

Technical Appendix

Catch the Pink Flamingo Analysis

Introduction	2
A: Data Exploration	4
DATA SET OVERVIEW	4
AGGREGATION	7
FILTERING	12
A: Classification	13
DATA PREPARATION	13
DATA PARTITIONING AND MODELING	15
EVALUATION	17
ANALYSIS CONCLUSIONS	18
A: Clustering	20
ATTRIBUTE SELECTION	20
TRAINING DATA SET CREATION	20
CLUSTER CENTERS	21
RECOMMENDED ACTIONS	24
B: Graph Analytics	25
GRAPH DATABASE FOR CHATS	25
LONGEST CONVERSATION CHAIN	29
RELATIONSHIP BETWEEN CHATTIEST USERS AND TEAMS	30
HOW ACTIVE ARE GROUPS OF THE CHATTIEST USERS	32
RECOMMENDED ACTIONS	33
List of Tables and Figures	34

Introduction

The goal of the present analysis is to suggest potential strategies to increase revenue from the game *Catch the Pink Flamingo*¹.

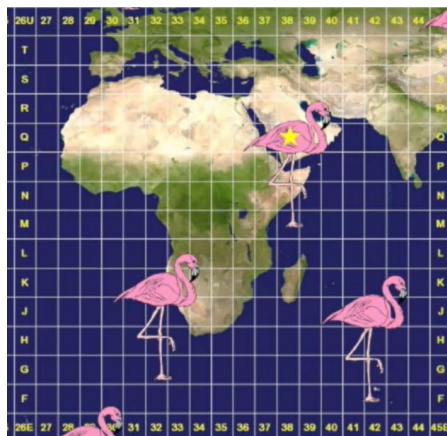


Figure.1: an in-game screenshot. source: Coursera

Two types of data are made available for the present analysis. The first type is in-game data and is analyzed in section A of the current appendix. The second type is on chat data and is presented in section B. The time frame covers 22 days in total between the 26th of May and the 16th June 2016. The game data covers 2,393 users among 109 teams. The users played 4,625 game sessions. In total, the users made 16,323 clicks on advertisements, 2,947 clicks on purchases and 755,806 clicks in-game.

¹*Disclaimer:* *Catch the Pink Flamingo* is a game of the fictitious company *Eglence Inc.* created in 2016 by the University of San Diego and Coursera for a series of [online courses on Big Data](#). The data is simulated and more information can be found on their [GitHub repository](#). The final capstone project was developed in partnership with [Splunk](#). The audience of this technical appendix and its presentation is the management of Eglence Inc. Finally, no flamingos were harmed during this analysis.

I start by exploring the data using Splunk. A dashboard has been set up to monitor the evolution of the current analysis and is shared in the app in Splunk.

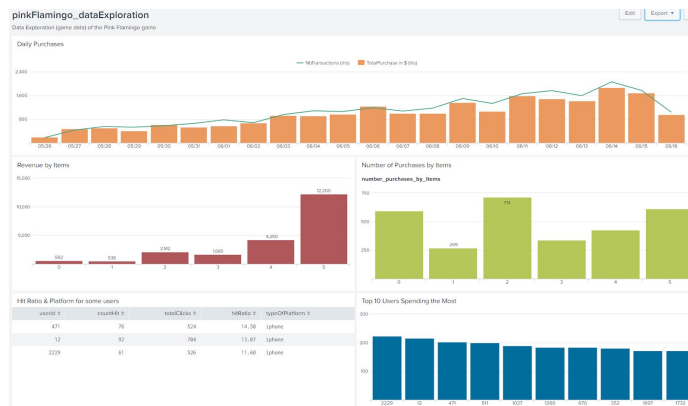


Figure.2: overview of the dashboard in Splunk

Afterwards, I use a decision tree to classify the users with the software KNIME. The goal is to identify characteristics of our users that can foster higher revenues and be explained easily.

Then, I apply a clustering method on users with Apache Spark through its API in Python. Can we combine some characteristics of our users and segment them through a new angle, a new perspective?

In the section B on chat data, I apply graph analytics techniques using Neo4J and its programming language Cypher. Even though we are not analyzing actual text and messages, the relationships among users and their teams will give us valuable insights on how our gamers community is organized.

A: Data Exploration

DATA SET OVERVIEW

The table below lists each of the game-data files available for analysis with a short description of what is found in each one.

File Name (extension: .csv)	Description	Fields (bolds are headers in file)
ad-clicks	each time a user clicks on an advertisement in the Flamingo app, a line is added to ad-clicks.csv	timestamp : [date] yyyy-mm-dd hh:mm:ss, when the click occurred on the advertisement. txId : [int] transaction's ID, unique among ad-clicks userSessionId : [int] session's ID of the user (who clicked) teamId : [int] ID of the team (of the user who clicked) adId : [int] ID of the ad clicked on adCategory : [string] category of the ad clicked on = {automotive, clothing, computers, electronics, fashion, games, hardware, movies, sports,}
buy-clicks	each time a user buys something in-app (in Flamingo app), a line is added to buy-clicks.csv	timestamp : [date] yyyy-mm-dd hh:mm:ss, when the purchase was made txId : [int] transaction's ID, unique among buy-clicks userSessionId : [int] session's ID of the user (who bought the item) team : [int] ID of the team (of the user who clicked) userId : [int] ID of the user (who bought the item) buyId : [int] ID of the item purchased price : [float] price of the item purchased
game-clicks	each time a user performs a click in the game, a line is added to game-clicks.csv	timestamp : [date] yyyy-mm-dd hh:mm:ss, when the click occurs clickId : [int] click's ID, unique among game-clicks userId : [int] user's ID (who clicked) userSessionId : session's ID of the user (who clicked) isHit : [boolean] 1 (one) if the click was on the targeted flamingo, 0 (zero) otherwise teamId : [int] ID of the team (of the user who clicked) teamLevel : [int] level of the team (of the user who clicked)

level-events	each time a team starts or finishes a level in the game, a line is added to level-events.csv	timestamp : [date] yyyy-mm-dd hh:mm:ss, when the event happens eventId : [int] event's ID, unique among level-events teamId : [int] team's ID teamLevel : [int] the level number (started or ended, see eventType) eventType : [string] the type of event, one of {'start', 'end'}
team	each time a team is created, a line is added to team.csv	teamId : [int] team's ID name : [string]: name of the team teamCreationTime : [date] yyyy-mm-dd hh:mm:ss, timestamp when the team was created teamEndTime : [date] yyyy-mm-dd hh:mm:ss, timestamp when the last member leaves and the team is ended. default=9999-12-31 23:59:59. strength : [float] team's strength, a measure of success currentLevel : [int] team's current level
team-assignments	each time a user joins a team, a line is added to team-assignments.csv	timestamp : [date] yyyy-mm-dd hh:mm:ss, when the user joined the team team : [int] ID of the team (that the user joined) userId : the ID of the user (who joined) assignmentId : [int] ID of assignment (of that user to that team at that time), unique among team-assignments
users	each time a user starts playing the game, a line is added to users.csv	timestamp : [date] yyyy-mm-dd hh:mm:ss, when the user first played the game userId : [int] ID assigned to this user nick : [string] nickname chosen by this user twitter : [string] twitter handle of this user dob : [date] yyyy-mm-dd, birthday of this user country : [string] a two-letter code of the country where this user lives
user-session	each time a session starts or ends, a line is added to user-session.csv.	timestamp : [date] yyyy-mm-dd hh:mm:ss, when the session starts/ends userSessionId : [int] ID for this session, unique among user-session userId : [int] user's ID teamId : [int] ID of the team (of this user) assignmentId : [int] ID of assignment of that user to that team (see team-assignments.csv) sessionType : [string] the type of event, one of {'start', 'end'}. Please note that a session denotes when a user starts/ends playing the game, or when a team moves to the next level (i.e. and then, the current session ends and a new one starts)

		teamLevel : [int] level of the team during that session platformType : [string] the type of platform used by the user during that session, {'android', 'iphone', 'linux', 'mac', 'windows', ...}
--	--	---

Table.1: the table above describes the game-data of Pink Flamingo in 8 CSV files: 3 on clicks made (on ad, bought item, in-game), 1 on level changes, 2 on teams (on their description and assignment) and 2 on users (on their description and on session).

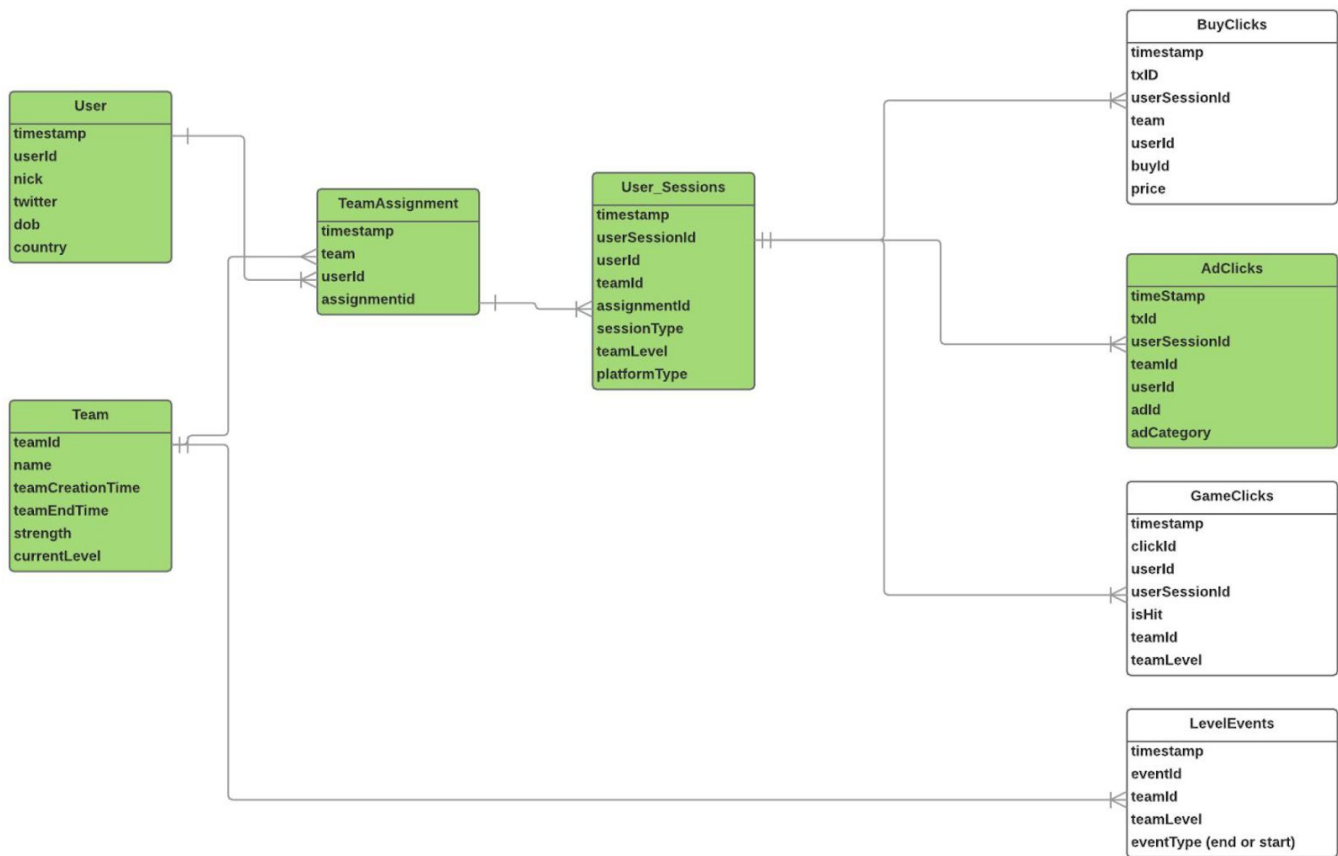


Figure.3: entity relationship diagram of Catch the Pink Flamingo game. source: Coursera.

