

Improving Revenue Generation on Catch the Pink Flamingo

Technical Appendix

Table of Contents

Splunk Data Exploration.....	2
Data Set Overview.....	2
Aggregation of Revenue Data	4
Filtering Top 10 Users by Expenditure	5
KNIME Decision Tree Analysis.....	6
Data Preparation.....	6
Attribute Creation	6
Attribute Selection	7
Data Partitioning and Modeling.....	8
Evaluation	9
Analysis Conclusions	9
Spark MLLib K-means Cluster Analysis.....	11
Attribute Selection	11
Training Data Set Creation	12
Cluster Centers.....	12
Log file from cluster analysis.....	13
Cost evaluation of cluster analysis.....	14
Recommended Actions	14
Neo4j Graph Analysis of Chat Data	15
Modeling Chat Data using a Graph Data Model	15
Creation of the Graph Database for Chats.....	15
Longest conversation chain and its participants.....	17
Analyzing relationship between top 10 chattiest users and top 10 chattiest teams	19
Measuring activity in groups of users via Cluster Coefficient.....	21

Splunk Data Exploration

Data Set Overview

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

File Name	Description	Fields
ad-clicks.csv	Records of instances where a user click on an advertisement banner shown in-game	timestamp : date and time of click txld : unique transaction identifier assigned to the click userSessionId : identifier tag of user's game session during which ad was clicked teamId : identifier tag of user's team during which ad was clicked userId : identifier tag of user who clicked the ad adId : identifier tag assigned to the advertisement banner that was clicked adCategory : Category of the advertisement that was clicked.
buy-clicks.csv	Instances of user purchasing an in-game item to aid gameplay.	timestamp : date and time of purchase txld : unique transaction identifier assigned to the purchase userSessionId : identifier tag of user's game session during which purchase was made team : identifier tag of user's team during which purchase was made userId : identifier tag of user who made the purchase buyId : identifier tag of the in-game item price : price value of the in-game item at time of purchase
users.csv	Profile of all registered users	timestamp : date and time of user registration id : identifier tag of user nick : in-game name of user, publicly viewable twitter : twitter username of user, where applicable dob : date of birth of user country : user country of origin
team.csv	Profile of teams formed	teamId : identifier tag of a team name : in-game name of the team, publicly viewable

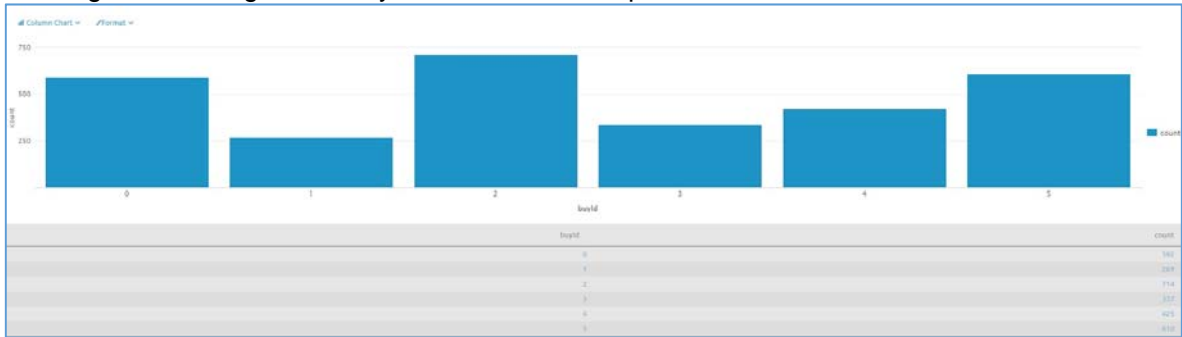
		<p>teamCreationTime: date and time at which team was formed.</p> <p>teamEndTime: date and time at which team was disbanded when last member leaves the team.</p> <p>strength: value of team strength, measured by rate of success during gameplay</p> <p>currentLevel: current game level at which team has progressed</p>
team-assignments.csv	Records of user being assigned to a team	<p>timestamp: date and time at which user was assigned to a team</p> <p>team: identifier tag of team which user was assigned to</p> <p>userId: identifier tag of user being assigned</p> <p>assignmentId: unique identifier tag of this assignment record</p>
level-events.csv	Records of lifecycle of game level for teams	<p>timestamp: date and time of event occurring for a particular level</p> <p>eventId: unique identifier tag for this event</p> <p>teamId: identifier of team involved in this event</p> <p>teamLevel: level value of this event</p> <p>eventType: type of event, i.e. either start (start of the game-level) or end (end of the game-level)</p>
user-session.csv	Records of lifecycle of game level for users	<p>timestamp: date and time of event for user at a level</p> <p>userSessionId: identifier tag of user's game session</p> <p>userId: identifier tag of user</p> <p>teamId: identifier of team which user belongs to</p> <p>assignmentId: team assignment identifier for the user</p> <p>sessionType: type of event, i.e. either start (start of the game-level) or end (end of the game-level)</p> <p>teamLevel: level value of this event</p> <p>platformType: device on which user played current session</p>
game-clicks.csv	Instance of every user game-click	<p>timestamp: date and time of recorded game-click</p> <p>clickId: unique identifier tag assigned to game-click</p> <p>userId: identifier tag of user who performed the game-click</p>

		userSessionId : identifier tag of gameplay session where game-click was performed isHit : binary value to indicate of game-click was carried out on a correct objective teamId : identifier tag of team which user belongs to when game-click was performed teamLevel : value of level at which game-click was performed
--	--	---

