# Technical Appendix
# Prepared by: Max Pan Ziyuan

## 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** |
|---|---|---|
| adclicks.csv | A line is added to this file when a player clicks on an advertisem ent in the Flamingo app. | timestamp: when the click occurre d.<br>txId: a unique id (within ad-clicks.log) for the click<br>userSessionid: the id of the user ses sion for the user who made the clic k<br>teamid: the current team id of the u ser who made the click<br>userid: the user id of the user who made the click<br>adId: the id of the ad clicked on<br>adCategory: the category/type of a d clicked on |
| buyclicks.csv | A line is added to this file when a player makes an in-app purchase in the Flamingo ap p. | timestamp: when the purchase was made.<br>txId: a unique id (within buy-clicks.log) for the purchase<br>userSessionId: the id of the user se ssion for the user who made the pu rchase<br>team: the current team id of the use r who made the purchase<br>userId: the user id of the user who made the purchase<br>buyId: the id of the item purchased<br>price: the price of the item purchas ed |
| users.csv | This file contains a line for each user playing the game. | timestamp: when user first played t he game.<br>userId: the user id assigned to the u ser.<br>nick: the nickname chosen by the u ser. |

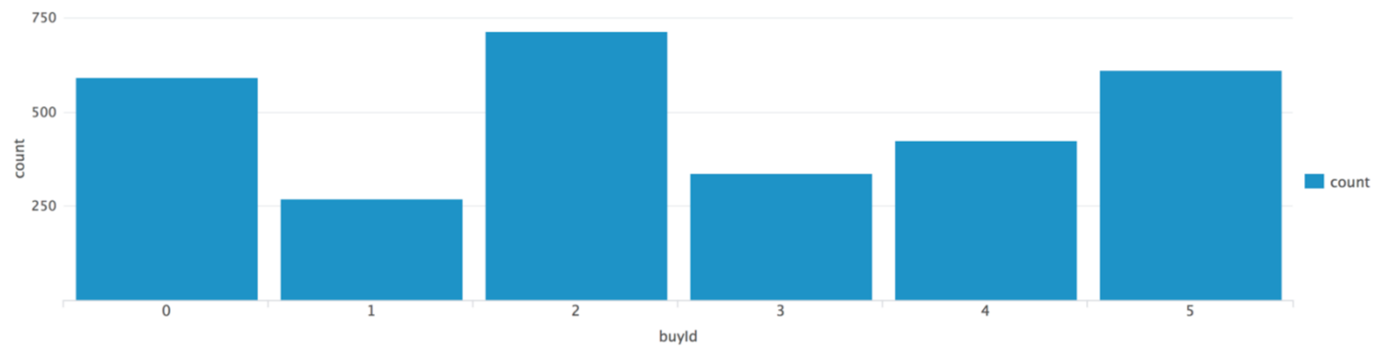
| | | |
|---|---|---|
| | | twitter: the twitter handle of the user.<br>dob: the date of birth of the user.<br>country: the two-letter country code where the user lives |
| team.csv | This file contains a line for each team terminated in the game. | teamId: the id of the team<br>name: the name of the team<br>teamCreationTime: the timestamp when the team was created<br>teamEndTime: the timestamp when the last member left the team<br>strength: a measure of team strength, roughly corresponding to the success of a team<br>currentLevel: the current level of the team |
| team-assignments.csv | A line is added to this file each time a user joins a team. A user can be in at most a single team at a time. | timestamp: when the user joined the team.<br>team: the id of the team<br>userId: the id of the user<br>assignmentId: a unique id for this assignment |
| level-events.csv | A line is added to this file each time a team starts or finishes a level in the game | timestamp: when the event occurred.<br>eventId: a unique id for the event<br>teamId: the id of the team<br>teamLevel: the level started or completed<br>eventType: the type of event, either start or end |
| user-session.csv | Each line in this file describes a user session, which denotes when a user starts and stops playing the game. Additionally, when a team goes to the next level in the game, the session is ended for each user in the team and a new one started. | timestamp: a timestamp denoting when the event occurred.<br>userSessionId: a unique id for the session.<br>userId: the current user's ID.<br>teamId: the current user's team.<br>assignmentId: the team assignment id for the user to the team.<br>sessionType: whether the event is the start or end of a session.<br>teamLevel: the level of the team during this session.<br>platformType: the type of platform of the user during this session. |