I remind here that the time horizon covers 22 days in total between the 26th of May and the 16th June 2016. Collecting more data, especially across different months or seasons, would be important to analyze trends or any seasonal pattern in our users' behaviors, and this should be part of a future analysis.

Currently, the data enables us to understand our users characteristics and behaviors through their personal information (age, geographic location), the time played, the platform they used, their clicks on advertisement, purchases and in-game, and also their team's characteristics (strength, levels information).

AGGREGATION

Using Splunk, I start to explore the data through a time perspective. How much time the players spend on the game can be analyzed through their user-session data. The durations of the users' sessions range from 0.5 hour to 98 hours. These sessions accumulate to over 11,694 days of time played with an average of 2.5 days per session. The data on user sessions count only 1,093 unique users compared to 2,393 in the data on users (i.e. the users.csv file). On the right bar chart below, 40.53% of the 1,093 users played less than 200 hours and 31.84% above 400 hours. The biggest fan accumulated a total of 470 hours played while some players enjoyed the game for just 30 minutes.

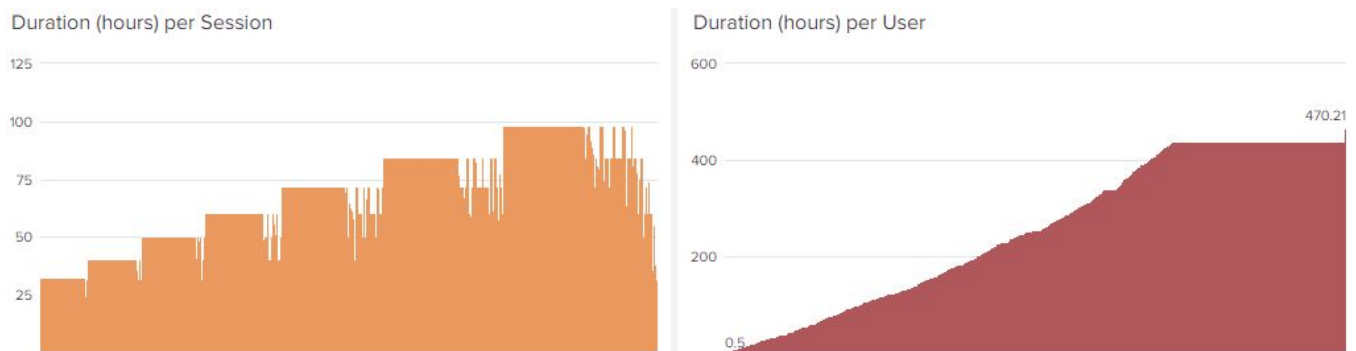


Figure.4: left bar chart represents the duration in hours per session (x-axis), ordered by session ID ascending. Right bar chart represents the duration in hours per user (x-axis), ordered by duration ascending.

Now, I turn to a daily analysis of our users. I aggregate the total amount spent and the number of transactions done per day. The trend is an increasing number of transactions per day (green line, right-hand side axis) and an increasing total amount in dollars spent buying items (orange bars, left-hand side axis). Culmination occurred on June 14th with 257 transactions and \$1,873 spent (by 195 users).

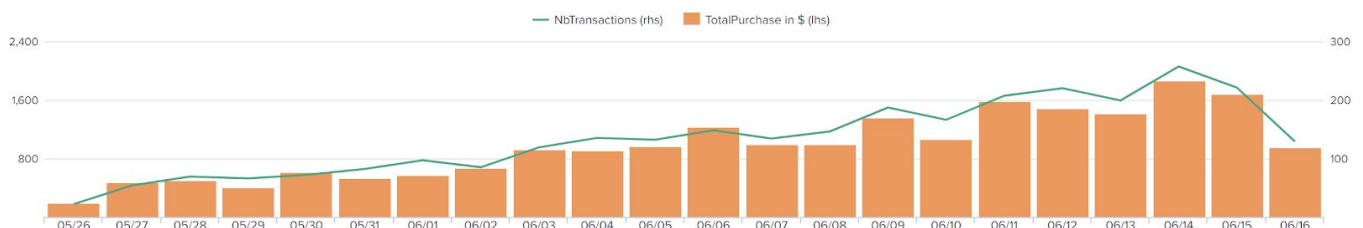


Figure.5: for this histogram, with days on the x-axis ranging from May 26th to June 16th, the orange bar represents the daily total amount of purchases in dollar terms (left-hand side axis) and the green line is the daily number of transactions (right-hand side axis, recall that a transaction ID represents the purchase of a single item).

For example, with this data, the further we go into time (from end of May to mid June), the higher the number of items purchased. Somewhat stabilizing around 200 items per day, with a top at 257 items on June 14th. Do the users tend to buy less at the end of the Month, and more in the first half of the month? Does a majority of the users receive a salary at the end of the month to explain such a trend? While I leave this for further studies, I focus on major aggregations in the following section.

Over the 22 days, the total amount spent buying items is \$ 21,407 and 6 unique items were available, each with their unique price from \$1 to \$20 (cf. table below):

Amount spent buying items	\$ 21,407
Number of unique items available to be purchased	6

Table.2: the table above shows aggregation statistics from the buy-clicks.csv file. There are six unique items available to be purchased, with ID = {0, 1, 2, 3, 4, 5} and respective price in dollars = {1, 2, 3, 5, 10, 20}.

I turn now through a perspective of the items. Mainly, what are the actual offers to the users and at what prices. The histogram shows how many times each item is purchased.

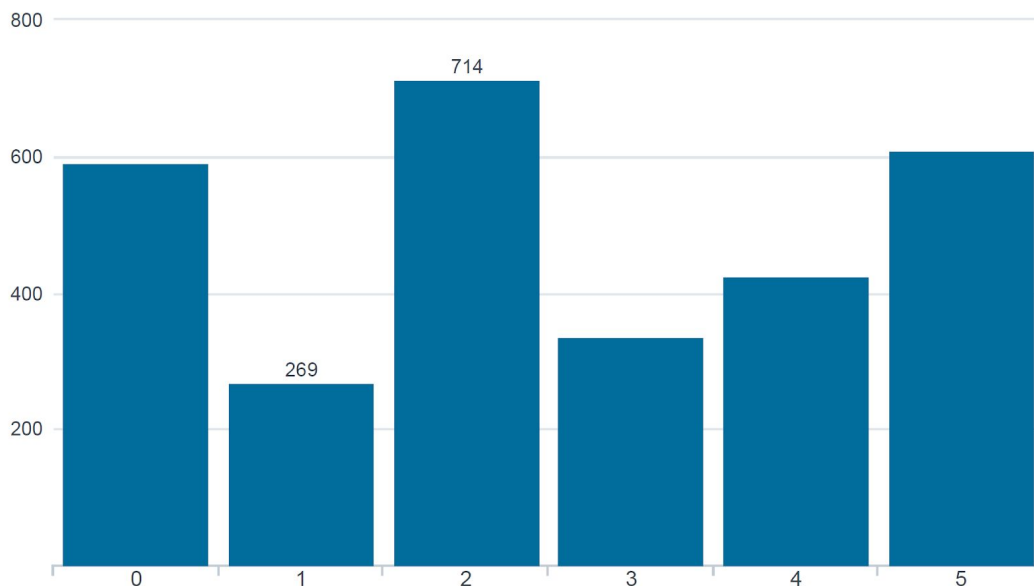


Figure.6: the bar chart above shows the number of purchases by item (identified by their ID). Item '2' was purchased 714 times, while item '1' was only purchased 269 times. Item '0' and '5' are both around 600 purchases each.

The figure shows how much money was made from each item:

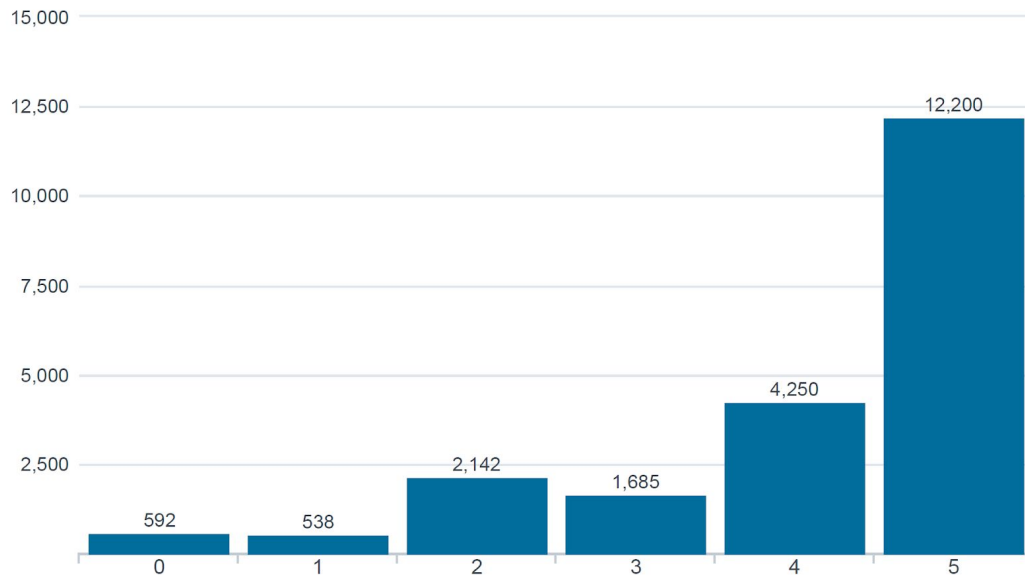


Figure.7: the bar chart above shows the revenue (\$) generated by each item (ID). The highest is item '5', at \$12,200, which is explained by its highest unit price (\$20) and a relatively high number of units sold (610 units vs an average of 491). On the contrary, item '0' has a low revenue due to its lowest unit price (\$1).

While there is no direct revenue analysis available from the advertisement data, what is the current advertisement display? Advertisements are tagged with the following category names: automotive, clothing, computers, electronics, fashion, games, hardware, movies and sports.