Aggregation of Revenue Data

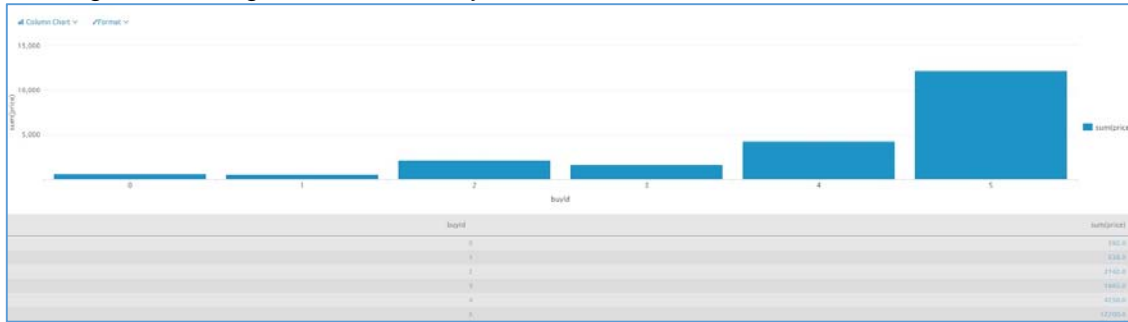
Amount spent buying items	\$21,407
# Unique items available to be purchased	6

A histogram showing how many times each item is purchased:



Query: `source="buy-clicks.csv" | stats sum(price) by buyId`

A histogram showing how much money was made from each item:



Query: `source="buy-clicks.csv" | stats sum(price) by build`

Filtering Top 10 Users by Expenditure

A histogram showing total amount of money spent by the top ten users (ranked by how much money they spent).



Query: `source="buy-clicks.csv" | stats sum(price) as "Total Expenditure ($)" by userid | sort - num("Total Expenditure ($")) | head 10`

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	11.6%
2	12	iphone	13.1%
3	471	iphone	14.5%

Query: `source="game-clicks.csv" (userId=2229 OR userId=12 OR userId=471) | stats sum(isHit) as hits, count as totalclicks by userId | eval "hit-ratio (%)" = hits / totalclicks * 100`

KNIME Decision Tree Analysis

Data Preparation

Analysis of combined_data.csv as carried out to investigate factors that may lead to classification of high-rollers.

Sample Selection

Item	Amount
Number of Samples	4619
Number of Samples with Purchases	1411

Attribute Creation

A new categorial attribute was created to enable analysis of players as broken into 2 categories (HighRollers and PennyPinchers). A screenshot of the attribute follows:

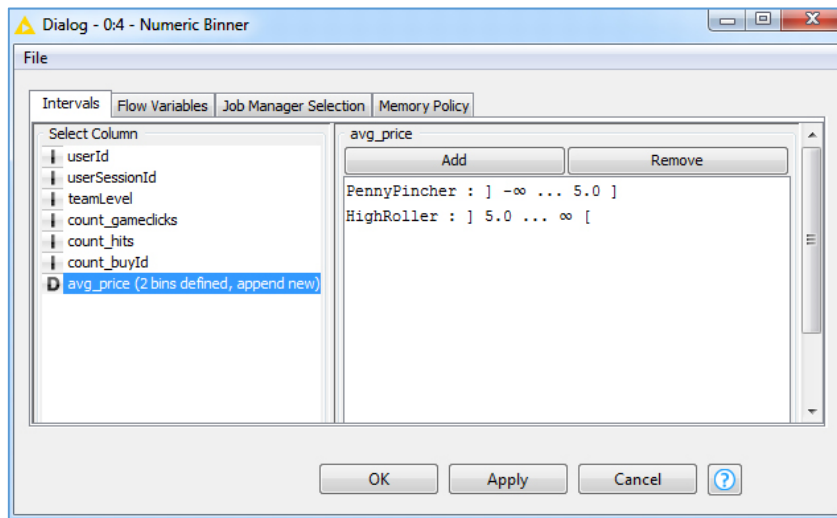
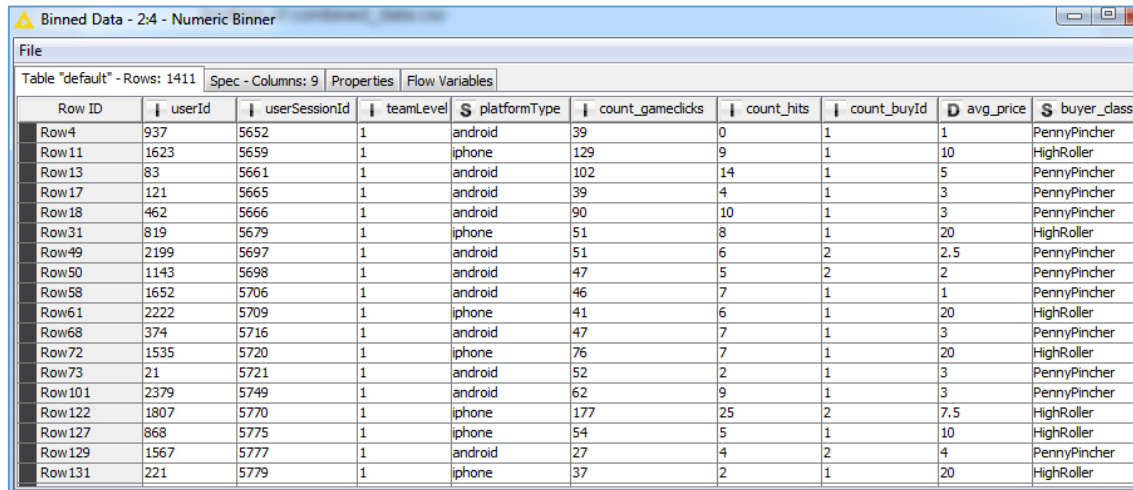


