Technical Appendix

By C.G.

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 (by Christina G)	**Description** **Note: All files below are connected with the app game, *Catch the Pink Flamingo*…**	**Fields**
ad-clicks.csv	*Contains an added new line whenever a player clicks on an advertisement in the app.*	**timestamp**: Time marked for whenever a click occurred. **txId**: A unique ID (within ad-clicks.log) for a click. **userSessionid**: The user session ID, for a user who made a click. **teamid**: The current team's ID, for a user who made a click. **userid**: The user ID, for a user who made a click. **adId**: The clicked-on advertisement's ID. **adCategory**: The category/type of the clicked-on advertisement.

buy-clicks.csv	*Contains an added new line whenever a player makes an in-app purchase.*	**timestamp**: The time marked for when the purchase was made. **txId**: A unique Id (within buy-clicks.log) for a purchase. **userSessionid**: The user session ID, for a user who made a purchase. **teamid**: The current team ID, for a user who made a purchase. **userId**: The user ID, for a user who made a purchase. **buyId**: The purchased item's ID. **price**: The purchased item's cost.
users.csv	*Features a line for every user who plays the game.*	**timestamp**: The time marked for when the user first played the game. **userId**: A user's assigned ID. **nick**: A nickname, selected by the user. **twitter**: A user's Twitter handle. **dob**: A user's date of birth. **country**: A two-letter country code, noting where the user lives.

team.csv	*Features a line for each team terminated in the game.*	**teamId**: The team's assigned ID. **name**: The name, selected by the team. **teamCreationTime**: The timestamp of when the team was formed. **teamEndTime**: The timestamp of when the last member left the team. **strength**: A measure of team strength. This roughly correlates to a team's success rate. **currentLevel**: The team's current level.
team-assignments.csv	*Contains an added new line whenever a user joins a team.* *Note: A user can belong – at most - to a single team at any given time.*	**timestamp**: The time marked for when a user joined the team. **teamid**: The current team ID. **userid**: The user ID. **assignmentId**: A unique Id, for this assignment.
level-events.csv	*Contains an added new line whenever a team starts or finishes a game level.*	**timestamp**: The time marked for when an event occurred. **eventid**: An event's unique ID. **teamid**: A team's unique ID. **teamLevel**: A team's level, started or completed in an event. **eventType**: The type of event – either "start" or "end".

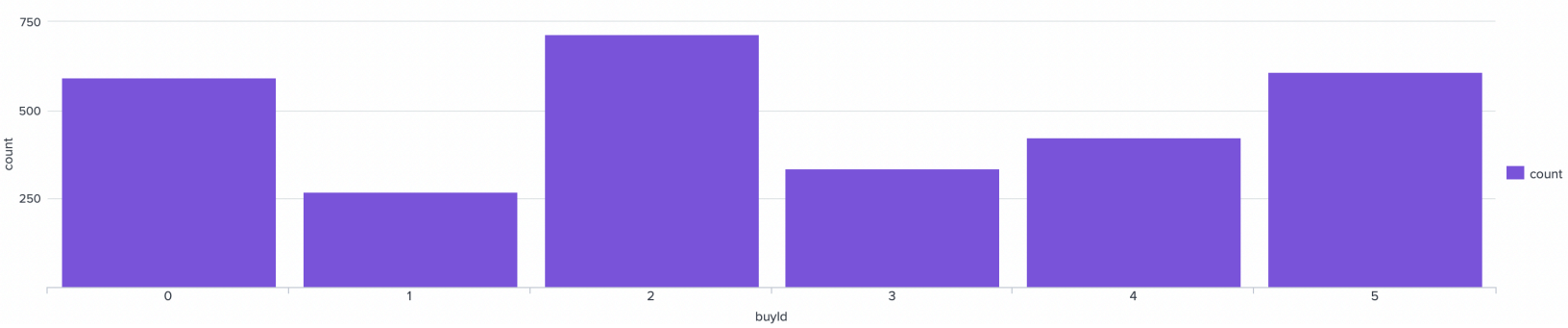
user-session.csv	*Features a line for each user session. A **user session** denotes when a user starts and stops playing the game.* *Note: Whenever a team advances to a new level, each players' user session is automatically ended, and a new user session begins.*	**timestamp**: The time marked for when an event occurred. **userSessionid**: A unique ID for the user session ID. **userId**: The current user's ID. **teamId**: The current team's ID. **assignmentId**: The team assignment Id, for the user to the team. **sessionType**: Denotes whether the event is at the start or end of a session. **teamLevel**: A team's level, during a session **platformType**: The type of platform a user is on during a session.
game-clicks.csv	*Contains an added new line whenever a user makes a game click.*	**timestamp**: Time marked for whenever a click occurred. **clickId**: A unique ID for a click. **userId**: The user ID, for a user who made a click. **userSessionid**: The user session ID, for a user who made a click. **isHit**: Denotes if click hit a flamingo (value is 1) or not (value is 0) **teamId**: The current team's ID, for a user who made a click. **teamLevel**: the current level of the user's team.