| | | |
|---|---|---|
| gameclicks.csv | A line is added to this file each time a user performs a click in the game. | timestamp: when the click occurred.<br>clickId: a unique id for the click.<br>userId: the id of the user performing the click.<br>userSessionId: the id of the session of the user when the click is performed.<br>isHit: denotes if the click was on a flamingo (value is 1) or missed the flamingo (value is 0)<br>teamId: the id of the team of the user<br>teamLevel: the current level of the team of the user |

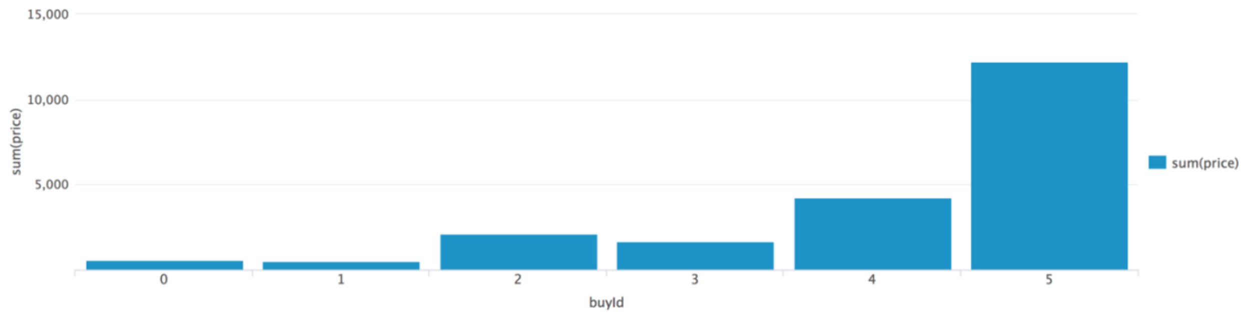
## Aggregation

| | |
|---|---|
| Amount spent buying items | 21407 |
| Number of unique items available to be purchased | 6 |

A histogram showing how many times each item is purchased:

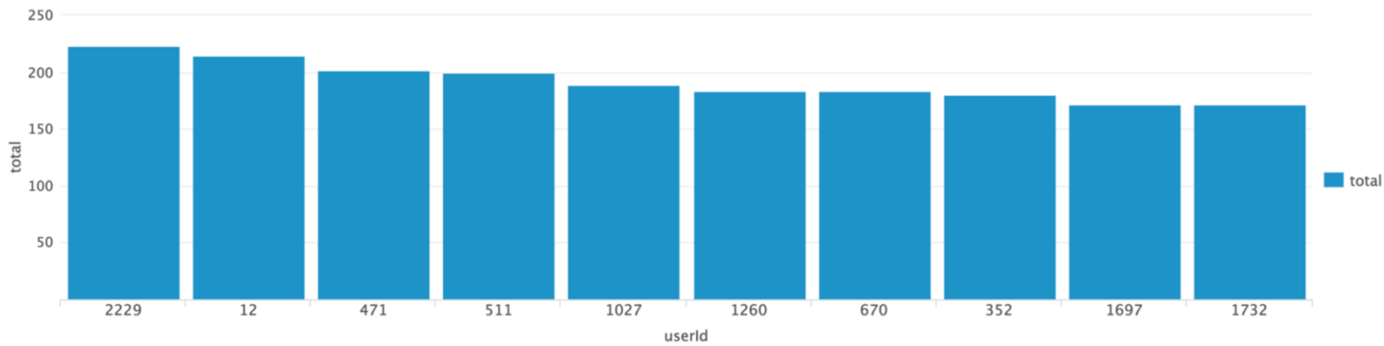



A histogram showing how much money was made from each item:




## Filtering

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




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 | 2 | iphone | 13.1 |
| 3 | 471 | iphone | 14.5 |

## Data Preparation

Analysis of combined_data.csv

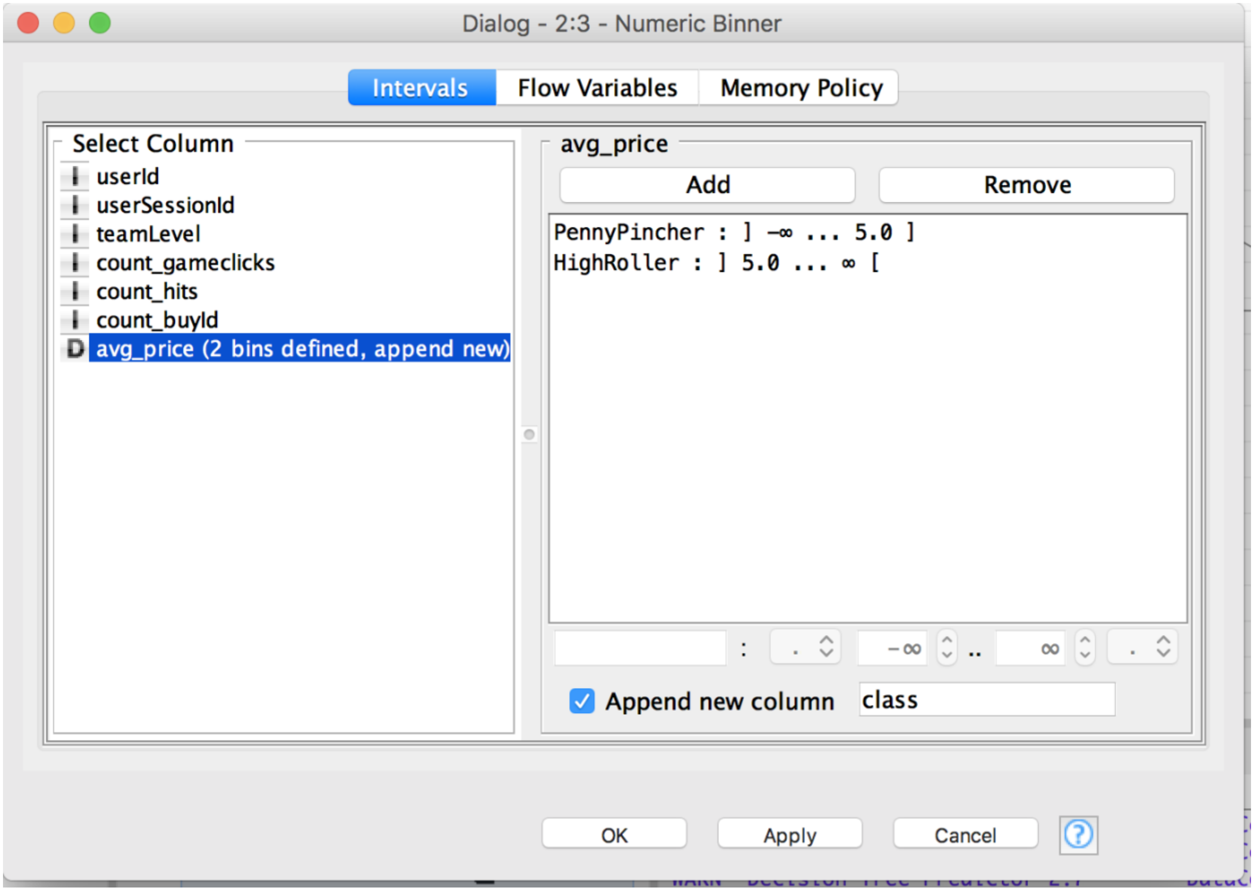
<u>Sample Selection</u>

| Item | Amount |
|---|---|
| # of Samples | 4619 |

| | |
|---|---|
| # of Samples with Purchases | 1411 |

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:




The column class is the new categorical attribute. The value is either PennyPincher or HighRoller. Its value depends on avg_price.

