TEAM F9 - NINJA KIWI

GAME ON!
BTD6 DATA ANALYSIS

TECHNICAL APPENDIX

PRESENTED BY

PEIHENG LI #PLI135
FAHAD SIDDIQUI #FSID783
SUNNY POON #HPPO600
VIGNESH JAIRAM #VJAI535
RAMEEN IFTIKHAR #RIFT670

Table of Contents

Introduction

As a gaming company, understanding player behaviour is very important to increase revenue and drive engagement for Ninja Kiwi, our client. Bloons TD 6, their flagship game, was released in 2018 and has since been a popular form of entertainment.

Player privacy is very important to the client, so they do not implement code that tracks a player's demographic information, and it is what makes them stand out. Despite that, they want to understand how the players engage with the game and group them so that they can target each group with additional content, accordingly, benefiting the players and the business.

The data they have is all psychographic i.e. how the players interact with different features of the game. We have information such as how many maps were created, what platform was the game played on, how many trophies were gained, and much more.

The goal of this project is to:

- understand these psychographics and group the players accordingly
- measure how different these groups are in terms of in-app purchases (IAP) and churn
- determine what groups would be the main focus for future content
- propose changes to improve certain groups' IAP spending and retention rates

Data Understanding

Data Source Description

We have been granted access to a database comprising 15 tables, each detailing various aspects of the game, such as user spending on in-app purchases and the number of times the app was used.

Table 1 below provides descriptions of each database table:

Table	**Description**
btd6_startsession	Information on platform and game version
btd6_starttrack	Records each instance of a user starting a new BTD6 map
btd6_endtrack	Matches starttrack records with game results
btd6_placetower	Logs each occurrence of a tower being placed
btd6_madet5	Logs when a player creates a tier 5 tower
btd6_madeparagon	Logs when a player creates a paragon tower
btd6_useactivatedability	Tracks the use of tower abilities within the game
btd6_usepower	Tracks the use of additional non-tower-based powers purchasable within the game
btd6_gainmonkeymoney	Records earnings of monkey money from winning levels, daily rewards, etc.
btd6_spendmonkeymoney	Tracks spending of monkey money on powers, heroes, skins, etc.
btd6_gaintrophies	Logs trophy gains from events, which can unlock specific items
btd6_spendtrophies	Tracks spending of trophies on powers, heroes, skins, etc.
btd6_unlockhero	Records the unlocking of new heroes using monkey money
btd6_unlockheroskin	Logs the unlocking of hero skins after the hero has been unlocked
btd6_buyiap	Contains details of all real-world money purchases made within the game

Data Cleaning and Preprocessing Steps

Our first goal was to create a table with variables that can be used for clustering and modelling. The final table contains 48 variables which we deemed relevant for our analysis.

Feature engineering involves creating new features to better capture the underlying patterns in the data. This process enhances the predictive power of the dataset. Several feature engineering techniques were employed:

- session_datediff: Calculated the difference between the maximum and minimum session dates to capture the activity span of users.

  ```
  a.max_sessiondate::date - a.min_sessiondate::date
  ```

- played_within_30days: Analysis revealed that 7% of users interacted solely within the last 60 days and not in the last 30 days, while 11.4% engaged within the last 30 days. Based on these findings and consultation with the client, we determined the 30-day threshold as the criterion for active users, with the remaining users categorised as churned. Therefore, we created a feature to indicate if a user played within the last 30 days.

  ```
  case

        when (select '2024-02-10'::date - a.max_sessiondate::date) <= 30 then
  1

        else 0

  end -- 2024-02-10 is the last day we have data for
  ```

- ios_flag, android_flag, steam_flag: Created binary flags to indicate which platforms were used by the user.

  Example: `MAX(CASE WHEN platform = 'ios' THEN 1 ELSE 0 END) AS ios_flag`

- avg_days_per_week: Average of the number of distinct days the app was opened per week.
- Different game modes were grouped and their proportions were calculated in this way:

  ```
  COUNT(CASE WHEN gamemode in ('Elite Boss', 'Regular Boss') THEN 1 END) *
  1.0/count(*)
  ```

- A flag iap_flag was created to indicate whether a user made an in-app purchase (1) or not (0).
- Win rate is the proportion of games won out of the total games played.

  ```
  COUNT(DISTINCT CASE WHEN win_loss_quit = 'won' THEN game_id END)*1.0/
  COUNT(DISTINCT game_id)
  ```

- In-App Purchases: Aggregated in-app purchase data to determine total spend (iap_spend), the number of items purchased, and proportions of different purchase types (monkeymoney_purchased_prop, cashmode_purchased_prop, etc.).

In the process of data preparation and cleaning, handling missing values is crucial to ensure the accuracy and reliability of the dataset. For features where a missing value indicates the absence of

an event or condition, zeros were used. For example, for features like num_free_heroes, num_paid_heroes, trophies_spent, and similar features, missing values were replaced with 0. This indicates that no heroes were unlocked or no trophies were spent by the user.

Moreover, recognizing the need for sufficient data per player for effective clustering, we filtered our users to include only those who have reached minimum 30 levels. This threshold ensures that users have interacted extensively with the game, as most features become accessible after this point, making it an appropriate cut-off. Hence, our final table only had approximately 5.9 million users.

To handle outliers in the continuous numeric variables, we limit extreme values to the 1st and 99th percentiles. Afterward, we standardise these variables to ensure that all features contribute equally to the clustering process.

```
#Defining numeric variables to deal with outliers and scaling

continuous_numeric_variables=['num_sessions','monkeymoney_gained_excl_iap','
avg_moneygained_persession', ] # there is a longer list but some columns have
been removed for the report

#Handling outliers: capping and flooring

for col in continuous_numeric_variables:

    low = df[col].quantile(0.01)

    high = df[col].quantile(0.99)

    df[col] = df[col].clip(lower=low, upper=high)
```

Exploratory Data Analysis (EDA) Results

While performing EDA on each table, we found interesting patterns representing users' preferences, playing style, activity durations, winning rate, etc. These initial observations were crucial to understand and get a feel of the data. The findings were visualised to elevate our comprehension of these patterns, transforming raw data into a compelling narrative of players behaviours. For instance, below we can see within table EDA for tables like buyiap, endtrack, and starttrack.

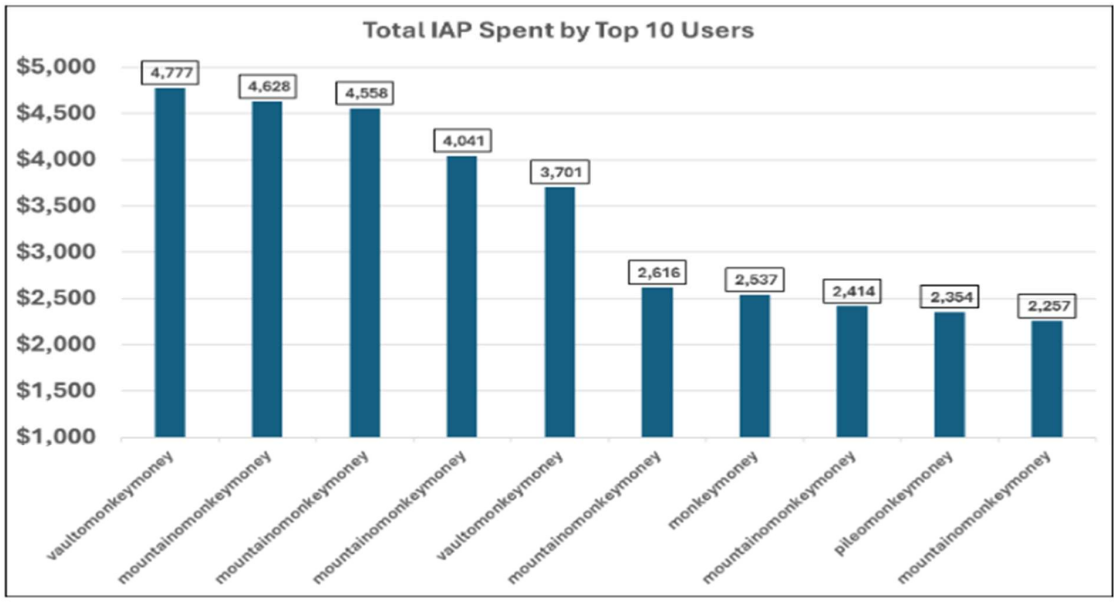


Figure 1, Total IAP Spent by Top 10 Users

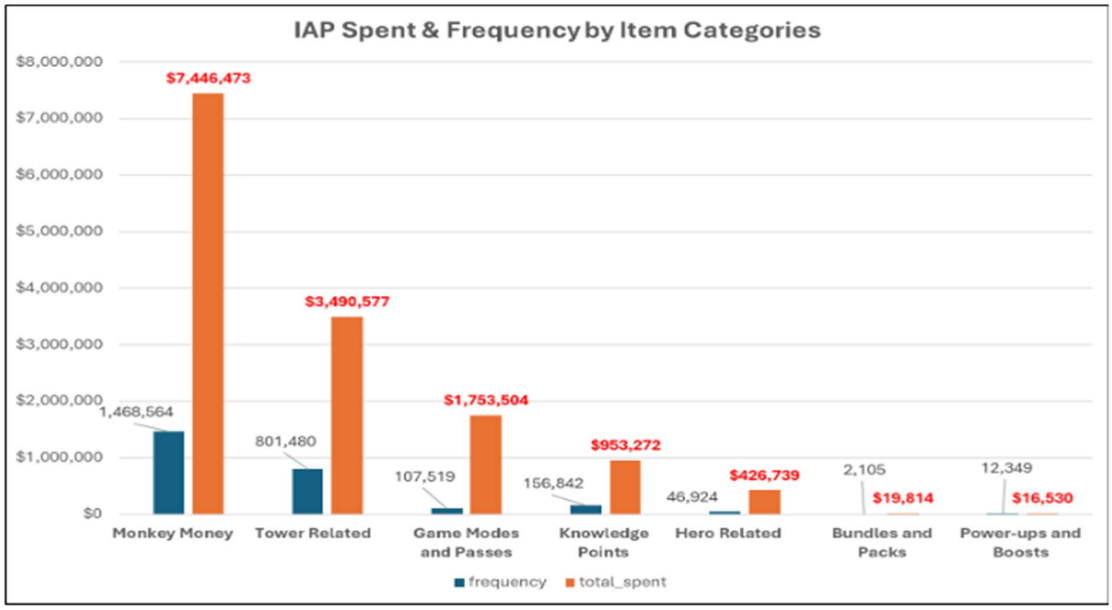


Figure 2, IAP Spent & Frequency by Item Categories

Figures 1 & 2 illustrate in-app purchase (IAP) behaviour in the game. The first chart shows the total IAP expenditure by the top 10 users, with spending ranging from $2,257 to $4,777 on items like `vaultomonkeymoney` and `mountainomonkeymoney`, and the type of item they spent the most on is labelled. The second chart provides a breakdown of IAP spending and purchase frequency across different item categories. `Monkey Money` leads in total spending at $7,446,473, while `Tower Related` items are the most frequently purchased at 801,480 times. These insights highlight the significant investment players make in specific categories, emphasising the popularity and financial impact of these items.

