



MOBILE GAMING MARKET

SC1015/CZ1115 Mini-Project

Lab Group: ECAD1

Group Members:

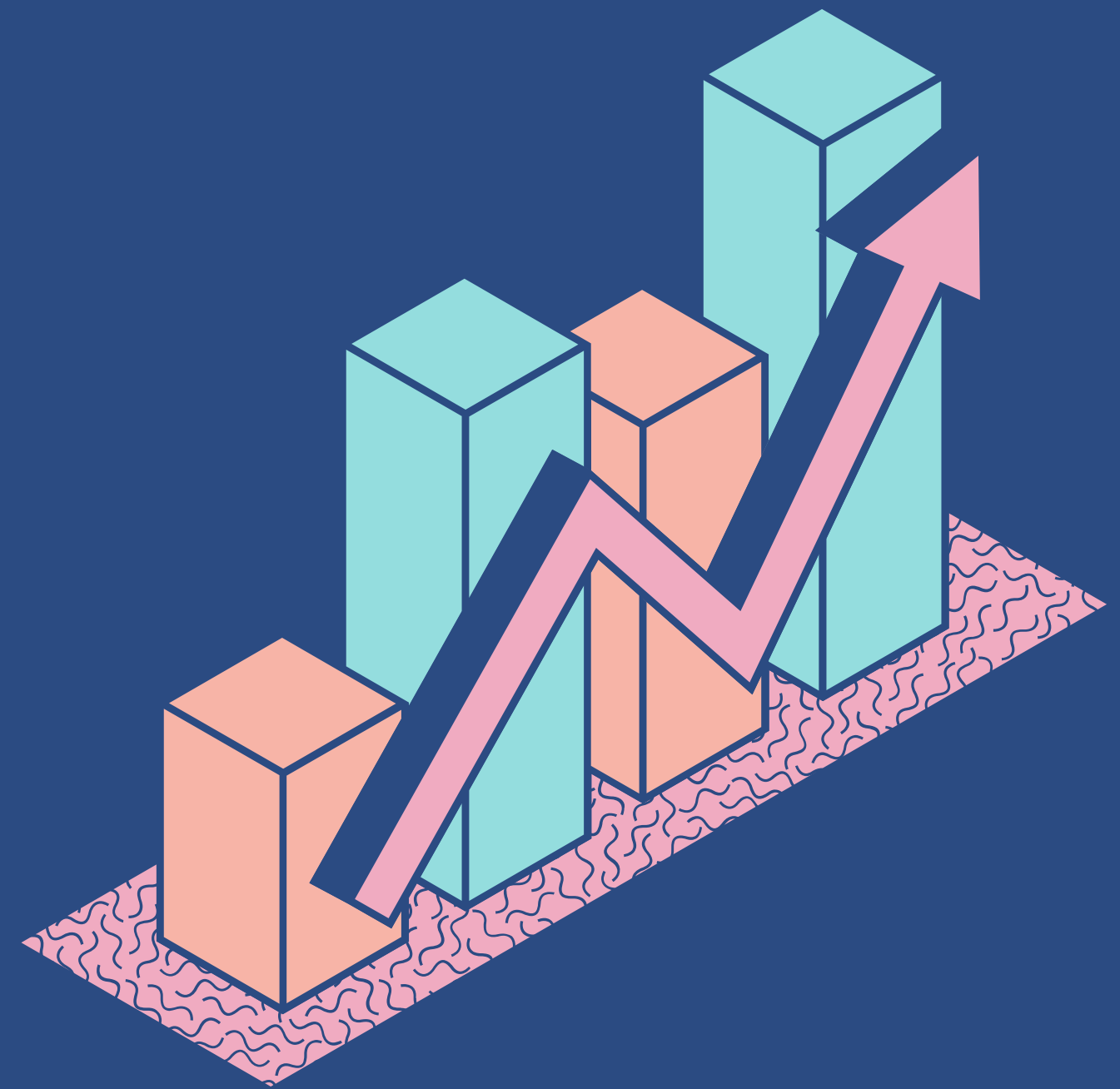
U2122236E Davyn Yam JunHao

U2022307C Faith Lee Kai Ling

Rising Trend of Mobile Gaming

2.69 billion mobile game players globally

Dominating application store revenue share at 66%





Genshin Impact

Within the first month of release:
15 million downloads
\$150 million revenue

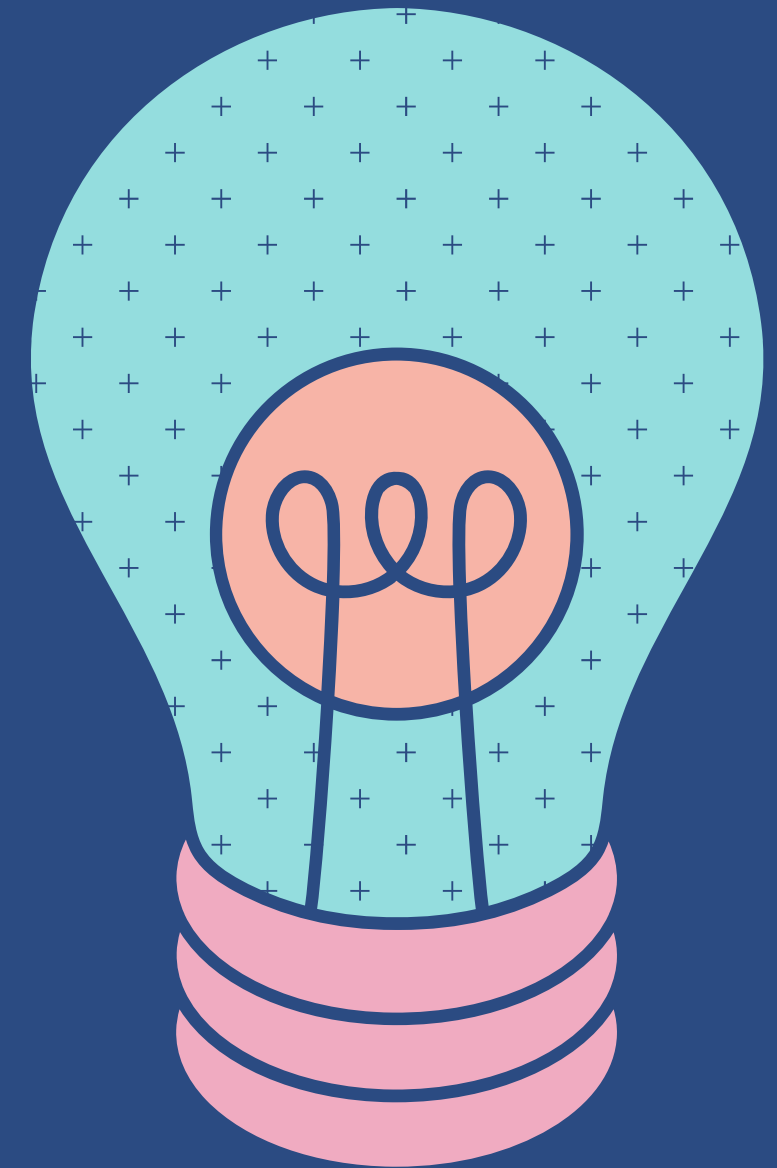
Hyper-Casual Games

Surpassing 10 billion
installs in 2020



Helping mobile game developers **increase the likelihood** of creating a successful application in the mobile game market, based on **factors that can be predetermined before release**

PROBLEM DEFINITION





Dataset used

'Google Play Store Apps', a dataset retrieved from Kaggle, was used to model the problem

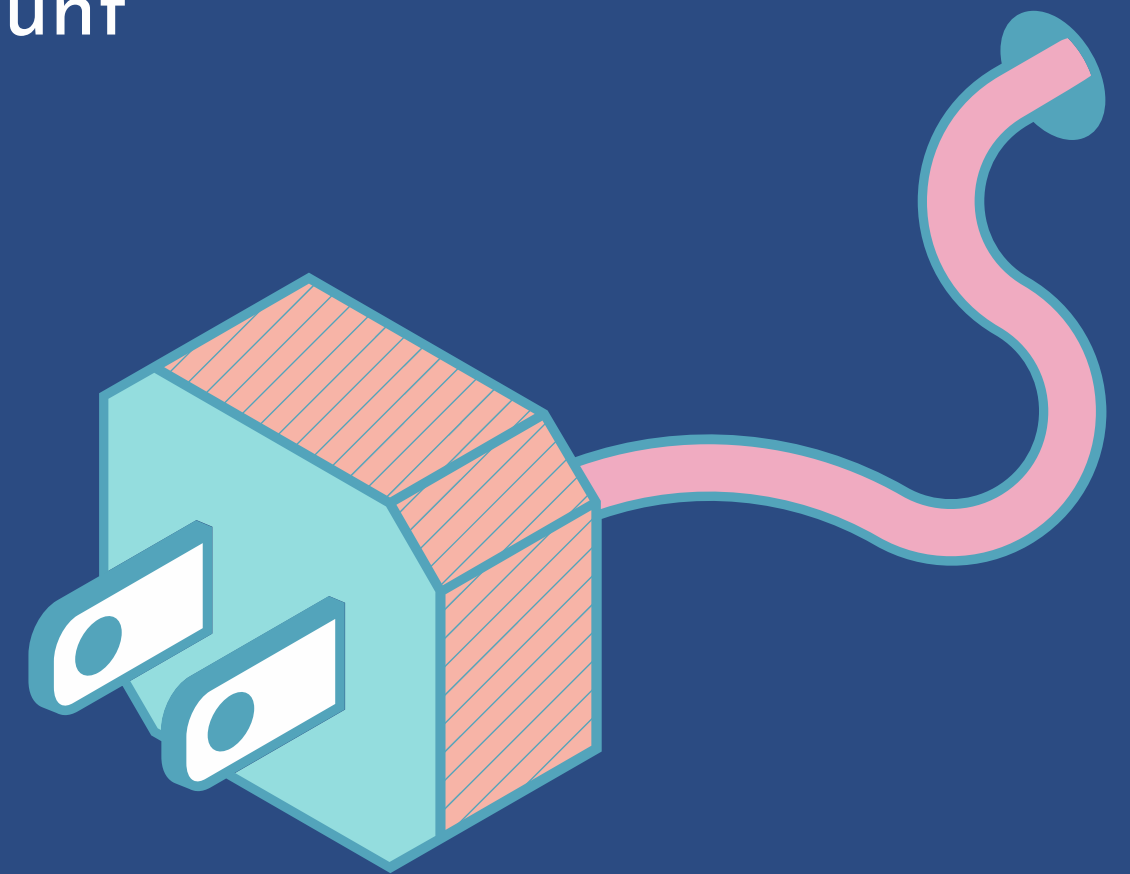
Variables considered

Predictor Variables:

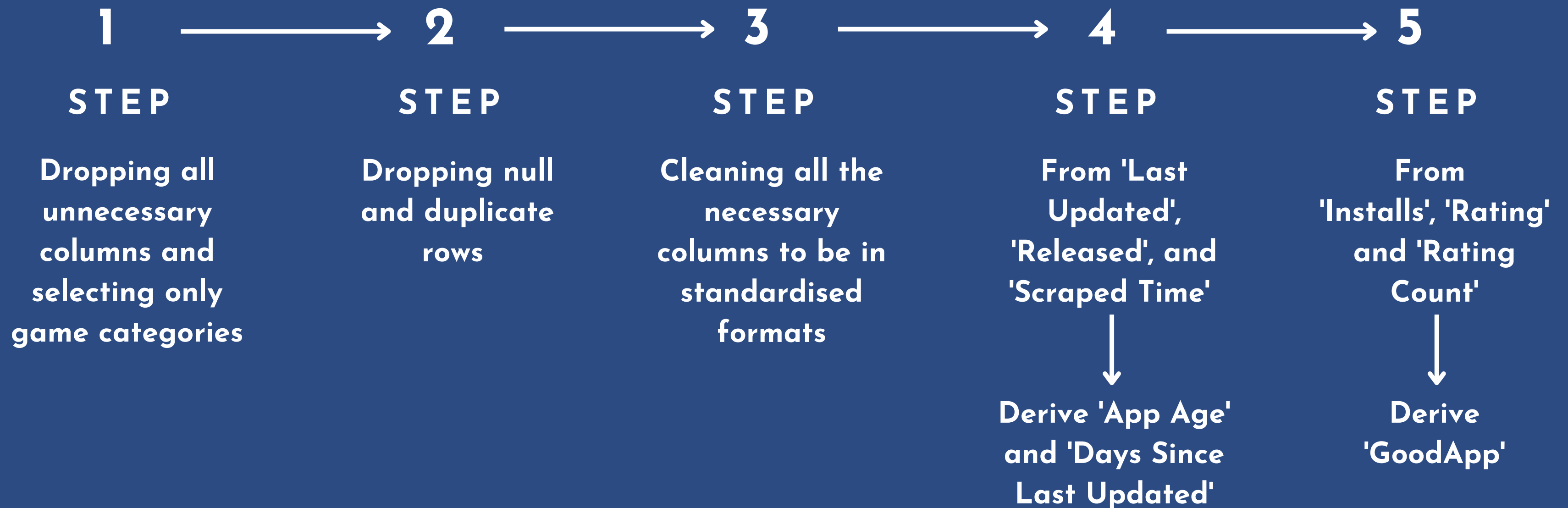
1. Price
2. Size
3. Content Rating
4. Ad Supported
5. In-App Purchases
6. Editors Choice
7. Released
8. Scraped Time
9. Last Updated

