catalogs':3, 'Education':4,
                                              'Entertainment':5,'Finance':6, 'Food & Drink':7, 'Games':8,
                                              'Health & Fitness':9,'Lifestyle':10, 'Medical':11, 'Music':12,
                                              'Navigation':13, 'News':14,'Photo & Video':15, 'Productivity':16,
                                              'Reference':17, 'Shopping':18,'Social Networking':19, 'Sports':20,
                                              'Travel':21, 'Utilities':22, 'Weather':23}).astype(int)
  data['cont_rating'] = data['cont_rating'].map({'4+': 1, '9+':2, '12+':3, '17+':4}).astype(int)
  np.unique(data['prime_genre'])
  np.unique(data['cont_rating'])

: array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
         18, 19, 20, 21, 22, 23])

: array([1, 2, 3, 4])
```

Data transformation2: rescale variable "size_bytes"

We conducted this transformation because the original values of "size_bytes" are in very large scale which will dominate the whole analysis process if we do not rescale them into the same scale as the scales of other variables.



```
data['size_MB'] = data['size_bytes'] /2**20
data['size_MB'].describe()

count    7197.000000
mean      189.909414
std       342.566408
min         0.562500
25%        44.749023
50%        92.652344
75%       173.497070
max      3839.463867
Name: size_MB, dtype: float64
```

Data transformation3: Define a weighted rating variable as dependent variable

The reason why we needed to define a new variable and the process how we defined it have been mentioned in Exploratory data analysis part step2 and we do not repeat it here.

5.2 Regression Analysis

Ideally, we wanted a prediction regression model to quantitatively tells us how the dependent variable changes as the independent variables change. But after we tried several models and evaluated their train performances, we found that none of them had very good performances as we expected. As you can tell from following table 1, the highest train score among those regression models is just 0.47. Therefore, we concluded that Regression models are not as good as expected (low train score) and we have to try the classification.

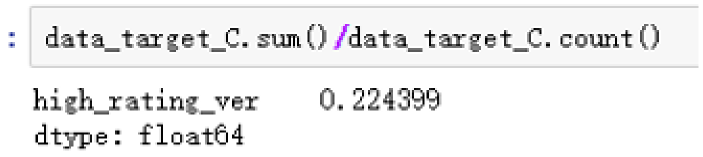
Table 1: Results of Each Model

Model name	Model best parameter	Train score
KNN Regression{'n_neighbors': 20}	{'n_neighbors': 20}	0.40719935
Simple Linear Regression		0.34
Linear Regression with Ridge	{'alpha': 10}	0.34635118
Linear Regression with Lasso	{'alpha': 0.01}	0.33918607
Simple Linear Support Vector Machine	{'C': 1}	0.22189784

5.3 Classification Analysis

Data Preprocessing 1:
Before we started to conduct classification, again, we had some data preprocessing to finish:

Data Preprocessing1: Define Binary Dependent Variable:
We defined a new dummy variable as the target variable for classification.
'high_rating_ver' =
1, if weighted rating is over 3.5
0, if weighted rating is not
Data Preprocessing2: Test Balance for Dataset



```
: data_target_C.sum()/data_target_C.count()