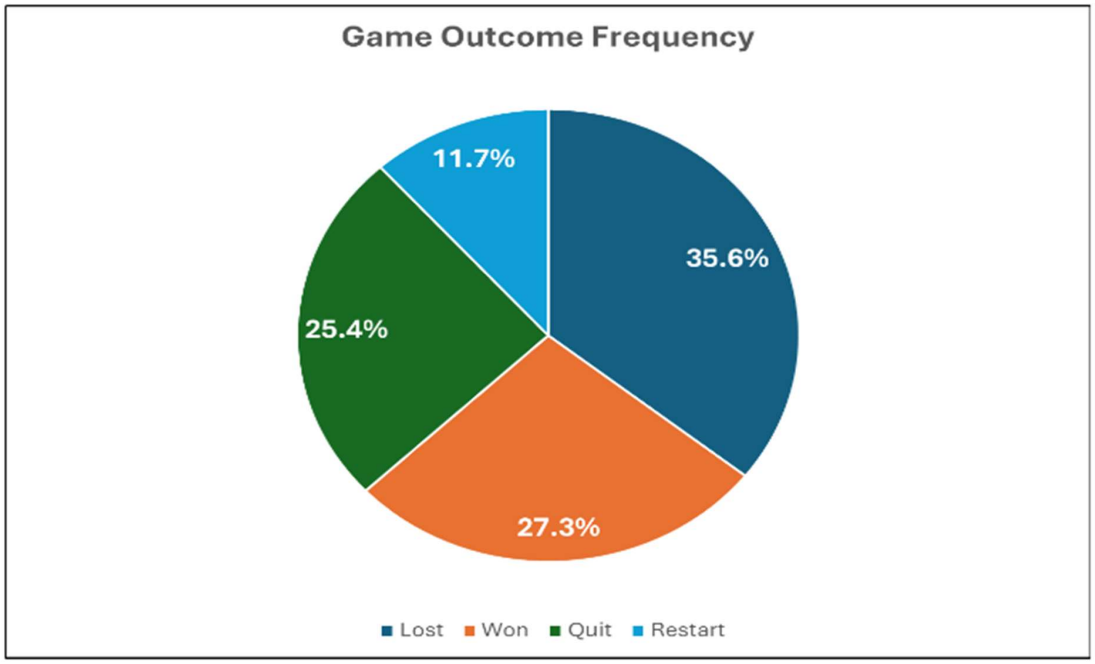


Figure 3, Player Outcomes Frequency

Figure 3 provides insights into game outcomes from the `endtrack` table. It shows the distribution of game outcomes, where 35.6% of games are lost, 27.3% are won, 25.4% are quit, and 11.7% are restarted. This highlights that a significant portion of players experience losses, with a substantial number quitting or restarting the game.

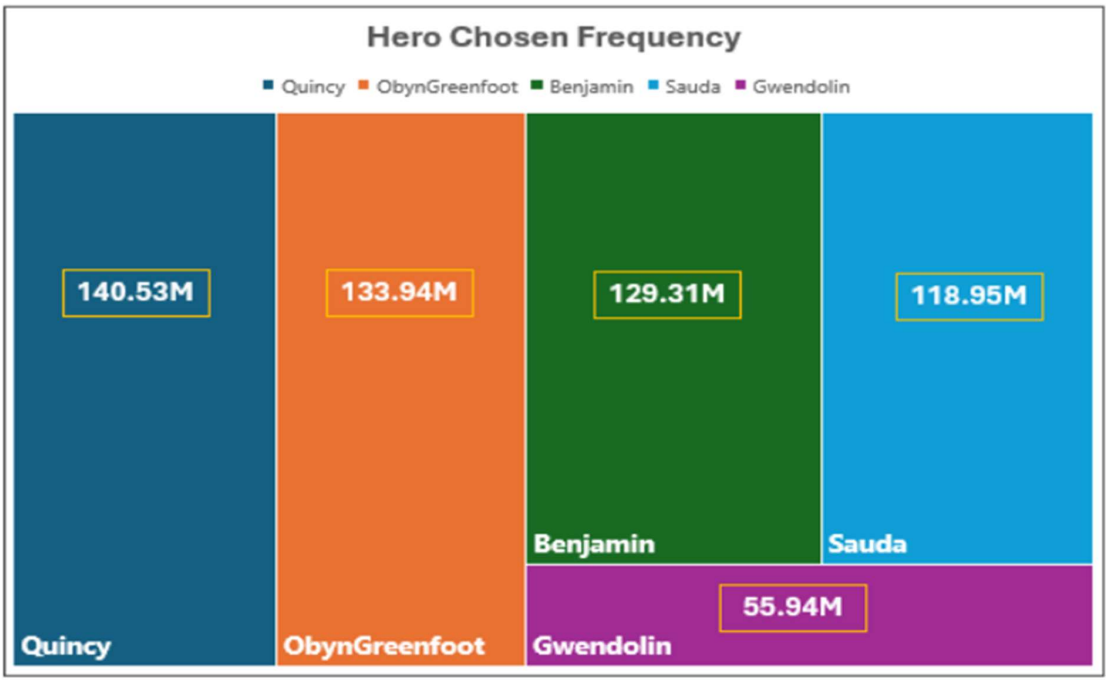


Figure 4, Hero Chosen Frequency

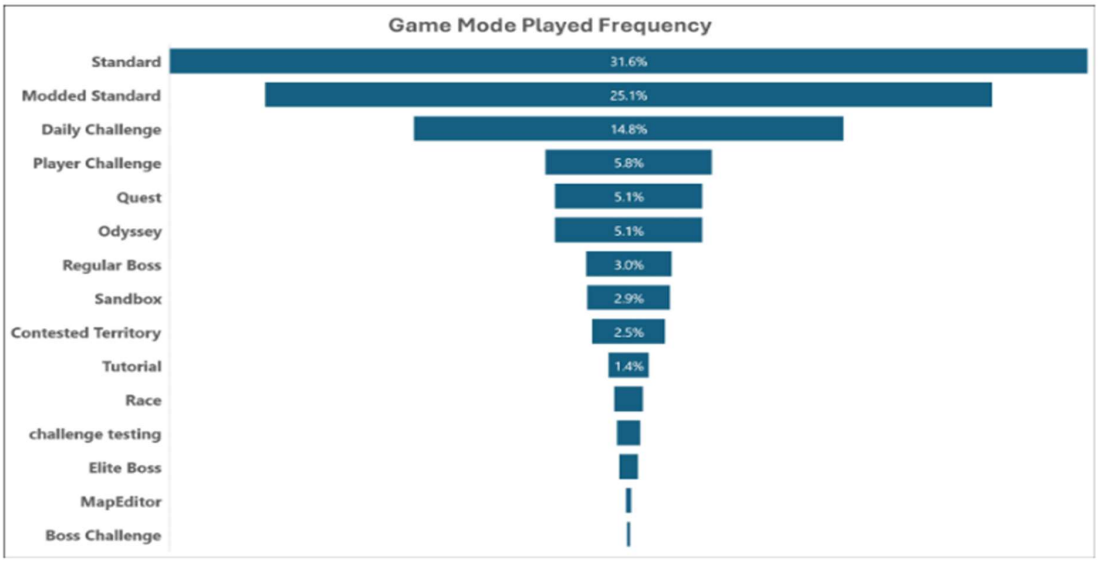


Figure 5, Game Mode Played Frequency

Figures 4 & 5 provide insights into hero selection and game mode preferences from the `starttrack` table. The first chart, "Hero Chosen Frequency," shows the distribution of heroes chosen by players. Quincy and ObynGreenfoot are the most frequently selected heroes, followed by Benjamin, Sauda, and Gwendolin. The second chart, "Game Mode Played Frequency," illustrates the frequency of different game modes played. Standard mode is the most popular at 31.6%, followed by Modded Standard at 25.1%, and Daily Challenge at 14.8%. Other modes, such as Player Challenge, Quest, and Odyssey, have lower but significant engagement rates. These insights highlight player preferences in hero selection and game modes.

While this was useful to gain some initial insights, we joined tables with each other wherever required to get a deeper understanding of player and game characteristics. Upon inspection, few tables that contained critical information were identified and became our focal point of analysis, such as, starttrack and endtrack tables which showed when a user started playing a game using which game mode, difficulty level, map etc. and what was the result of that game.

By joining the starttrack and gainmonkeymoney tables (see Figure 6), we identified which game modes generate the most Monkey Money. Modded Standard leads with 1,907,286 Monkey Money, followed by Standard with 1,124,021 and Daily Challenge with 612,989. Quest and Sandbox also contribute significantly, while less popular modes include challenge testing, Odyssey, and various Boss challenges. This analysis highlights player preferences and engagement levels across different game modes.

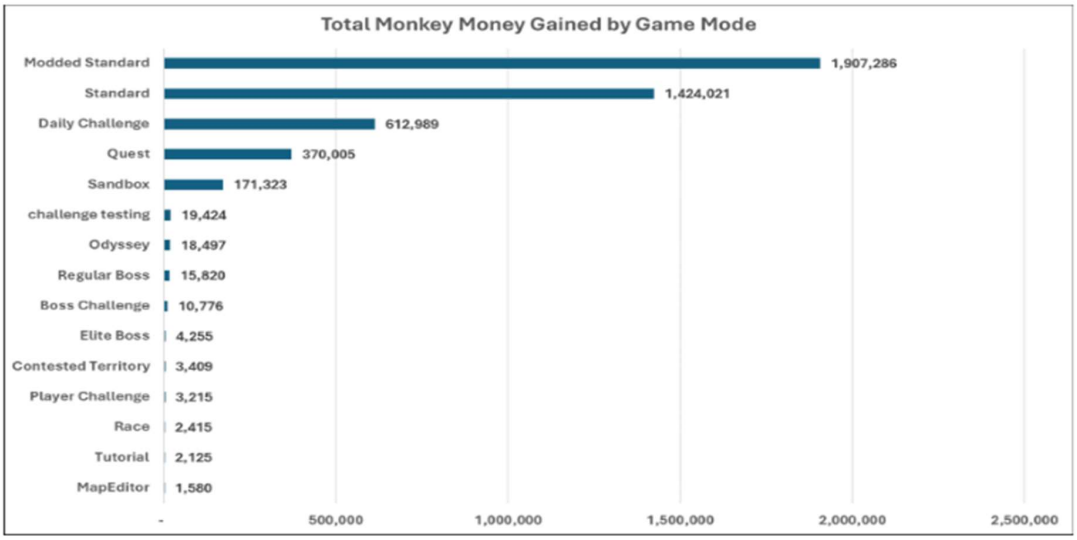


Figure 6, Monkey Money Gained by Game Mode

While investigating the endtrack and madeparagon tables, we identified that DartMonkey is the most frequently used Paragon across all outcomes (see Table 2). Players quit 2,471,203 times with DartMonkey, lost 424,559 times, won 352,733 times, and restarted 1,365 times. This indicates DartMonkey's popularity and significant role in gameplay strategies, regardless of the outcome.

win_loss_quit	mode_unit_name	total_paragons
quit	DartMonkey	2,471,203
lost	DartMonkey	424,559
won	DartMonkey	352,733
restart	DartMonkey	1,365

Table 2, Total Paragons Used by Unit Name and Outcomes

After thorough EDA, we gained a better understanding of the dataset, enabling us to select the relevant variables for the final dataset. This final table would then be used in both SPSS and Python to perform machine learning algorithms to achieve the business objective.