Response Variables:

1. Installs
2. Rating
3. Rating Count



Data preparation and cleaning



Data preparation and cleaning

- Removed apps with prices not in USD
- Converted size to kilobytes
- Removed apps with 'Unrated' content rating
- Converted released date, last updated date, and scraped time into datetime
 - Released : Date the application was released
 - Last Updated : Date the application was last updated
 - Scraped Time : Date the data was scraped
- Derived 'AppAge' and 'DaysSinceLastUpdated'
 - $\text{AppAge} = \text{Scraped Time} - \text{Released}$
 - $\text{DaysSinceLastUpdated} = \text{Last Updated} - \text{Scraped Time}$

Deriving predictor variable

Rating

Using the mean of rating,
rounding down to 4.0

Installs

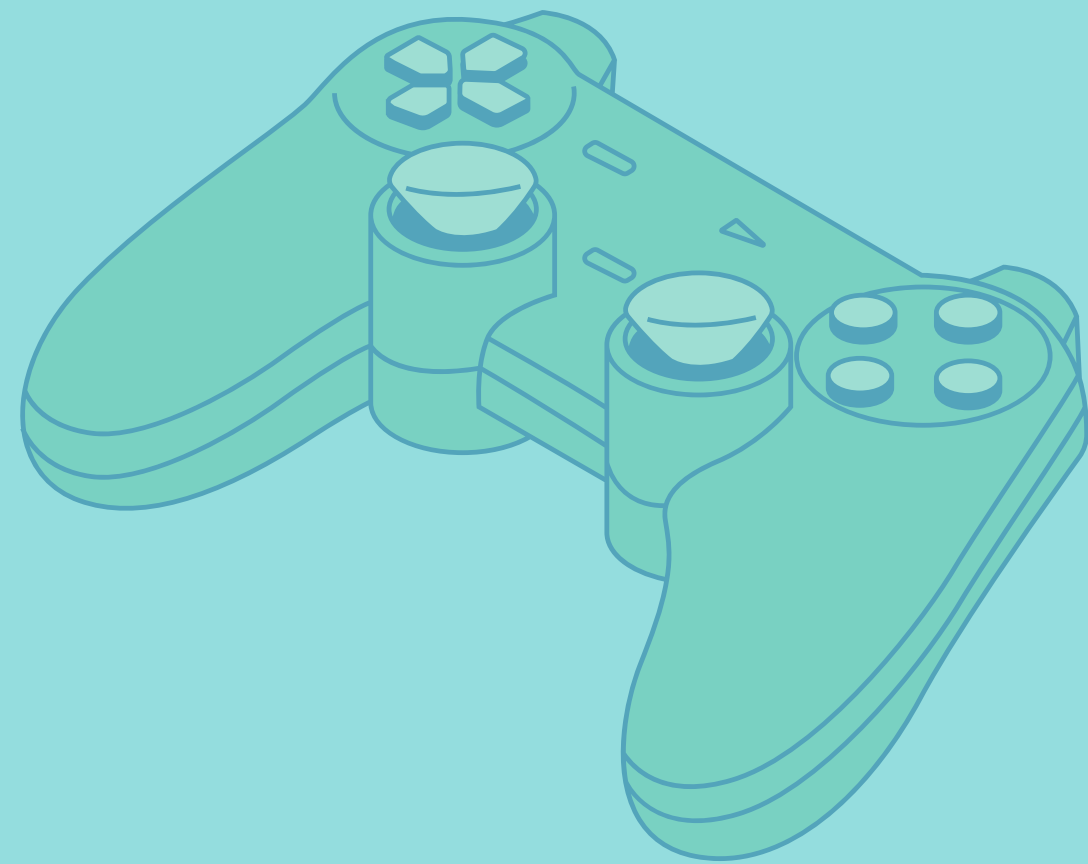
Using the midpoint of
installs, about 500,000

Rating Count

Using the median of 50
for rating count

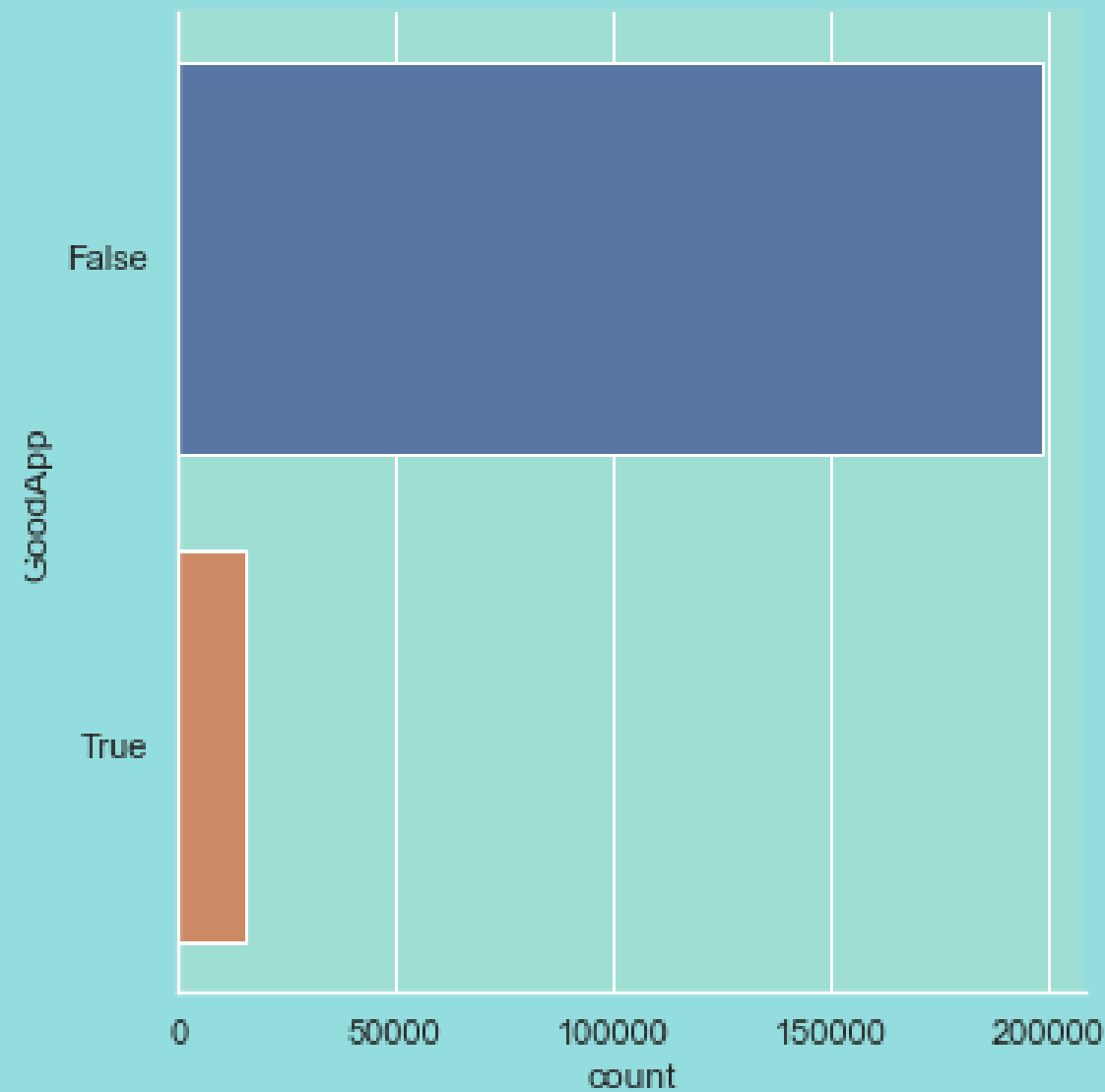
The success factor or 'GoodApp' variable condition:

'Installs' \geq 500,000 and 'Rating' \geq 4.0 and 'RatingCount' \geq 50



Exploratory Data Analysis

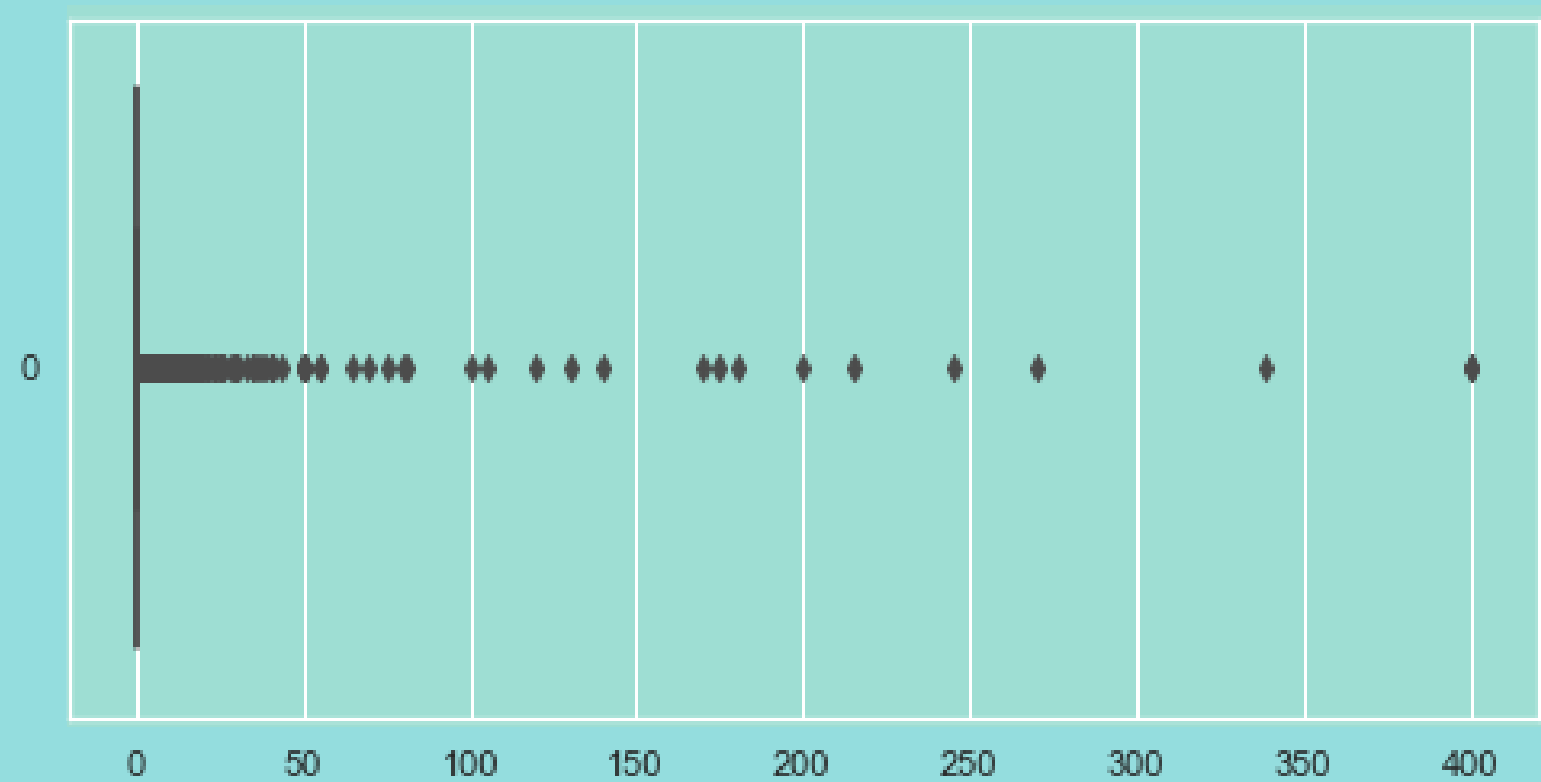
Response Variable



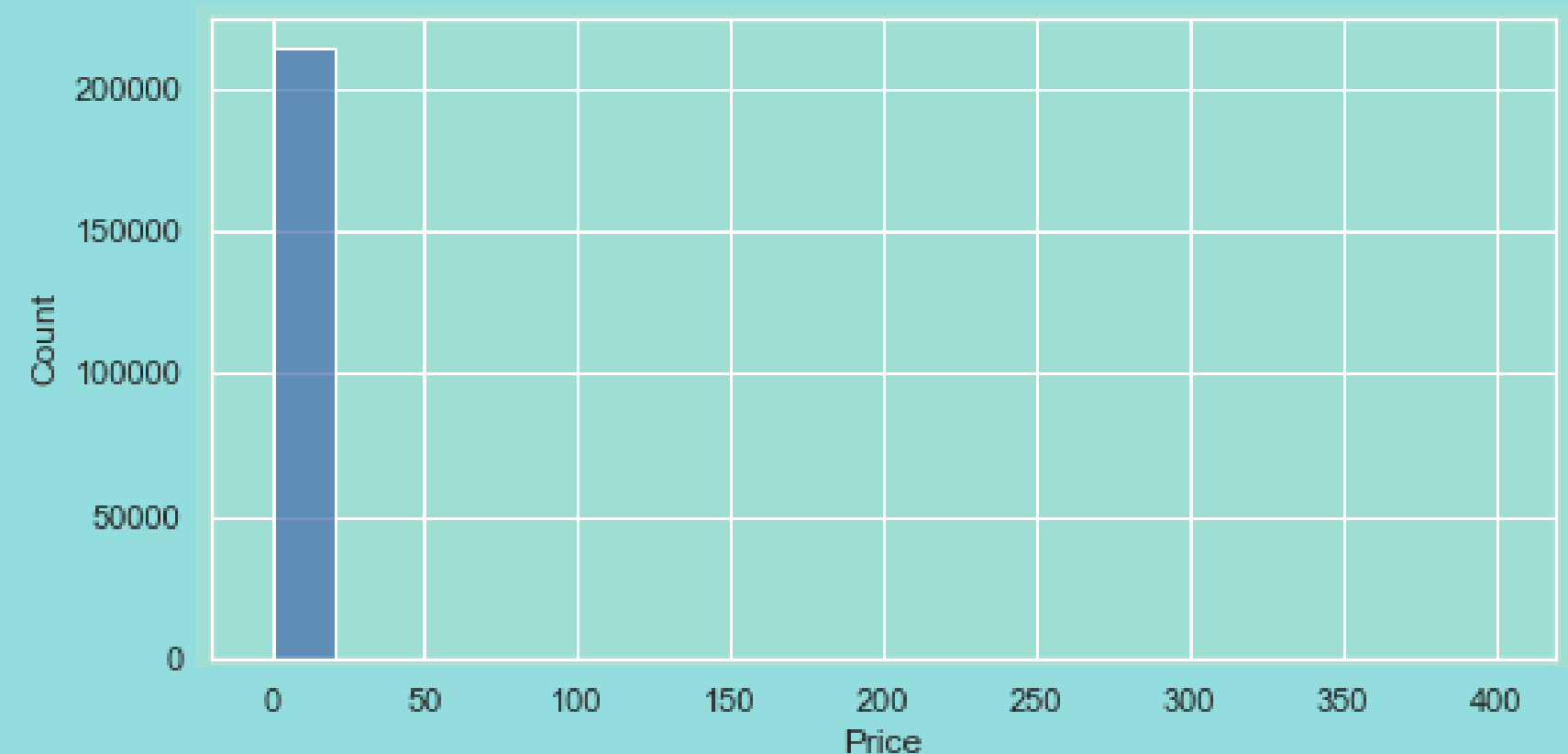
Count plot for 'GoodApp'

- Significant class imbalance, with the minority class being our class of interest (the positive class; 'GoodApp' == true)
- Expected, because there are many more not so good games than good ones
- Would result in poor predictive accuracy of that class as the machine learning 'ignores' that class
- Class imbalanced has to be fixed in the training data

Predictor Variable: 'Price'



Boxplot for 'Price'



Histogram for 'Price'

- Distribution positively skewed, many outliers at the upper end
- Majority of games are free, developers should consider letting people download their app for free to compete well in the market

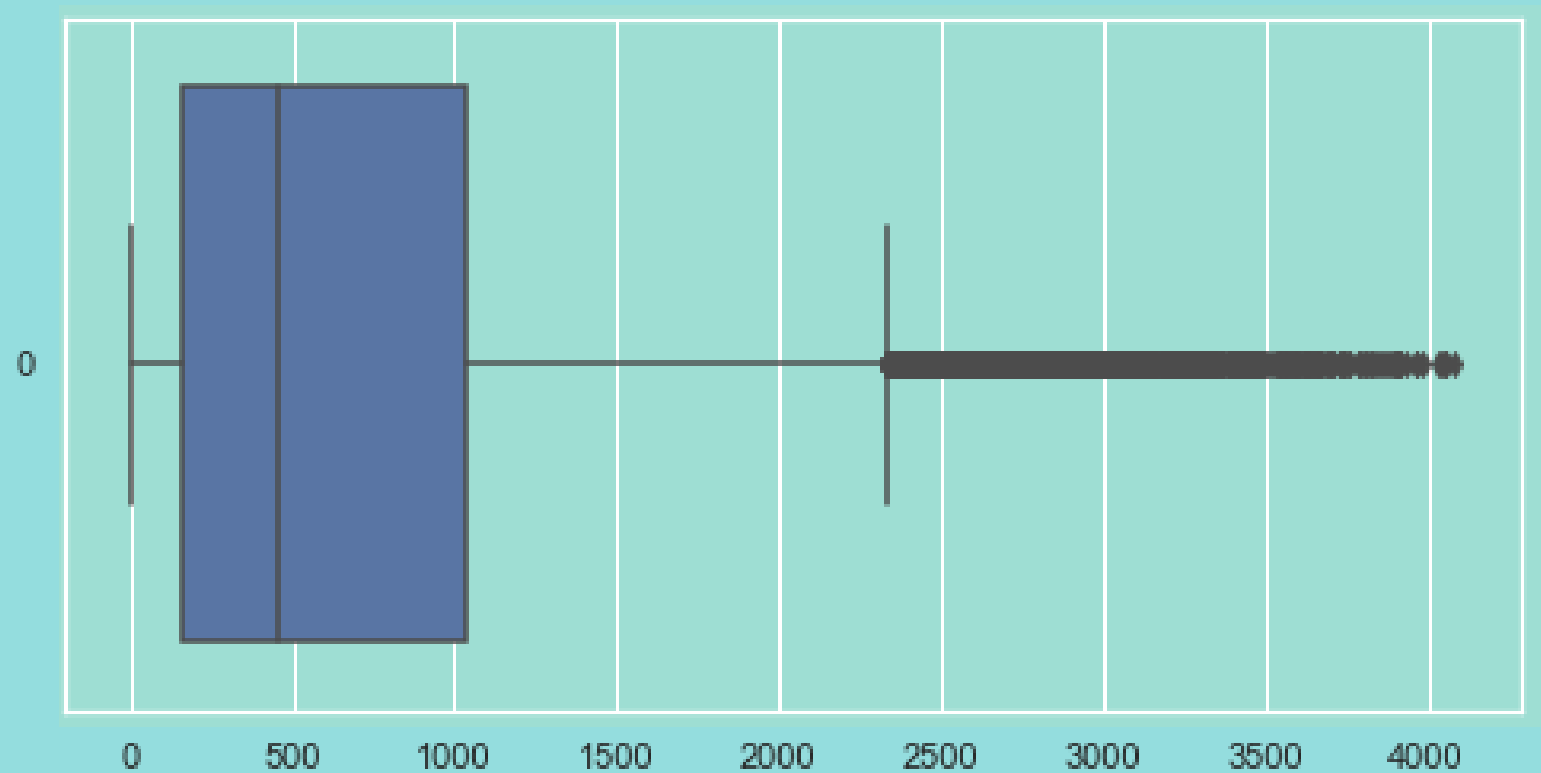
'Price' vs 'GoodApp'



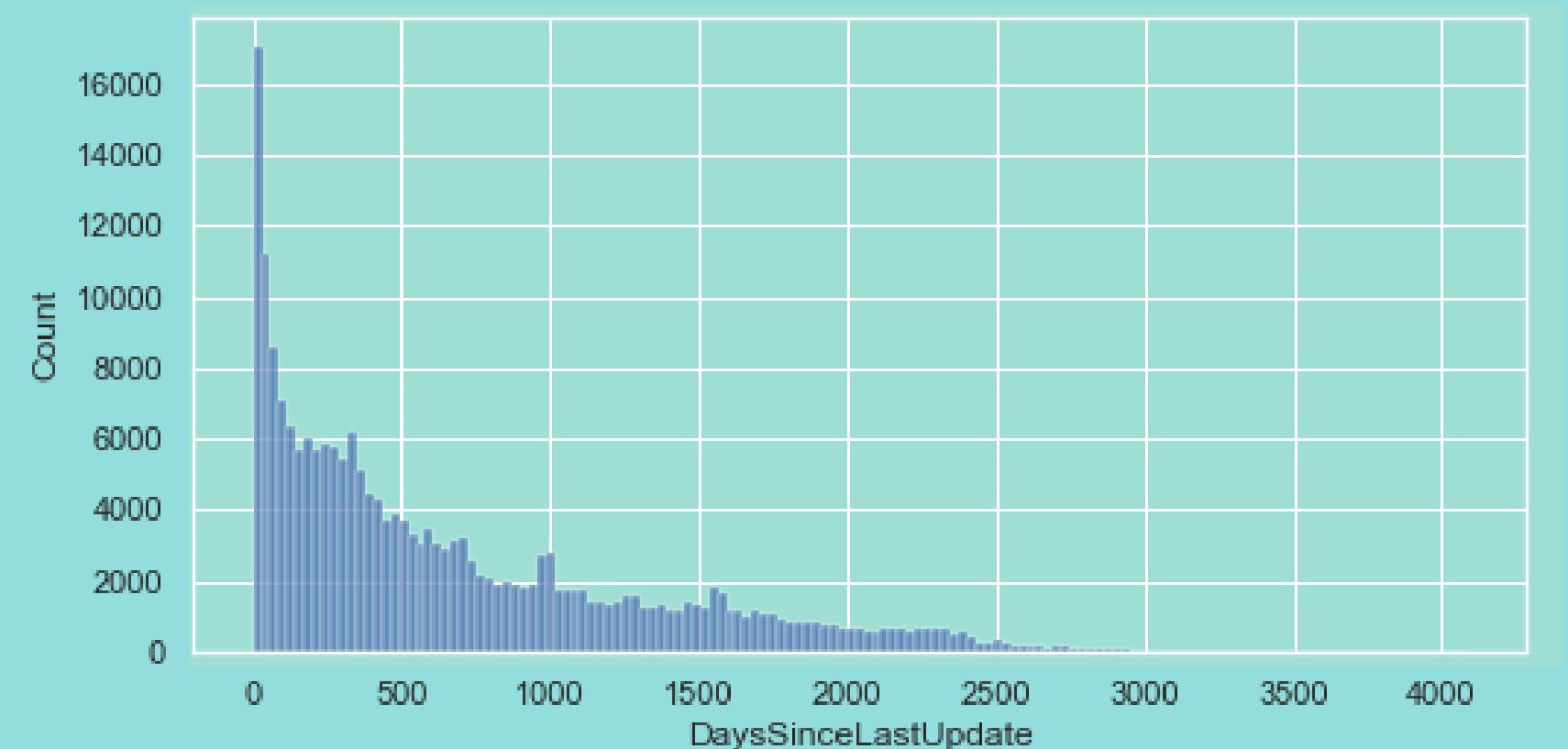
Strip Plot for 'Price' vs 'GoodApp'

- Good apps tend to not charge high prices upfront
- There are no good apps that charge exorbitantly high prices

Predictor Variable: 'DaysSinceLastUpdate'



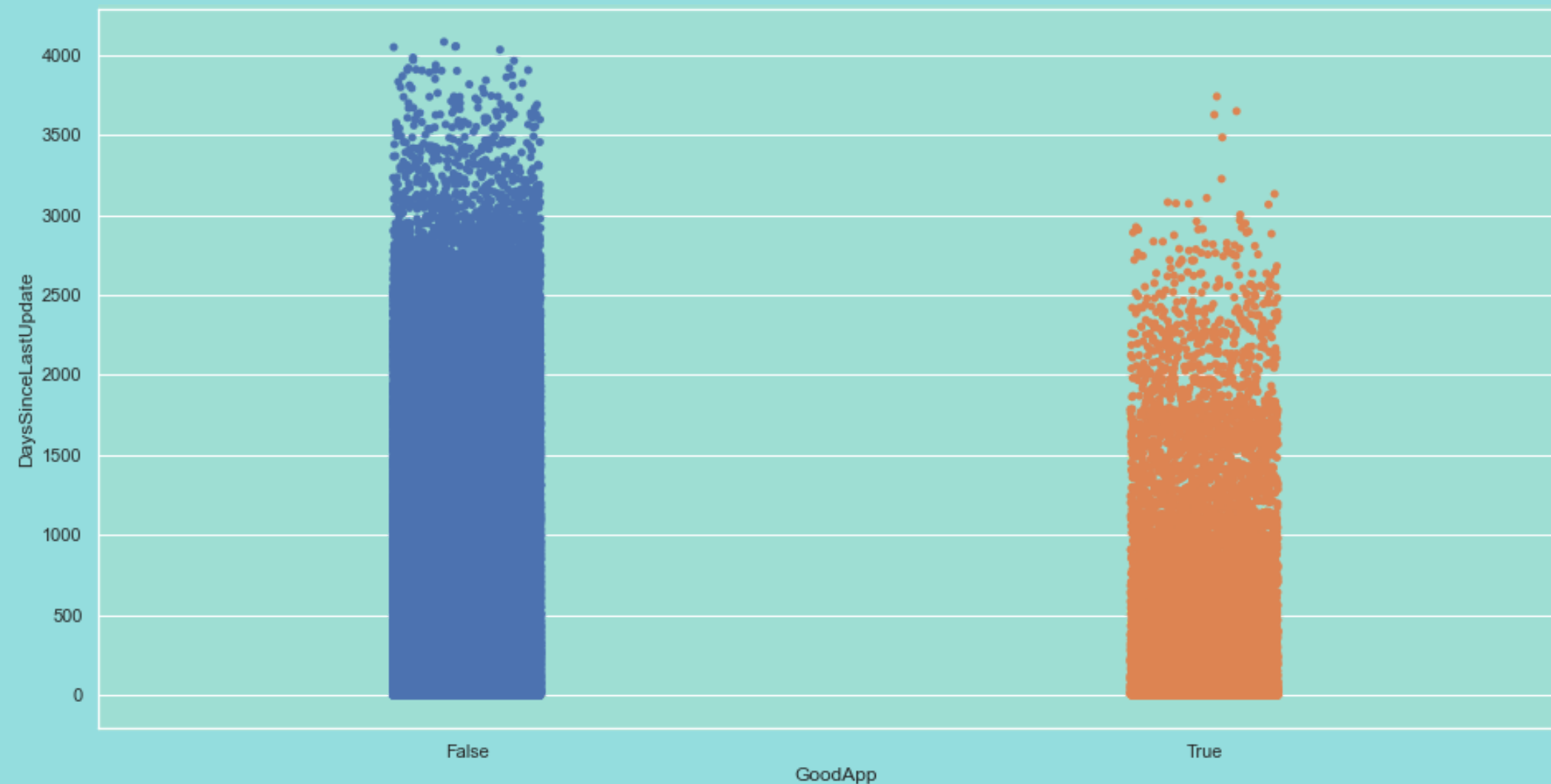
Boxplot for 'DaysSinceLastUpdate'