high_rating_ver    0.224399
dtype: float64
```

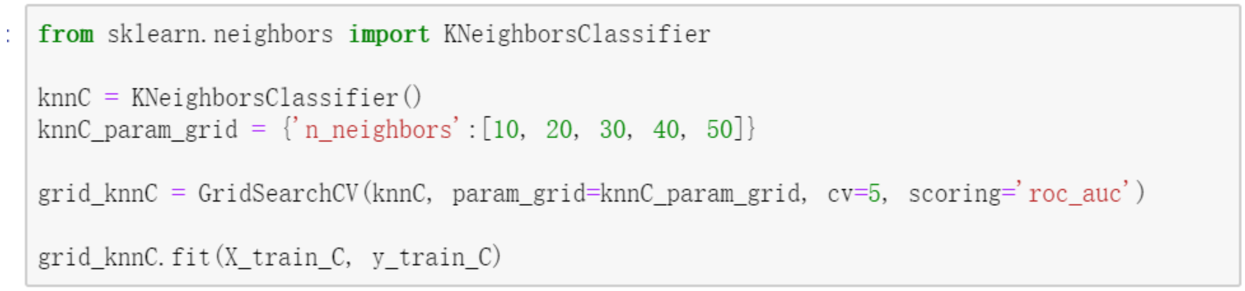
The number 0.224399 told us that only roughly 22% records are labeled as 'high_rating_ver' =1, which means our dataset is very imbalanced. Thus, we cannot just simply use accuracy to evaluate predictive performance. Instead, we use ROC and AUC scores.

5.4 Model Selection

General Idea for Classification Models
We tried different classification models, and those models gave us different results and scores. But basically, we used similar ideas to build models. We generated those ideas as following:
1.Use grid search with cross validation to get the best parameter for the model
2.Train model with the best parameter
3.Predict
4.Evaluation for prediction
Examples are selected and displayed as follows:

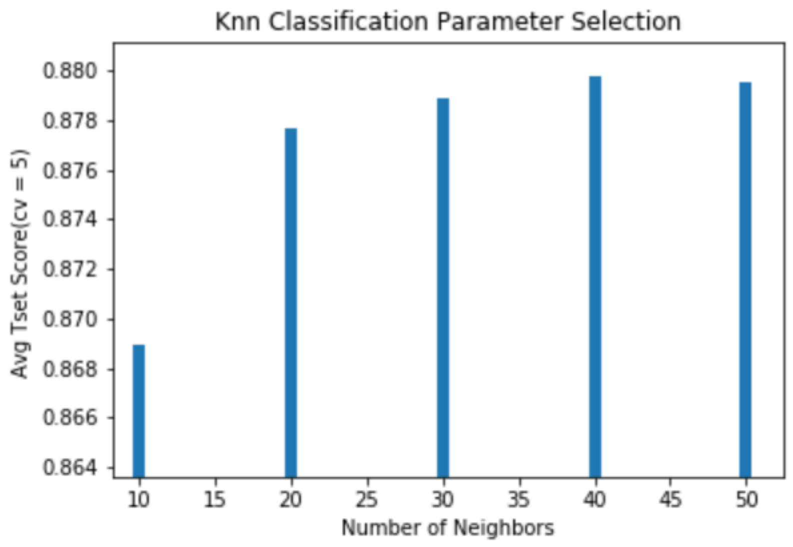
KNN Classification

For KNN Model we need to determine the number of neighbors to use in the algorithm:



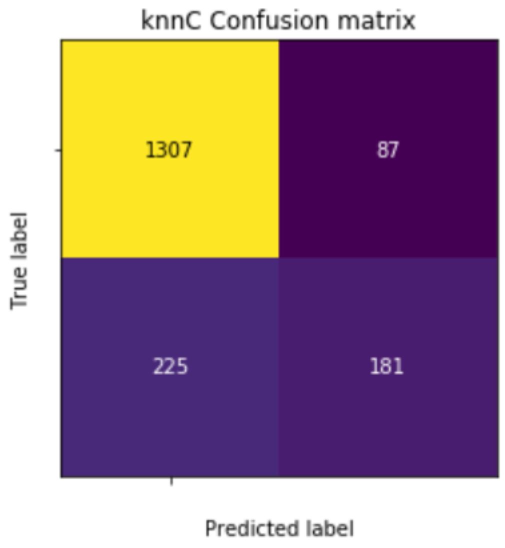
```
from sklearn.neighbors import KNeighborsClassifier

knnC = KNeighborsClassifier()
knnC_param_grid = {'n_neighbors':[10, 20, 30, 40, 50]}

grid_knnC = GridSearchCV(knnC, param_grid=knnC_param_grid, cv=5, scoring='roc_auc')