Ad Clicks per category per day

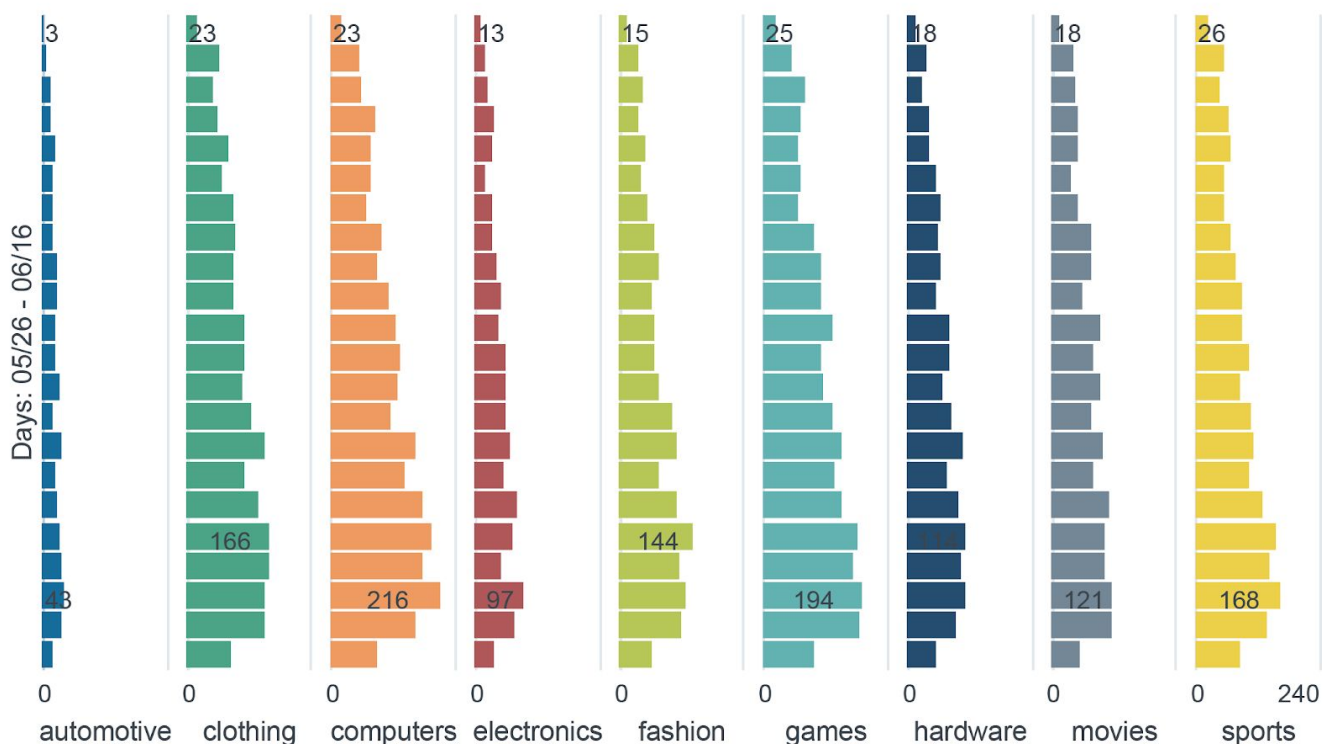


Figure.8: each bar represents the total clicks in a day for a given category. The categories are sorted alphabetically. For most of the categories, the bar chart exhibits an increasing trend with peaks on the 12th or 14th of June. The top three peaks are computers (216 clicks), games (194) and sports (168). The bottom peaks are automotive (48), electronics (97) and hardware (114).

Interestingly, in terms of total clicks over the entire period per category, the top three categories are computers (2638 clicks), games (2601) and clothing (2340). From figure.22, the latter shows a slightly more stable evolution over the 22 days of observation. The bottom three categories are automotive (566), electronics (1097) and hardware (1588).

With the help of level-events.csv file, I analyze the average time that the teams have spent in each level. From level 2 with 41 hours to level 7 with 98 hours. The strength of a team is supposed to be following such a trend because the strength increases as the team moves to next levels.

Average Time per Level

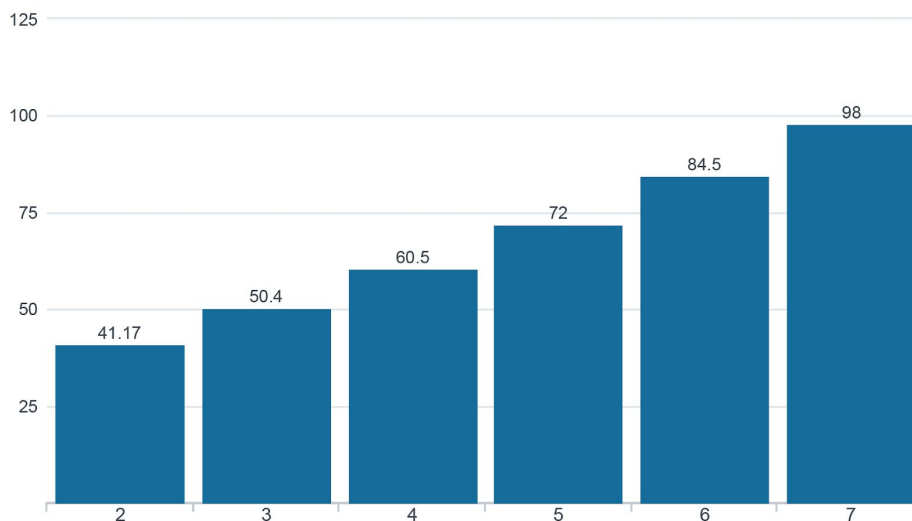


Figure.9: the bar chart above represents the average time in hours spent by level. The levels are sorted in increasing order from level 2 to level 7. Only finished levels' times are taken into account for the average. Level 1 is discarded due to missing data at the current stage of our analysis.

Final view will be on the hit ratio, defined by the actual hit on a flamingo during the game, divided by the total clicks the user made. Below, the left bar chart is per user and the right one is per team. The main point is that only a few users and one team actually deviates from the overall average of 11.12% hit ratio.

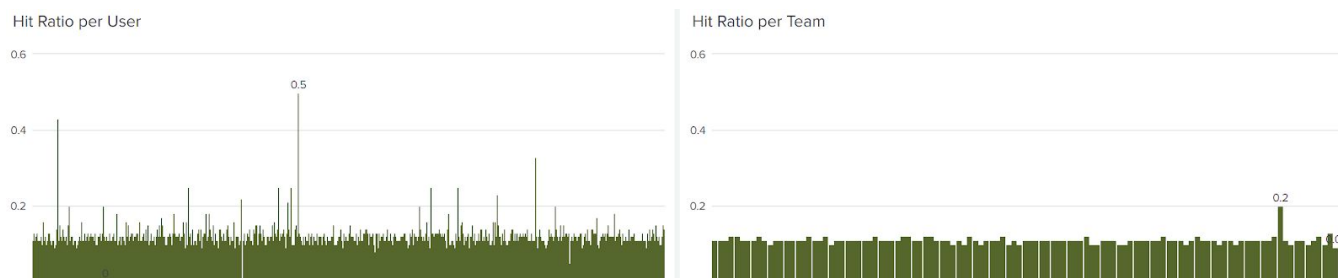


Figure.10: the primary purpose of the bar charts is to highlight that only a few users actually deviate from the average hit ratio of 11.12%. The top user is at 50% hit ratio and one team is at 20%.

As a last remark, please note that the analysis of users per geographic locations is left for a future study. However, I note that each country Egence Inc. covers has either 9 or 10 users. The map is a per country count and the chart bar is a count per continent.

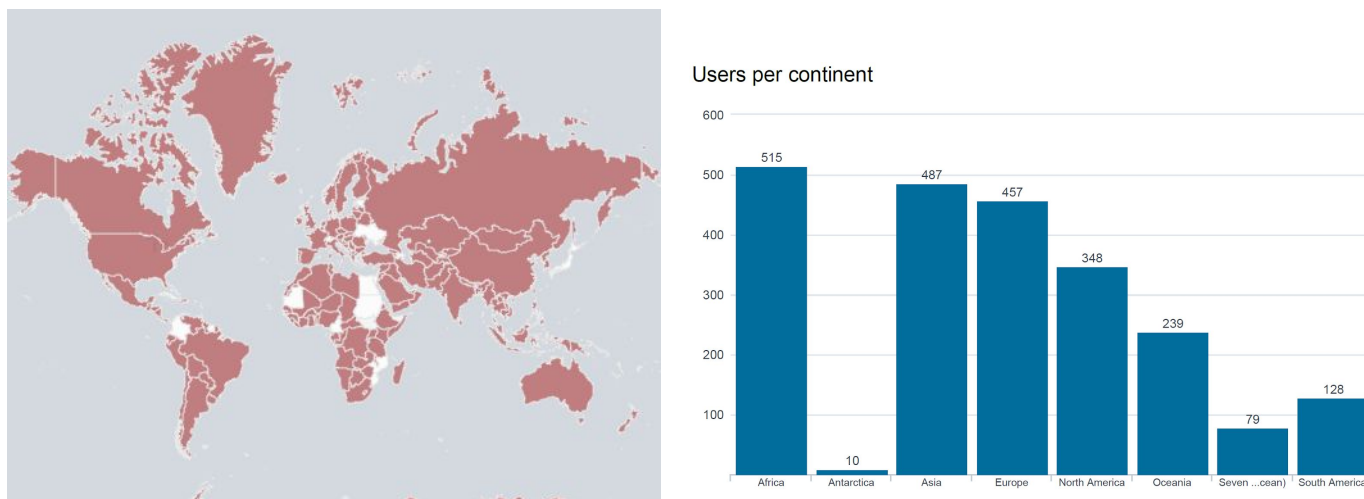


Figure.11: on the left, red countries have 10 users and the white ones have 9 users. On the right, aggregation by continents shows that there are 515 in Africa, 487 in Asia, 475 in Europe, 348 in North America, 239 in Oceania, 128 in South America, 79 in the Seven seas and 10 in Antarctica.

FILTERING

In the following, I will identify users based on their spending habits, platform used and accuracy in the game.

A bar chart showing the total amount of money spent by the top ten users.

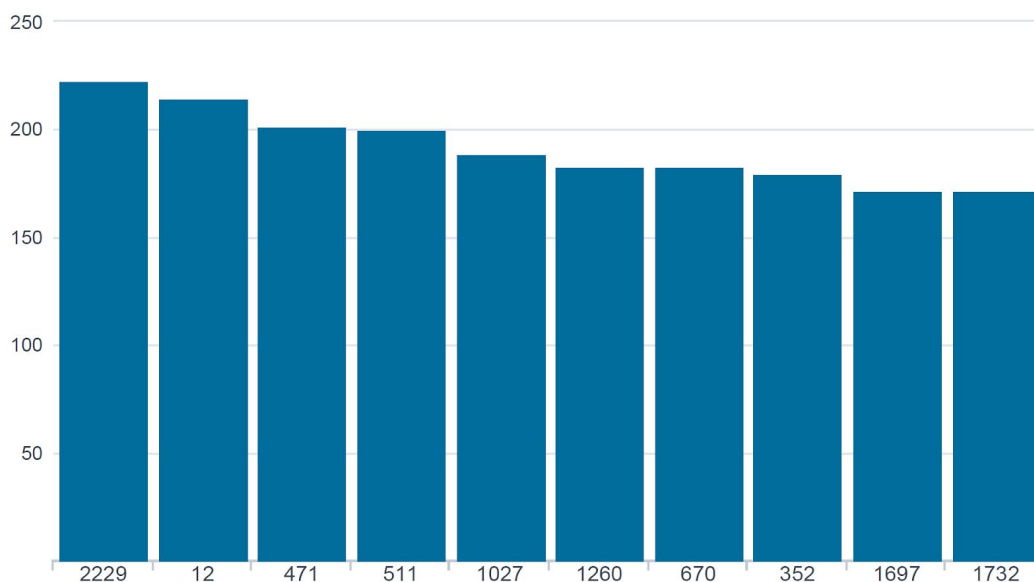


Figure.12: the histogram above shows the top 10 users ranked by how much they have spent buying items. The top-10 has spent \$1,919 in total. The user with ID '2229' has spent \$223. The 9th and 10th users spent \$172 each.