Histogram for 'DaysSinceLastUpdate'

- Distribution positively skewed, many outliers at the upper end
- Small number of out-of-date apps, but a majority of apps are up-to-date. This could also mean that the app was newly released on the play store

'DaysSinceLastUpdate' vs 'GoodApp'

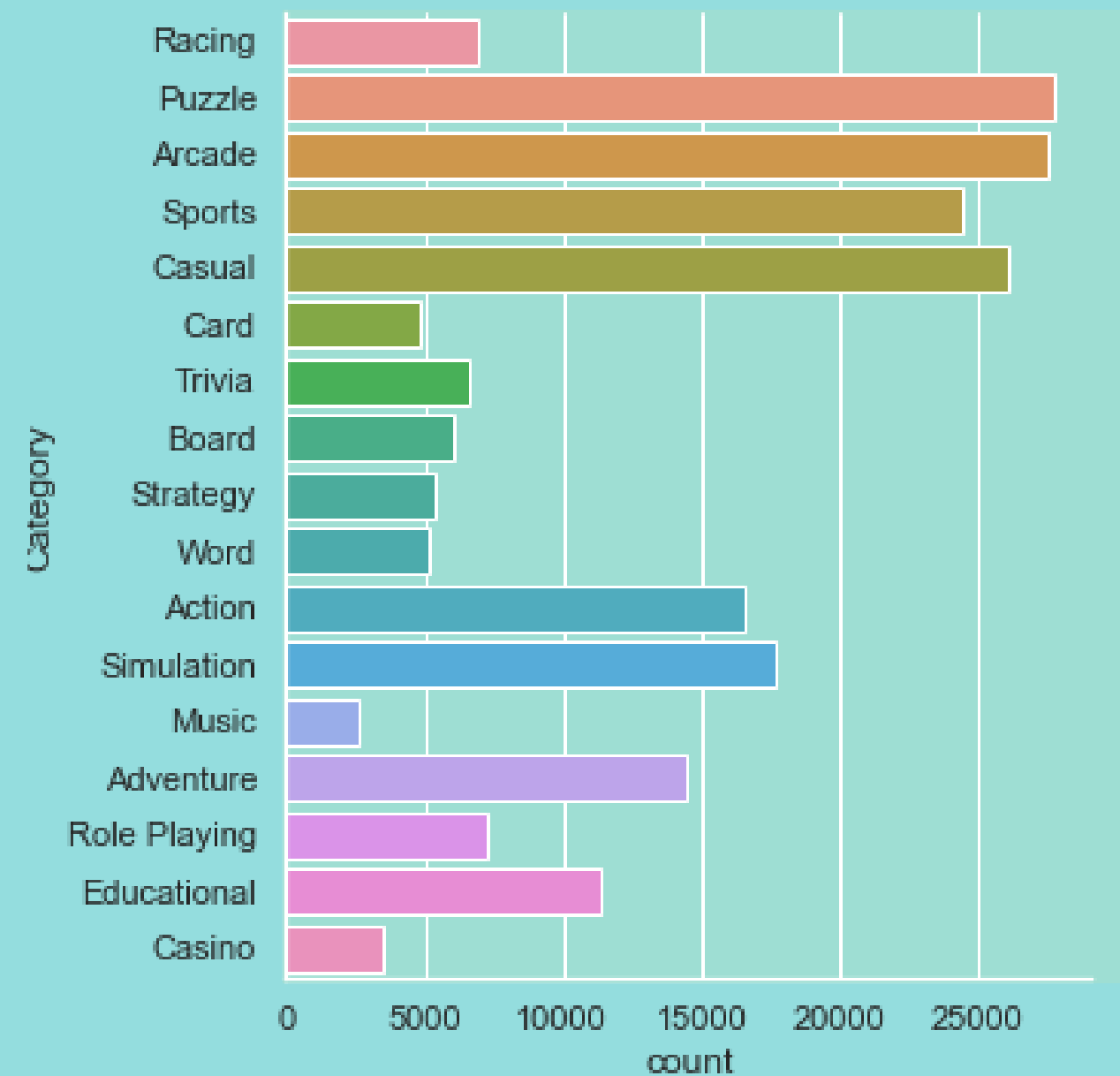


Strip Plot for 'DaysSinceLastUpdate' vs 'GoodApp'

- Very out-of-date app are unlikely to be good apps
- But updating your app frequently also does not guarantee it will be successful

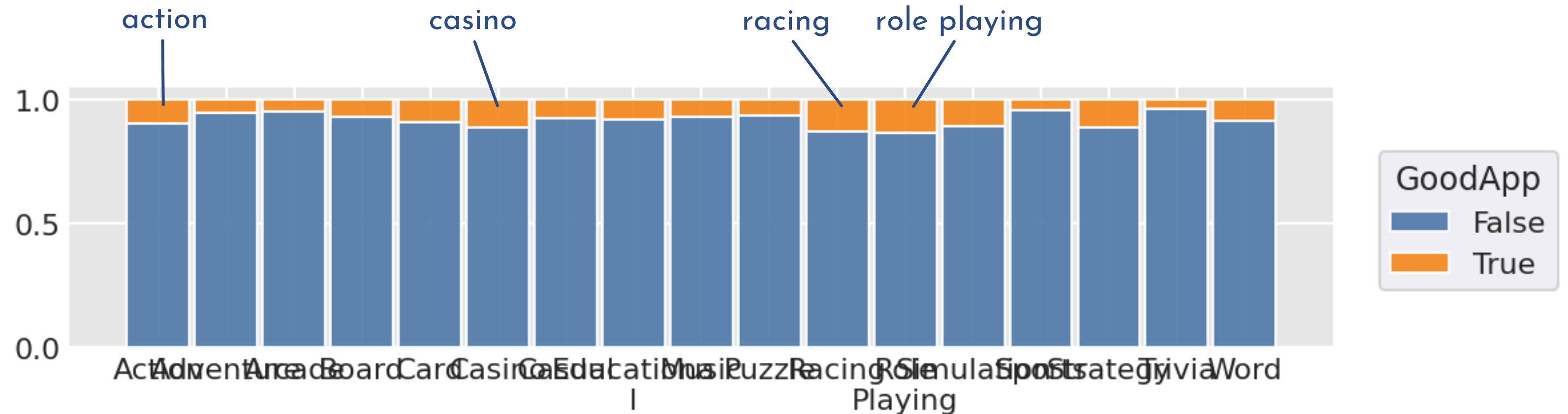
Predictor Variable: 'Category'

The most popular categories of games developed are:
Puzzle > Arcade > Casual > Sports



Count Plot for 'Category'

'Category' and 'GoodApp'



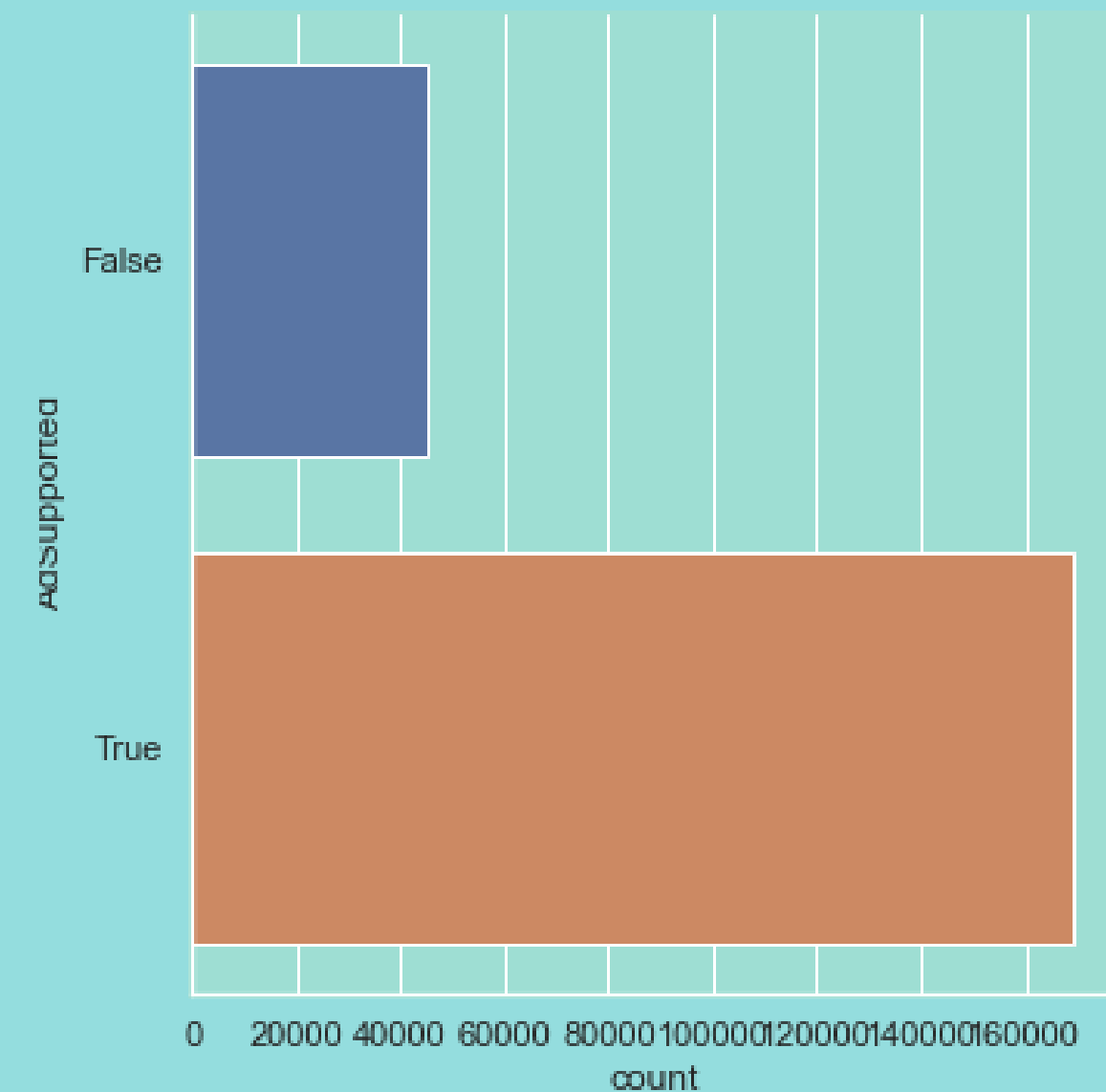
Proportion of 'GoodApp' for 'Category'

- However, the aforementioned categories don't produce the highest proportion of good apps
- Certain categories have a higher proportion of good apps, eg action, casino, racing, and role playing; developers can consider creating such games

Predictor Variable: 'AdSupported'