Aggregation

Amount spent buying items (by Christina G)	$21,407

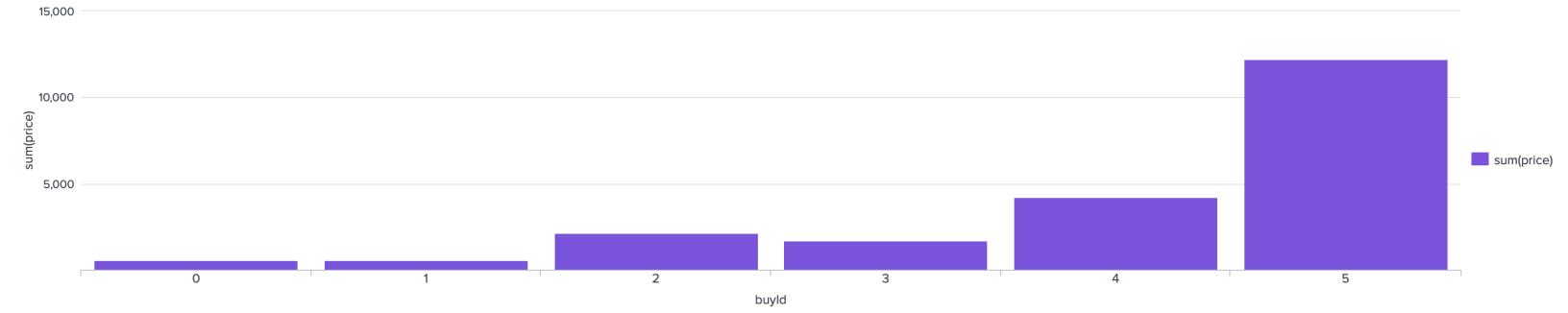
Number of unique items available to be purchased	6

A histogram showing how many times each item is purchased:



In the game, buyID 2 was the most purchased item, while buyID 1 was the least purchased item.

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



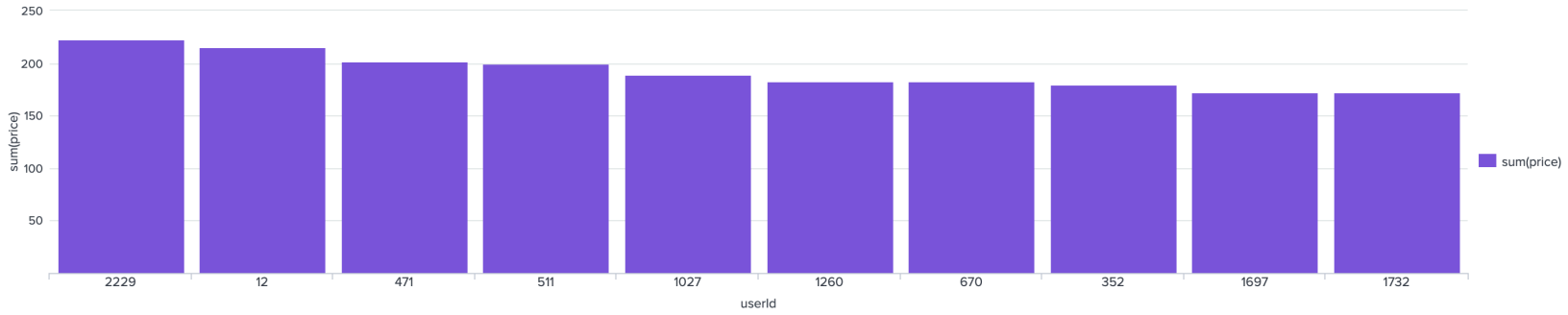
The most profitable item was buyID 5. The least profitable item was buyID 1, closely followed behind by buyID 0.

After analyzing both histograms above, it is clear that buyID 1 is the least popular item. The game makes least profit from buyID 1, and the item is purchased the least. In contrast, buyID 0 is purchased frequently, likely because it may be the cheapest item in the game. Meanwhile, buyID 5 is a popular item despite its high price tag. Although there are exceptions, the more popular an item is, the greater the revenue will be.

Note that price doesn't necessarily correlate with popularity for all items. For instance, buyID 2 was the most purchased item, yet it made the game little profit. This may be because buyID 2 could correspond to a helpful game boost that players use frequently, but it is not necessarily an expensive item.

Filtering

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



In the histogram above, users' unique IDs are used to determine how much money they have spent in a game. When a user purchases an item, the purchase is linked to the user's unique ID. The top ten users above have spent the most amount of money in the game compared to other players. The top 3 users, in order of highest spender, are: user 2229, user 12, and user 471.

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

Rank (by Christina G)	**User Id**	**Platform**	**Hit-Ratio (%)**
1	2229	iPhone	11.60%
2	12	iPhone	13.07%
3	471	iPhone	14.50%

Based on the table above, the more money a player spends, the lower the Hit-Ratio the Player may have. This is likely because purchases may award game players an advantage in the game that can boost their health or hit other players in battle. It is also worth noting that the top three players all have iPhones. iPhones may give the top players an advantage over other mobile platform users. It also may be the case that those who can afford an iPhone can also afford to make more game purchases compared to other platform users.