Data Preparation

For the Python part, we randomly selected 100k rows to analyse. For the SPSS Statistics, we used all data in the final table.

Feature Selection Techniques:

Important features that are useful for building predictive models enhance model robustness and make machine learning models more efficient, faster, and more accurate in predicting the future.

Correlation Analysis:

We conducted a correlation coefficient matrix analysis (Figure A9) to enhance the efficiency and robustness of our machine learning models. The primary objective of this analysis is to identify and select features that contribute meaningfully to the model. Features with a correlation coefficient less than 0.2 or greater than 0.8 are typically excluded to ensure they provide significant and unique information. Additionally, correlation analysis helps to identify correlations between features, which is crucial for avoiding multicollinearity. By addressing multicollinearity, we can prevent redundant or highly interdependent variables from distorting the model's performance and interpretability.

Recursive Feature Elimination (RFE):

RFE is a powerful feature selection method that builds a model iteratively and removes the weakest feature one at a time. This process is repeated until the desired number of features is reached. The effectiveness of this method lies in its continuous optimisation of the feature set, allowing the model to use only those features that significantly impact the prediction results, greatly enhancing the model's accuracy and operational efficiency. After several iterations, based on the data from Figure 7, we selected the ten most critical features for our selected Gradient Boosting Classifier model. These features have been empirically validated as the most crucial for enhancing model performance and ensuring the model's efficiency and accuracy in practical applications.

```
Selected features for Gradient Boosting Classifier: Index(['num_sessions', 'monkeymoney_gained_excl_iap', 'session_datediff',
       'num_paid_heroes', 'monkeymoney_spent', 'num_game_modes',
       'num_map_played', 'event_games_prop', 'other_games_prop',
       'current_rank'],
      dtype='object')
```

Figure 7, Selected Important Features

```
# Selecting the best model from the previous step - Gradient Boosting Classifier,
# selecting features, performing hyper parameter tuning and cross validation

#Initializing the Gradient Boosting Classifier
gbm_classifier = GradientBoostingClassifier(random_state=42)

# RFE to select the top 10 features
selector_class = RFE(gbm_classifier, n_features_to_select=10, step=1)
selector_class = selector_class.fit(X_train_churn, y_train_churn)
X_train_class_selected = selector_class.transform(X_train_churn)
X_test_class_selected = selector_class.transform(X_test_churn)

#Displaying the chosen variables
selected_features_class = X_churn.columns[selector_class.support_]
print("Selected features for Gradient Boosting Classifier:", selected_features_class)
```

Figure 8, Feature Importance Using RFE

Data Splitting Techniques:

To ensure the robustness of our models and verify their effectiveness, we utilised the `train_test_split` method from the `sklearn.model_selection` package to divide our sample data into training and testing sets. This method involves random sampling from our dataset to create these splits, which helps to maintain a similar distribution of data in both the training and testing sets. This step is crucial for building predictive models as it mitigates the risk of model bias and ensures that the model can generalise well to new, unseen data.

In our implementation, we chose a split of 75:25 for the training and test set. It provides a substantial amount of data for training purposes while reserving enough instances for an effective evaluation of the model's performance on unseen data. Choosing the correct balance helps prevent the model from overfitting.

This is the code snippet about splitting data into train data and test data:

```
#Model building - defining dependent and independent variables
X_churn = df.drop(['played_within_30days'], axis=1)
y_churn = df['played_within_30days']

#Splitting the data into training and testing sets
X_train_churn, X_test_churn, y_train_churn, y_test_churn = train_test_split(X_churn, y_churn, test_size=0.25, random_state=42)
```

Figure 9, Train and Test Data Split

Modelling

Model Selection

SPSS:

To ensure our analysis was meaningful and catered to distinct player experiences, we categorised players into three groups based on their current rank: beginner, intermediate, and experienced players. This categorization was crucial for understanding different levels of engagement and behaviour patterns across various stages of player experience. The criteria for these categories were as follows:

- **Beginner Players:** Current rank of 30 to 60
- **Intermediate Players:** Current rank of 61 to 100
- **Experienced Players:** Current rank of 101 to 155

For player segmentation, we employed clustering techniques:

- **2-Step Clustering**: We used the silhouette score to evaluate the clustering performance and ensure distinctiveness among the clusters.
- **K-Means Clustering**: Applied to categorise players based on their behavioural features (output discussed below).

We performed cluster modelling separately for each player category (beginner, intermediate, experienced) to account for the differences in their behaviour patterns.

After several iterations, we identified seven key variables that yielded the most meaningful clusters:

- **Boss_event_prop -** Proportion of Boss Event Games out of all games played
- **Event_game_prop -** Proportion of Event Games out of all games played
- **Normal_games_prop -** Proportion of Standard Games out of all games played
- **Continue_moneyspent_prop -** Proportion of spending on Continue out of all monkey money spent
- **Hero_moneyspent_prop -** Proportion of spending on Hero out of all monkey money spent
- **Knowledge_moneyspent_prop -** Proportion of spending on Knowledge out of all monkey money spent
- **Power_moneyspent_prop -** Proportion of spending on Power ability out of all monkey money spent

General Findings

Across all player levels, the clustering analysis (Figure A1 and A2) provided several key insights:

- High Pay Rate and Spending: Clusters 1 and 2 consistently showed high pay rates, higher spending amounts, and lower churn rates. These players tend to engage more with non-standard game modes, particularly Cluster 1, which plays event game modes extensively and spends on Heroes, Knowledge, and Continues.

- Behavioural Patterns: Player behaviours evolve over time. Initially, players engage with standard games to gain experience and upgrade their heroes and knowledge. As they progress, they explore more game modes, and their spending preferences shift accordingly.

Beginner (Lvl 30-60; Figures A3 & A4)

Cluster 1 (53.3%)

- Players in this cluster predominantly play normal games (0.83) and spend a significant proportion of their monkey money on continues (0.28). This suggests they may struggle with game difficulty and frequently use continues to progress. Their lack of engagement with event games (0.03) and boss events (0.01) indicates they are still familiarising themselves with the basic mechanics and are not yet confident enough to tackle more complex game modes.

Cluster 2 (19.1%)

- These players spend a large proportion of their monkey money on heroes (0.48) and knowledge (0.38), indicating a focus on character development and strategic improvement. Although they still primarily play normal games (0.81), they engage more with event games (0.05) compared to Cluster 1.

Cluster 3 (17%)

- Players in this cluster spend significantly on Power ability (0.92). They primarily play normal games (0.8) and have low engagement with event games (0.04). This suggests they focus on using power ability to achieve their short-term goal of winning the game.

Cluster 4 (10.5%)

- These players engage more with event games (0.38) and boss event games (0.1) at the beginner stage.

Intermediate (Lvl 61-100: Figures A5 & A6)

Cluster 1 (23.7%)

- This cluster plays a higher proportion of boss events (0.07) and event games (0.42), and spends significantly on heroes (0.36) and power (0.14). The lower win rate (26%) despite high spending might suggest they prioritise participation and resource acquisition over competitive success.

Cluster 2 (10.9%)

- Players in this cluster exhibit lower spending proportions but with a notable focus on normal games (0.82), with lower IAP spend (1.83) and moderate win rate (43%).

Cluster 3 (37.1%)

- These players spend significantly on heroes (0.66) and knowledge (0.14), indicating a focus on strategy and character development. They have a balanced approach to game mode participation.

Cluster 4 (28.3%)

- These players also spend heavily on continue (0.3) and participate significantly in normal games (0.81) with moderate win rate (40%).

Experienced (Lvl 101-155: Figures A7 & A8)

Cluster 1 (16.5%)

- This cluster includes the biggest IAP spenders, with high proportions of spending power (0.37) relatively. These players have balanced game mode preferences. While relatively high portion spending on power (0.37) could be the result of focus on gaining advantages through spending and participating in diverse game modes.

Cluster 2 (31.8%)

- These players spend less, focusing on heroes (0.51). Their preference for balanced game modes suggests they enjoy both strategic and character development aspects of the game.

Cluster 3 (20.5%)

- These players spend significantly on knowledge (0.23) and heroes (0.51), indicating a strategic and developmental focus.

Cluster 4 (31.2%)

- Players in this cluster spend moderately across all categories. Their high engagement and highest game frequency suggest a deep interest in the game. Their lower win rate (28%) and high participation rate indicates a focus on participation, enjoying the game's content without necessarily achieving the highest competitive success and paying.

Summary of clustering results across three different levels of players can be seen in Table 3.