Fig 1: Configuring the Numeric Binner node



Row ID	userId	userSessionId	teamLevel	platformType	count_gamedicks	count_hits	count_buyId	avg_price	buyer_class
Row4	937	5652	1	android	39	0	1	1	PennyPincher
Row11	1623	5659	1	iphone	129	9	1	10	HighRoller
Row13	83	5661	1	android	102	14	1	5	PennyPincher
Row17	121	5665	1	android	39	4	1	3	PennyPincher
Row18	462	5666	1	android	90	10	1	3	PennyPincher
Row31	819	5679	1	iphone	51	8	1	20	HighRoller
Row49	2199	5697	1	android	51	6	2	2.5	PennyPincher
Row50	1143	5698	1	android	47	5	2	2	PennyPincher
Row58	1652	5706	1	android	46	7	1	1	PennyPincher
Row61	2222	5709	1	iphone	41	6	1	20	HighRoller
Row68	374	5716	1	android	47	7	1	3	PennyPincher
Row72	1535	5720	1	iphone	76	7	1	20	HighRoller
Row73	21	5721	1	android	52	2	1	3	PennyPincher
Row101	2379	5749	1	android	62	9	1	3	PennyPincher
Row122	1807	5770	1	iphone	177	25	2	7.5	HighRoller
Row127	868	5775	1	iphone	54	5	1	10	HighRoller
Row129	1567	5777	1	android	27	4	2	4	PennyPincher
Row131	221	5779	1	iphone	37	2	1	20	HighRoller

Fig 2: buyer_class column (right-most column of table) created with Numeric Binner node

A Numeric Binner node was used to create the new category entitled buyer_class, as shown in the 2 figures above. Categorical values 'PennyPincher' and 'HighRoller' were assigned using the avg_price values as a reference, i.e. records with average price being \$5.00 or lesser being classified as 'PennyPincher', while others with any average price value greater than \$5.00 will be classified as 'HighRoller'.

The creation of this new categorical attribute was necessary because it would serve as the reference vector (a.k.a model answer) for training and subsequently scoring the Decision Tree model.

Attribute Selection

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

Attribute	Rationale for Filtering
userId	This is an arbitrarily assigned identification tag for players in the game, and have no logical bearing on deciding whether a player is a penny-pincher or a high-roller.
userSessionId	This is yet another arbitrarily assigned tag to identify a particular gameplay session, and itself also has no bearing on deciding whether a player is a penny-pincher or a high-roller.
avg_price	Since this attribute is used to generate the new categorical attribute (a.k.a label) for testing and scoring of the Decision Tree model, itself cannot be included within the training and testing dataset, otherwise the Decision Tree would most likely be built solely on it in order to achieve 100% accuracy, which defeats the original objective of analysing player in-game behavior/host environmental factor that may give rise to his/her buyer classification.

Data Partitioning and Modeling

The data was partitioned into train and test datasets.

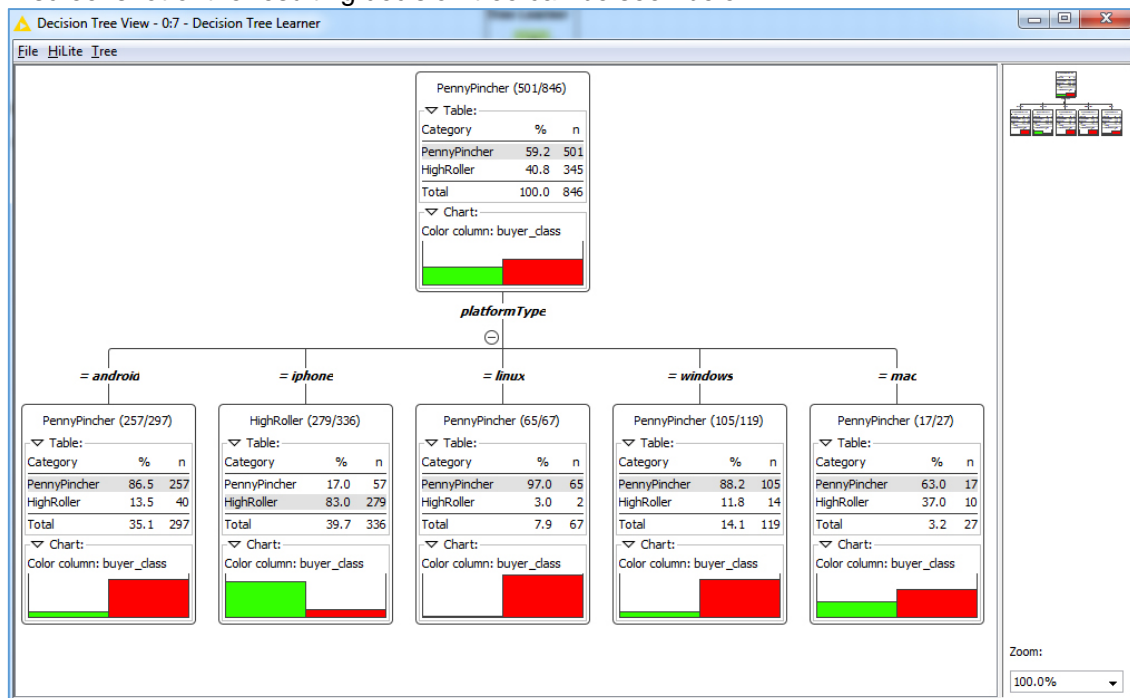
The **train** data set was used to create the decision tree model.

The trained model was then applied to the **test** dataset.

This is important because the train dataset consists of records with known class labels to facilitate the building of the classification model (in this case a Decision Tree), while the test dataset would contain records with unknown labels, thus serving as an unbiased means of evaluating the performance of the trained model.