grid_knnC.fit(X_train_C, y_train_C)
```

Figure 21: KNN Classification Parameter Selection

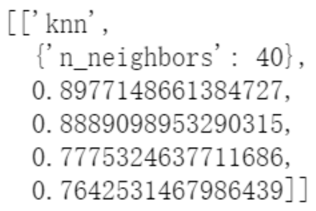


As the result of grid search displays (figure 21), 40 is the best number of neighbors to fit our model.
Then we did prediction using test dataset based on the KNN model we have just fitted. And the result of Confusion Matrix is following figure 22):

Figure 22: KNN Confusion Matrix



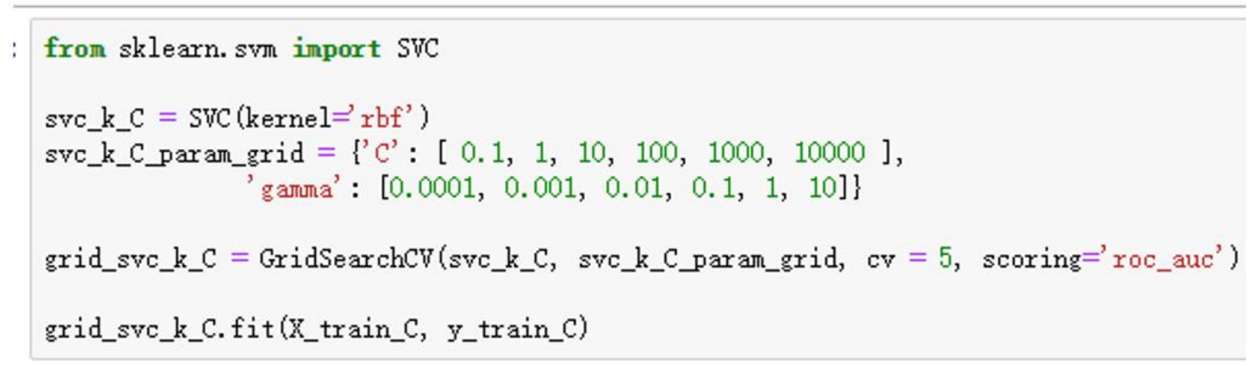
We summarized the information of the model, for the convenience of later compare among models.