Player Level	Beginner (Lvl 30-60)				Intermediate (Lvl 61-100)				Experienced (Lvl 101-155)			
Silhouette Score	0.4 (Fair)				0.4 (Fair)				0.3 (Fair)			
Cluster	1	2	3	4	1	2	3	4	1	2	3	4
Cluster Size	53.3%	19.1%	17.0%	10.5%	23.7%	10.9%	37.1%	28.3%	16.5%	31.8%	20.5%	31.2%
Input												
boss_event_prop	0.01	0.01	0.01	0.1	0.07	0.02	0.02	0.02	0.13	0.03	0.02	0.04
continue_moneyspent_prop	0.28	0.05	0.04	0.17	0.16	0.09	0.08	0.3	0.15	0.1	0.36	0.14
event_games_prop	0.03	0.05	0.04	0.38	0.42	0.07	0.08	0.05	0.25	0.11	0.18	0.55
hero_moneyspent_prop	0	0.48	0	0.15	0.36	0.1	0.66	0.06	0.22	0.51	0.11	0.44
knowledge_moneyspent_prop	0.02	0.38	0.02	0.07	0.16	0.72	0.14	0.08	0.13	0.22	0.23	0.19
normal_games_prop	0.83	0.81	0.8	0.36	0.38	0.82	0.79	0.81	0.54	0.8	0.71	0.35
power_moneyspent_prop	0.06	0.06	0.92	0.17	0.14	0.04	0.05	0.28	0.37	0.04	0.06	0.05
Evaluation Fields												
iap_flag	3.5%	13.9%	5.0%	6.4%	18.7%	8.8%	19.4%	16.1%	38.8%	24.8%	23.0%	24.0%
iap_spend	0.44	2.09	0.91	1.07	4.8	1.83	3.82	4.42	24.26	6.4	10.14	6.64
played_within_30days	14.6%	11.0%	11.4%	12.5%	30.6%	24.7%	28.7%	31.8%	50.2%	50.3%	45.4%	53.5%
avg_games_per_week	9.41	14.58	9.66	15.8	33.81	22.12	22.84	16.48	36.94	38.3	35.94	61.68
current_rank	41.26	49.99	41.44	45.23	77.65	73.05	75.3	72.44	120.48	114.49	120.57	118.33
win_rate	44%	48%	45%	32%	26%	43%	41%	40%	33%	41%	40%	28%

Table 3: Summary of Clustering Results by Players' Level

Python:

In addition to our SPSS clustering, the players' behavioural variables were also used as input to come up with meaningful clusters in Python. For more detailed clustering and specific segmentation, players were classified by their level using 'current_rank' as the criterion (same as the SPSS clustering). All the different cluster models follow the same template.

For churn prediction and IAP prediction which are binary classification problems, three appropriate and comparable algorithms were fit on the data to choose the best performing one. The best-performing model in each case was chosen to perform feature selection, hyper-parameter tuning, and cross-validation to further validate results.

Model Justification:

K-means clustering (Kevin Arvai, 2019) was chosen for clustering in Python due to its simplicity, efficiency, and scalability in handling large datasets. The algorithm is well-suited for our data as it aims to partition the players into k clusters based on their psychographic characteristics, facilitating the identification of distinct player segments. K-means clustering is particularly effective when the number of clusters (k) is known or can be estimated, allowing for targeted analysis and recommendations.

For both binary classification problems, three algorithms were considered: Logistic Regression, Random Forest Classifier, and Gradient Boosting Classifier (Demir. N, 2015). Logistic Regression was selected for its simplicity and interpretability, providing a baseline for comparison. Random Forest was chosen for its ability to handle high-dimensional data and capture nonlinear relationships.

Gradient Boosting was selected due to its superior performance in both binary classification problems, leveraging boosting techniques to improve model accuracy.

Model Hyper-parameter Tuning:

Although we initially hypothesised that 4 clusters would be optimal based on prior insights, we still performed the elbow method to validate this assumption. By plotting the within-cluster sum of squares (WCSS) against different values of k, the point where the WCSS starts to diminish returns was identified as the optimal k. This approach ensured that our preconception was supported by data-driven evidence, providing a robust justification for choosing 4 clusters. Although for some cluster models, the elbow method revealed that other values might be optimal, k was chosen to be 4, keeping domain knowledge in mind.

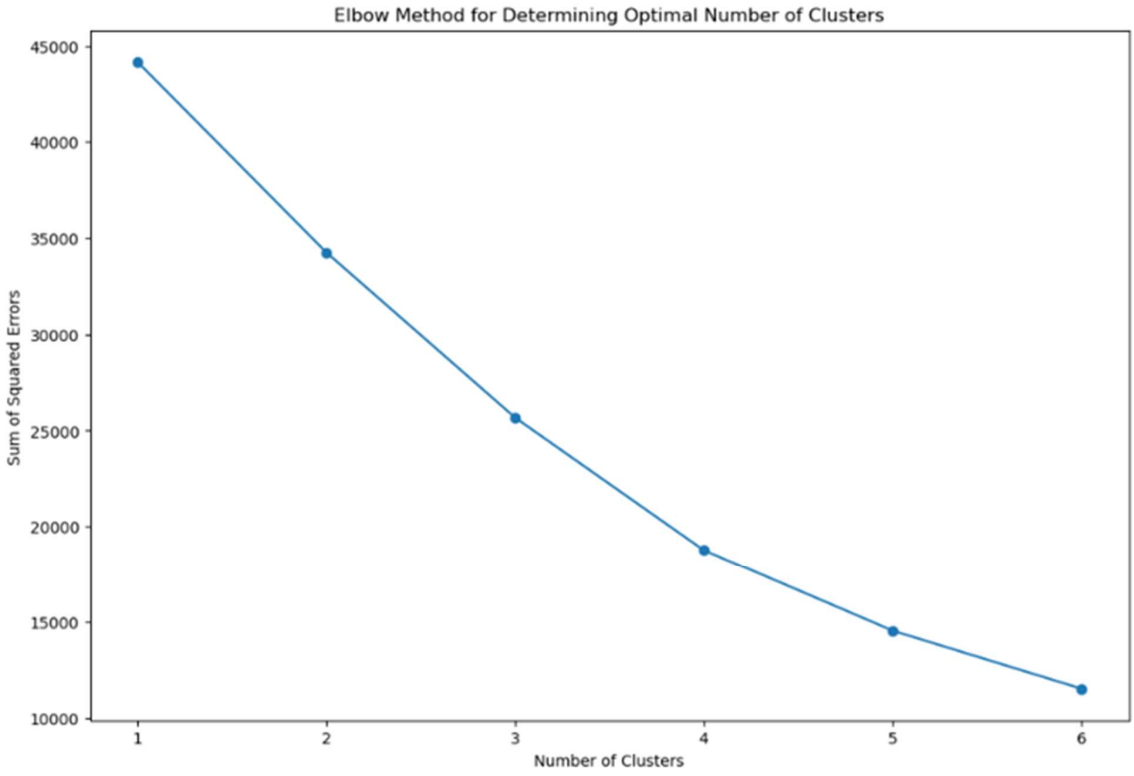


Figure 10: Elbow method for the clustering of all players indicating that 4 is the optimal k.

For the Gradient Boosting Classifiers aimed at predicting player churn and IAP purchase, a meticulous hyperparameter tuning process was conducted to optimise the model's performance. The parameters fine-tuned included:

- **n_estimators:** The number of boosting stages to be run, tested at 50 and 100.
- **max_depth**: The maximum depth of the individual trees, tested at 3 and 6.
- **learning_rate**: The step size shrinkage to be applied at each boosting step, tested at 0.01 and 0.2.

This tuning was performed using GridSearchCV, which exhaustively searches over the specified parameter grid to find the best combination of hyperparameters. The scoring metric used was roc_auc, which evaluates the model based on the Area Under the Receiver Operating Characteristic Curve, providing a balance between precision and recall.

The cross-validation process (Abhishek, 2023) involved splitting the training data into 2 folds (cv=2) to validate the model's performance on different subsets of the data, ensuring robustness and generalizability. The choice of 2 folds balances the need for computational efficiency with the goal of robust model evaluation.

```
Fitting 2 folds for each of 8 candidates, totalling 16 fits
[CV] END ...learning_rate=0.01, max_depth=3, n_estimators=50; total time=   6.1s
[CV] END ...learning_rate=0.01, max_depth=3, n_estimators=50; total time=   6.1s
[CV] END ..learning_rate=0.01, max_depth=3, n_estimators=100; total time=  12.3s
[CV] END ..learning_rate=0.01, max_depth=3, n_estimators=100; total time=  13.0s
[CV] END ...learning_rate=0.01, max_depth=6, n_estimators=50; total time=  15.2s
[CV] END ...learning_rate=0.01, max_depth=6, n_estimators=50; total time=  14.6s
[CV] END ..learning_rate=0.01, max_depth=6, n_estimators=100; total time=  27.5s
[CV] END ..learning_rate=0.01, max_depth=6, n_estimators=100; total time=  28.3s
[CV] END ....learning_rate=0.2, max_depth=3, n_estimators=50; total time=   6.2s
[CV] END ....learning_rate=0.2, max_depth=3, n_estimators=50; total time=   6.2s
[CV] END ...learning_rate=0.2, max_depth=3, n_estimators=100; total time=  12.8s
[CV] END ...learning_rate=0.2, max_depth=3, n_estimators=100; total time=  12.5s
[CV] END ....learning_rate=0.2, max_depth=6, n_estimators=50; total time=  13.0s
[CV] END ....learning_rate=0.2, max_depth=6, n_estimators=50; total time=  13.4s
[CV] END ...learning_rate=0.2, max_depth=6, n_estimators=100; total time=  25.7s
[CV] END ...learning_rate=0.2, max_depth=6, n_estimators=100; total time=  25.6s
Best hyper-parameters: {'learning_rate': 0.2, 'max_depth': 3, 'n_estimators': 100}
```

Figure 11: Verbose output from the GridSearchCV process and the optimum hyper-parameters yielding the best ROC-AUC – Question 2

```
Fitting 2 folds for each of 8 candidates, totalling 16 fits
[CV] END ...learning_rate=0.01, max_depth=3, n_estimators=50; total time=   6.3s
[CV] END ...learning_rate=0.01, max_depth=3, n_estimators=50; total time=   6.5s
[CV] END ..learning_rate=0.01, max_depth=3, n_estimators=100; total time=  12.7s
[CV] END ..learning_rate=0.01, max_depth=3, n_estimators=100; total time=  12.6s
[CV] END ...learning_rate=0.01, max_depth=6, n_estimators=50; total time=  12.9s
[CV] END ...learning_rate=0.01, max_depth=6, n_estimators=50; total time=  13.0s
[CV] END ..learning_rate=0.01, max_depth=6, n_estimators=100; total time=  27.0s
[CV] END ..learning_rate=0.01, max_depth=6, n_estimators=100; total time=  26.2s
[CV] END ....learning_rate=0.2, max_depth=3, n_estimators=50; total time=   6.1s
[CV] END ....learning_rate=0.2, max_depth=3, n_estimators=50; total time=   6.1s
[CV] END ...learning_rate=0.2, max_depth=3, n_estimators=100; total time=  12.4s
[CV] END ...learning_rate=0.2, max_depth=3, n_estimators=100; total time=  13.8s
[CV] END ....learning_rate=0.2, max_depth=6, n_estimators=50; total time=  13.2s
[CV] END ....learning_rate=0.2, max_depth=6, n_estimators=50; total time=  14.5s
[CV] END ...learning_rate=0.2, max_depth=6, n_estimators=100; total time=  28.7s
[CV] END ...learning_rate=0.2, max_depth=6, n_estimators=100; total time=  24.9s
Best hyper-parameters: {'learning_rate': 0.2, 'max_depth': 3, 'n_estimators': 100}
```

Figure 12: Verbose output from the GridSearchCV process and the optimum hyper-parameters yielding the best ROC-AUC – Question 3

Cross-validation is a critical step in our model-tuning process. It involves partitioning the training data into k subsets (folds), where the model is trained on k-1 folds and validated on the remaining fold. This process is repeated k times, with each fold used exactly once as the validation data. The advantages of this approach include:

- **Reducing Overfitting:** By validating the model on different subsets of data, we ensure that the model does not overfit to a specific portion of the data.
- **Robust Performance Metrics:** Cross-validation provides a more reliable estimate of model performance by averaging the results from multiple folds, reducing the variance in the performance metrics.
- **Model Stability:** Ensures that the model's performance is consistent and not dependent on a particular train-test split.