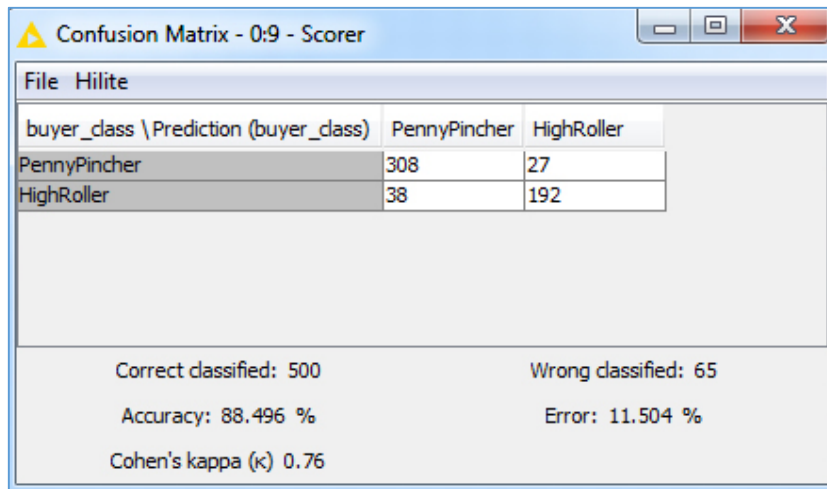
When partitioning the data using sampling, it is important to set the random seed because it is necessary to replicate the same partitioned datasets for repeated executions of the training and scoring process, where we may intend to evaluate the effects on performance based on different Decision Tree configurations, whilst keeping the partitions constant across these all runs.

A screenshot of the resulting decision tree can be seen below:



Evaluation

A screenshot of the confusion matrix can be seen below:



buyer_class \ Prediction (buyer_class)	PennyPincher	HighRoller
PennyPincher	308	27
HighRoller	38	192

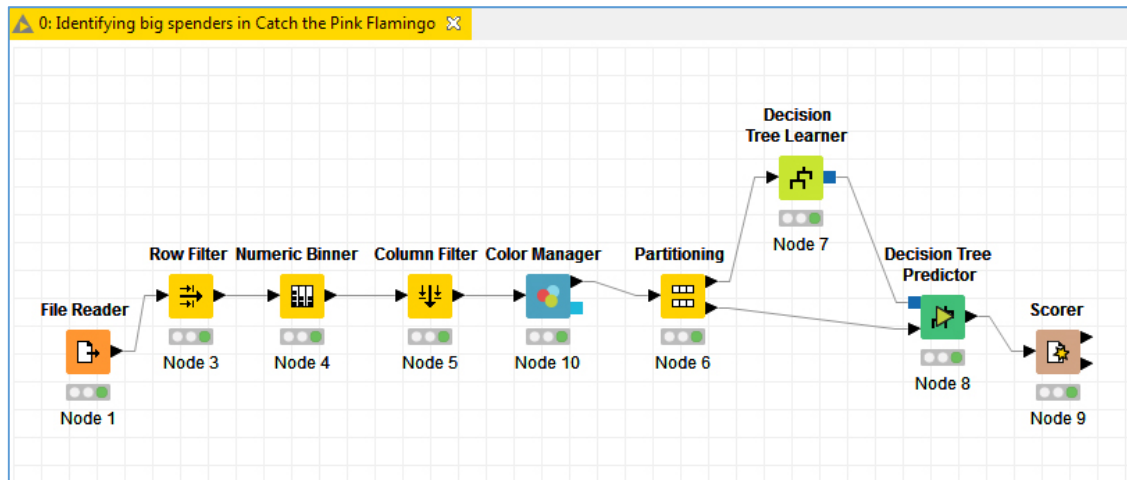
Correct classified: 500 Wrong classified: 65
 Accuracy: 88.496 % Error: 11.504 %
 Cohen's kappa (κ) 0.76

As seen in the screenshot above, the overall accuracy of the model is 88.5%

For PennyPincher, 308 records were correctly classified, while 27 were erroneously evaluated to be HighRollers. On the other hand, 192 HighRollers were correctly classified, while 38 were wrongly evaluated as PennyPinchers.

Analysis Conclusions

The final KNIME workflow is shown below:



Based on the Decision Tree that was formulated, players using iOS device has been classified as HighRollers, while players of other platforms have been classified as PennyPinchers.

Specific Recommendations to Increase Revenue
1. Eglence Inc. could consider apportioning more of their marketing budget to promote the game on the iOS platform. This could lead to an increase in iOS users picking up the game and lead to increase revenue as they exhibit a high likelihood, compared to players on other platforms, of being highrollers.
2. Provide better rewards for iOS players to write a review for the game, so as to incentivise them to submit favourable reviews, which would improve the ranking of the game on the iOS app-store, and potentially lead to a greater player base.

Spark MLLib K-means Cluster Analysis

Attribute Selection

Attribute	Rationale for Selection
Average buy-click count	<p>Every in-game purchase made by users is recorded as a buy-click. Over time, as users make more purchases whilst making their way through the game levels, the average number of buy-clicks per level can be aggregated. This attribute measures a user's propensity to make an in-game purchase as they make progress through the game.</p> <p>Count all the buy-clicks for a user, then divide it by the number of sessions played.</p>
Average ad-click count	<p>Same as the in-game purchase, every ad-click is recorded, and we can subsequently monitor the propensity of a user clicking on an advertisement banner as they play the game. Over time, some users would click ads more often than others, thus it serves as a metric for comparison.</p> <p>Count all the ad-clicks for a user, then divide it by the number of sessions played.</p>
Average expenditure	<p>There is a range of in-game items for users to purchase. As such, we can study the average expenditure per level. This would provide us with an opportunity to study highrollers against those who may not spend at all.</p> <p>Sum up all the item-prices paid by a user, then divide it by the number of sessions played.</p>