```
[['knn',
  {'n_neighbors': 40},
  0.8977148661384727,
  0.8889098953290315,
  0.7775324637711686,
  0.7642531467986439]]
```

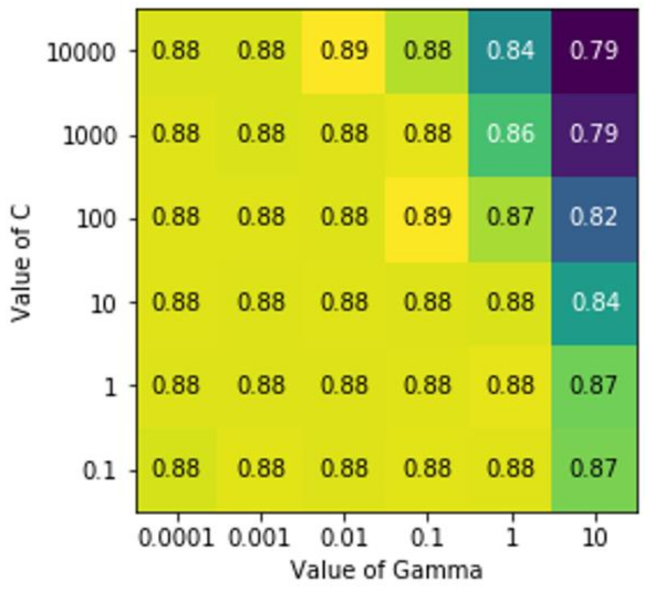
RBF Kernelized SVM

For SVM model with RBF Kernel we need to determine the best value of gamma and the proper penalty for magnitude of the data to use in the algorithm:



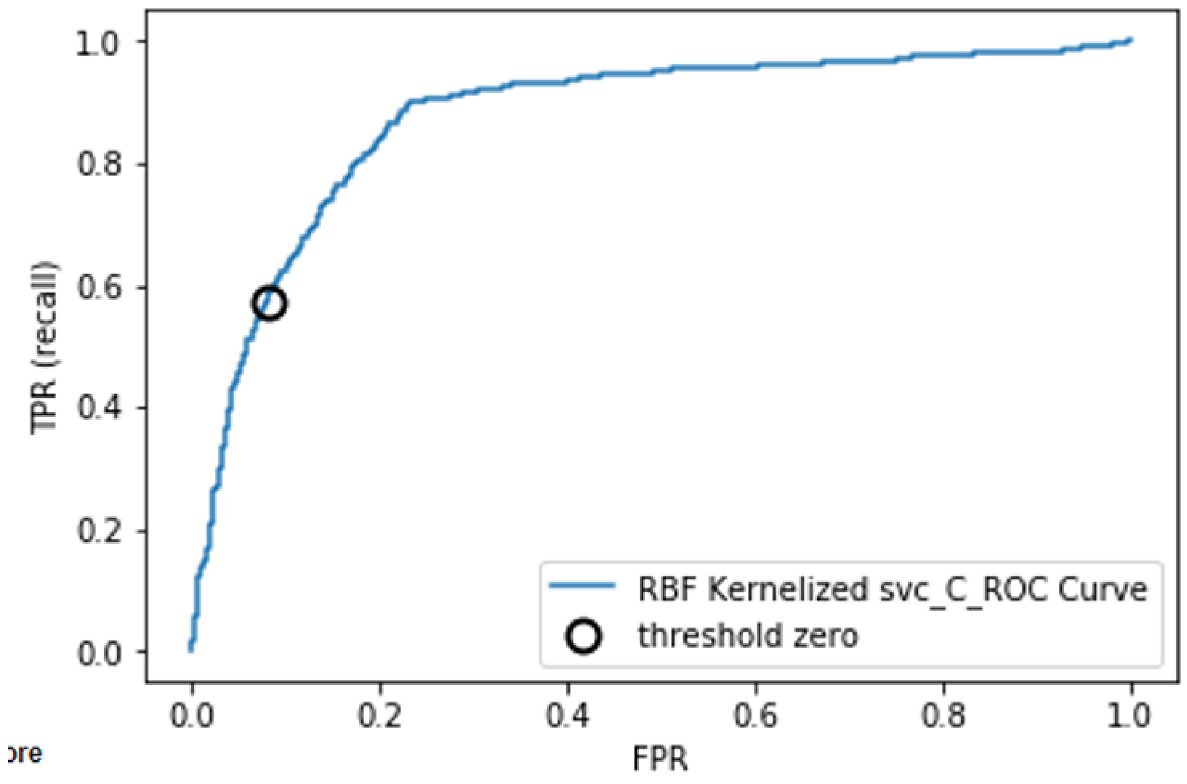
```
from sklearn.svm import SVC

svc_k_C = SVC(kernel='rbf')
svc_k_C_param_grid = {'C': [ 0.1, 1, 10, 100, 1000, 10000 ],
                 'gamma': [0.0001, 0.001, 0.01, 0.1, 1, 10]}

grid_svc_k_C = GridSearchCV(svc_k_C, svc_k_C_param_grid, cv = 5, scoring='roc_auc')