The creation of this new categorical attribute was necessary because the decision tree classifier needs a class label for each data record.

Attribute Selection

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

| **Attribute** | **Rationale for Filtering** |
|---|---|

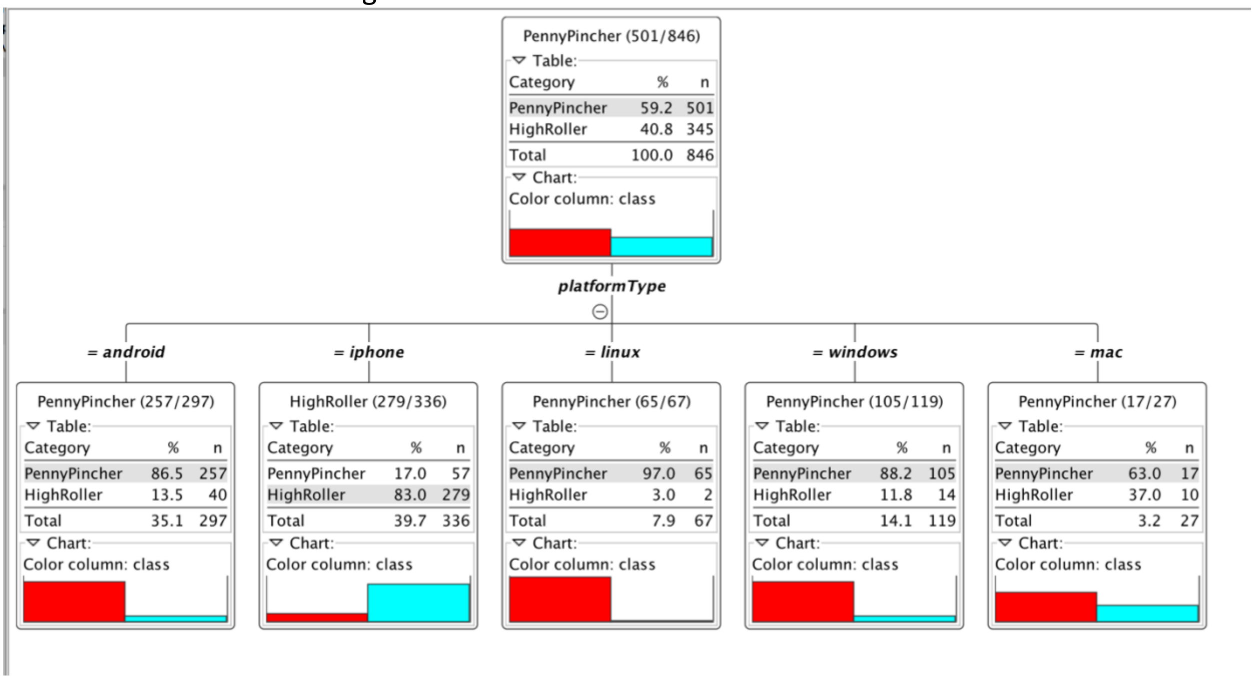
| | |
|---|---|
| userId | UserId doesn't represent any actual properties about the user. It is just an identifier. |
| userSessionId | Similar to userId. It is just an identifier. |
| avg_price | We have already converted avg_price into the class label. So we do not need it any more. |

## Data Partitioning and Modeling

The data was partitioned into train and test datasets.
The training data set was used to create the decision tree model.
The trained model was then applied to the testing dataset.
This is important because we need both training and testing dataset. And they normally