Most games run ads

Rather than charge an upfront cost
for downloading the app



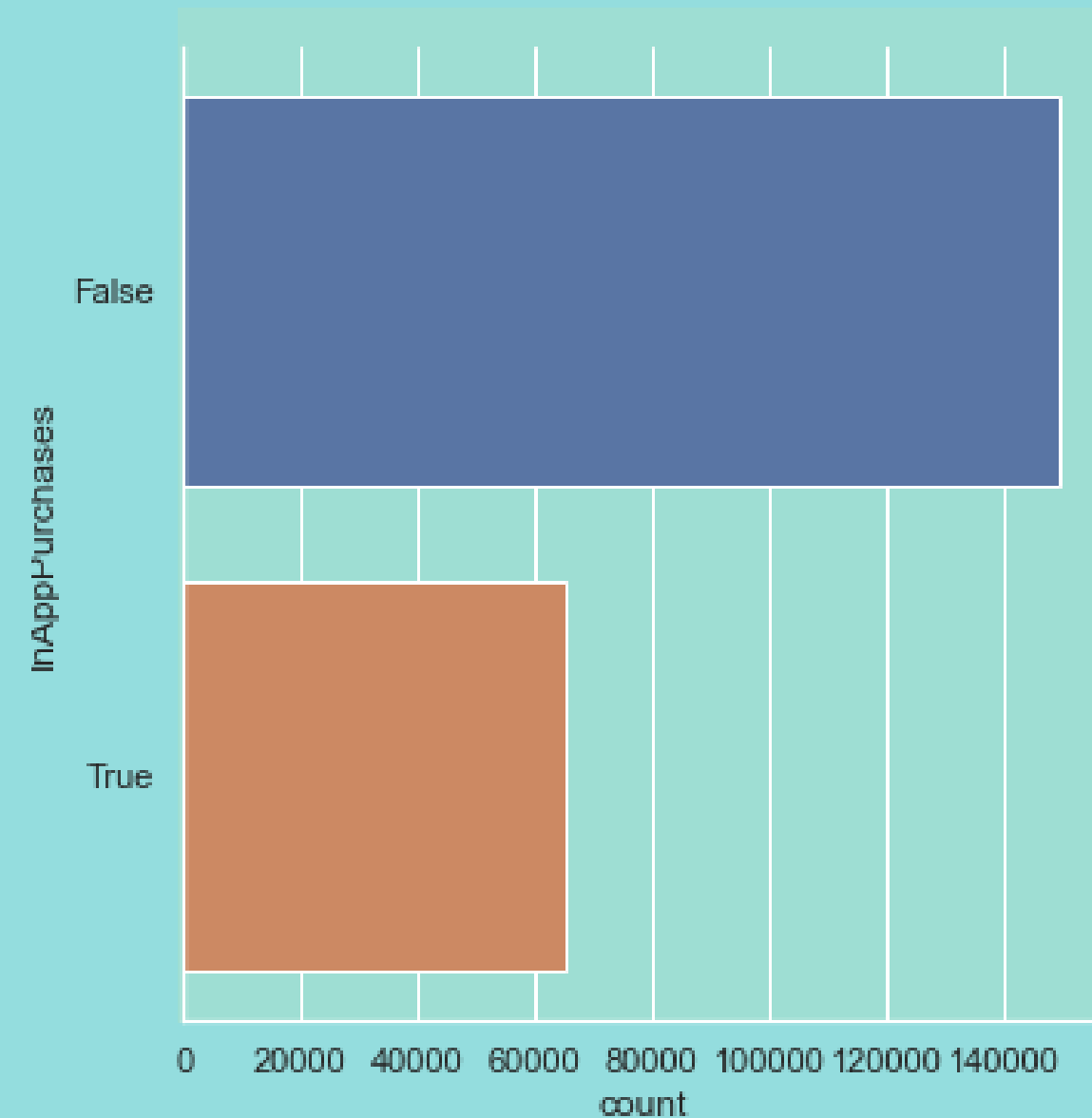
Count Plot for 'AdSupported'

Predictor Variable: 'InAppPurchases'

Most games do not have in-app purchases

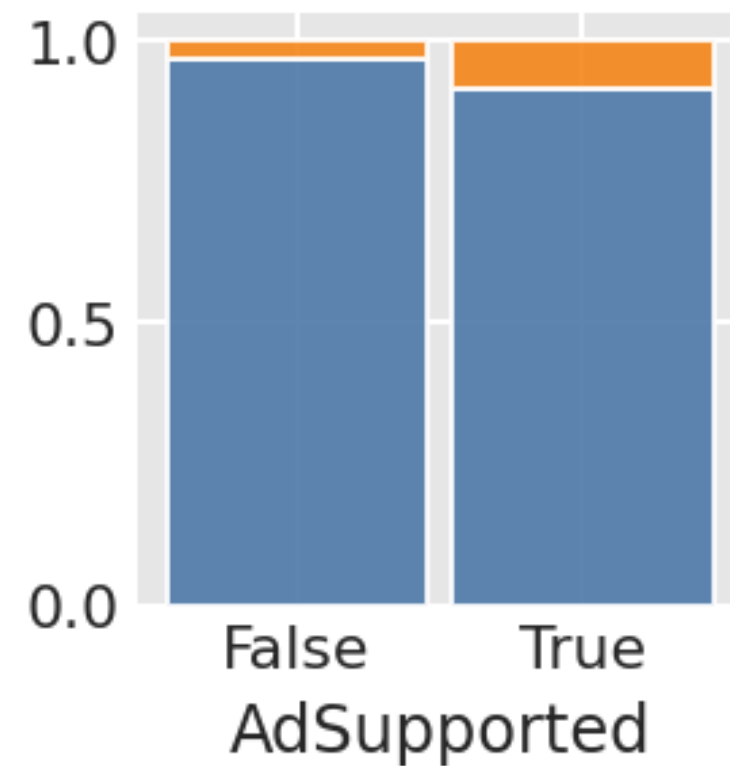
Meaning that there are no premium features or pay-to-win strategies

Running ads seems to be most popular way of earning app revenue

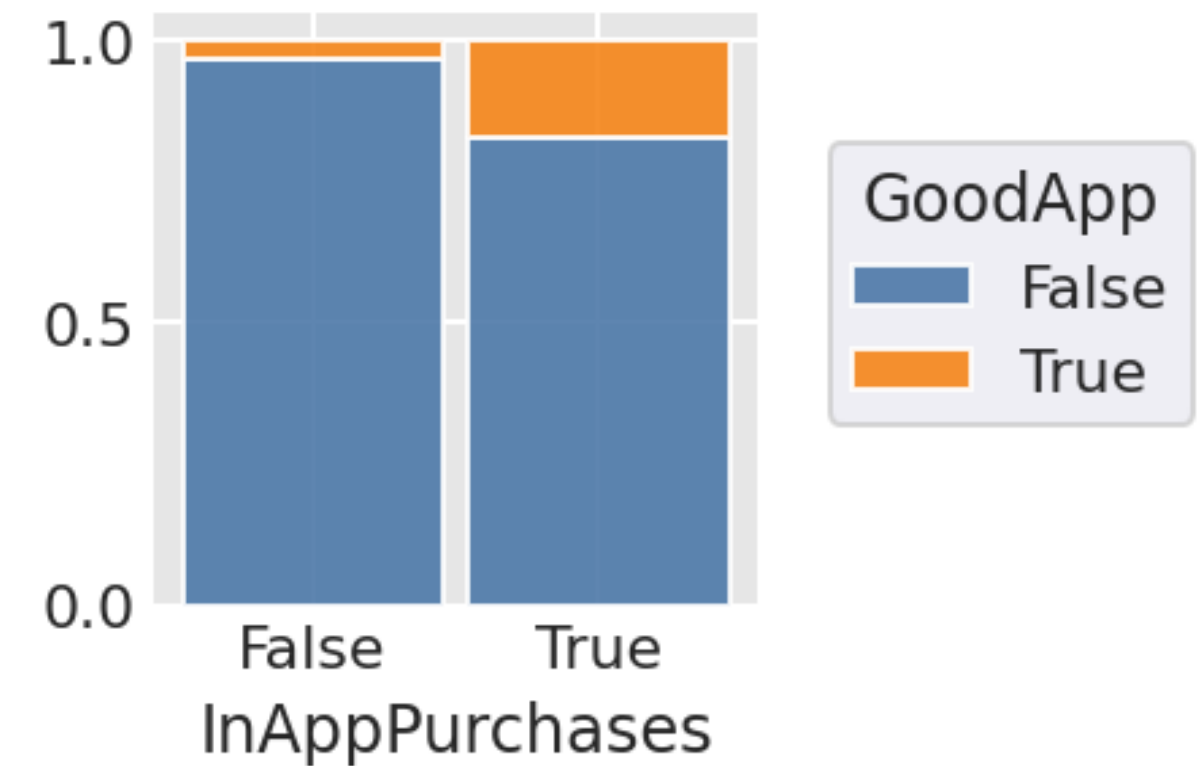


Count Plot for 'InAppPurchases'

In app purchases, ads, and 'GoodApp'



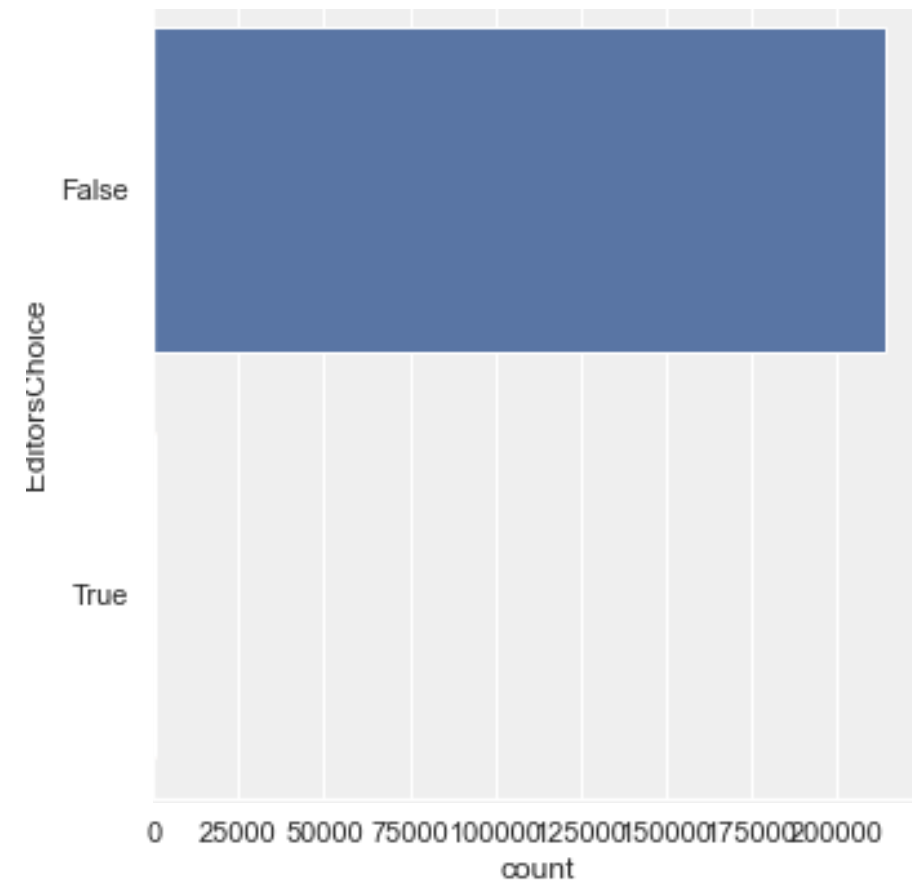
Proportion of 'GoodApp' for 'AdSupported'



Proportion of 'GoodApp' for 'InAppPurchases'

- Games that run ads have a higher proportion of good apps
- Games that have in app purchases have a higher proportion of good apps
- Could be that since developers earn a decent amount from it, they put more effort into the game and hence the app is more likely to be good

'EditorsChoice' and 'GoodApp'



Count Plot of 'EditorsChoice'



Proportion of 'GoodApp' for 'EditorsChoice'

- As expected, very few apps are editor's choice, as it's a coveted title
- A large proportion of editor's choice apps are good apps. The 'EditorsChoice' category is after all, looking out for the same qualities of an app as us, but including greater domain knowledge than our metrics
- Developers should study and learn from editor's choice games

Data preparation for machine learning

70:30 train-test split



To create train and
test set

pandas
get_dummies() function



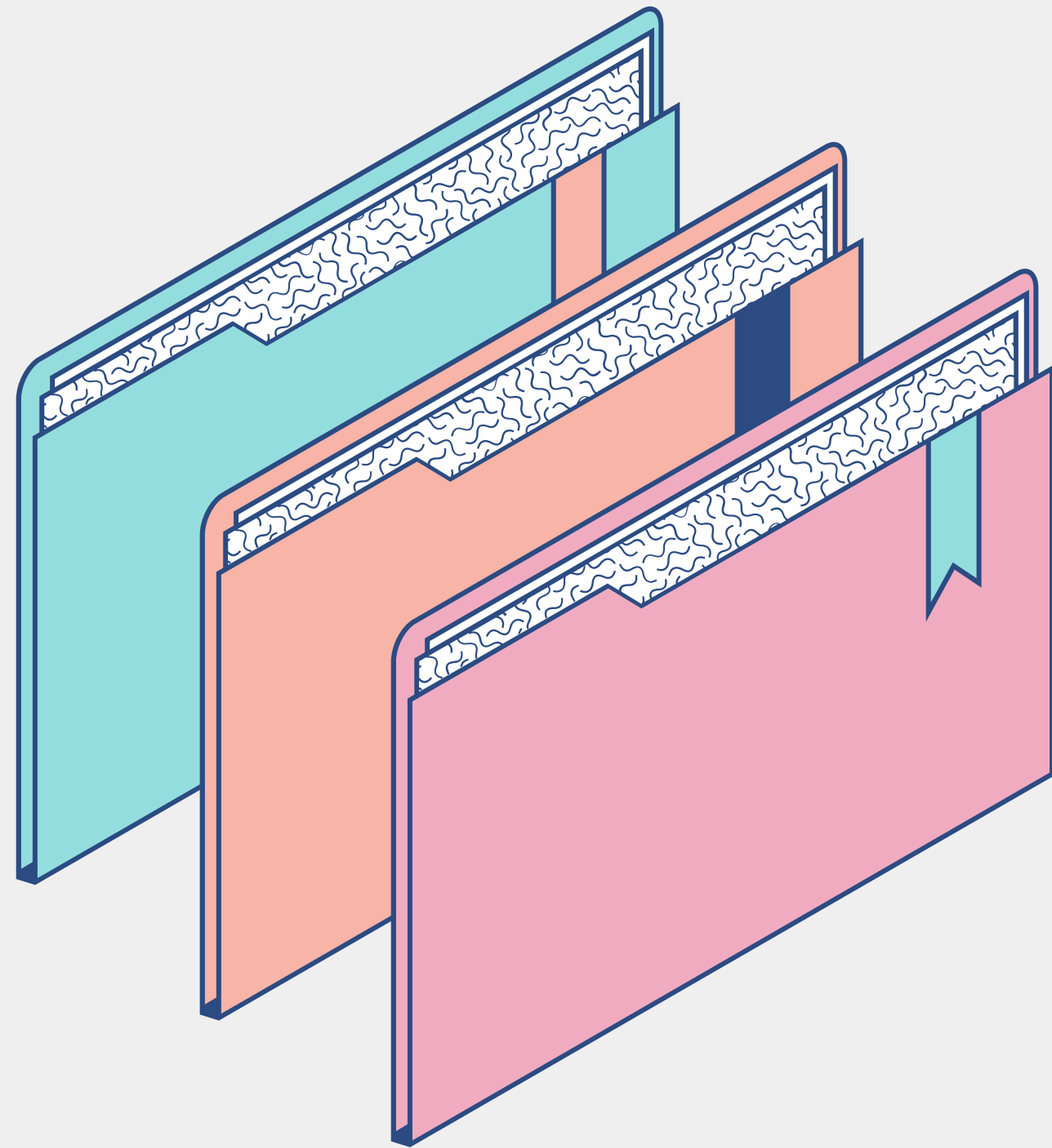
To encode categorical
data

Synthetic Minority
Oversampling Technique
(SMOTE)*



To correct class
imbalance in train set

*SMOTE oversamples the minority class, by creating
"new" minority class data points from existing data