Training Data Set Creation

The training data set used for this analysis is shown below (first 10 lines):

avgPurchaseCount	avgAdClicks	avgExpPerSession
0	0	0
0.714285714	5.142857143	8.571428571
0	0	0
0	0	0
0	0	0
1.142857143	4.857142857	3.714285714
2	6.5	31.5
1	5	6.5
1.5	5.666666667	3.333333333
0	0	0

Dimensions of the training data set (rows x columns): **1091** rows by **3** columns

Number of clusters created: **3**

Cluster Centers

Cluster #	Cluster Center
1	Average buy-clicks: 0.02 Average expenditure: 0.03 Average ad-clicks: 0.23 <i>Cluster size: 589; 54.0% of dataset</i>
2	Average buy-clicks: 0.94 Average expenditure: 4.30 Average ad-clicks: 5.83 <i>Cluster size: 390; 35.7% of dataset</i>
3	Average buy-clicks: 1.58 Average expenditure: 20.19 Average ad-clicks: 5.71 <i>Cluster size: 112; 10.3% of dataset</i>

These clusters can be differentiated from each other as follows:

Cluster 1 (a.k.a *freeloaders*) has captured majority of the users that essentially play the game without spending at all, and typically clicks only 1 advertisement in every 4-5 games. They provide the base-line behavior of users in-game.

Cluster 2 (a.k.a *penny-pinchers*) is characterised as users who makes close to 1 purchase per level, with expenditure of around \$4.30 within a level. They also display a propensity to click on close to 6 advertisements per level, which is a much higher rate than the *freeloaders*.

Cluster 3 (a.k.a *high-rollers*) is different from the penny-pinchers cluster, making an average of close to 1.58 purchases per level (68% more than *penny-pinchers*), and spending almost \$20 per level (470% more than *penny-pinchers*). While ad-click rate is just slightly lower by 2% than *penny-pinchers*, they also click close to 6 advertisements per level, a much higher rate than the *freeloaders*.

[Log file from cluster analysis](#)

Highlighted text indicates details of the model selected for evaluation.

```
[SAMPLE DATASET]
[array([ 0., 0., 0.]), array([ 0.71428571, 8.57142857, 5.14285714]), array([ 0., 0., 0.]), array([ 0., 0., 0.]), array([ 0., 0., 0.]),
array([ 1.14285714, 3.71428571, 4.85714286]), array([ 2., 31.5, 6.5]), array([ 1., 6.5, 5.]),
array([ 1.5, 3.33333333, 5.66666667]), array([ 0., 0., 0.])]
```

```
[TRAIN DATASET SHAPE]
(1091, 3)
```

```
[K]=1
[CENTER(S)]
[array([ 0.50749203, 3.62997822, 2.79677448])]
[COST]=59925.05543342645
[CLUSTER SIZES]
dict_items([(0, 1091)])
```

```
[K]=2
[CENTER(S)]
[array([ 1.37703252, 17.03899811, 5.90320122]), array([ 0.35365747, 1.25772443, 2.24720167])]
[COST]=23212.28217540738
[CLUSTER SIZES]
dict_items([(0, 164), (1, 927)])
```

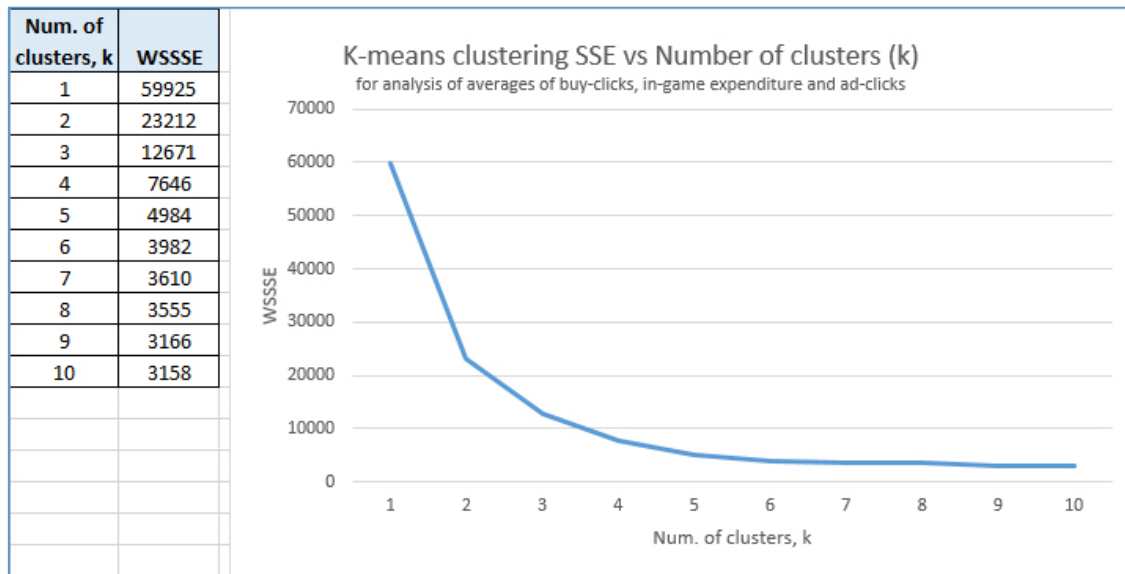
```
[K]=3
[CENTER(S)]
[array([ 0.93948718, 4.30493547, 5.8309768 ]), array([ 1.57476616, 20.18807802, 5.71264881]),
array([ 0.01850594, 0.03449349, 0.23324844])]
[COST]=12671.3123894749
[CLUSTER SIZES]
dict_items([(0, 390), (1, 112), (2, 589)])
```

```
[K]=4
[CENTER(S)]
[array([ 1.16061224, 11.48923622, 5.95703231]), array([ 0.01357597, 0.02816291, 0.15800116]),
array([ 1.84887218, 25.41132331, 5.46215539]), array([ 0.87687397, 2.79847867, 5.72487983])]
[COST]=7645.50590694292
[CLUSTER SIZES]
dict_items([(0, 140), (1, 577), (2, 57), (3, 317)])
```

```
[K]=5
[CENTER(S)]
[array([ 0.83032237, 2.41210121, 5.65188594]), array([ 1.14658163, 9.47171122, 6.06686224]),
array([ 0.01146922, 0.02439024, 0.14227642]), array([ 1.54826531, 19.3221017, 5.38472789]),
array([ 2.28541667, 35.36238095, 6.1639881 ])]
[COST]=4983.973681127357
[CLUSTER SIZES]
dict_items([(0, 291), (1, 141), (2, 574), (3, 69), (4, 16)])
```