grid_svc_k_C.fit(X_train_C, y_train_C)
```

Figure 23: Parameter for RBF Kernelized SVM

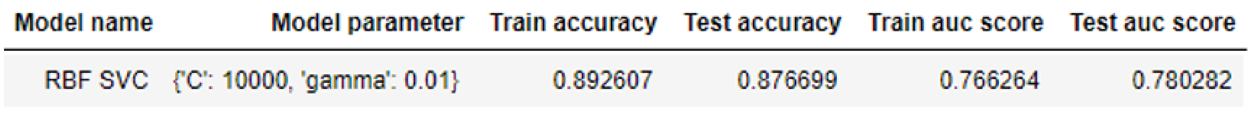


As you can tell from the matrix (figure 23), the best parameters for RBF Kernelized SVM are C=10000 and Gamma=0.01. And we trained the model with its best parameters.
Then we did prediction using test dataset based on the SVM model we have just fitted.

Figure 24: ROC Plot



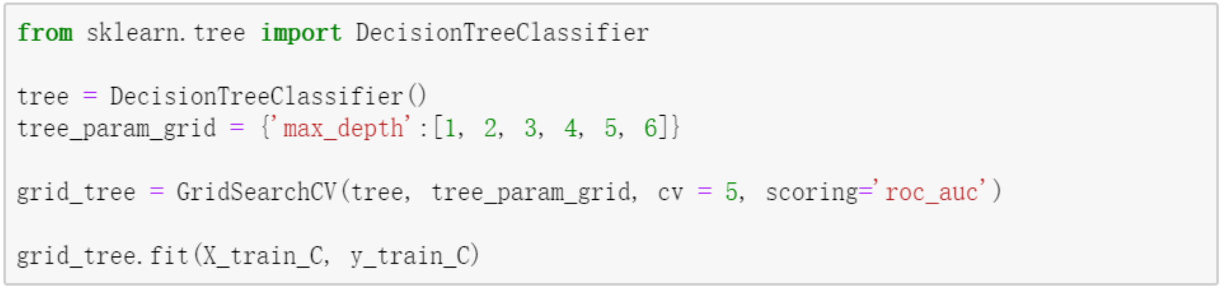
The ROC plot (figure 24) told us that this RBF Kernelized SVM had very low chance to predict the value as positive while it is negative. And it had very high chance to predict the value as positive when it is positive. And the area under the ROC curve is significantly large, indicating a high prediction performance. As the following table shows, the exact value is 0.78, which is acceptable to us.



Model name	Model parameter	Train accuracy	Test accuracy	Train auc score	Test auc score
RBF SVC	{'C': 10000, 'gamma': 0.01}	0.892607	0.876699	0.766264	0.780282

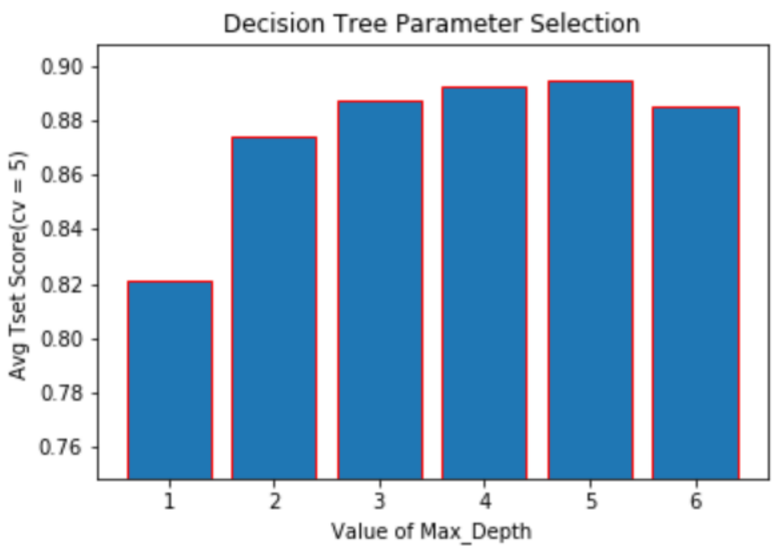
Decision Tree

For Decision Tree Model we need to determine the max depth of the tree in the algorithm:



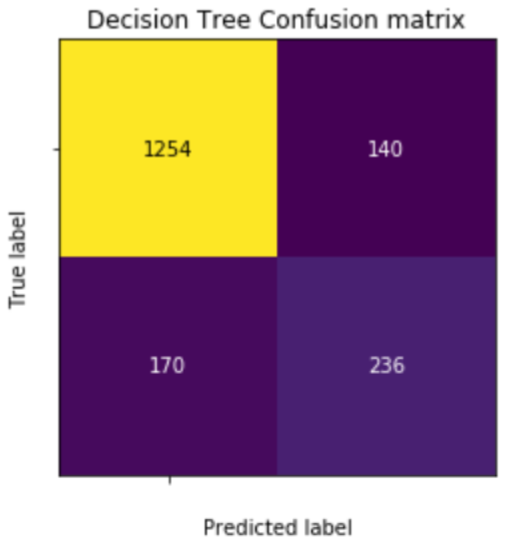
```
from sklearn.tree import DecisionTreeClassifier