Query Examples

Number of items available for purchase:

There were six game items available for purchase. The most purchased item was buyID 2. The least purchased item was buyID 1.

```
source="buy-clicks.csv" | stats count by buyId
```

buyId	count
0	592
1	269
2	714
3	337
4	425
5	610

Most profitable items:

Of the six items available for purchase, buyID 5 was the most profitable while buyID 1 was the least profitable.

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

buyId	sum(price)
0	592.0
1	538.0
2	2142.0
3	1685.0
4	4250.0
5	12200.0

***Top three buying users*:**

The user IDs of the top 3 players was found in the previous query. This user ID is the same across different game files and can help us learn the player's mobile platform type and hit-ratio. The following queries are for the top purchasing user, 2229. The same queries can be applied for other users simply by changing the userId field to another user's ID.

- *Hit-Ratio*

```
source="game-clicks.csv"  userId=2229 | stats avg(isHit)
```

- *Platform*

```
source="user-session.csv"  userId=2229 | stats count by platformType
```

Data Preparation

Analysis of combined_data.csv

By C.G.

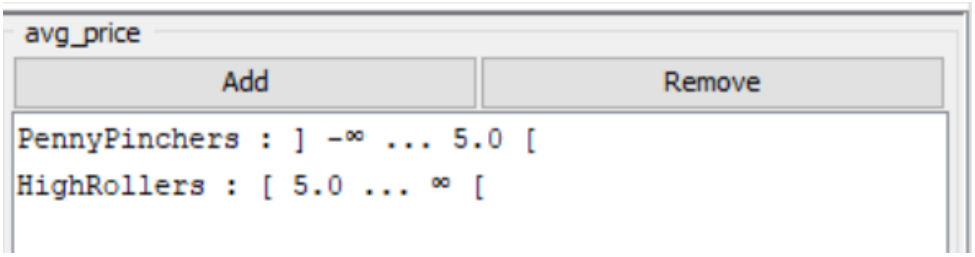
Sample Selection

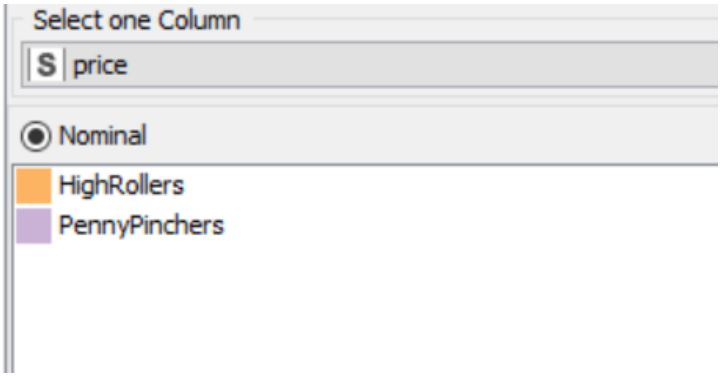
Item	Amount
# of Samples	4,619
# of Samples with Purchases	1,411

Attribute Creation

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

Creation of "price" attribute, by partitioning the "avg_price" variable…





Categorical Attribute Rationale

For the categorical price attribute, I divided the users into two categories - high spenders (a.k.a., "HighRollers") and low spenders (a.k.a., "PennyPinchers.") I used two colors visually

distinctive from one another – a warm tone vs a cool tone - to avoid possible confusion when these spenders are displayed in a graph.

The "HighRollers" are denoted with a warm orange – since they burn through their money, while the PennyPinchers are denoted with a cool lavender – since they have an ice-grip on their spending habits.

The creation of this new categorical attribute was necessary because… *we are trying to predict which users are likely to fit in one of the following spending categories – "HighRollers" (who spend more than five dollars) or "PennyPinchers" (who spend five dollars or less).*

Attribute Selection

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

Attribute	**Rationale for Filtering**
userSessionId	With "users" is our target variable, it is best to disregard userSessionId. Both variables highly correlate with one another and will thus weaken the strength of our conclusions between users and their spending habits.
count_gameclicks	When reading in the file, a quick statistics check reveals this variable relatively follows the uniform distribution across all values. For this reason, we can conclude that we will not likely find any more information about this relatively constant variable.
count_hits	Similar to "count_gameclicks", this variable follows the uniform distribution. Further analysis will likely not reveal more information about this variable.
avg_price	Since "avg_price" is already replaced by its categorical version called "price", "avg_price" is no longer needed. Including "avg_price" would result in the tree essentially memorizing the data to predict "price", since both variables are essentially the same.

Data Partitioning and Modeling

By C.G.

The data was partitioned into train and test datasets.

The *train set (60 percent of the original* data set) was used to create the decision tree model.

The *train* model was then applied to the *remaining 40% of the data, to create the test* dataset.