Model Training and Evaluation Metrics:

The K-means algorithm iteratively assigns players to the nearest cluster centroid, recalculates the centroids, and reassigns players until convergence. This process was repeated multiple times with different initializations to ensure robustness and avoid local minima. The run time for the full clustering process in Python was 15 minutes. The Silhouette Score was employed to evaluate clustering quality. The Silhouette Score assesses the cohesion and separation of clusters by comparing intra-cluster and inter-cluster distances.

Both binary classification tasks follow a similar model structure. The dependent and independent variables were defined. Relevant variables were selected to ensure that no data leakage happens. The data was split into training and testing sets with a 75-25 split. Each of the three models (with

default hyper-parameters) was fit on the training data. Following this, the models were evaluated on the test data using the following evaluation metrics:

- Accuracy: Measures the overall correctness of the model.
- Precision: Indicates the proportion of positive identifications that were correct.
- Recall: Reflects the proportion of actual positives that were identified correctly.
- F1 Score: A harmonic mean of precision and recall, balancing the two metrics.
- ROC-AUC Score: Evaluates the model's ability to distinguish between classes.

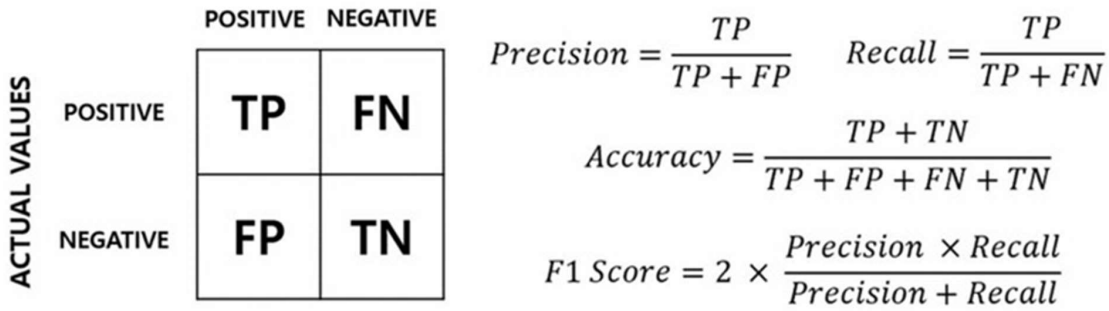


Figure 13: Confusion Matrix and performance metrics in the classification models

The best-performing models based on the ROC-AUC score were chosen to further improve upon and validate results. The ROC-AUC score is essential for both player churn prediction and in-app purchase prediction as it effectively balances true positive and false positive rates, ensuring robust performance across all decision thresholds. This is crucial for making balanced recommendations to the company by accurately identifying both churners and non-churners, as well as purchasers and non-purchasers. Additionally, the ROC-AUC score handles class imbalance well, providing a more reliable measure of model performance compared to metrics like accuracy, which can be misleading in imbalanced datasets.

For both classification tasks, the chosen model, Gradient Boosting Classifiers, were trained on the training dataset using the best hyperparameters found through GridSearchCV. The models iteratively build decision trees, where each new tree attempts to correct the errors of the previous ones. The same evaluation metrics used to pick the best-performing model are also used to assess the performance here. This particular step was very time-consuming and was streamlined to achieve the best possible outcome, keeping in mind computational constraints. The fine-tuned Gradient Boosting Classification models took 35 minutes for churn prediction and 25 minutes for IAP prediction.

Evaluation

Model Comparison

Clustering:

The comparison of Silhouette Scores between Python's K-means clustering and SPSS's Two-step clustering reveals notable differences based on player levels. For the overall data, both methods achieve a Silhouette Score of 0.4, indicating moderate cluster cohesion and separation. In the beginner level (30-60), Python's K-means clustering significantly outperforms SPSS with a Silhouette Score of 0.5 compared to 0.4, suggesting better-defined clusters. For intermediate players (61-100), SPSS achieves a higher Silhouette Score of 0.4 versus Python's 0.3, indicating more cohesive clusters in SPSS. However, for experienced players (101-155), both methods yield similar scores of 0.3, reflecting lower cluster quality due to increased variability in player behaviour. These differences can be attributed to the inherent characteristics of each algorithm. K-means, with its focus on minimising variance within clusters, excels in scenarios with well-defined, homogeneous groups, while Two-step clustering effectively handles mixed data types and large datasets but may struggle with noise and outliers. This comparison underscores the importance of choosing the appropriate clustering technique based on the specific characteristics and distribution of the data.

Level	Python : K-means Silhouette Score	SPSS: Two-step Clustering Silhouette Score
All	0.4	0.4
30 - 60	0.5	0.4
61 - 100	0.3	0.4
101 - 155	0.3	0.3

Table 4: A table showing the Silhouette Scores of the different cluster models.

Churn Prediction:

Logistic Regression, known for its simplicity and interpretability, had lower precision and recall , resulting in a low F1 score and a moderate ROC-AUC score . This is due to its linear nature, which struggles with capturing complex patterns in the data. Random Forest performed better, leveraging multiple decision trees to improve accuracy and precision . However, its recall and F1 score were only slightly improved, indicating it could still miss some patterns due to overfitting on the training data, despite being more robust against overfitting than Logistic Regression. Gradient Boosting emerged as the best, showing superior performance across all metrics. This model effectively handles complex data patterns through iterative improvements, which helps in capturing subtle patterns that other models might miss.

Despite a decrease in the ROC-AUC score to 0.640518 after feature selection, hyper-parameter tuning, and cross-validation, Gradient Boosting was chosen due to its overall balance and robustness. The slight performance drop is attributed to the exclusion of some important variables during feature selection and the specific hyper-parameter values used. The detailed tuning and validation process, including grid search and cross-validation, ensured the model’s robustness by

minimising overfitting and enhancing generalisation. This comprehensive approach confirmed Gradient Boosting as the best choice for churn prediction in this context, balancing complexity and interpretability with robust performance.

	Model	Accuracy	Precision	Recall	F1 Score	ROC-AUC Score
0	Logistic Regression	0.82692	0.618009	0.270259	0.376063	0.788716
1	Random Forest	0.83772	0.694135	0.284560	0.403645	0.806657
2	Gradient Boosting	0.83812	0.693340	0.289119	0.408074	0.815608

Table 5: Evaluation metrics of the three models

```
Gradient Boosting Classifier Evaluation Metrics:
Accuracy: 0.83948
Precision: 0.6810883140053524
Recall: 0.31647668393782385
F1 Score: 0.43214942691382485
ROC-AUC Score: 0.6405183915352068
```

Figure 14: Evaluation metrics for the final chosen Gradient Boosting Classifier model – Question 2

IAP Prediction:

Despite its simplicity and efficiency, Logistic Regression's low recall and F1 score indicate it struggles with the imbalanced nature of the dataset, failing to identify a significant number of positive instances (players making an IAP). The Random Forest classifier showed an improved performance, The higher precision and recall demonstrate Random Forest's robustness in handling class imbalance by effectively capturing the minority class (IAP instances). The initial Gradient Boosting model, without hyperparameter tuning indicates a strong balance between precision and recall.

After feature selection, hyperparameter tuning, and cross-validation, the chosen Gradient Boosting model's performance showed some decline in ROC-AUC to 0.587963 but improved recall and F1 score, suggesting the model became better at identifying positive instances at the cost of some overall discriminatory power. This decline can be attributed to the complexity of the hyperparameter space and the stringent feature selection process, which might have omitted some influential variables, highlighting the trade-off between model complexity and interpretability.

	Model	Accuracy	Precision	Recall	F1 Score	ROC-AUC Score
0	Logistic Regression	0.90000	0.495614	0.045236	0.082905	0.787974
1	Random Forest	0.91040	0.728723	0.164532	0.268452	0.834703
2	Gradient Boosting	0.90928	0.695578	0.163731	0.265068	0.838769

Table 6: Evaluation metrics of the three models

```
Gradient Boosting Classifier Evaluation Metrics:
Accuracy: 0.90948
Precision: 0.6690647482014388
Recall: 0.18614891913530823
F1 Score: 0.2912621359223301
ROC-AUC Score: 0.5879638027371503
```

Figure 15: Evaluation metrics for the final chosen Gradient Boosting Classifier model – Question 3

Sensitivity Analysis

Recursive Feature Elimination systematically removed less important features and built models iteratively, ultimately selecting the top 10 features for the Gradient Boosting Classifier. For question 2: churn prediction, the tuned Gradient Boosting model post RFE showed a slight improvement compared to the initial model in recall and F1 score, albeit with a reduced ROC-AUC score, indicating enhanced sensitivity in identifying churners at the expense of overall discriminatory power.

After applying RFE to the initial Gradient Boosting model for the IAP prediction, the accuracy and recall improved, but the ROC-AUC dropped, suggesting better identification of IAP instances. This sensitivity analysis highlights the trade-off between including more features for comprehensive data representation and selecting fewer, more influential features to enhance model sensitivity, demonstrating the nuanced balance between model complexity and performance.

Feature	Importance
num_sessions	0.126366
monkeymoney_gained_excl_iap	0.044757
session_datediff	0.281249
num_paid_heroes	0.077193
monkeymoney_spent	0.133496
num_game_modes	0.021831
num_map_played	0.165649
event_games_prop	0.014884
other_games_prop	0.092189
current_rank	0.042387

Figure 16: Feature importance scores of input variables for question 2

Feature	Importance
ios_flag	0.070018
avg_moneygained_persession	0.025987
knowledge_moneyspent_prop	0.031173
continue_moneyspent_prop	0.057596
hero_moneyspent_prop	0.391144
heroskin_moneyspent_prop	0.127149
power_moneyspent_prop	0.127068
num_map_played	0.046222
coop_mode_prop	0.048921
current_rank	0.074722

Figure 17: Feature importance scores of input variables for question 3

Deployment

To deploy the two-step clustering models developed in SPSS and the Gradient Boosting models for churn and IAP prediction in BTD6, we would integrate these models into the game's backend. This involves exporting SPSS model results and applying clusters to new data through batch processing. For real-time deployment, SPSS logic can be converted to Python code and served via RESTful APIs using Flask or FastAPI on AWS or Google Cloud. Automated model retraining can be handled using Apache Airflow or Jenkins. An analytics dashboard with Tableau or Power BI will monitor model performance and provide actionable insights.