tree = DecisionTreeClassifier()
tree_param_grid = {'max_depth':[1, 2, 3, 4, 5, 6]}

grid_tree = GridSearchCV(tree, tree_param_grid, cv = 5, scoring='roc_auc')

grid_tree.fit(X_train_C, y_train_C)
```

Figure 25: Decision Tree Parameter Selection

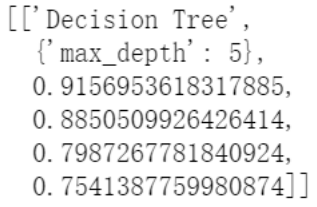


As the result of grid search displays (figure 25), 4 is the best max depth of the tree to fit our model.
Then we did prediction using test dataset based on the Decision tree model we have just fitted. And the result of Confusion Matrix (figure 26) is following:

Figure 26: Decision Tree Confusion Matrix



We summarized the information of the model, for the convenience of later compare among models.



```
[['Decision Tree',
  {'max_depth': 5},
  0.9156953618317885,
  0.8850509926426414,
  0.7987267781840924,
  0.7541387759980874]]
```

Random Forest

Also, we tried to fit some ensemble model to see if there is any portfolio to get better prediction performance. Let's take Random Forest for an example.

Firstly, we need to determine the best values of parameters to use in the algorithm:
Bootstrap: True for bagging method and false for pasting method
Max_depth: the maximum depth for each individual tree
Max_features: the number of features to be selected in the modeling
Min_samples_leaf: the minimum number of samples included in each leaf.
N_estimators: the number of trees for the ensemble random forest.

```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier()
rf_param_grid = { 'bootstrap': [True,False],
                  'max_depth': [13, 15, 17],
                  'max_features': [7, 8],
                  'min_samples_leaf': [15, 20, 25],
                  'n_estimators': [100, 200, 300]}

grid_rf = GridSearchCV(rf, rf_param_grid, cv = 5, scoring='roc_auc', n_jobs = -1)

grid_rf.fit(X_train_C, y_train_C)
```

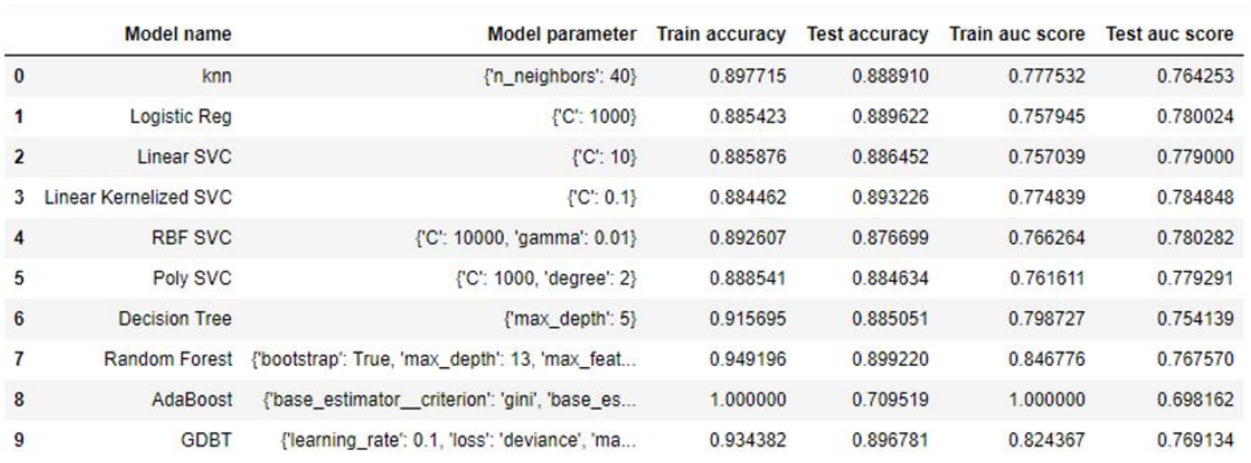
```
grid_rf.best_params_
```