This is important because… *the train dataset is used to make predictions on test dataset. Only the training data has the modifications from prepping the data. That way, the final model is not a tailored to the test data. This final model accounts for differences between the data it is shown, since it has no prior knowledge of the testing data and can thus make viable predictions on other unseen data as well.*

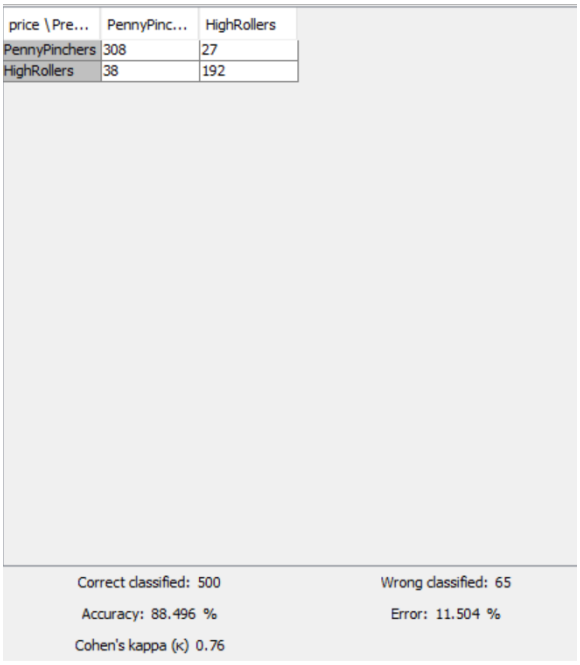
When partitioning the data using sampling, it is important to set the random seed because… *a random seed allows the experiment to be repeated and replicated with the same results each time.*

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

Evaluation

By C.G.

A screenshot of the confusion matrix can be seen below:



price \ Pre...	PennyPinc...	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

As seen in the screenshot above, the overall accuracy of the model is 88.496%!

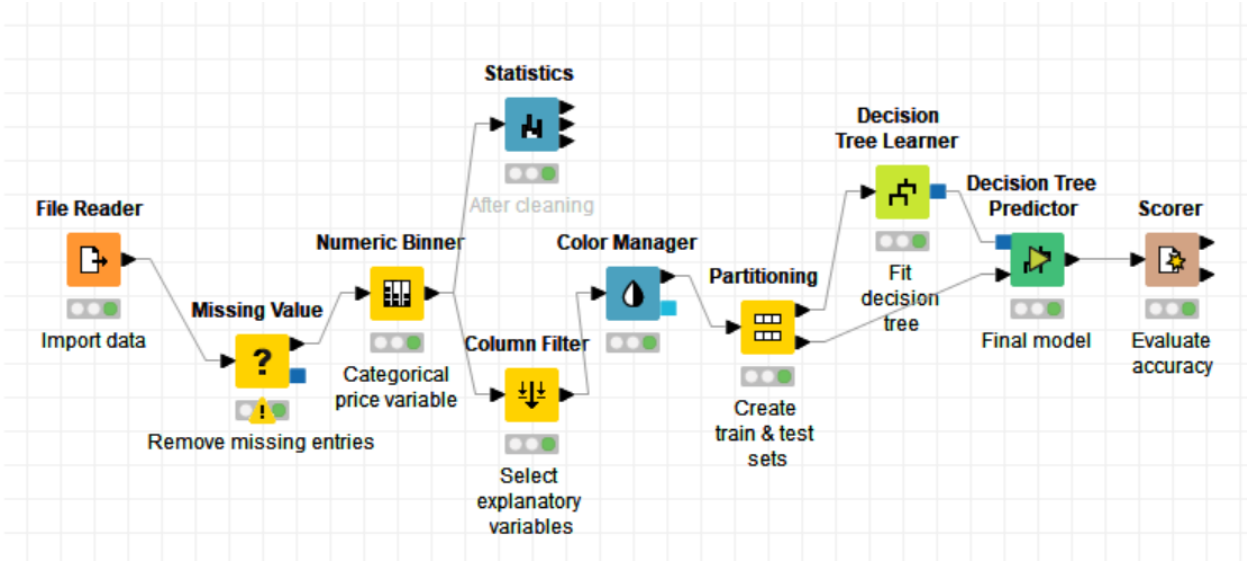
According to the confusion matrix above, out of 335 users who are actually PennyPinchers, 308 were correctly classified as PennyPinchers; the remaining 27 were misclassified as HighRollers. Out of the 230 users who are actually HighRollers, 192 were correctly classified as HighRollers; the remaining 38 were misclassified as PennyPinchers.

In sum, the decision tree correctly classifies users with an 88.496% accuracy, with 500 users correctly classified and 65 users misclassified.

Analysis Conclusions

By C.G.

The final KNIME workflow is shown below:



What makes a HighRoller vs. a PennyPincher?

Based on the model, the platform users play on strongly predicts whether a user is a Highroller or a PennyPincher; the evidence is in the decision tree's five root nodes that divide users by the platform they use. In general, HighRollers tend to be iPhone users. PennyPinchers, on the other hand, tend to be any other platform user – especially Linux users.