Machine Learning Techniques

Predict whether a new game idea will be good, with factors that can be predetermined, helping developers know whether to embark on that project

Random Forest

- Concept of bagging: bootstrapping the dataset + aggregating each individual tree's prediction
- Creates numerous decision trees
- Trees are created independent of each other; each tree is different from each other due to bootstrapping and random subset feature selection
- Each individual tree's prediction constitutes a 'vote'
- The majority vote becomes the final prediction

AdaBoost

- New trees are added to correct the errors made by the previous tree, until the data is predicted perfectly or a maximum number of trees are created
- Creates decision stumps (one level decision tree)
- Trees are dependent on each other
- Gives higher 'sample weight' to hard-to-predict cases (cases that it misclassified), for the next tree

Logistic Regression


- Models the probability of an outcome
- Estimates the coefficients of the linear combination of the input variables
- Calculates the log odds of an outcome given the input predictor variables
- Uses the logistic function to convert log odds to a probability

Random Forest with XGBoost

- XGBoost implements the gradient boosting decision tree algorithm
- New models are created that predict the residuals of prior models, and then are added together to make the final prediction
- Uses gradient descent algorithm to minimise the loss when adding new models
- XGBoost can be configured to support random forest

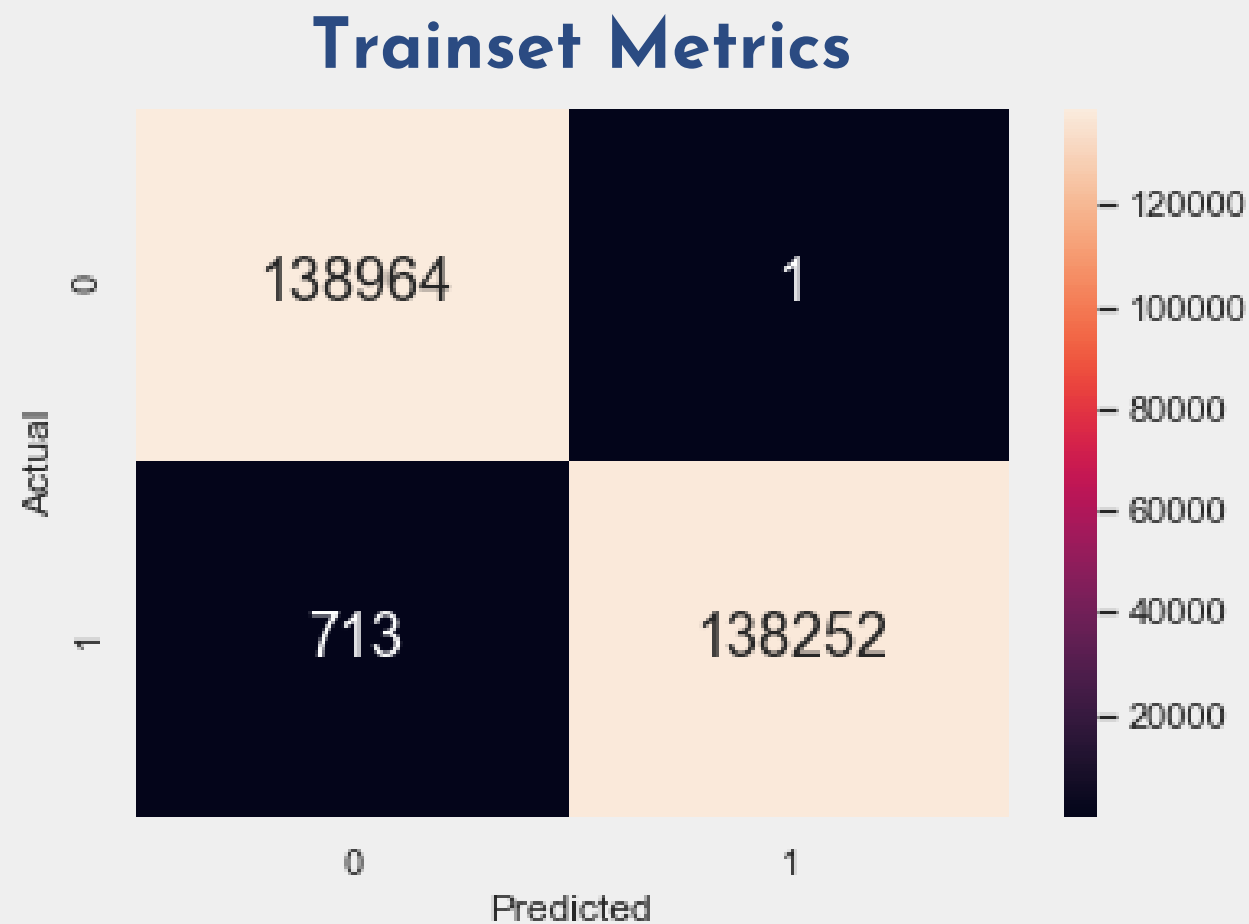
Evaluation parameters

- Train set
 - How good of a fit the model is on its training data
- Test set
 - Tells us the predictive accuracy of the model, on unseen, test data
- True positive rate
 - How well the model predicts the positive class
- False negative rate
 - How badly did the model predict the positive class
- True negative rate
 - How well the model predicts the negative class
- False positive rate
 - How badly did the model predict the negative class

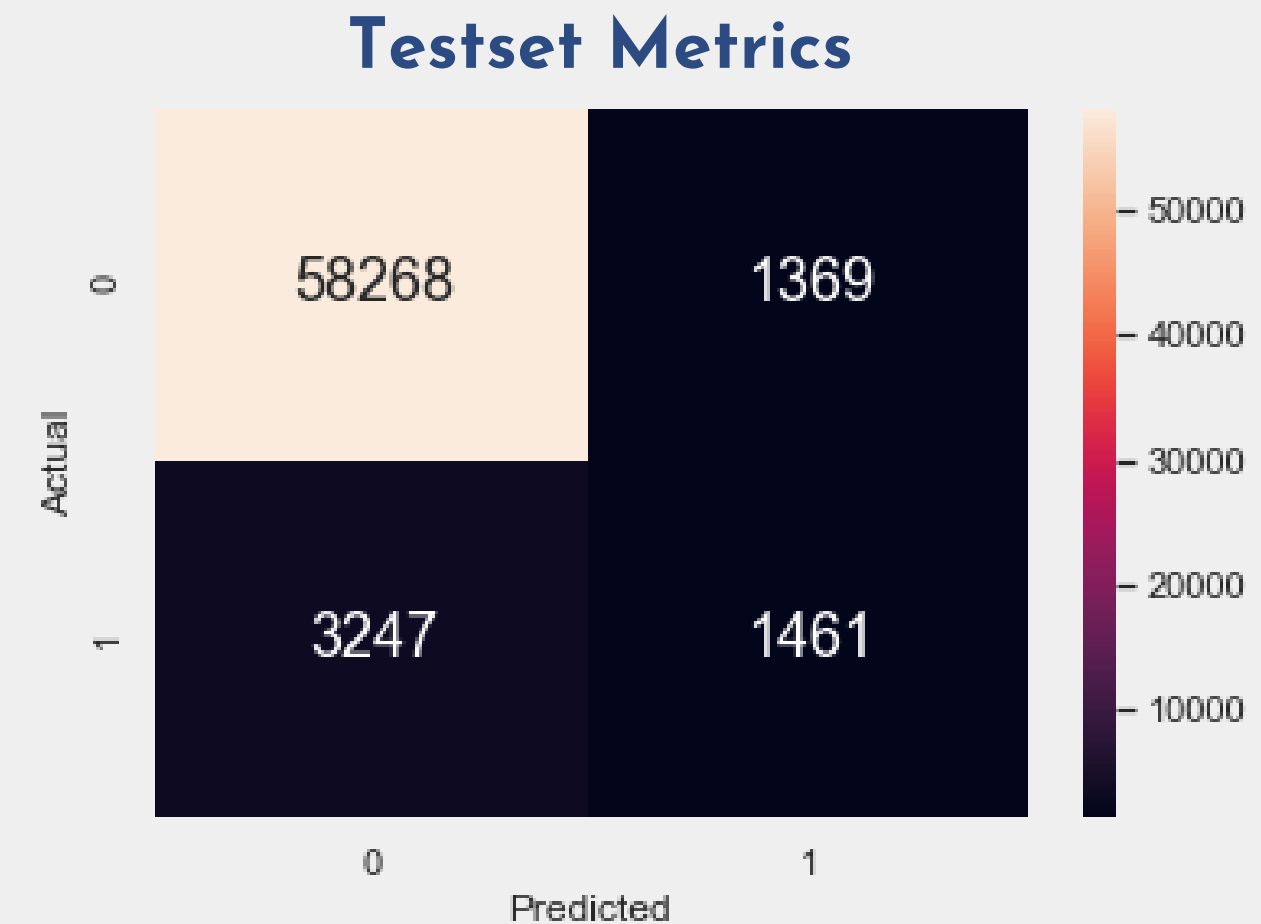


We care about both the positive and negative class, as we want to know whether a game will be good or bad. Hence we study all these metrics!