<Log truncated for brevity. $k=6$ onwards not included here>

Cost evaluation of cluster analysis



$k=3$ picked for evaluation of cluster analysis.

Recommended Actions

Action Recommended	Rationale for the action
Assign higher value ads to users who make at least 1 purchase per game.	<p>The cluster analysis reveal that users who make at least 1 in-game purchase per level will have a tendency to click up close to 6 ads per level. This could optimise the revenue generated from 'premium-rate' ads.</p> <p>The 'common-rate' ads can be assigned to the freeloaders.</p>
Bundle multiple in-game purchases at a slight discount.	<p><i>Higher-rollers</i> who make close to 1.5 purchases per level tend to spend much more than the <i>penny-pinchers</i> (470%, i.e. almost 4 to 5 times more in the long run), thus a slight-discount on bulk purchases can entice this group continue with this behavior in the long term (perhaps maybe spend even more), or encourage the penny-pinchers to spend more. This would also lead to greater revenue generated from in-game purchases.</p>

Neo4j Graph Analysis of Chat Data

Modeling Chat Data using a Graph Data Model

The graph model illustrates the dynamics of chat interaction between ‘Catch the Flamingo’ users. A user of a team can create a chat-session, where other users on the same team are free to join and subsequently leave. Interaction is initiated when a user creates a post (also known as chat-item), to which other users can reply to with their own chat-item, or simply mention it as a quick reply. Such posts are linearly linked to one another as a response, and all of these posts are part of the same team chat-session. All relationships between entities of this graph model are universally marked with a timestamp, while the entities are identified by a unique ID value.

Creation of the Graph Database for Chats

The following table shows the CSV data files that were utilized for graph analysis.

CSV file	Field	Description
chat_create_team_chat	user.id	User ID value
	team.id	Team ID value
	teamchatsession.id	ID value assigned to the chat-session
	timestamp	Date/time of creation of chat-session and ownership assignment of chat-session to team.
chat_join_team_chat	user.id	User ID value
	teamchatsession.id	Team chat-session ID value
	timestamp	Date/time of user joining team chat-session
chat_leave_team_chat	user.id	User ID value
	teamchatsession.id	Team chat-session ID value
	timestamp	Date/time of user leaving team chat-session
chat_item_team_chat	user.id	User ID value
	teamchatsession.id	Team chat-session ID value
	chatitem.id	Post ID value

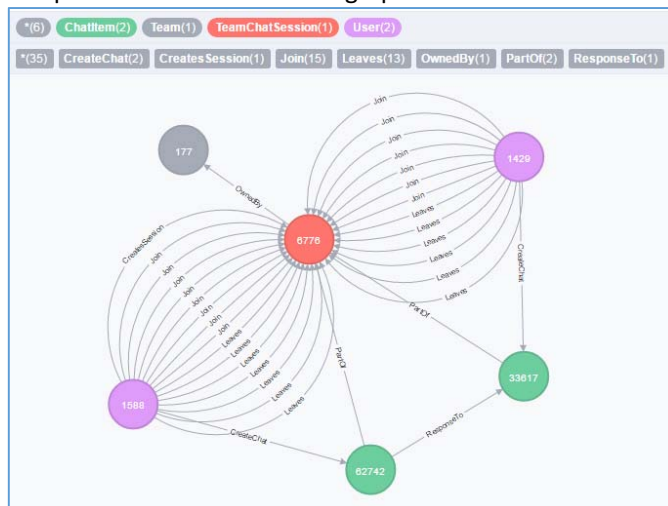
	timestamp	Date/time of post creation by user and linking of post to team chat-session.
chat_mention_team_chat	chatitem.id	Post ID value
	user.id	User ID value
	timestamp	Date/time of user mentioning a particular post as a quick reply.
chat_respond_team_chat	chatitem.id	Post -1 ID value
	chatitem.id	Post-2 ID value
	timestamp	Date/time of post-1 created as a response to post-2

When loading a CSV file, each row is parsed for extracting ID values to create nodes, and timestamp values to create edges. Consider the following load command:

```
LOAD CSV FROM "file:/chat_create_team_chat.csv" AS row
MERGE (u:User {id: toInt(row[0])})
MERGE (t:Team {id: toInt(row[1])})
MERGE (c:TeamChatSession {id: toInt(row[2])})
MERGE (u)-[:CreatesSession{timeStamp: row[3]}]->(c)
MERGE (c)-[:OwnedBy{timeStamp: row[3]}]->(t)
```

1st line loads the csv file from the specified location and parses each line as a 'row' variable. 2nd, 3rd and 4th lines create User, Team and TeamChatSession nodes, with respective ID values extracted from the 'row' tuple using index values in sequential notation. 4th and 5th lines create the CreatesSession and OwnedBy edges with respective timestamp values extracted in similar manner, to link up the nodes. The MERGE command used to create the node/edge if it does not exist in the graph, or merge the attributes into already existing nodes/edges.