Specific Recommendations to Increase Revenue
1. **Create ads based on platform type to give users incentives to make more purchases.** These ads could be more frequent among Linux users, for example, since these users contain the most amount of PennyPinchers (based on percentage).
2. **Make purchases have greater power in the game.** For instance, if a purchase normally grants the user one leveling boosting point, make it three. By investing more purchasing power into the game, users may feel like they are making a valuable purchase and thus spend more.

Attribute Selection

By C.G.

features_used = [“totalAdClicks”, “totalBuyClicks”, “revenue”]

Attribute	**Rationale for Selection**
userId (index)	Both files have this attribute in common. This unique ID can help track users’ actions, from the ads they watched to the purchases they make.
totalAdClicks	The total number of ads each user clicked on in the game.
totalBuyClicks	The total number of purchases each user made in the game.
revenue	This is the renamed “price” attribute. This captures the total profit the game made.

Training Data Set Creation

By C.G.

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

```
trainingDF = combinedDF[["totalAdClicks", "totalBuyClicks", "revenue"]]
trainingDF.head(n=5)
```

	totalAdClicks	totalBuyClicks	revenue
0	44	9	21.0
1	10	5	53.0
2	37	6	80.0
3	19	10	11.0
4	46	13	215.0

Dimensions of the final data set: **543 x 3**

```
trainingDF.shape
```

```
(543, 3)
```

# of clusters created: **3**

```
print(my_kmmodel.centers)
```

```
[array([ 40.87037037,   9.75925926, 138.24074074]), array([25.29532164,  4.28947368, 15.23684211]), array([34.63945
578,  6.45578231, 59.12244898])]
```

Cluster Centers

By C.G.

The code used in creating cluster centers is given below:

```
my_kmmodel = KMeans.train(parsedData, 3, maxIterations=10, initializationMode="random")
```

Cluster centers formed are given in the table below:

Cluster #	Center
1	[40.87037037, 9.75925926, 138.24074074]
2	[25.29532164, 4.28947368, 15.23684211]
3	[34.63945578, 6.45578231, 59.12244898]

The indices for each cluster are as follows: **[totalAdClicks, totalBuyClicks, revenue]**

These clusters can be differentiated from each other as follows:

Cluster 1 is different from the others in that users who click on the most ads and make the most purchases also generate the highest revenue in the game.

Cluster 2 is different from the others in that users who click on the least ads and make the least purchases also generate the lowest revenue in the game.

Cluster 3 is different from the others in that on average, users who click on the more ads and make more purchases significantly increase the game's revenue. This cluster does not generate the highest revenue, compared to Cluster 1.

Cluster 3 is a great comparison point among the other clusters. Analyzing all clusters together, we can conclude that there is a strong, positive correlation among the

total amount of ads users clicked on, the total amount of game purchases, and the total amount of revenue that the game generates from ads and purchases.

Below you can see the summary of the train data set:

```
trainingDF.describe()
```

	totalAdClicks	totalBuyClicks	revenue
count	543.000000	543.000000	543.000000
mean	29.373849	5.419890	39.349908
std	15.216343	3.244713	41.221737
min	1.000000	1.000000	1.000000
25%	16.000000	3.000000	10.000000
50%	30.000000	5.000000	25.000000
75%	42.000000	7.000000	55.000000
max	67.000000	16.000000	223.000000

Recommended Actions

By C.G.

Action Recommended	Rationale for the action
Increase ads that advertise in-app purchases.	Since ad clicks are highly correlated with in-app purchases, the game would increase revenue if the players saw more advertisements that encouraged them to make in-app purchases. This should be included in addition to advertisements related to different products than the game.
Decrease the time ads are shown. Use the extra time to show more ads.	The game can incorporate more ads, without forcing users to waste more time watching ads, by simply decreasing the time ads are shown to users. The extra time can then be used to show more ads in order increase revenue derived from advertisers.
Analyze purchases among users who generated the least profit in the game. Then, create custom advertisements.	The game can cater advertisements to infrequent buyers in order to prompt these players to make more in-app purchases. This can be accomplished by analyzing the products these users buys and marketing these products towards these users. For instance, if the infrequent buyers only purchase game boosts, these users can be shown more game-boost advertisements rather than showing users other items they likely will never buy.

Graph Analytics

By. C.G.

This analysis will use Neo4J to explore how a team of players interact in chat sessions while playing the fictional game, *Catch the Pink Flamingo.*

Modeling Chat Data using a Graph Data Model

Describe the graph model for chats in a few sentences. Try to be clear and complete.

The chat data model illustrates how game users interact in chats they can engage in while playing the game. Users can chat with other team members in a chat session. The graph also illustrates chat properties, such as when a player joins or leaves a chat. A player can take part of different chat sessions. When a player is mentioned or responds to a chat, the game also creates a timestamp to document this interaction. The model can also be used to learn more about players' social interactions, including determining which users are the most and the least active in the chat rooms.