The following table shows the user id, platform, and hit-ratio percentage for the top three buying users:

Rank	User Id	Platform	Hit-Ratio (%)
1	2229	iphone	11.60%
2	12	iphone	13.07%
3	471	iphone	14.50%

Table.3: the table above shows, for the top-3 buying users, their id, platform and hit-ratio percentage. aggregation statistics from the buy-clicks.csv file. Among all users, the hit-ratio (%) ranges from 0% to 50% with an average of 11.09% (mode 12.50%, median 11.07%, over 1,119 users that bought items).

A: Classification

DATA PREPARATION

Now, I will use KNIME to classify the users. For technicality purposes, I use a data file (combined_data.csv) that combines the data from log files user-session.csv, buy-clicks.csv, and game-clicks.csv. As we go through the classification, please recall that there are six unique items available to be purchased with their respective price of \$1, \$2, \$3, \$5, \$10 and \$20.

Sample Selection

Item	Amount
# of Samples	4,619
# of Samples with Purchases	1,411

Table.4: from an original sample of 4,619 users, ~30% of them have made in-game purchases.

Attribute Creation

A new categorical attribute was created to enable analysis of players as broken into 2 categories: HighRollers and PennyPinchers. To achieve these two categories, I divided the average price into two classes: if purchases cost more than \$5, the class is marked as 'HighRollers', if purchases cost \$5 or less it is marked as 'PennyPinchers'. Therefore, 'HighRollers' are users that purchase items at a price of \$10 or \$20. 'PennyPinchers' have a wider offering with four items with price \$1, \$2, \$3 and \$5. A KNIME screenshot of the attribute follows:

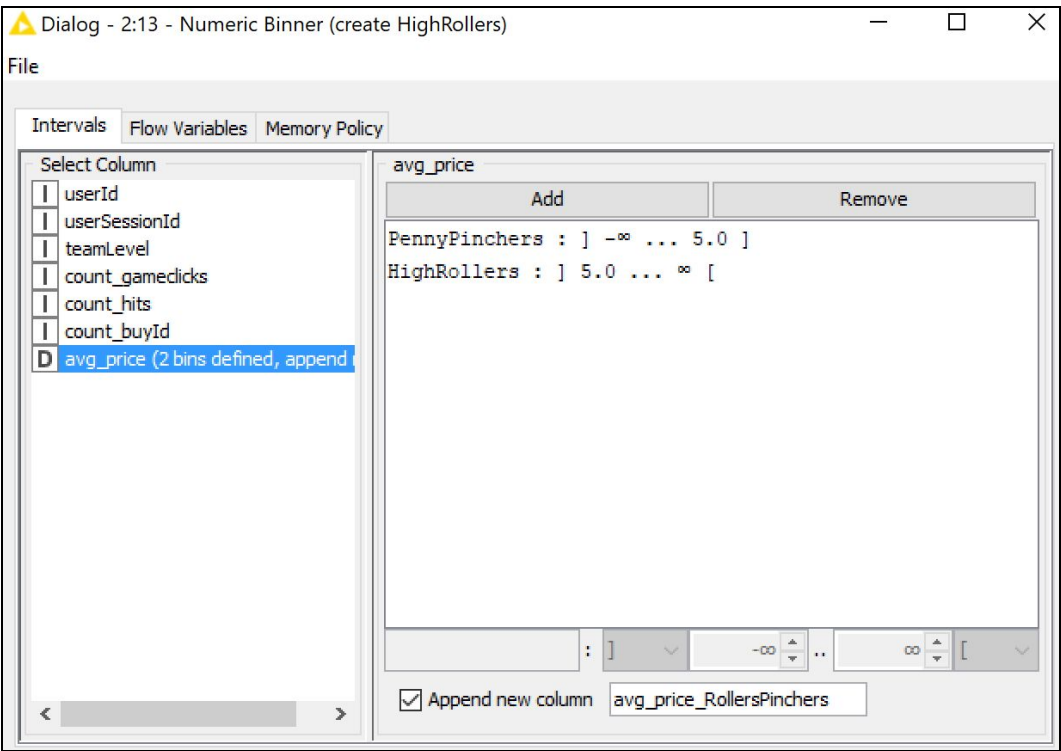


Figure.13: the above screenshot from KNIME shows a numeric binner node. Based on the column avg_price, the value ‘PennyPinchers’ is assigned to users having an avg_price below or equal to \$5.0, and a value ‘HighRollers’ is assigned to users with avg_price above \$5.0. To keep original data in the avg_price column, a new binned column called ‘avg_price_RollersPinchers’ is created.

The original data is kept and a new column (new categorical bin) is created and has values True or False.

The creation of this new categorical attribute was necessary because our data exploration showed interesting statistics from an important number of users buying only lower priced items. By categorizing the users into ‘PennyPinchers’ and ‘HighRollers’, the goal is to better understand the characteristics of these important users that have valuable potential to the revenue of Eglence Inc.

Attribute Selection

The following attributes were filtered from the dataset for the following reasons:

Attribute	Rationale for Filtering
avg_price	Excluded because this is the label that is actually needed in the current classification. This is the data on which the new column

	'avg_price_RollersPinchers' was created.
count_buyId	The number (ID) of the record for an item bought cannot bring additional value to the classification, and it is not expected to have any predictive power.
userId	The number (ID) of the user cannot bring additional value to the classification, and it is not expected to have any predictive power.
userSessionId	The number (ID) of the user session cannot bring additional value to the classification, and it is not expected to have any predictive power.

Table.5: the table above explains why the source column and three identity number columns were deleted.

DATA PARTITIONING AND MODELING

The data was partitioned into train and test datasets:

- The train dataset was used to create (i.e. train) the decision tree model.
- The trained model was then applied to the test dataset.

This is important because the trained model has to perform well on data that it has not yet seen. With the test dataset, it can be analyzed whether the model generalizes well (i.e. performs well on new data) or if it is overfitted (i.e. performs poorly on new data).

Stratified sampling was selected. Consequently, the distribution of classes of the partitioned data will be approximately similar to the one of the original dataset. Additionally, when partitioning the data using sampling, it is important to set the random seed because it enables us to have reproducible results. With a random seed set, the same partitions are returned at each node's execution.

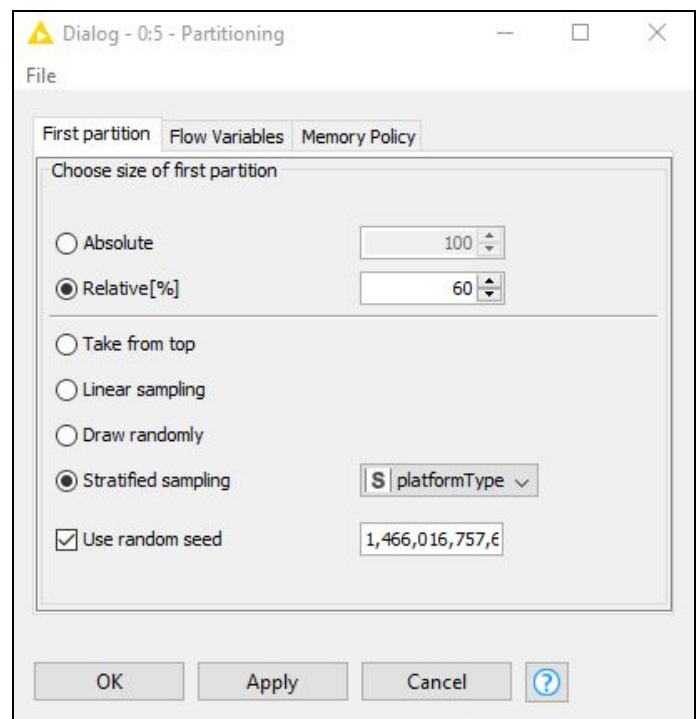


Figure.14: partitioning screenshot: 60%-40%, stratified sampling using random see.

Preparing the decision tree

A decision tree learner was configured. It enables us to set up a pruning technique. This technique reduces the size of the tree by removing sections of the tree that have little power to classify instances. Consequently, the complexity of the final classifier is reduced and so is the chance of an overfitted model. A screenshot of the resulting decision tree can be seen below. Due to pruning, no internal nodes remain and only leaf nodes, the platform type, remain:

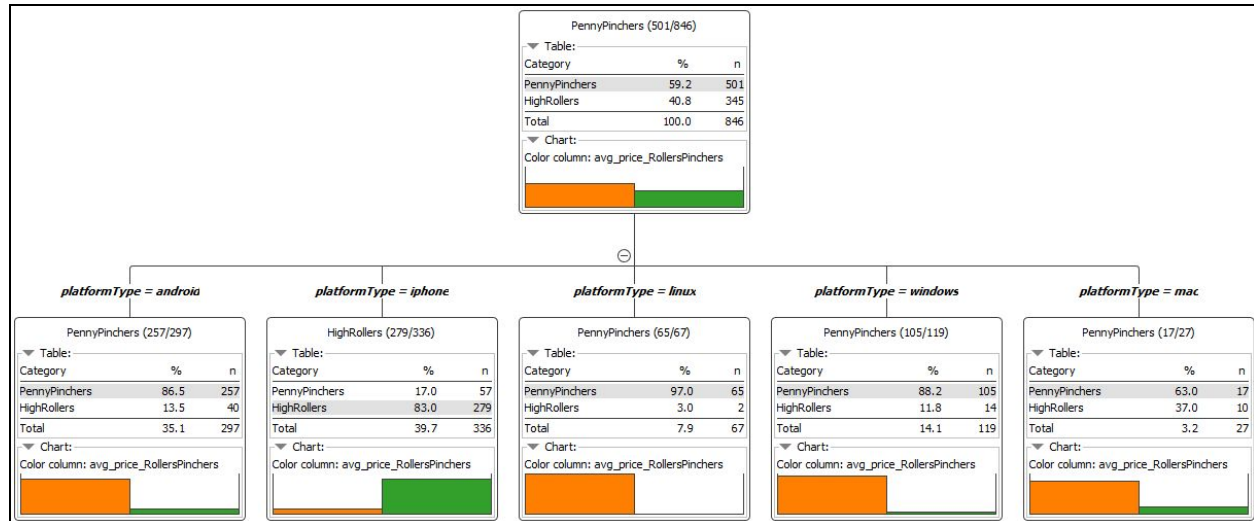


Figure.15: with all details, this is the decision tree on the test dataset. The greens are the 'HighRollers' which are clearly more present (in percentage) in users with 'iphone' platform type. The oranges are the 'PennyPinchers' and are present in all other platform types, especially 'linux' in percentage terms.