Sample screenshot of created graph:



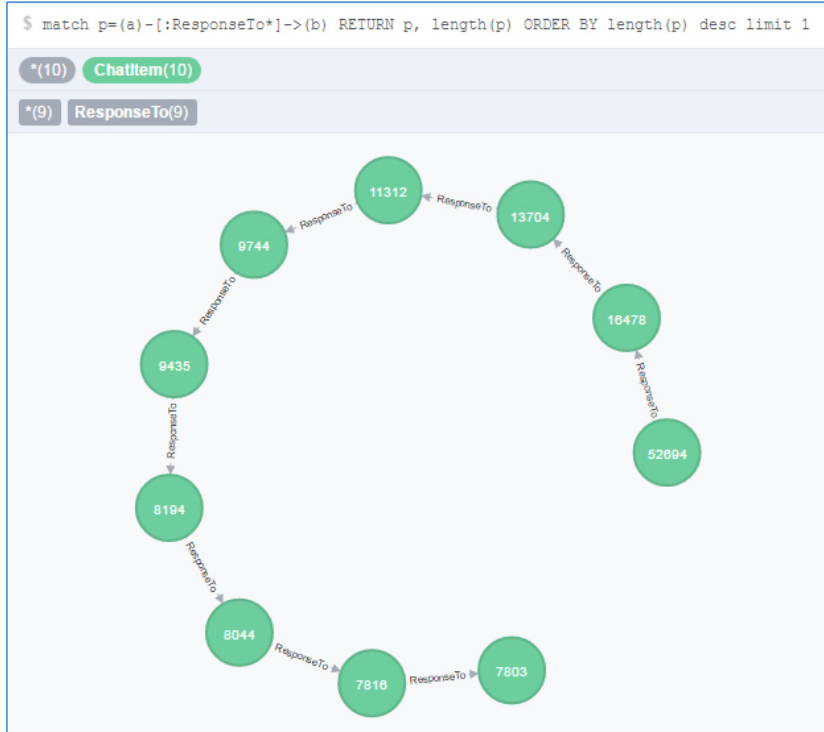
Longest conversation chain and its participants

Longest conversation chain has path length of 9, i.e. **10 Chatitem nodes**.

Query:

```
match p=(a)-[:ResponseTo*]->(b)
RETURN p, length(p)
ORDER BY length(p) desc limit 1
```

Graph:

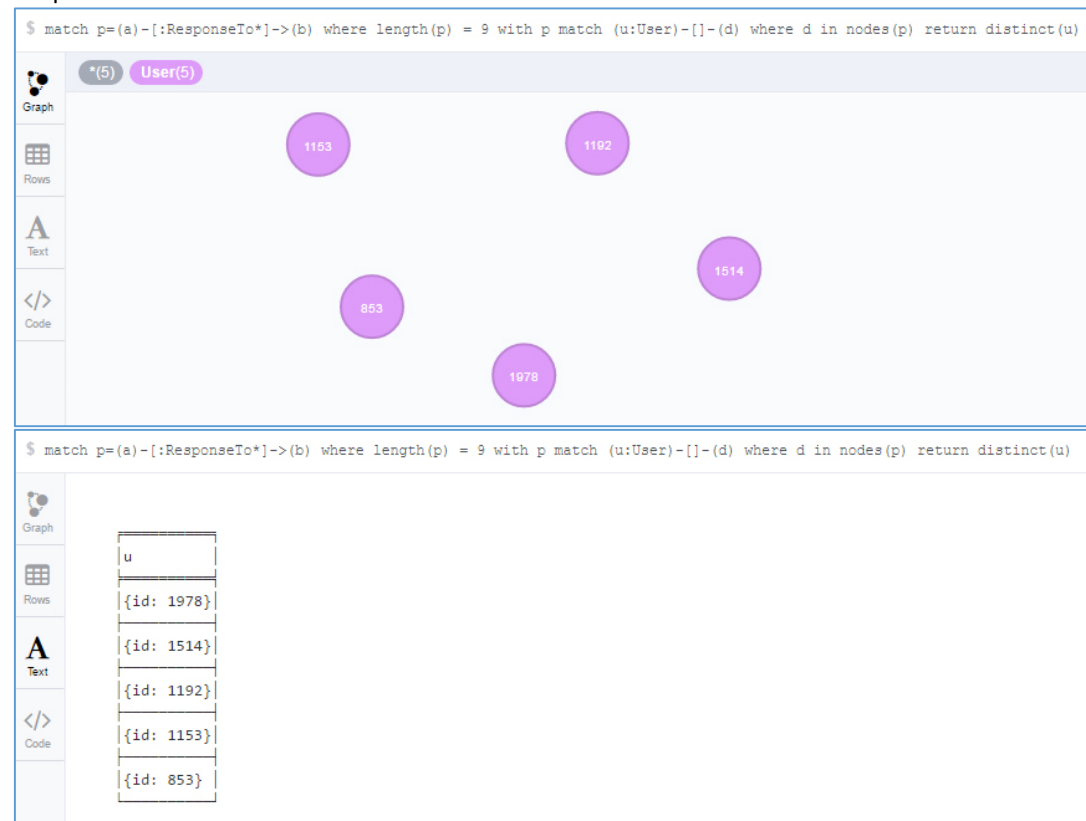


Now that the longest path has been determined to be 9, a subsequent query is used to find all the distinct users related to the 10 ChatItem nodes that are part this path. **5 unique users** were extracted.

Query:

```
match p=(a)-[:ResponseTo*]->(b)
where length(p) = 9
with p
match (u:User)-[]-(d)
where d in nodes(p)
return distinct(u)
```

Graph:



Analyzing relationship between top 10 chattiest users and top 10 chattiest teams

Match for directional CreateChat relationship from User node to ChatItem node, then count the relationship links between the 2 mentioned nodes and subsequently return the sort list of user IDs, together with number of CreateChat edges (a.k.a out-degrees) that stem from them.