Creation of the Graph Database for Chats

Describe the steps you took for creating the graph database. As part of these steps

i) Write the schema of the 6 CSV files

Overview

userId: The unique ID of a game user

teamId: The unique ID of team of players

teamChatSessionId: The unique ID of a team's chat session

timestamp: The record of time for when an event occurred

chatItemId: The unique ID of a user's chat activity

chatItemId_one: A user's conversation with another team member

chatItemId_two: A team member's response to another user's conversation

File Name	Description	Fields
chat_create_team_chat.csv	*Creates a chat session between a user and their team*	userId teamId teamChatSessionId timestamp
chat_join_team_chat.csv	*Denotes the interactions between a user and a team's chat session*	userId teamChatSessionId timestamp
chat_leave_team_chat.csv	*Indicates when a user has left a team's chat session*	userId teamChatSessionId timestamp
chat_item_team_chat.csv	*Depicts a user's chat activity in a chat session*	userId teamChatSessionId chatItemId timestamp
chat_mention_team_chat.csv	*Represents when a user is mentioned in a conversation*	chatItemId userId timestamp
chat_respond_team_chat.csv	*Demonstrates when a user responds to another user in a conversation*	chatItemId_one chatItemId_two timestamp

ii) Explain the loading process and include a sample LOAD command

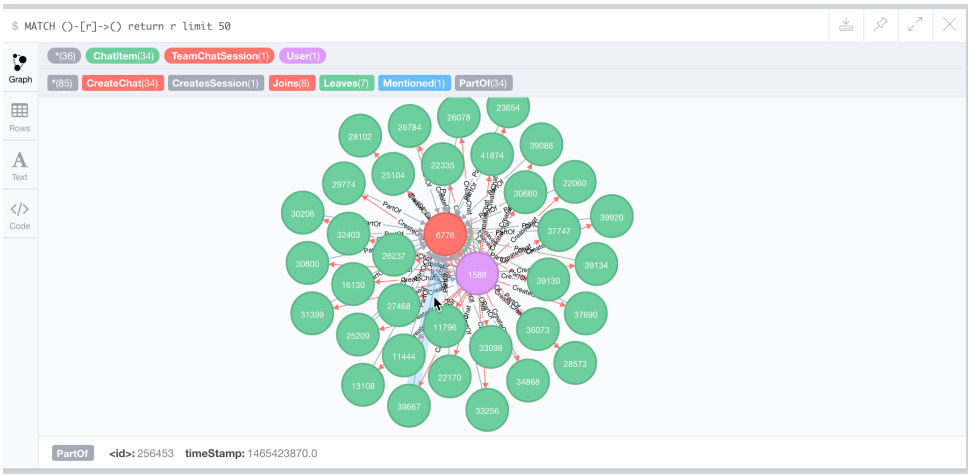
The LOAD command uses the Cypher query language to upload a file type, such as a CSV, into the NEO4J database. The LOAD command follows the specified directory path to locate the file on a computer. Each line in the file is read one row at a time. The Load command then can be used to create nodes. A column value will be converted into an integer to create values for the node.

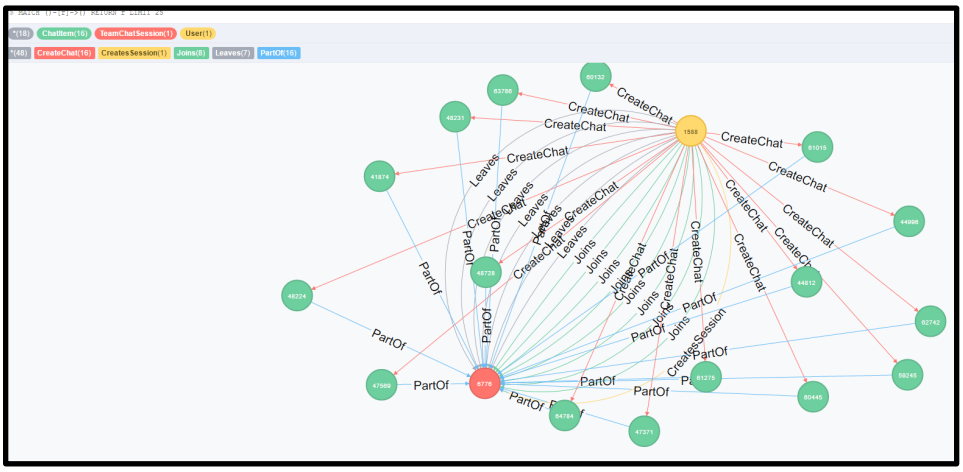
Example:

```
LOAD CSV FROM
"file:/big_data_capstone_datasets_and_scripts/ch
at_mention_team_chat.csv" AS row
MERGE (i:ChatItem {id: toInteger(row[0])})
MERGE (u:User {id: toInteger(row[1])})
MERGE (i)-[:Mentioned{timeStamp:
toInteger(row[2])}]→(u)
```

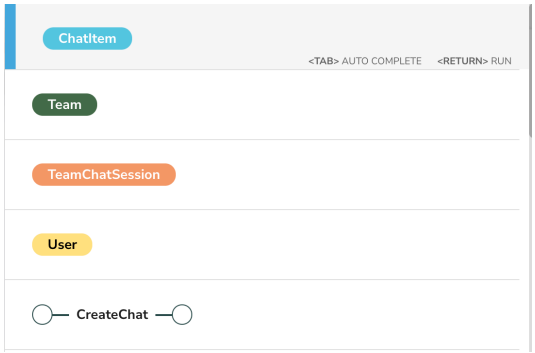
In the example above, the first line follows the file path directory and reads the file **chat_mention_team_chat.csv** one line at a time to create ChatItem nodes. In the following lines, the same logic is repeated - each line is read individually to create User nodes. In the last two lines, an edge called Mentioned is created between ChatItem nodes and User nodes. The Mentioned edges are located in the third column of the file and contain a timestamp property.