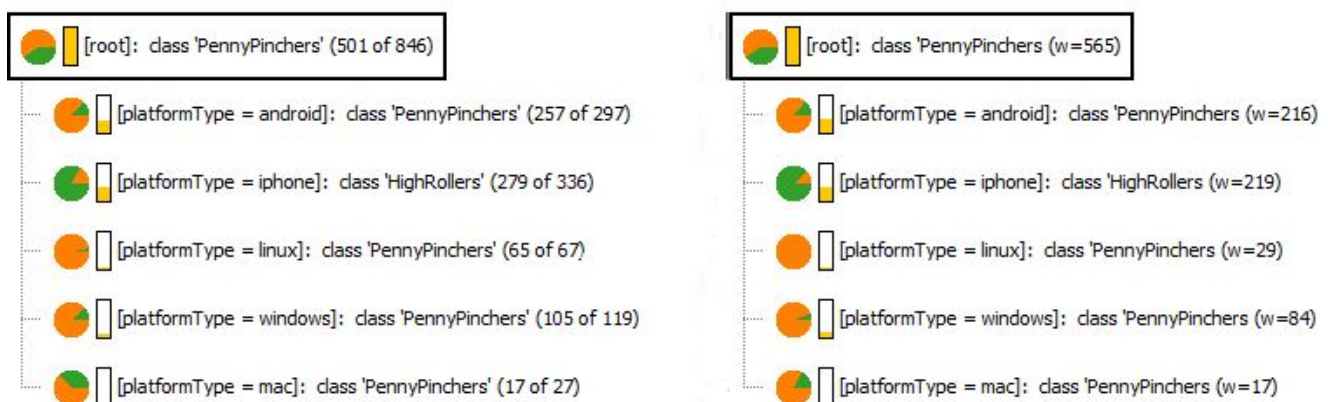
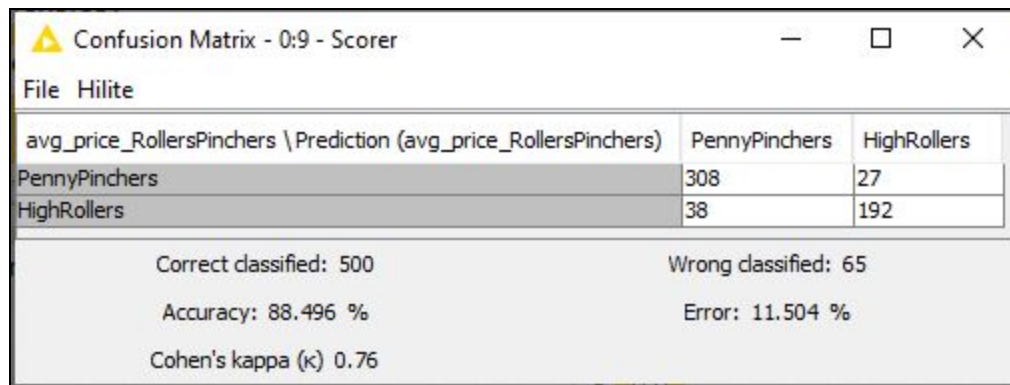


Figure.16: in simpler form, the left part represents the decision tree from the train dataset. The right decision tree is from the test dataset. The results are relatively similar and indicate potentially good generalization.

EVALUATION

A screenshot of the confusion matrix can be seen below:



avg_price_RollersPinchers \ Prediction (avg_price_RollersPinchers)	PennyPinchers	HighRollers
PennyPinchers	308	27
HighRollers	38	192

Correct classified: 500	Wrong classified: 65
Accuracy: 88.496 %	Error: 11.504 %
Cohen's kappa (κ) 0.76	

Figure.17: confusion matrix from KNIME with predicted class label in columns and true class label in rows. There are 500 correct predictions: 308 true positive and 192 true negative. There are 65 wrong predictions: 27 false negative (type II error) and 38 false positive (type I error).

65 values were wrongly classified. 27 were predicted HighRollers instead of PennyPinchers, and 38 PennyPinchers instead of HighRollers. 500 values were correctly classified. 308 were PennyPinchers and 192 for HighRollers.

As seen in the screenshot above, the overall accuracy of the model is 88.496%. The false negative rate is 8.06%: this is the fraction of true PennyPinchers that are not detected by the model. The false positive rate is 10.98%: this is the fraction of PennyPinchers that the model detects that are not real PennyPinchers.

To overcome some limitations of accuracy, I calculated the precision and the recall. The precision is 89.02%: this is a measure of exactness and calculates the percentage of PennyPinchers predicted as positive which are actually in a positive class. The recall is 91.94%: this is a measure of completeness and calculates the percentage of PennyPinchers that the model correctly identified.

These two measures complete each other because the precision does not tell anything about the number of actual PennyPinchers that the model predicted incorrectly, and the recall does not tell anything about how many actual HighRollers were incorrectly labeled as PennyPinchers.

Consequently, I calculate the F-measure which combines (evenly weighted in my case) the precision and recall. The F-measure equals 90.46%.

ANALYSIS CONCLUSIONS

The final KNIME workflow is shown below:

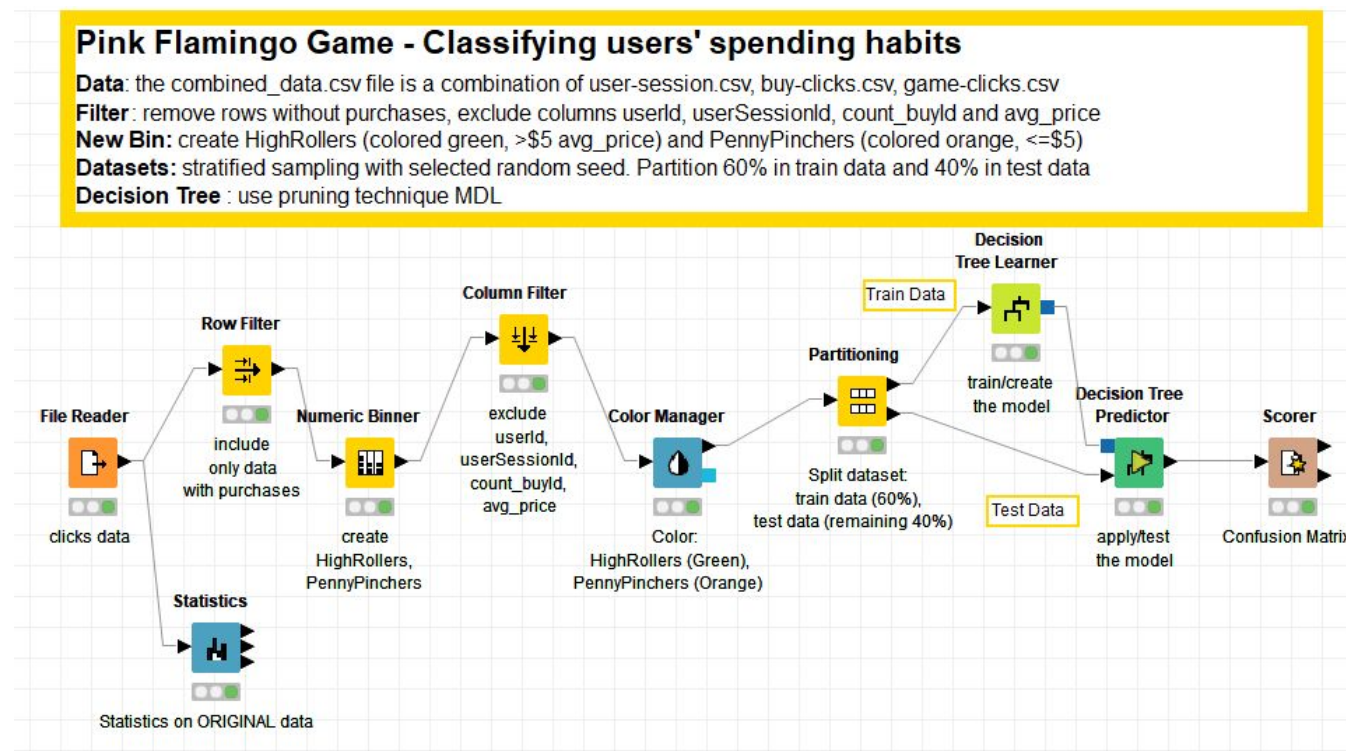


Figure.18: the final workflow from KNIME.

What makes a HighRoller vs. a PennyPincher?

Based on the classification obtained through the decision tree, users playing on an ‘iphone’ platform seem to have a higher capacity to buy expensive items. On the contrary, Linux followed by Windows users seem to be focused only on cheap items. Mac and Android users could have a potential for expensive items, but overall they are more keen on cheap items like Linux and Windows users. The rest of the metrics (team level, count of game clicks, count of hits) have less impact and have been pruned away.

Specific Recommendations to Increase Revenue
1. Continue to offer a wide selection of cheaper items to Linux, Windows, Mac and Android users
2. Provide a richer offering of ‘luxury’ items to iphone users. Because iPhone users are valuable customers of the (more) expensive items, it is crucial to ensure solid and rich updates toward that platform to keep our valuable clients with a tendency to offer a wide range of ‘luxury’ items.
3. Refine the analysis and prices with very cheap, normal, and expensive prices Could we find more pertinent results by segmenting our items between really cheap (e.g. on sale), normal and luxury (e.g. \$20, \$30, ...) items? Do iphone users have more potential above \$20 price tag? Can we split Linux, Windows, Mac and Android users between very cheap and normal prices?
4. Leverage advertisement clicks data Further analyses and studies should include classifications that include advertisement clicks data. As there is no actual revenue from ads, assumptions should be made on a certain percentage of revenue generated by each advertisement or each click for specific categories.
5. Age classification The date of birth of our gamers should be leveraged as well. For example, do adult players exhibit different behaviors?

Table.6: recommendations to increase revenue based on classification

A: Clustering

ATTRIBUTE SELECTION

features_used = ["age", "totalspent", "adClicks"]

Attribute	Rationale for Selection
age	the age of the user. Does a younger player behave with other characteristics than an adult player? The data indicates a mean of 37.69 years with 14yrs as standard deviation (youngest has 14-year, and the more mature player 69-year old).
totalspent	Per user: total amount of money spent purchasing items Distinguish the user's willingness to spend. Can we trigger a new buying habit to the users that don't buy much? Can we leverage even better the items sold to our best buyers?
adClicks	Per user: total number of clicks on ads Who are more prone to click on ads, who are more curious? Can we ask why some players are not clicking and what is not interesting to them? Are the ones not interested in ad, actually interested in something else?

Table.7: attribute selection for clustering

TRAINING DATA SET CREATION

The training data set used for this analysis is shown below (first 5 lines):

age	total spent	ad clicks
28.0	63.0	19
19.0	20.0	39
55.0	28.0	37
47.0	95.0	34
56.0	53.0	41

Table.8: first five lines of our training data set which includes three columns: age, total spent and advertisement clicks per user. The entire data set has 543 rows.

The dimensions of the final data set is 543 rows x 3 columns. The number of clusters created is 3.

For the current study, a visual guess has been applied to select the number of clusters. However, specific methods, such as the elbow test, will be applied in the next study to ensure proper identification of the number of clusters.