Query:

```
MATCH (u:User)-[r:CreateChat]->(ci:ChatItem)
RETURN u.id as node, count(r) as Outdegree
ORDER BY Outdegree desc LIMIT 10
```

Graph:

\$ MATCH (u:User)-[r:CreateChat]->(ci:ChatItem) RETURN u.id as node, count(r) as Outdegree ORDER BY Outdegree desc LIMIT 10

node	Outdegree
394	115
2067	111
209	109
1087	109
554	107
516	105
1627	105
999	105
668	104
461	104

Chattiest Users

Users	Number of Chats
394	115
2047	111
209*	109
1087*	109

*A tie for 3rd placing between user 209 and 1087, both with 109 chatItems

To find the chattiest teams, we match for all PartOf relationships linking ChatItem to TeamChatSession, where the latter is subsequently linked to a Team via OwnedBy relationship. Again we count the PartOf relationships for every TeamChatSession, isolate the top 10 with the highest numbers, then discover the Teams through the OwnedBy relationship.

Query:

```
MATCH (ci:ChatItem)-[po:PartOf]->(tcs:TeamChatSession)-[o:OwnedBy]->(t:Team)
RETURN t.id as team, count(po) as Outdegree
ORDER BY Outdegree DESC LIMIT 10
```

Graph:

\$ MATCH (ci:ChatItem)-[po:PartOf]->(tcs:TeamChatSession)-[o:OwnedBy]->(t:Team) RETURN t.id as team, count(po) as Outdegree ORDER BY Outdegree DESC LIMIT 10

team	Outdegree
82	1324
185	1036
112	957
18	844
194	836
129	814
52	788
136	783
146	746
81	736

Chattiest Teams

Teams	Number of Chats
82	1324
185	1036
112	957

Modify the query for top 10 chattiest users to trace the users back to their respective team. This facilitates the investigation of whether the top 10 chattiest users reside within the top 10 chattiest teams.

Query:

```
MATCH (u:User)-[r>CreateChat]->(ci:ChatItem)-[po:PartOf]->(tcs:TeamChatSession)-
[o:OwnedBy]->(t:Team)
RETURN u.id as node, count(r) as Outdegree, t.id as team
ORDER BY Outdegree desc LIMIT 10
```

Graph:

\$ MATCH (u:User)-[r>CreateChat]->(ci:ChatItem)-[po:PartOf]->(tcs:TeamChatSession)-[o:Own]

node	Outdegree	team
394	115	63
2067	111	7
209	109	7
1087	109	77
554	107	181
999	105	52
516	105	7
1627	105	7
461	104	104
668	104	89

As shown above, with the exception of user ID:999 (6th chattiest user) belonging to Team ID:52 (7th chattiest team), the other 9 users hail from teams that are not part of the top 10 chattiest teams list. Hence, it can be concluded the chattiest users do not usually hail from the chattiest teams.

Measuring activity in groups of users via Cluster Coefficient

Activity is defined as user interacting with each other, i.e. 2 user nodes connected to each other via an edge for this definition to hold true. Given the present graph, we can connect 2 users in this manner based on either of the 2 following conditions:

1. User A is interacting with User B if A creates a ChatItem that B mentions, and both A and B are not referring to the same person. The following query creates the edge fulfilling this condition:

```
MATCH (u1:User)-[cc>CreateChat]->(ci:ChatItem)-[m:Mentioned]->(u2:User)
WHERE u1.id <> u2.id
MERGE (u1)-[:InteractsWith]->(u2)
```

2. User A is interacting with User B if A creates a ChatItem that is in response to a ChatItem that B has created on, and both A and B are not referring to the same person. The following query creates the edge fulfilling this condition:

```
MATCH (u1:User)-[cc1>CreateChat]->(ci1:ChatItem)-[rt:ResponseTo]->(ci2:ChatItem)-[cc2>CreateChat]->(u2:User)
WHERE u1.id <> u2.id
MERGE (u1)-[:InteractsWith]->(u2)
```

Interactivity between 2 users is true if at any one of the above condition has been fulfilled at least once, hence the MERGE command is to ensure that at most 1 *interactsWith* edge can be formulated between 2 separate User nodes.

Density of activity can be measured by the clustering coefficient of User nodes. For a User node x with outgoing edges connected to k other User nodes, these k nodes are known as x 's neighbours, and the clustering coefficient of x is defined as the ratio of the total count of directed *InteractsWith* edges amongst neighbours themselves and the maximum possible number directed edges that could exists amongst themselves (which is $(k * (k-1))$ as the denominator).

Note:

A ratio value denotes that it can be between 0 and 1. A value of 0 means that either a node x has no neighbours, or only 1 neighbour, or have at least 2 neighbours but every neighbour is not connected to each other at all. A value of 1 means that a node x has at least 2 neighbours and that every neighbour is bi-directionally connected to each other.

Based on the formula described above, the following query would calculate the Cluster Coefficient of a targeted node with id=123:

```
match (u1:User { id: 123 })-[iw1:InteractsWith]->(u2:User)
with collect(u2.id) as neighbours, count(u2) as k
match (u3:User)-[iw2:InteractsWith]->(u4:User)
where (u3.id in (neighbours)) and (u4.id in (neighbours)) and (u3.id <> u4.id)
with count(iw2) as numerator, (k * (k - 1) * 1.0) as denominator
return numerator/denominator as clusterCoefficient
```

The above query is subsequently used to calculate the Cluster Coefficient for the top 3 chattiest users.

Most Active Users (based on Cluster Coefficients)

User ID	Coefficient
394	0.75
2047	0.92
209*	1
1087*	0.73

*Users 209 and 1087 shared 3rd placing for top chattiest user, as shown in the earlier table, thus both are considered for investigation here