References

Abhishek (2023). Cross Validation in Machine Learning. GeeksforGeeks https://www.geeksforgeeks.org/cross-validation-machine-learning/

Demir. N (2015). Ensemble Methods: Elegant Techniques to Produce Improved Machine Learning Results. Developers Ensemble Methods in Machine Learning | Toptal®

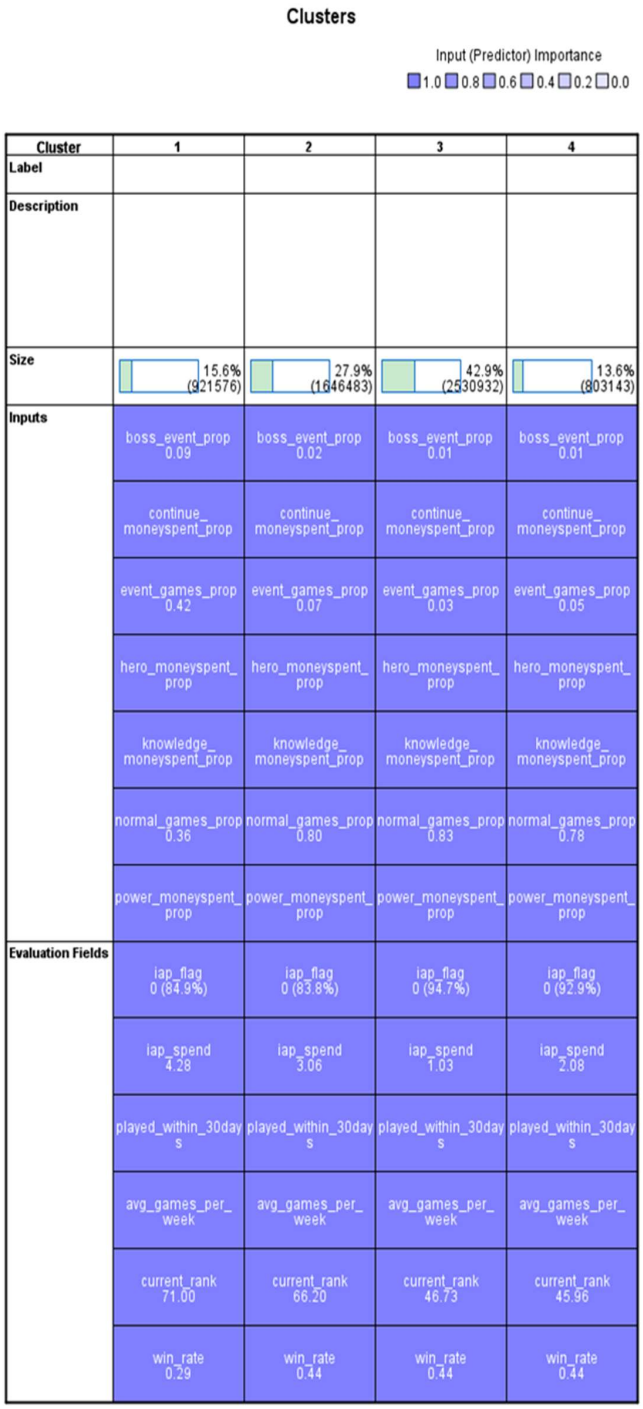
Kevin Arvai (2019) K-Means Clustering in Python: A Practical Guide. Real Python https://realpython.com/k-means-clustering-python/

Appendix

A1	*Cluster Result for All Level Players*
A2	*Cluster Comparison for All Level Players*
A3	*Clustering output for Beginner Players (Level 30-60)*
A4	*Cluster Comparison for Beginner Players (Level 30-60)*
A5	*Clustering output for Intermediate Players (Level 61-100)*
A6	*Cluster Comparison for Intermediate Players (Level 61-100)*
A7	*Clustering output for Experienced Players (Level 101-155)*
A8	*Cluster Comparison for Experienced Players (Level 101-155)*
A9	*Correlation coefficient matrix.*

Link to the Data and Code:

https://drive.google.com/drive/folders/1XI2UIFoTlvjDJ7koRvJTeKeVfXMUHwsg?usp=drive_link



Cluster	1	2	3	4
Label				
Description				
Size	15.6% (921576)	27.9% (1646483)	42.9% (2530932)	13.6% (803143)
Inputs	boss_event_prop 0.09	boss_event_prop 0.02	boss_event_prop 0.01	boss_event_prop 0.01
	continue_ moneyspent_prop	continue_ moneyspent_prop	continue_ moneyspent_prop	continue_ moneyspent_prop
	event_games_prop 0.42	event_games_prop 0.07	event_games_prop 0.03	event_games_prop 0.05
	hero_moneyspent_ prop	hero_moneyspent_ prop	hero_moneyspent_ prop	hero_moneyspent_ prop
	knowledge_ moneyspent_prop	knowledge_ moneyspent_prop	knowledge_ moneyspent_prop	knowledge_ moneyspent_prop
	normal_games_prop 0.36	normal_games_prop 0.80	normal_games_prop 0.83	normal_games_prop 0.78
	power_moneyspent_ prop	power_moneyspent_ prop	power_moneyspent_ prop	power_moneyspent_ prop
Evaluation Fields	iap_flag 0 (84.9%)	iap_flag 0 (83.8%)	iap_flag 0 (94.7%)	iap_flag 0 (92.9%)
	iap_spend 4.28	iap_spend 3.06	iap_spend 1.03	iap_spend 2.08
	played_within_30day s	played_within_30day s	played_within_30day s	played_within_30day s
	avg_games_per_ week	avg_games_per_ week	avg_games_per_ week	avg_games_per_ week
	current_rank 71.00	current_rank 66.20	current_rank 46.73	current_rank 45.96
	win_rate 0.29	win_rate 0.44	win_rate 0.44	win_rate 0.44