CLUSTER CENTERS

The code used in creating cluster centers is given below:

```
>featuresUsed = ['age', 'totalspent', 'adClicks']  
>...  
>kmeans = KMeans(k=3, seed=1)  
> model = kmeans.fit(scaledData)  
>centers = model.clusterCenters()  
>print(centers)
```

Cluster centers formed are given in the table below

Cluster #	Center [age, total spent, ad clicks]	color of the center
1 <blue>	-0.1753, -0.4221, -0.9201	yellow
2 <orange>	0.2617, -0.2114, 0.8060	green
3 <red>	-0.1650, 2.0273, 0.7702	black

Table.9: the three centers with coordinates on the axis [age, total spent, ad clicks]. The three centers do not move much on the age-axis. The red-cluster's center (point black in 3D-plot) shows a strong difference on the total-spent-axis. The blue-cluster's center (point yellow in 3D-plot) is different from the orange-cluster's center (point green) and red-cluster's center based on the ad-clicks-axis.

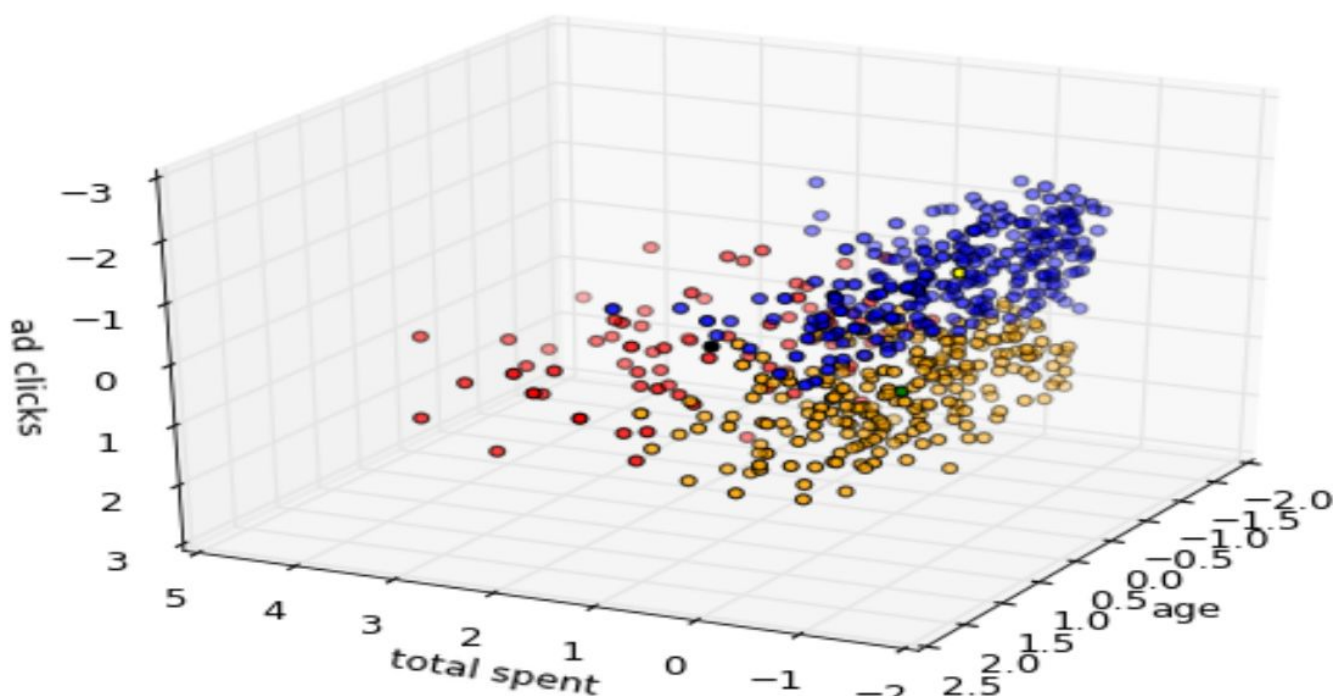


Figure.19: a 3D view of ad-clicks, total spent and age attributes with the three resulting clusters in red, blue and orange. Their respective centers are the black, yellow and green dot respectively.

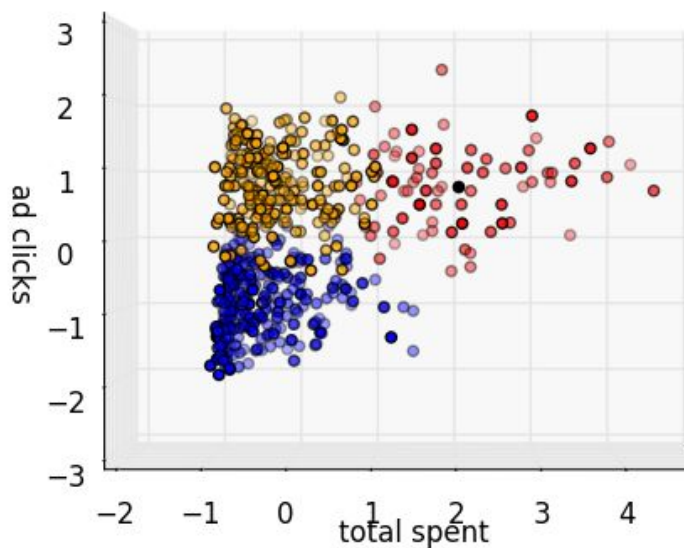


Figure.20: comparing advertisement clicks and total spent by users reveals interesting clusters. The orange cluster has gamers who tend to spend little and click relatively heavily on advertisements. The blue cluster also

has gamers that spend little but they also click less on ads. Finally, the red cluster represents the gamers that spent the most and click the most on ads - they also seem to be a smaller group.

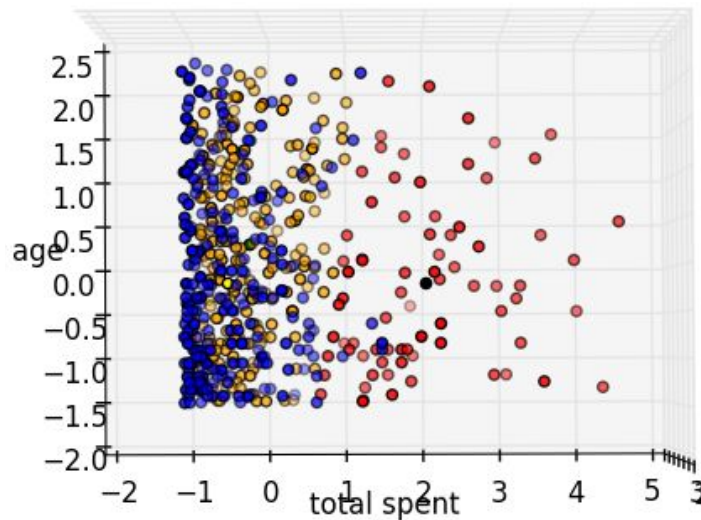


Figure.21: any separation between minor and adult gamers are actually not distinguishable. In other words, the three clusters do not show clear separation on the y-axis here. On the total spent, however, the blue and the orange clusters have gamers that tend to spend little. The red cluster has gamers that tend to spend more.

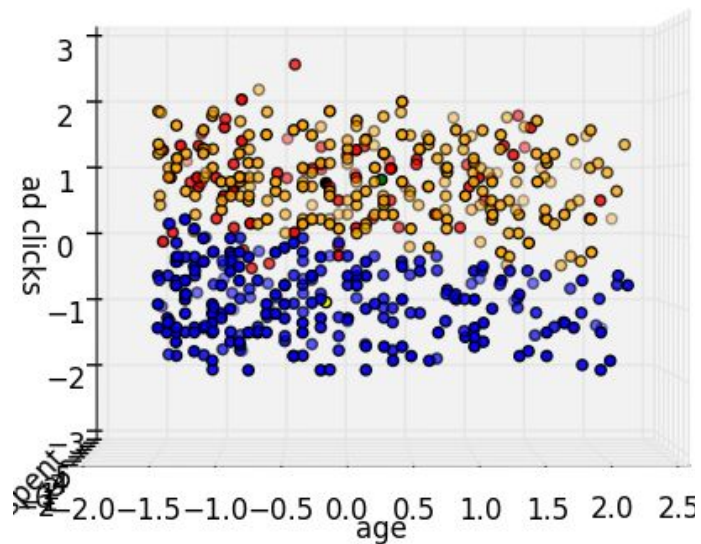


Figure.22: the vertical split from advertisement clicks is clear. However, it is more complicated to see any split based on the age of our gamers which could be questioned.

These clusters can be differentiated from each other as follows:

Cluster <blue> is different from the others in that these are the users who both spent the least on purchases and click the least on advertisements.

Cluster <orange> is somehow in-between in that these are the users who spent also less on purchases, but actually clicks heavily on advertisements. Additionally, they seem to be as numerous as the users in cluster <blue>.

Cluster <red> is different from the others in that these are the users who spent the most and click the most on advertisements. These should be the gamers that contribute the most to our revenue, at least on an individual basis as they are much less.

	features	prediction
0	[-0.6230, 0.5737, -0.6818]	0
1	[-1.2514, -0.4694, 0.6326]	1
2	[1.2622, -0.2753, 0.5012]	1
3	[0.7036, 1.3500, 0.3040]	2
4	[1.3321, 0.3311, 0.7641]	1

Table.10: the first five rows of the train data set

RECOMMENDED ACTIONS

Action Recommended	Rationale for the action
Send survey to users who spend little and click little	Indirect impact, understand our users better. The idea is to understand better why the users are actually not clicking on ads and not buying. We cannot simply imply that they don't have the revenue for it. They are possibly hidden lucrative users.
Analyse the Pricing of Advertisements	This is particularly important for users in cluster orange because they tend to spend less, but are among the best candidates for advertisements. Can we find the most lucrative ad categories for these users?
For the users that clicks more on ads and spend more, increase prices	As those users have more resources (i.e. revenue) to spend on items and clicks more on ads, we can expect a positive direct impact by a marginal increase in price. We can increase marginally up to (or just before) the point when they tend to buy less items.

Table.11: recommended actions based on clustering analysis.

B: Graph Analytics

In this section, I model the chat data. The data consists of numeric values only and no text analysis takes place here. It can still be of added-value for Eglence, Inc. through proper graph analytics. Indeed, I seek to understand the strength and directions of relationships among the community. This community can be represented with nodes (i.e. the users, the teams, the chats, ...) and edges (i.e. the link or connection between nodes).

The graph model that I used has several types of nodes: `userId`, `teamId`, `TeamChatSessionId` and `ChatItem`. The graph model has multiple types of edges: `CreateSession`, `OwnedBy`, `Joins`, `Leaves`, `CreateChat`, `PartOf`, `Mentioned`, and `ResponseTo`. The graph model is illustrated below in Figure.23.