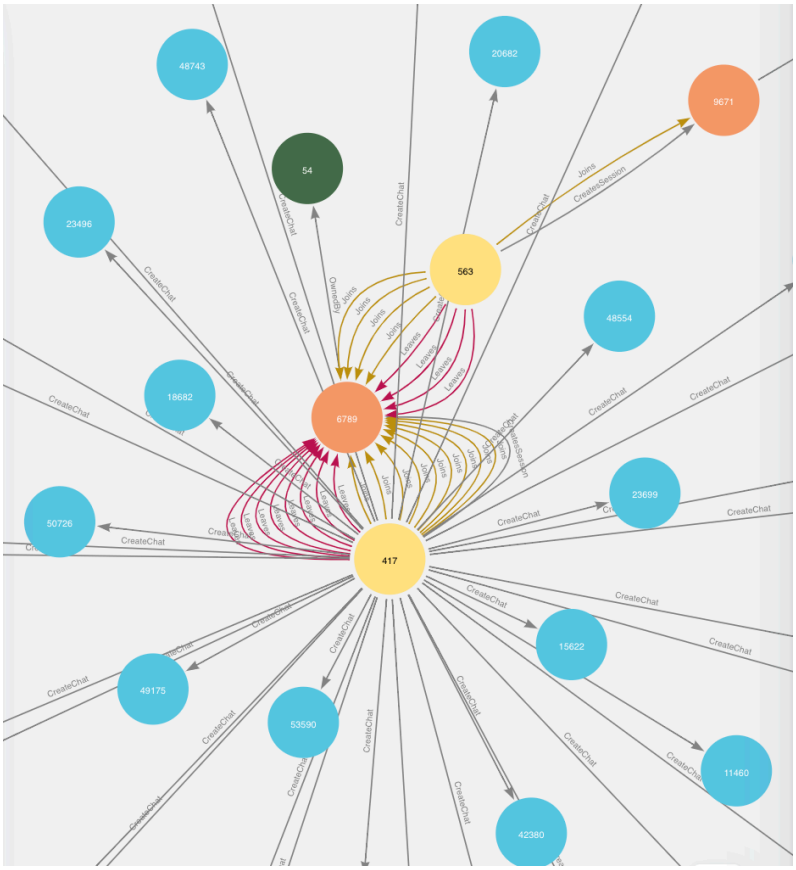
iii) Present a screenshot of some part of the graph you have generated. The graphs must include clearly visible examples of most node and edge types. Below are two acceptable examples. The first example is rendered in the default Neo4j distribution, the second has had some nodes moved to expose the edges more clearly. Both include examples of most node and edge types.





Example:





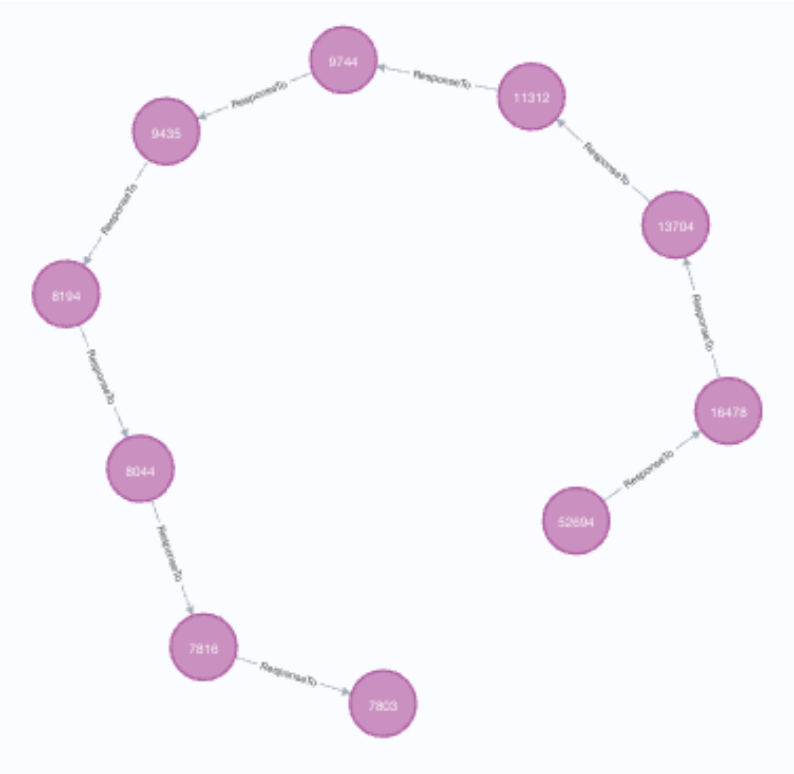
Finding the longest conversation chain and its participants

Report the results including the length of the conversation (path length) and how many unique users were part of the conversation chain. Describe your steps. Write the query that produces the correct answer.