Figure A1

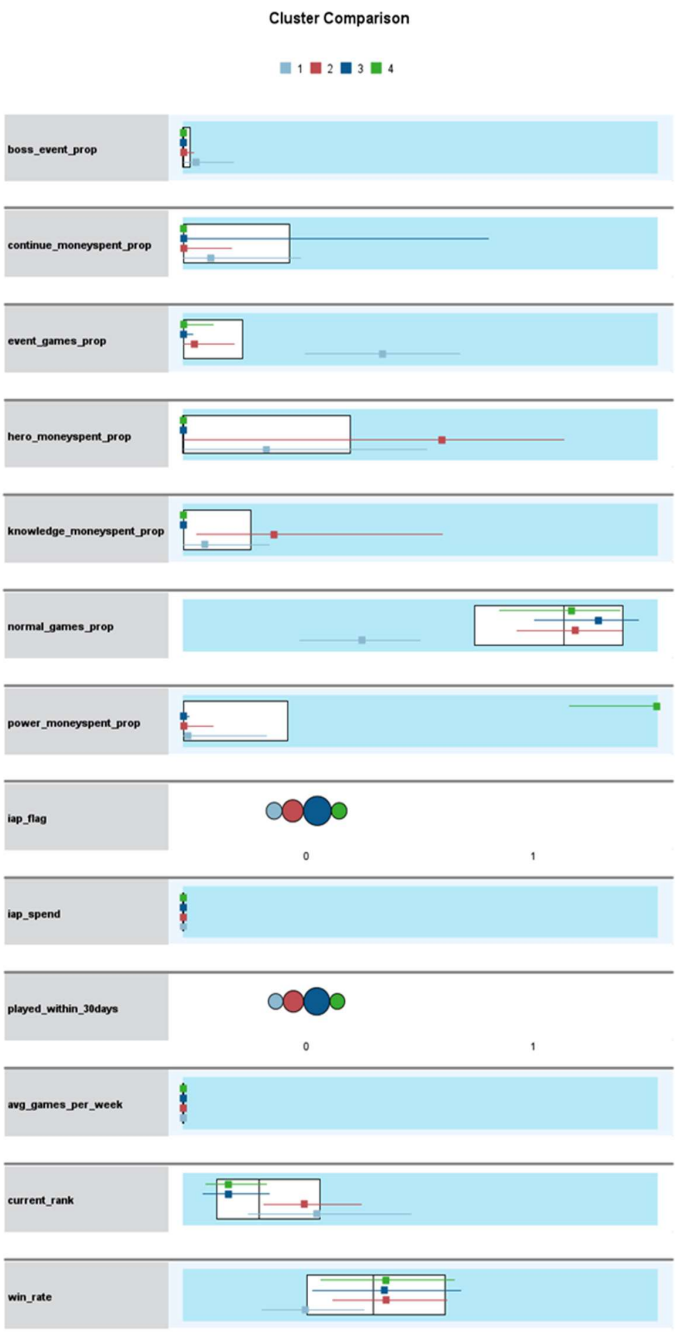


Figure A2

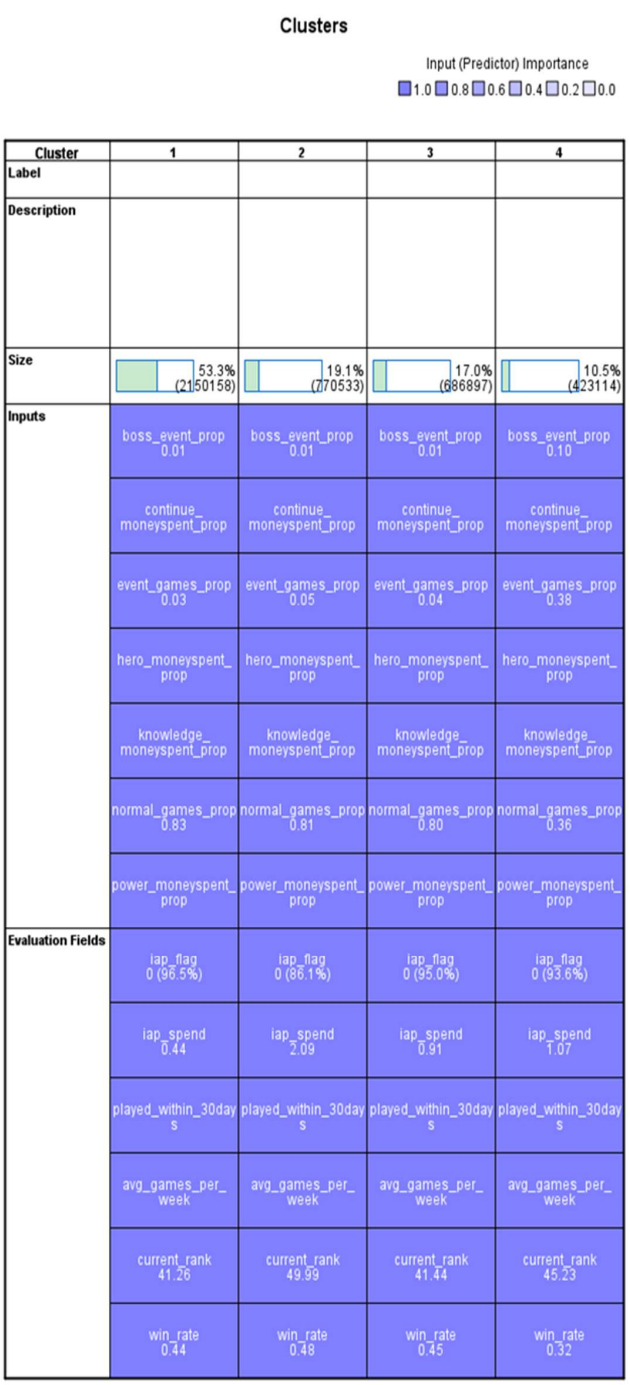


Figure A3

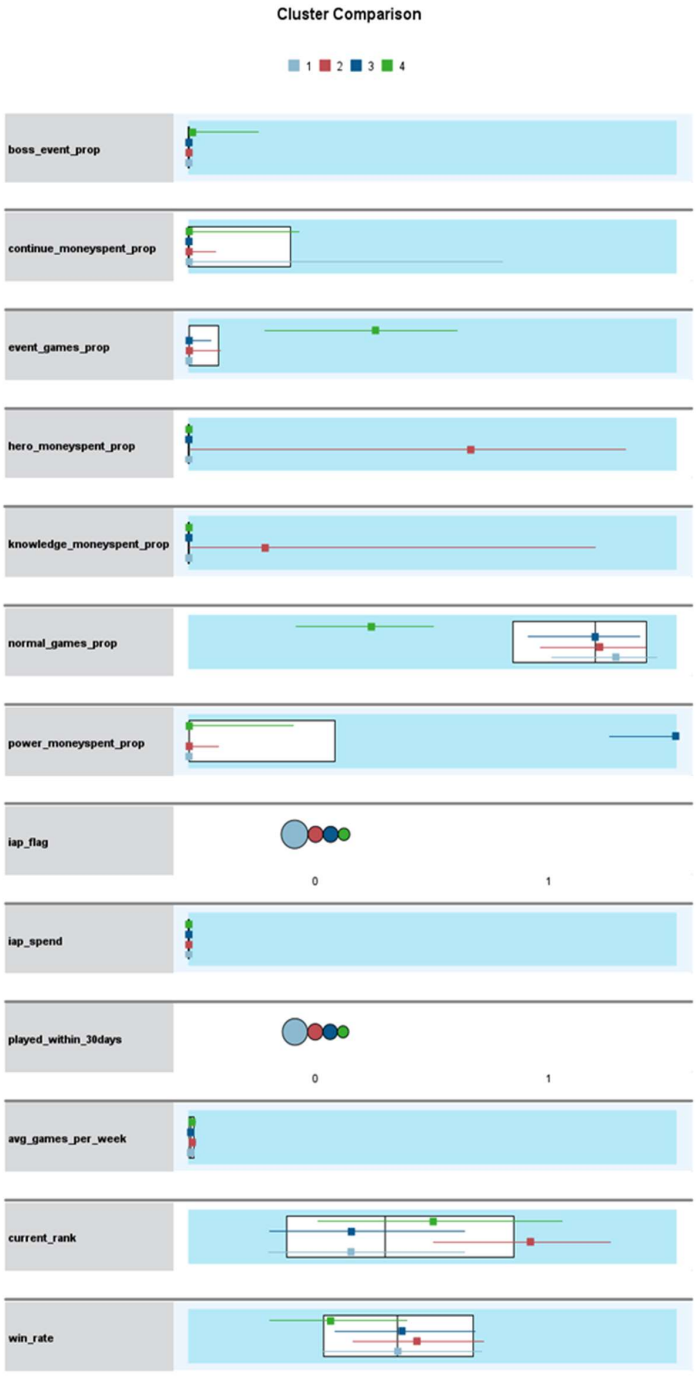
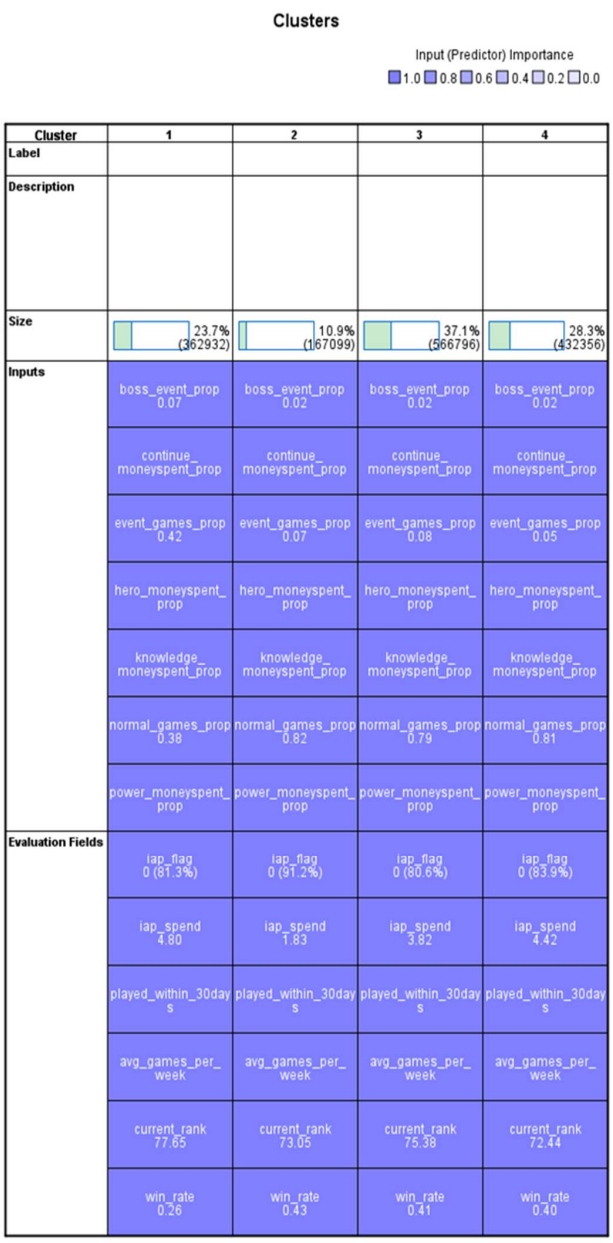


Figure A4



Cluster	1	2	3	4
Label				
Description				
Size	23.7% (362932)	10.9% (167099)	37.1% (566796)	28.3% (432356)
Inputs	boss_event_prop 0.07	boss_event_prop 0.02	boss_event_prop 0.02	boss_event_prop 0.02
	continue_ moneyspent_prop	continue_ moneyspent_prop	continue_ moneyspent_prop	continue_ moneyspent_prop
	event_games_prop 0.42	event_games_prop 0.07	event_games_prop 0.08	event_games_prop 0.05
	hero_moneyspent_ prop	hero_moneyspent_ prop	hero_moneyspent_ prop	hero_moneyspent_ prop
	knowledge_ moneyspent_prop	knowledge_ moneyspent_prop	knowledge_ moneyspent_prop	knowledge_ moneyspent_prop
	normal_games_prop 0.38	normal_games_prop 0.82	normal_games_prop 0.79	normal_games_prop 0.81
	power_moneyspent_ prop	power_moneyspent_ prop	power_moneyspent_ prop	power_moneyspent_ prop
Evaluation Fields	iap_flag 0 (81.3%)	iap_flag 0 (91.2%)	iap_flag 0 (80.6%)	iap_flag 0 (83.9%)
	iap_spend 4.80	iap_spend 1.83	iap_spend 3.82	iap_spend 4.42
	played_within_30day s	played_within_30day s	played_within_30day s	played_within_30day s
	avg_games_per_ week	avg_games_per_ week	avg_games_per_ week	avg_games_per_ week
	current_rank 77.65	current_rank 73.05	current_rank 75.38	current_rank 72.44
	win_rate 0.26	win_rate 0.43	win_rate 0.41	win_rate 0.40