Random Forest



Overall Accuracy Rate: 99.74
TPR: 99.49 FNR: 0.51
TNR: 100.00 FPR: 0.00

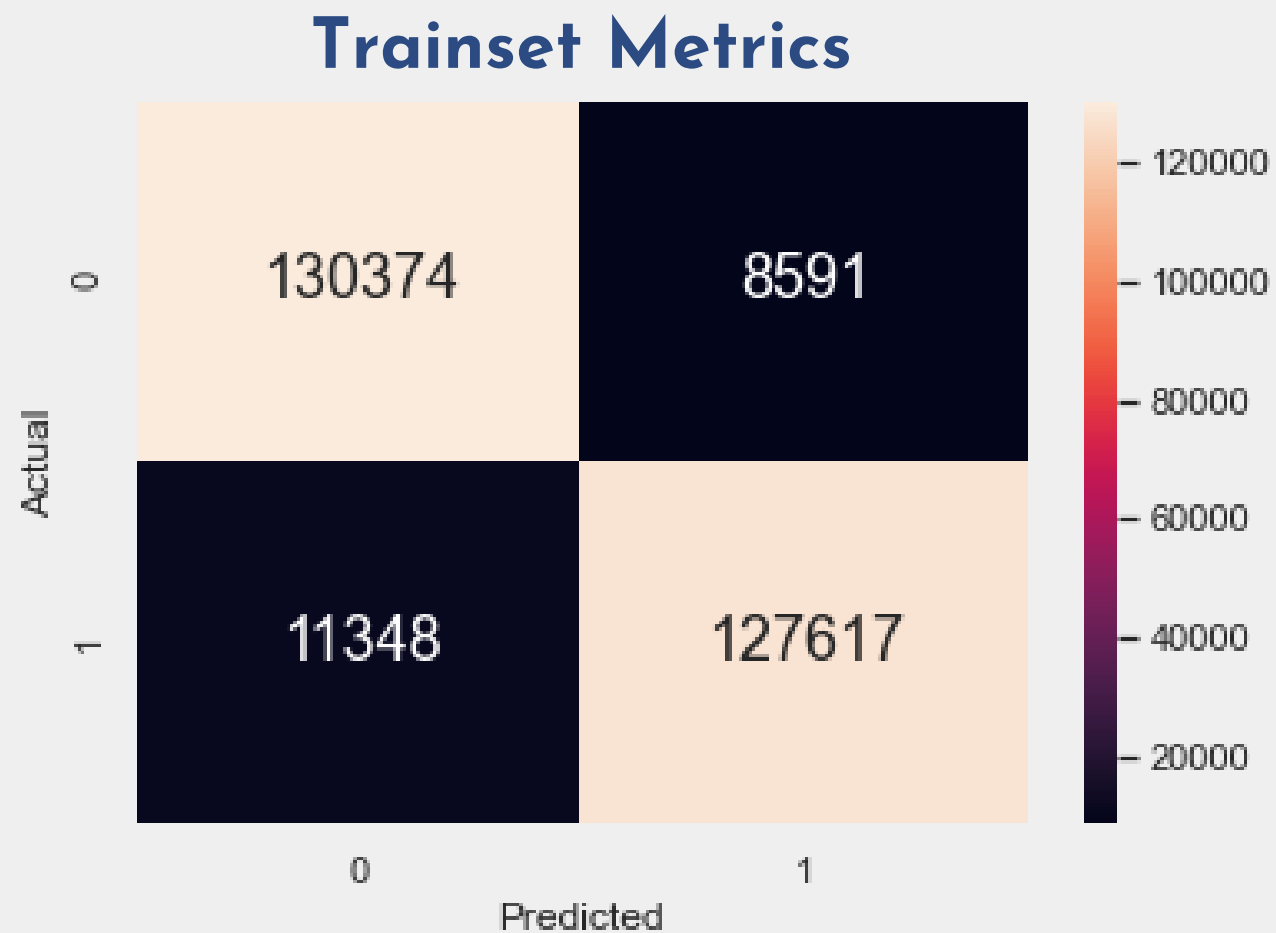


Overall Accuracy Rate: 92.83
TPR: 31.03 FNR: 68.97
TNR: 97.70 FPR: 2.30

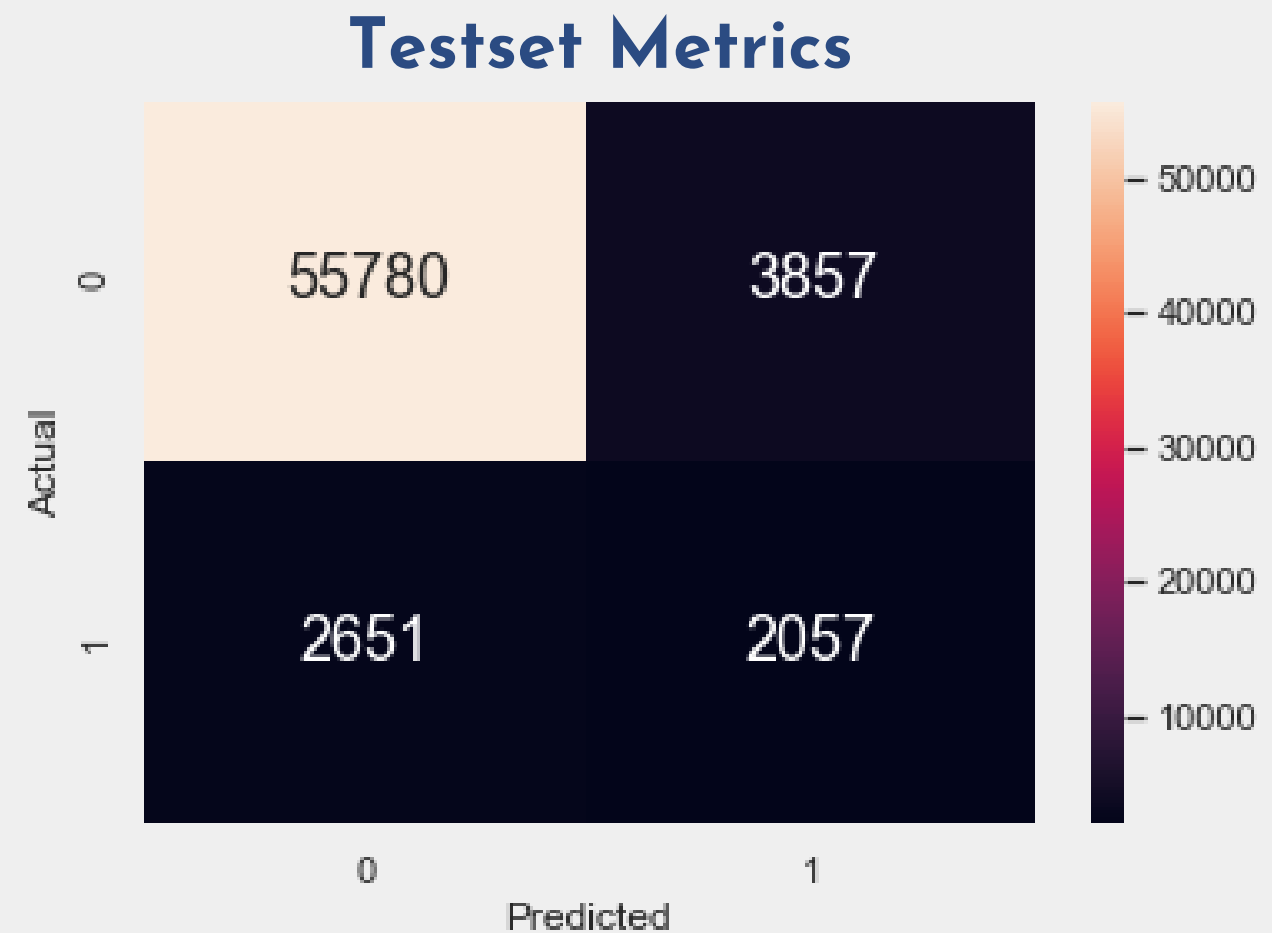
Performed very well on the trainset: good fit on training data

In testset, very poor predictions for the positive class: poor predictive accuracy on unseen data of the positive class

AdaBoost



Overall Accuracy Rate: 92.64
TPR: 91.79 FNR: 8.21
TNR: 93.48 FPR: 6.52

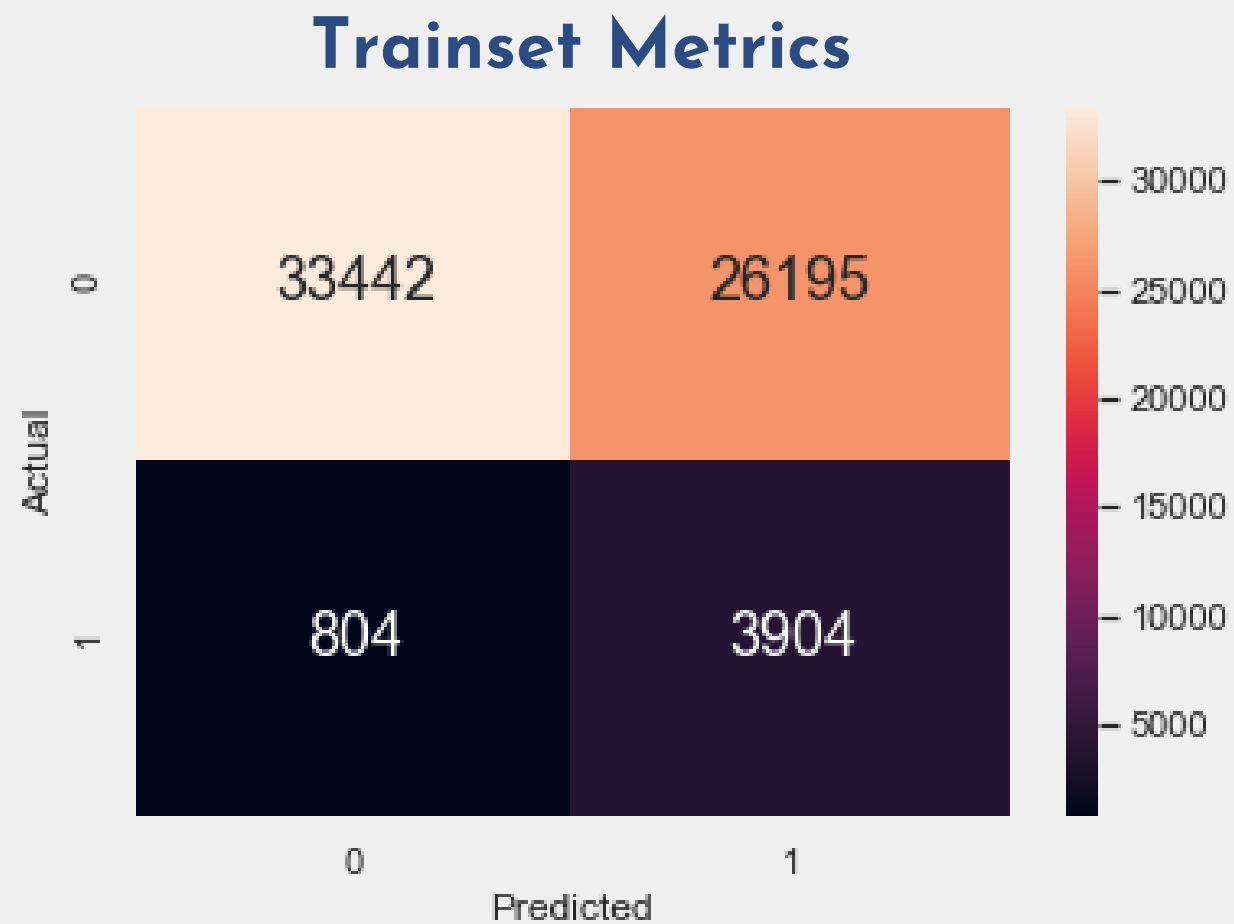


Overall Accuracy Rate: 89.89
TPR: 43.69 FNR: 56.31
TNR: 93.53 FPR: 6.47

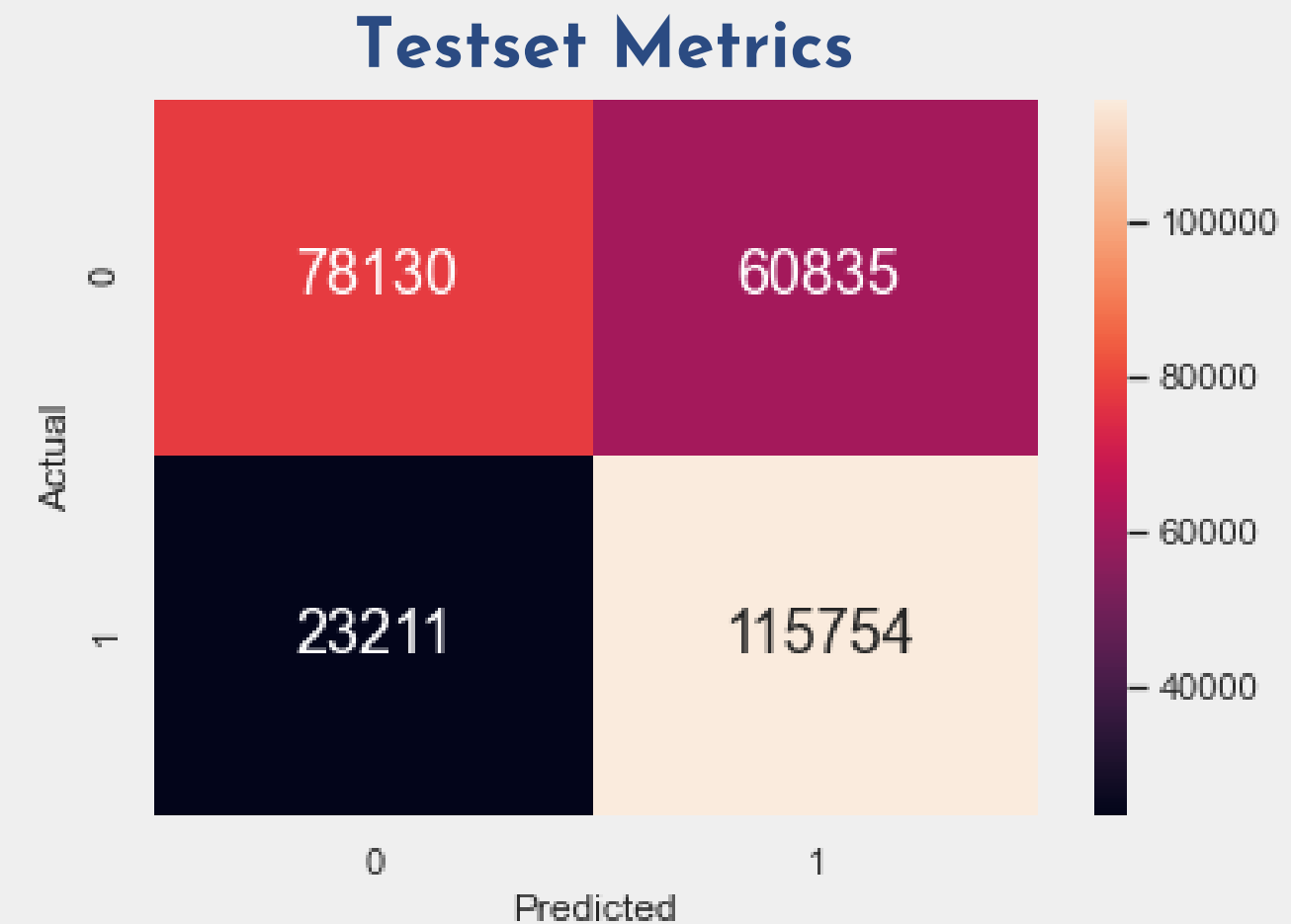
Performed well on the trainset: good fit on training data

In testset, unable to predict the positive class well: poor predictive accuracy on unseen data of the positive class