```
{'bootstrap': True,
 'max_depth': 13,
 'max_features': 7,
 'min_samples_leaf': 15,
 'n_estimators': 200}
```

Then we find out the proper parameters for random forest model. After fitting and predicting, we came to following results:

```
[['Random Forest',
  {'bootstrap': True,
   'max_depth': 13,
   'max_features': 7,
   'min_samples_leaf': 15,
   'n_estimators': 200},
  0.9491958986366973,
  0.8992197383579166,
  0.8467760068634149,
  0.7675703090262339]]
```

Compare and select models
For each candidate model we applied the same method. Here are the summarized results:



	Model name	Model parameter	Train accuracy	Test accuracy	Train auc score	Test auc score
0	knn	{'n_neighbors': 40}	0.897715	0.888910	0.777532	0.764253
1	Logistic Reg	{'C': 1000}	0.885423	0.889622	0.757945	0.780024
2	Linear SVC	{'C': 10}	0.885876	0.886452	0.757039	0.779000
3	Linear Kernelized SVC	{'C': 0.1}	0.884462	0.893226	0.774839	0.784848
4	RBF SVC	{'C': 10000, 'gamma': 0.01}	0.892607	0.876699	0.766264	0.780282
5	Poly SVC	{'C': 1000, 'degree': 2}	0.888541	0.884634	0.761611	0.779291
6	Decision Tree	{'max_depth': 5}	0.915695	0.885051	0.798727	0.754139
7	Random Forest	{'bootstrap': True, 'max_depth': 13, 'max_feat...	0.949196	0.899220	0.846776	0.767570
8	AdaBoost	{'base_estimator__criterion': 'gini', 'base_es...	1.000000	0.709519	1.000000	0.698162
9	GDBT	{'learning_rate': 0.1, 'loss': 'deviance', 'ma...	0.934382	0.896781	0.824367	0.769134

As you can tell, Linear Kernelized SVM with Test AUC score=0.784848 was the best model we got. In the future, if we have any new application records, we can just use this model to predict if they are highly rated applications or not.

6. Findings and managerial implications

- The number of paid apps decreases as the price increases and very few apps are priced above $30
- Free apps are more popular than paid apps
- The 4 most popular categories are Games, Entertainment, Education, and Photo & Video
- According to the text analysis, free RPG games whose size is between 100MB and 250MB are more likely to be successful
- Linear Kernelized SVM is an optimal model to predict the performance of future apps

7. Conclusions

After all the process above, we managed to figure out the question: Which kind of apps in Apple iOS app store is more likely to get higher ratings? The answer is **Free RPG games whose size is between 100MB and 250MB,** indicating a specific set of characters. And certainly, we also have some higher-level findings indicating the key attributes for different categories, **multi-dimensionally**. We are then capable of giving advices on the development of apps, for chasing a higher weighted rating ---- which may significantly increase the probability of making success for an app on the market.

Besides, we also trained a machine learning model for predicting the latest version of an app based on the current apps data on the Apple iOS app store. **The model could significantly help us briefly predict the performance of the new version of an app** ---- and such works would be great helpful for us to ensure the advices we are giving is on the right direction, making the model a reliable tool to support our business. Also, it could supply reasonable evidence for marketing level advices we give, which may help make us understandable for our clients.

8. Appendix

The following three IPython notebooks are related to the corresponding parts in this report.

1. *Data Description.ipynb*

2. *Exploratory Data Analysis (Text Analysis).ipynb*

3. *Models and Analysis.ipynb*