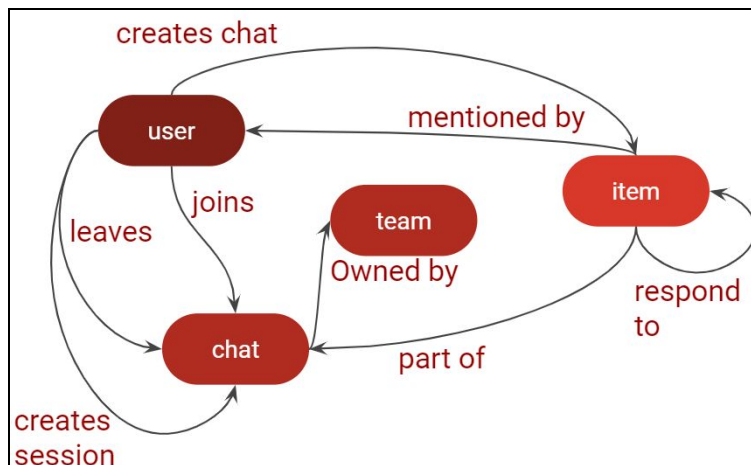


Figure.23: a schema illustrating the graph created in Neo4J. A user can create, join or leave a chat session. The chat session is owned by a team. and a chat item is part of a chat session. A chat item can mention a user and can respond to another chat item. A user can also create a chat item.

GRAPH DATABASE FOR CHATS

Firstly, I describe three steps taken to create the graph database: (i) the schema of files used, (ii) the load process, and (iii) a screenshot example from the graph itself. Afterwards, I will seek the longest conversation chain, investigate the relationship between the chattiest users and teams, and calculate the cluster coefficients. Finally, I will conclude this section with recommended actions.

i) Schema of the 6 CSV files

chat_create_team_chat	<p>When a user creates a new chat for its team, a line is added to this file</p> <p>userId [int] contains the id of the user</p> <p>teamId [int] contains the id of the team that the user is part of</p> <p>teamChatSessionId [int] contains the id of the chat session the user is part of</p> <p>timestamp [date] contains the timestamp when the user created the chat session</p>
chat_item_team_chat	<p>Creates nodes labeled ChatItems. Also create an edge labeled "PartOf" from the ChatItem node to the TeamChatSession node. This edge should also have a timeStamps property.</p> <p>userId [int] contains the id of the user</p> <p>teamChatSessionId [int] contains the id of the chat session the user is part of</p> <p>chatItemId [int] contains the id of the chat item</p> <p>timestamp [date] contains the timestamp of when the user created the chat item</p>
chat_join_team_chat	<p>Creates an edge labeled "Joins" from User to TeamChatSession. The columns are the User id, TeamChatSession id and the timestamp of the Joins edge.</p> <p>userId [int] contains the id of the user</p> <p>teamChatSessionId [int] contains the id of the chat session the user is part of</p> <p>timestamp [date] contains the timestamp of when the user joined the chat session</p>
chat_leave_team_chat	<p>Creates an edge labeled "Leaves" from User to TeamChatSession. The columns are the User id, TeamChatSession id and the timestamp of the Leaves edge.</p> <p>userId [int] contains the id of the user</p> <p>teamChatSessionId [int] contains the id of the chat session the user is part of</p> <p>timestamp [date] contains the timestamp of when the user left the chat session</p>
chat_mention_team_chat	<p>Creates an edge labeled "Mentioned".</p> <p>chatItemId [int] contains the id of the chat item</p> <p>userId [int] contains the id of the user</p> <p>timestamp [date] contains the timestamp of when the chat item mentioned a user</p>
chat_respond_team_chat	<p>A line is added to this file when a player with chatid2 responds to a chat post by another player with chatid1.</p> <p>chatid1 [int] contains the id of the chat item 1 (first)</p> <p>chatid2 [int] contains the id of the chat item 2 (second)</p> <p>timestamp [date] contains the timestamp of when the first chat item responds to the second chat item, creating a thread of conversation.</p>

Table.12: schema of the 6 CSV files on chat data.

ii) Loading process

Using Neo4J and its language Cypher, I load the data files, create their respective nodes and edges. The following example illustrates the loading process (with actual code in orange that forms a query to be executed at once):

1) the file 'chat_respond_team_chat.csv' is read from its path and is iterated row by row:

```
LOAD CSV FROM "file:/chat_respond_team_chat.csv" AS row
```

2) Then, a node for the chat item One is created. Its type is converted from string to integer. The 'row[0]' indicates that the first item (i.e. index zero) of the row corresponds to the chat item One:

```
MERGE (iOne:ChatItem {id: toInteger(row[0])})
```

3) Similar instruction is written to create chat item Two's node. This time, the index one is used:

```
MERGE (iTwo:ChatItem {id: toInteger(row[1])})
```

4) Finally, the edge 'ResponseTo' is created. It connects the nodes 'iOne' (chat item One) to 'iTwo'. Additionally, an attribute 'timeStamp' is added from the third index of the row:

```
MERGE (iOne)-[:ResponseTo{timeStamp: row[2]}]->(iTwo)
```

iii) Screenshot of generated graph

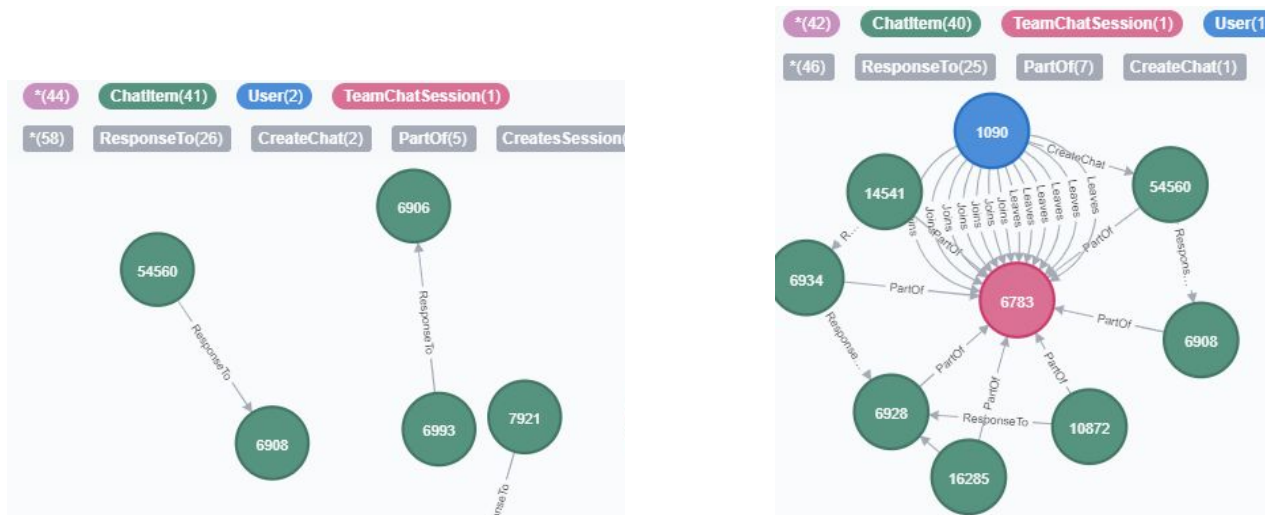


Figure.24: The left graph is what the example query (explained in point ii) loading process) actually created. These are some of the chat-item-One and -Two nodes (green) and RespondTo edges, and for example, chat-item-One (id 54560) responds to chat-time-Two (id 6908). In the right graph, I expanded the child relationships of the chat-item-One node with id 54560: it shows that it is part of team-chat-session id 6783, and so is the chat-item-Two node with id 6908. The blue node represents a user node (id 1090) that creates the chat-item-One node (id 54560), joins and leaves the team-chat-session id 6783.

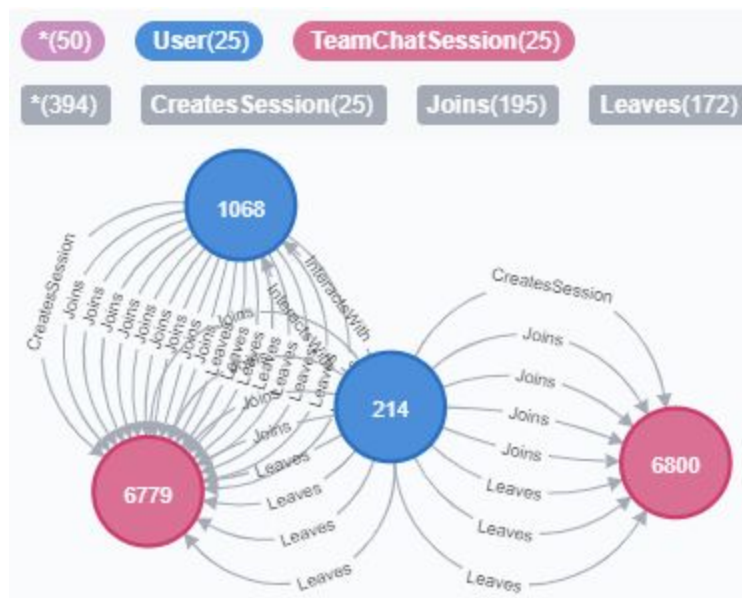


Figure.25: not directly related to the example query, this graph shows two user nodes (id 1068, 214) and two team-chat-session nodes (id 6779, 6800). User 1068 creates session 6779, and user 214 creates session 6800. In addition, Joins and Leaves edges are clearly visible.

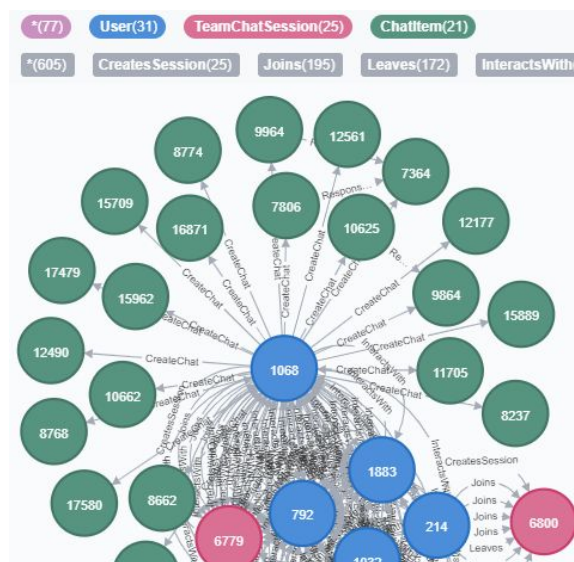


Figure.26: from Figure.25, for user 1068, I expanded its child relationships. This results in multiple chat-item nodes (green) being displayed on the graph, each with an edge CreateChat. Additional user

nodes (blue) are shown as the user 1068 interacts with them. The un-readable edges are an accumulation of Joins, Leaves and InteractsWith edges mainly.

LONGEST CONVERSATION CHAIN

To find the longest conversation chain, I make use of the 'ResponseTo' edge in the following query:

```
MATCH path=()-[:ResponseTo*]->()  
RETURN length(path)  
ORDER BY length(path) DESC LIMIT 3
```

That particular query is built by creating a variable 'path' and returning its length. This will return the length of any edge chains with the label 'ResponseTo'. As only the longest one is of interest, the results are ordered by the length from longest to shortest and only the top three are shown. The longest edge chain has a length of 9, followed by two edge chains of length 8 each.