Logistic Regression



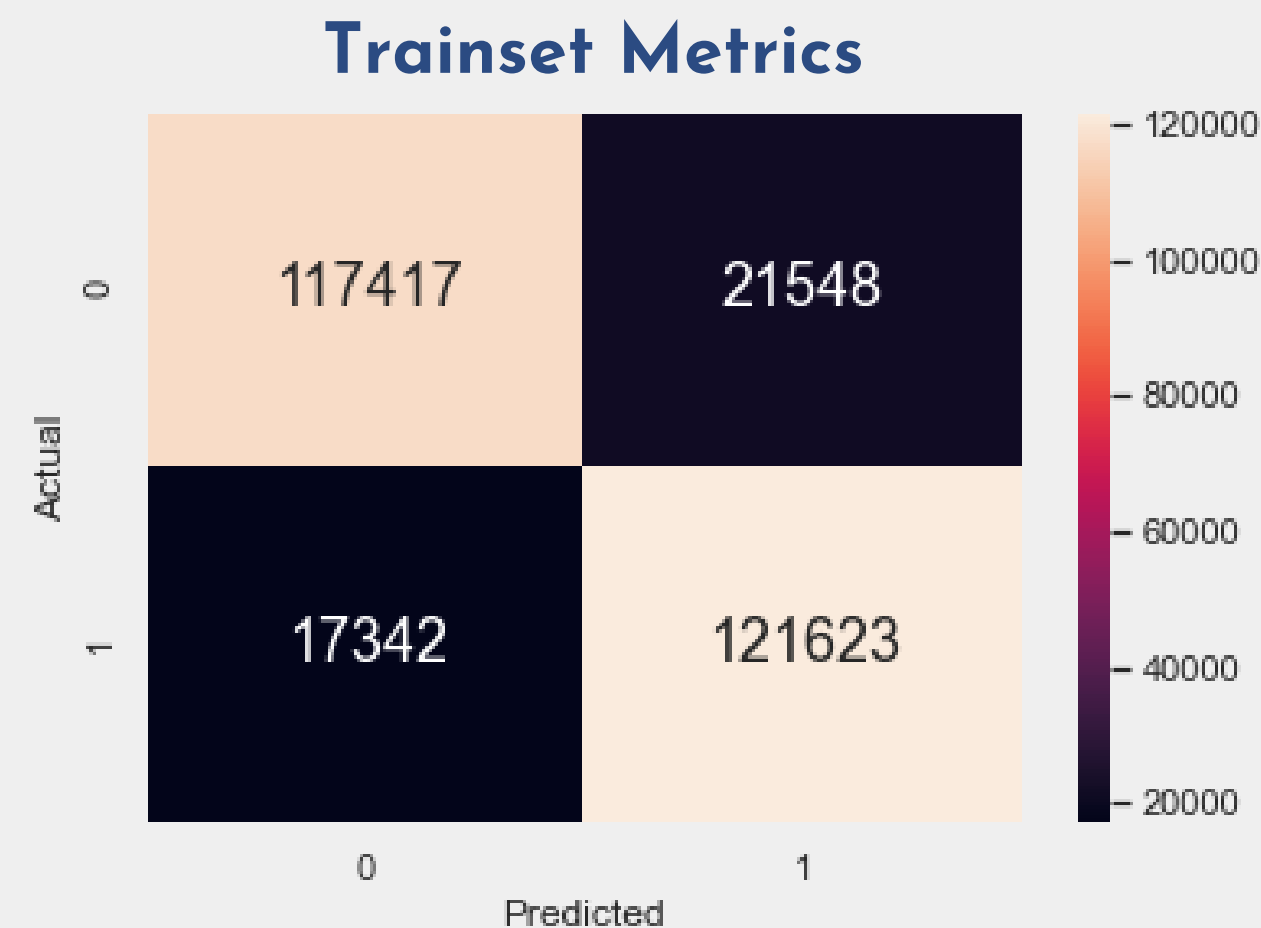
Overall Accuracy Rate: 69.76
TPR: 83.30 FNR: 16.70
TNR: 56.22 FPR: 43.78



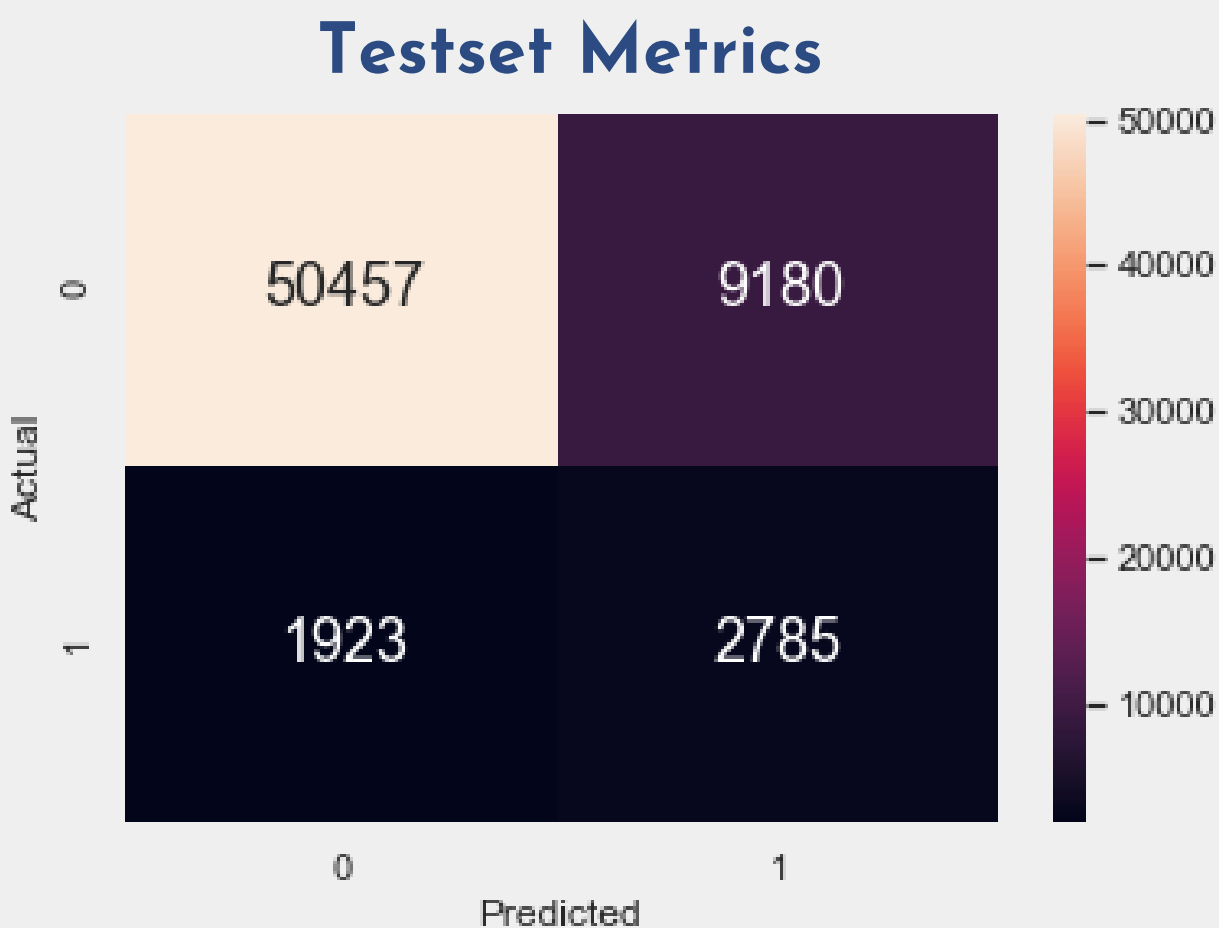
Overall Accuracy Rate: 58.04
TPR: 82.92 FNR: 17.08
TNR: 56.08 FPR: 43.92

Predicted the positive class well in training data, but did poorly on the negative class
In testset, unable to predict the negative class well: poor predictive accuracy on unseen data of the negative class

Random Forest with XGBoost



Overall Accuracy Rate: 85.97
TPR: 87.56 FNR: 12.44
TNR: 84.38 FPR: 15.62



Overall Accuracy Rate: 82.74
TPR: 59.15 FNR: 40.85
TNR: 84.61 FPR: 15.39

Good accuracy metrics on the trainset: model fits well on training data
Best balance between predicting the positive and negative class in the testset well, out of all the models

Comparing predictive accuracy

Decision Tree Models

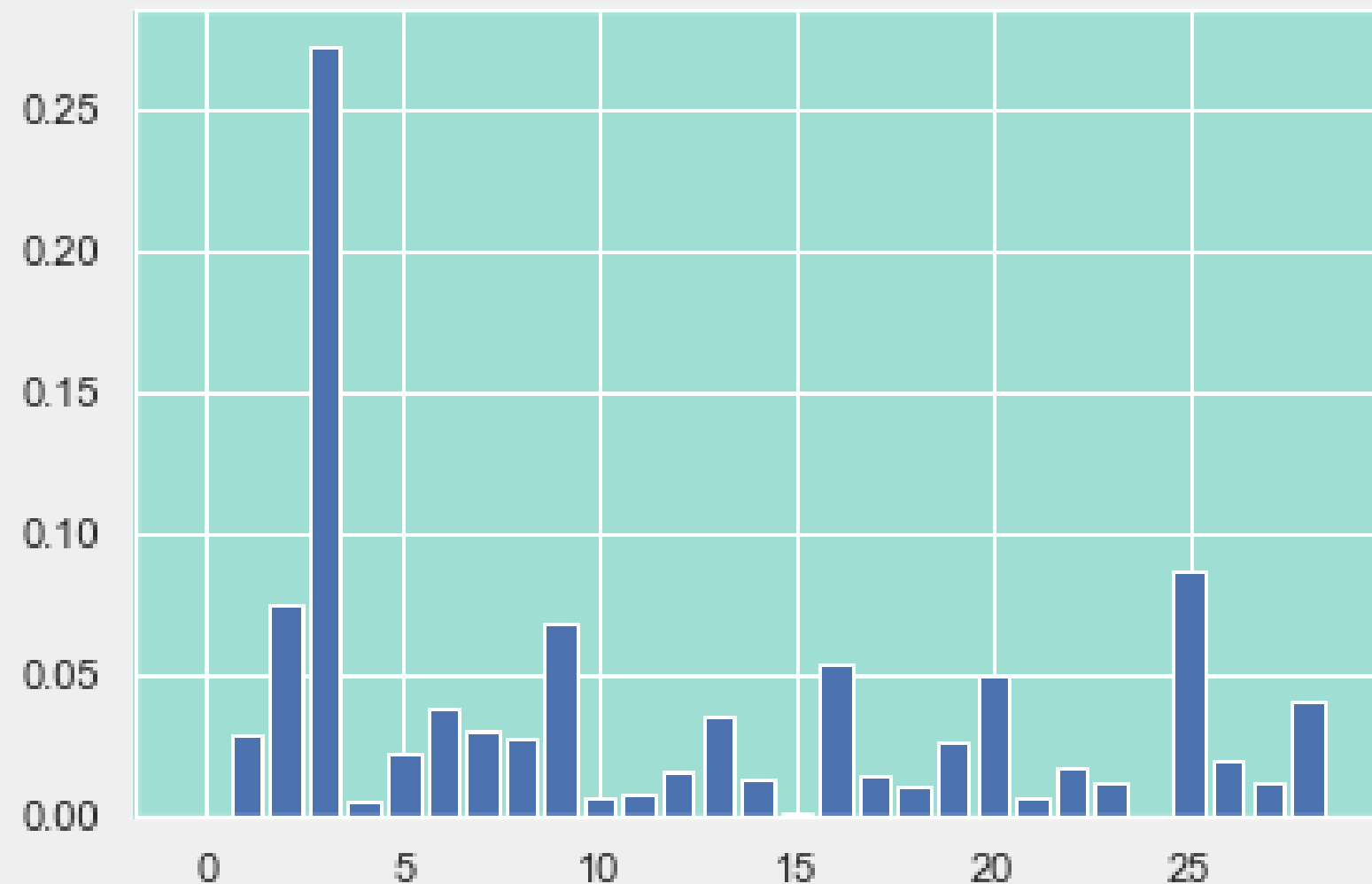
	Random Forest	AdaBoost	Random Forest with XGBoost	Logistic Regression
TPR	31.03	43.69	59.15	82.92
FNR	68.97	56.31	40.85	17.08
TNR	97.70	93.53	84.61	56.08
FPR	2.30	6.47	15.39	43.92

Random forest with XGBoost had the best balance between predicting the positive and negative class decently, out of all the models, although prediction of the negative class is not the best. Hence, we suggest developers to use this model for prediction.

Feature importance

- We are able to extract the feature importance from the decision tree models
- We cannot get feature importance from logistic regression, unless we scale the numeric variables
- Since we chose random forest with XGBoost, we shall extract the feature importance from that model

Feature importance



Feature Importance Derived from RF w XGB

- Index 3, corresponding to 'InAppPurchases' had the highest feature importance by far
- Followed by 'ContentRating_Everyone' > 'AdSupported' > 'Category_Arcade' > 'Category_Puzzle'

Insights from feature importance

- In-app purchases
 - Add features that incentivise players to spend money to hasten in-game progression, or win more easily
- Content rating
 - Have the games be appropriate for all audiences, to reach a greater target audience
- Ads supported
 - It is fine to run in game ads, be sure to do it tactfully eg. tasteful ads that help a player progress faster
- Game category
 - Some categories are bound to be more successful than others

Analysis limitations

- Need better predictor variables
 - More relevant data, such as the number of people on the development team behind each app, would help improve model accuracy
- SMOTE alters the distribution of data, such that it does not reflect the true distribution of good apps in real life

Conclusion

- Using our random forest with XGBoost model, mobile game developers can:
 - Predict the likelihood of their game's success using the model
 - Modify the game pre-development variables, to improve their likelihood of success based on the model



Thank you!

Greater project detail can be found in our
Jupyter notebooks on GitHub!