Path Length between chat nodes: 9

Longest conversation chain: 10

```
neo4j$ MATCH z=(x)-[:ResponseTo*]→(y)
       RETURN z, length(z)
       ORDER BY length (z) desc limit 1
```



Number of unique Users: 5

```
MATCH l =(x:ChatItem)-[:ResponseTo*]→(y:ChatItem)
WHERE length(l)=9
WITH l
Match (u:User)-[:CreateChat]→(i:ChatItem)
WHERE (i IN NODES(l))
RETURN count (distinct u)
```

As shown above, the longest path between the chat nodes is 9. Therefore, the longest conversation chain in the game has 10 chats, since there are 9 edges that connect the 10 chat nodes. This is found by determining the number of chat responses between participants in the chat rooms.

There are also 5 unique users in the longest conversation chain. The number of unique users is determined by analyzing the number of distinct user IDs among the chats in the conversation thread.

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

Describe your steps from Question 2. In the process, create the following two tables. You only need to include the top 3 for each table. Identify and report whether any of the chattiest users were part of any of the chattiest teams.

Top 10 Chattiest Users:

```
MATCH (u:User)-[:CreateChat]→(i:ChatItem)
RETURN u.id as Users, count(u.id) as Number_Chats
ORDER BY count (u.id) desc limit 10
```

	Users	Number_Chats
1	394	115
2	2067	111
3	1087	109
4	209	109
5	554	107
6	1627	105
7	999	105
8	516	105
9	668	104
10	461	104

The top 10 users that participate in the most amount of chats are listed above. This is found by analyzing the number of unique users who participate in conversations in chat sessions.

Top 10 Chattiest Teams:

```
MATCH (i: ChatItem)-[:PartOf]→(c: TeamChatSession)-[:OwnedBy]→(n)
RETURN n.id as Teams, count(n.id) as Number_Chats
ORDER BY count (n.id) desc limit 10
```

	Teams	Number_Chats
1	82	1324
2	185	1036
3	112	957
4	18	844
5	194	836
6	129	814
7	52	788
8	136	783
9	146	746
10	81	736

The top 10 teams that participate in the most amount of chats are listed above. This is found by analyzing the number of unique teams who participate in chat sessions.

Top 3: Users and Teams

Chattiest Users

Users	Number of Chats
394	115
2067	111
1087	109

Chattiest Teams

Teams	Number of Chats
82	1,324
185	1,036
112	957

Finally, present your answer, i.e. whether or not any of the chattiest users are part of any of the chattiest teams.

Answer: *User 999 on Team 52*

```
MATCH (u: User)-[:CreateChat]→(i: ChatItem)-[:PartOf]→(c: TeamChatSession)-[:OwnedBy]→(n)
RETURN distinct u.id as Users, n.id as Teams, count(c) as Number_Chats
ORDER BY count(c) desc limit 10
```

	Users	Teams	Number_Chats
1	394	63	115
2	2067	7	111
3	209	7	109
4	1087	77	109
5	554	181	107
6	1627	7	105
7	516	7	105
8	999	52	105
9	461	104	104
10	668	89	104

One user, 999, is part of one the chattiest teams, 52. To determine whether the chattiest users were among the chattiest times, the nodes and edges from the query above were combined. Cross-referencing the combined table above with the individual chattiest user and chattiest team tables, it is evident that user 999 is also part of team 52.

How Active Are Groups of Users?

Describe your steps for performing this analysis. Be as clear, concise, and as brief as possible. Finally, report the top 3 most active users in the table below.

This final query had multiple steps that can be divided into three main parts. First, a neighborhood of users was constructed. By labeling four users in the same proximity, or neighborhood, as one another, the next step involves analyzing interactions. Since users are in the same neighborhood, self-interactions among the group go neighbors must be eliminated in order to avoid over-inflating the cluster-coefficient. Finally, the last step is calculating the cluster coefficient. The cluster coefficient measures the degree in which the user nodes cluster together in a neighborhood, a.k.a., are most socially active in the game. The formula is below, where C is the clustering coefficient, A is a neighborhood of game users, and k is a neighbor, a.k.a., a game user.

$$C_i = \frac{\sum A_{ij} \cdot A_{jk} \cdot A_{ki}}{k_i \cdot (k_i - 1)}$$

The three steps are simplified below:

Step 1) Construct neighborhood of users

Step 2) Eliminate neighborhood interactions

Step 3) Determine the clustering coefficient of top users

Most Active Users (based on Cluster Coefficients)

User ID	Coefficient
209	0.9524
554	0.9048
1087	0.8