As I know the longest edge chain has a length of 9, I can obtain the number of unique users with the following query:

```
MATCH path=()-[:ResponseTo*]->()  
WHERE length(path)=9  
MATCH uCount=(u:User)-[:CreateChat]->(i:ChatItem)  
WHERE (i IN NODES(path))  
RETURN count(DISTINCT u), count(uCount)
```

I use a clause 'WHERE' to filter edge chain of length 9 only and use a second 'MATCH' to create a subquery with a path named 'uCount'. The latter account for any user 'u' that creates any chat item 'i'. I use a second clause 'WHERE' to select only the chat items that are part of edge chains of length 9 ('path'). Finally, I return a distinct count of user 'u' which is 5, and the number of chat items (10).

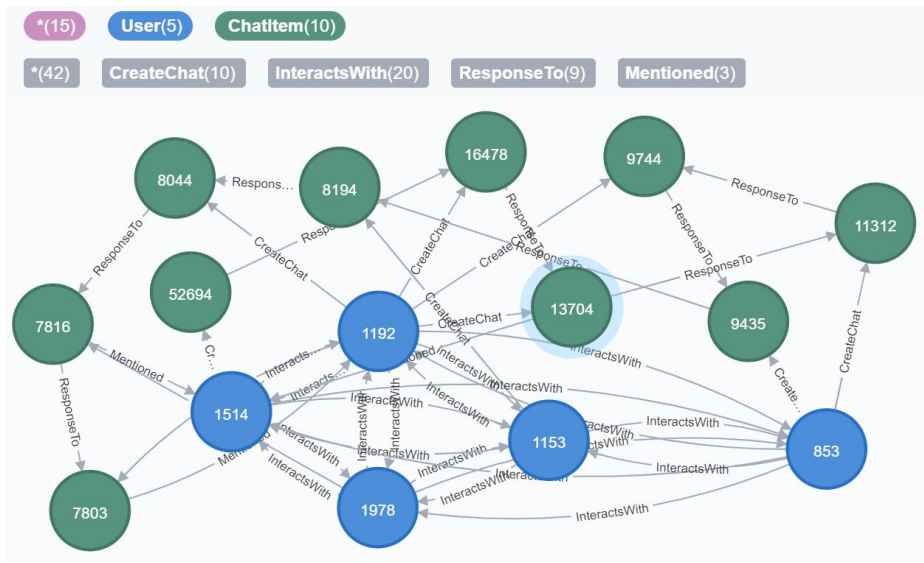


Figure.27: the longest edge chain (with 10 green nodes, the chat items) and the unique users (5 unique blue nodes).

For Egence, such queries can give the strength of users' interaction among themselves. As there are hundreds of users interacting and reading, writing, both constructive and destructive exchanges can take place. This is only a first step for Egence to leverage chat data.

RELATIONSHIP BETWEEN CHATTIEST USERS AND TEAMS

Chattiest Users

To find the users who chat the most, I match any user creating an edge 'CreateChat' and count the number of edges created by each user. Only the top 10 counts are returned and three of them are shown in the table.13.

```
MATCH (n:User)-[r>CreateChat]->()
RETURN n.id ,count(r)
ORDER BY count(r) DESC LIMIT 10
```

Users	Number of Chats
394	115
2067	111
1087	109

Table.13: also the query returns the top 10 users chatting the most, only three are displayed for convenience.

Chattiest Teams

To find the teams speaking the most in chat, I match any chat item 'c' that is part of a team-chat-session that is subsequently owned by a team 't'. Then, I count the number of chats 'c' of

each team and return the results in descending order - limited to ten in the script but showing the top three in the table.14 for convenience.

```
MATCH (c:ChatItem)-[:PartOf]->(:TeamChatSession)-[:OwnedBy]->(t:Team)
RETURN t.id, count(c) AS chats
ORDER BY chats DESC LIMIT 10
```

Teams	Number of Chats
82	1324
185	1036
112	957

Table.14: also the query returns the top 10 teams chatting the most, only three are displayed for convenience.

Chattiest Users in the chattiest Teams?

To find if any of the chattiest users are part of any of the chattiest teams, I combine the two 'MATCH' clauses and results in a long query that finds any user creating a chat item, that is part of a team-chat-session, that is owned by a team. By returning the number of chats with their matching user id and team id, and ordering them in descending order, I conclude that only the chattiest user with id 999 was part of the chattiest team with id 52.

```
MATCH
(n:User)-[:CreateChat]->(c:ChatItem)-[:PartOf]->(:TeamChatSession)-[:OwnedBy]->(t:Team)
RETURN n.id, t.id, count(c) AS chats
ORDER BY chats DESC LIMIT 10
```

For Egence Inc., In a community, it can be a great value to identify the member with the biggest voice or greatest influence in a sense. Through such reports, Egence Inc. is able to target the potentially most influential user, whether this user's influence is direct or not. Same goes for the most chattiest team. Now, can we learn more about this user's characteristics and behaviors? How should the sales team contact them? What offers could we provide these chattiest users and teams with?

HOW ACTIVE ARE GROUPS OF THE CHATTIEST USERS

As a final part of the graph analysis, I will calculate the cluster coefficients². This measure will help define how tightly connected the users are. The coefficient is equal to one if, for a given node u, each neighbor of u is connected to the neighborhood (i.e. all neighbors of u). A coefficient of zero would

² The local clustering coefficient of a node in a graph quantifies how close its neighbours are to being a complete graph (a clique). I use the formula for the directed graph as edges are mainly unidirectional ([Wikipedia](#)).

mean that the neighbors of u are only connected to u and not among themselves. The calculation is the result of the following four steps:

1) Connect mentioned users

I start by creating a new edge 'InteractsWith' to identify the neighborhoods when users mention each other. In other words, this edge is defined by matching any user ($u1$) creating a chat that mentions another user ($u2$). The 'MERGE' ensures to keep a directed graph.

```
MATCH (u1:User)-[:CreateChat]-(i:ChatItem)-[:Mentioned]->(u2:User)
MERGE (u1)-[:InteractsWith]->(u2)
```

2) Connect users responses with the chat creator

In our graph, another way to identify neighborhoods is by checking the thread of conversations through the edge 'ResponseTo'. I will add these to the already created edge 'InteractsWith'. Again, the 'MERGE' ensures to keep a directed graph.

```
MATCH (u1:User)-[:CreateChat]-(i1:ChatItem)-[:ResponseTo]->(i2:ChatItem)
WITH u1, i1, i2
MATCH (u2)-[:CreateChat]-(i2)
MERGE (u1)-[:InteractsWith]->(u2)
```

3) Eliminate all self interaction

Should a user ($u1$) respond to a chat that she actually created, a doublon would be created. I delete them by the following cypher query:

```
MATCH (u1)-[r:InteractsWith]->(u1)
DELETE r
```

4) Create the clustering coefficients

For the given chattiest users (cf. earlier query), I retrieve a collection of distinct ' $u2$ ' users that 'interactsWith' other chattiest users. This collection is called 'neighbours'. I collect the number of distinct nodes and call it ' k '. Then, I proceed to a subquery with them: for each neighbor, count the unique interactions ($r2$) they ($u3$) have with their own neighbors ($u4$). When there more than one edge between two neighbors, it is counted as 1. It keeps the calculation for a directed graph in that way.

```
MATCH (u1:User)-[r1:InteractsWith]->(u2:User)
WHERE u1.id IN [394,2067,1087,209,554,1627,516,999,668,461]
AND (u1.id <> u2.id)
WITH COLLECT(u2.id) AS neighbours, COUNT(DISTINCT r1) AS k, u1
MATCH (u3:User)-[r2:InteractsWith]->(u4:User)
WHERE (u3.id IN neighbours) AND (u4.id IN neighbours) AND (u3.id <> u4.id)
WITH u1, u3, u4, k,
CASE WHEN COUNT(r2) > 0 THEN 1
ELSE 0
```



```
END AS result
RETURN u1.id, SUM(result)*1.0/(k*(k-1)) AS coef
ORDER BY coef DESC LIMIT 10
```

Most Active Users based on Cluster Coefficients (among chattiest users)

User ID	Coefficient
209	1.0
668	1.0
999	0.9464

Table.15: among the chattiest users, users with id 209, 668 and 999 are the most active.

Interestingly, a cross-reference with the game-data shows that user id 209 has not made any purchases nor did she click on any advertisement. However, she did play over 437 hours with an average hit ratio of 13%. Similarly, user 668 played 275 hours with a hit ratio at 14% on average. User 999 does not have any of these records.

Among the top 10 active users, user id 1087 did purchase seven items (1x #0, 3x #1, 2x #2, 1x #5) for a total of \$33. In addition, she clicked on 33 advertisements across multiple categories such as computers, hardware and sports. Finally, she plays on a mac and evolved up to the level 6 currently. As six out of seven items are below the \$5 threshold and that she plays on a mac, she is a penny pincher.

This cross-reference is for illustration purposes. If Egence, Inc. wants to reward the most active users, it has to approach them with the most suited gift or promotion.

RECOMMENDED ACTIONS

For the graph analytics, the recommendation are

- to take the analysis a step further for cross-references with the game data
- to take user id 1087 as a potential candidate for promotion and gift. Create scenarios of potential reactions and analysis feedbacks and impacts.
- Extend to top 20, 30 most active users to find diverse users and test potential promotion.

List of Tables and Figures

Table.1: Game-data files overview	page 4-6
Table.2: Amount spent in total and number of unique items	8
Table.3: Top-3 buying users with platform and hit-ratio	13
Table.4: Number of samples for classification (decision tree)	13
Table.5: Attribute selection for classification (decision tree)	14-15
Table.6: Recommendations to increase revenue based on classification	19
Table.7: Attribute selection for clustering	20
Table.8: Training data set examples	20
Table.9: Cluster centers	21
Table.10: Features and prediction examples	24
Table.11: Recommendations to increase revenue based on clustering	24
Table.12: Schema of the chat data	26
Table.13: Chattiest users	30
Table.14: Chattiest teams	31
Table.15: Most Active Users based on Cluster Coefficients (among chattiest users)	33
Figure.1: in-game screenshot	page 2
Figure.2: Splunk Dashboard overview	3
Figure.3: Entity Relationship Diagram	6
Figure.4: Duration in hours per session, per user	7
Figure.5: Daily total amount of purchases and number of transactions	7-8
Figure.6: Number of purchases by items	8
Figure.7: Revenue by items	9
Figure.8: Total clicks per advertisement categories	10
Figure.9: Average time in hours per level	11
Figure.10: Hit ratio per user, per team	11
Figure.11: Users by countries, by continent	12
Figure.12: Top 10 users spending the most	12
Figure.13: KNIME - numeric binner	14
Figure.14: KNIME - partitioning	15
Figure.15: KNIME - decision tree (full) on test dataset	16
Figure.16: KNIME - decision tree (simplified) on train vs test dataset	16
Figure.17: KNIME - confusion matrix	17
Figure.18: KNIME - workflow overview	18
Figure.19: Cluster - 3D plot	22
Figure.20: Cluster - ad clicks vs total spent	22
Figure.21: Cluster - age vs total spent	23
Figure.22: Cluster - ad clicks vs age	23
Figure.23: Schema of graph data	25
Figure.24: Graph examples generated by loading data	27
Figure.25: Graph example - user nodes and team-chat-session nodes	28
Figure.26: Graph example - expanded child relationships from Figure.25	28
Figure.27: Graph example - the longest edge chain	30