Figure A5

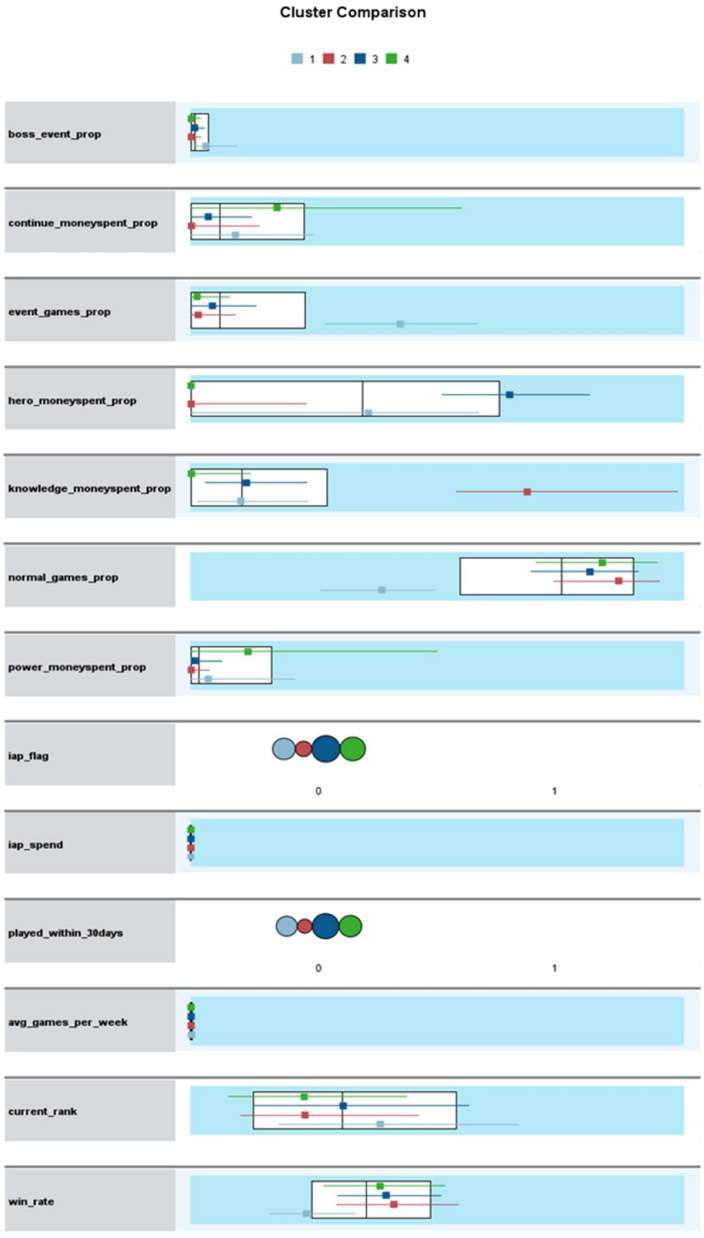


Figure A6

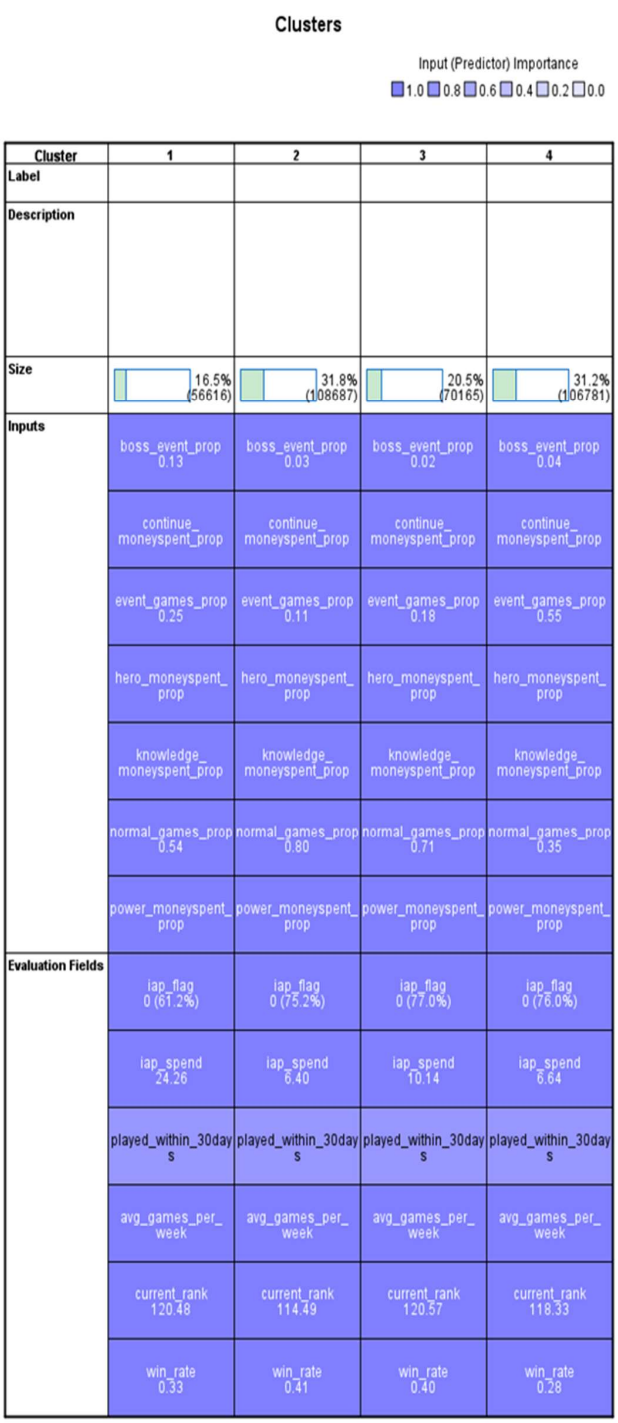


Figure A7

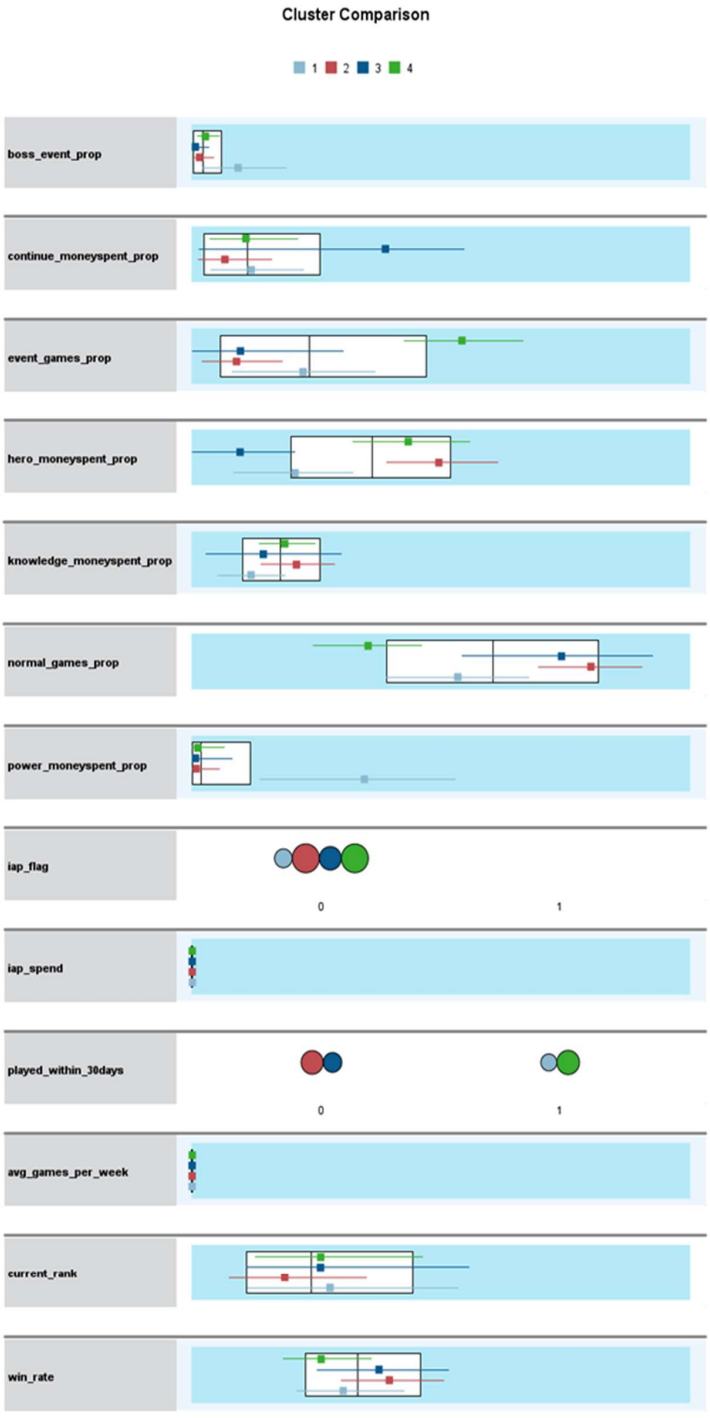


Figure A8

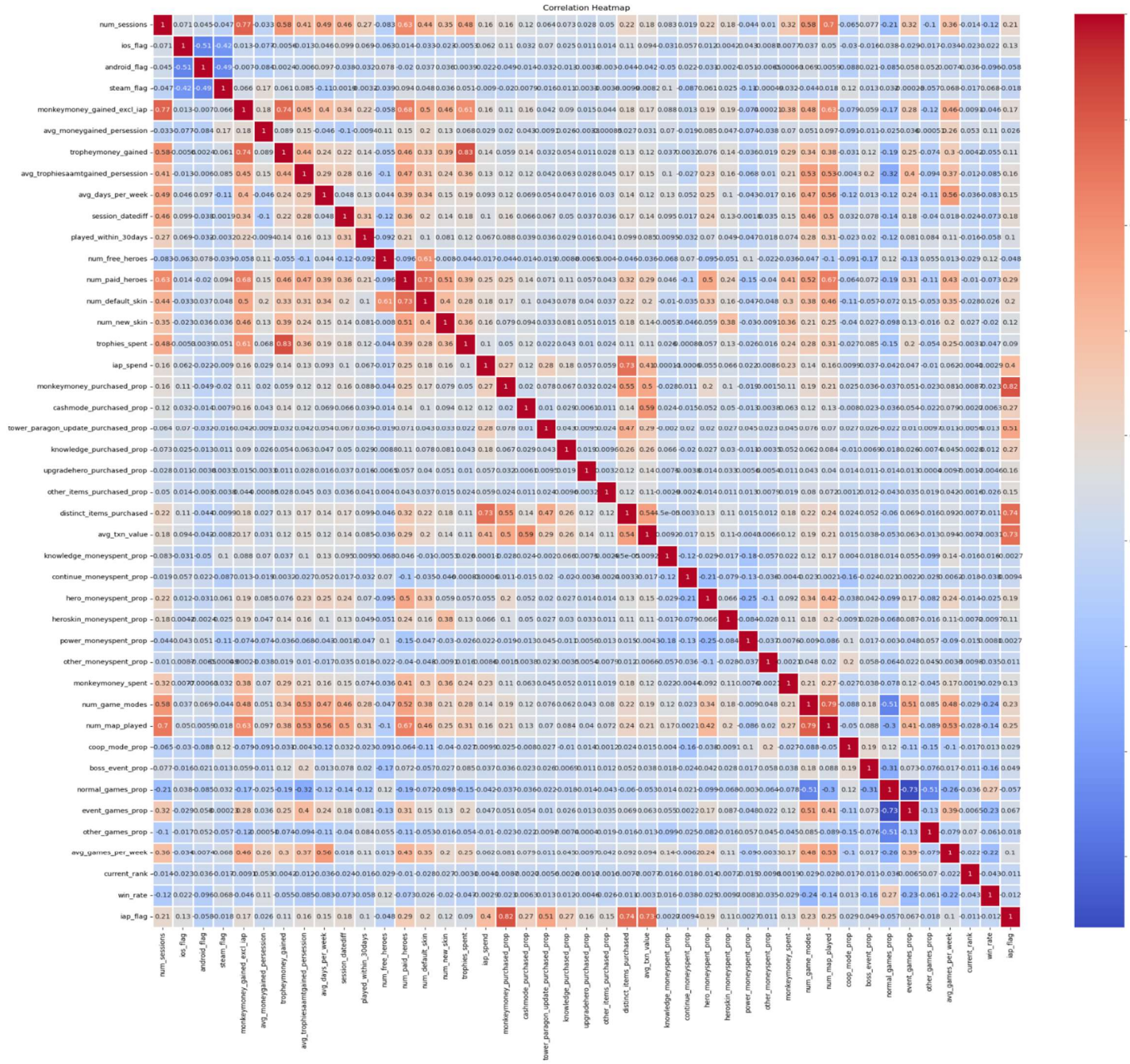


Figure A9, Correlation Coefficient Matrix