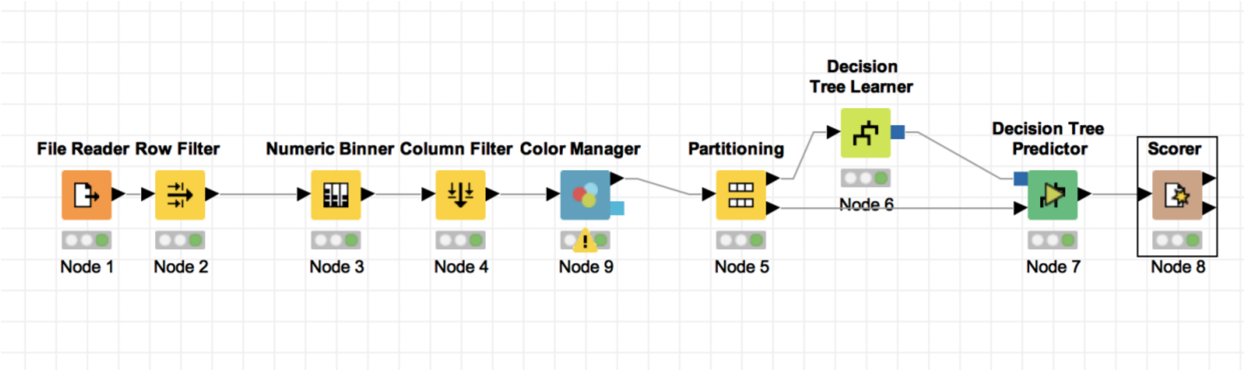
When partitioning the data using sampling, it is important to set the random seed because this will let us get the same partitioning result every time, and hence get the reproducible result.

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




## Analysis Conclusions

The final KNIME workflow is shown below:




What makes a HighRoller vs. a PennyPincher?

Based on the structure of the decision tree, the users using iPhone are more likely to be HighRoller, while the users on the other platforms tend to be PennyPincher.

| **Specific Recommendations to Increase Revenue** |
|---|
| 1. Compare the differences between the iOS app and the interface on the other platforms, and try to find the possible reasons (i.e. there might be some features only available on iPhone) why iOS users are more likely to spend more. |
| 2. After finding the possible reasons mentioned above, launch A/B testing for some new features accordingly on all the platforms except for iOS, and analyse which features could help to increase the revenue. |

## Evaluation

A screenshot of the confusion matrix can be seen below:

| class \ Pre... | PennyPinc... | HighRoller |
|---|---|---|
| PennyPinc... | 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 88.496%

The value 308 at top-left cell means that 308 PennyPinchers were correctly classified.
The value 27 at top-right cell means that 27 PennyPinchers were incorrectly classified as HighRoller.
The value 38 at bottom-left cell means that 38 HighRollers were incorrectly classified as PennyPinchers.
The value 192 at bottom-right cell means that 192 HighRollers were correctly classified.

## Training Data Set Creation

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

In [153]:
```
users.toPandas().head(5)
```

Out[153]:

| | userId | sum(price) | sum(count_gameclicks) | count(adId) |
|---|---|---|---|---|
| **0** | 231 | 63.0 | 262 | 19 |
| **1** | 2032 | 20.0 | 638 | 39 |
| **2** | 233 | 28.0 | 250 | 37 |
| **3** | 433 | 0.0 | 34 | 11 |
| **4** | 34 | 95.0 | 665 | 34 |

Dimensions of the training data set (rows x columns) : 529*3

# of clusters created: 4

## Attribute Selection

| **Attribute** | **Rationale for Selection** |
|---|---|
| Sum of total spending | This is an important indicator of how much the users spent in the game. |
| Total count of game clicks | This attribute indicates how often the users play the game. |
| Total count of ad clicks | This attributes indicates how often the users click on the ad. |

## Cluster Centers

| **Cluster #** | **Cluster Center** |
|---|---|
| 1 | [-0.0967, 0.02876, 0.8414]] |
| 2 | [2.322, 0.0707, 0.8612] |
| 3 | [-0.0656, 2.5974, 0.1903] |
| 4 | [-0.4427, -0.5243, -0.9249] |

These clusters can be differentiated from each other as follows:

Cluster 1 is different from the others in that the users click on the ads very often but seldom pay for the items.

Cluster 2 is different from the others in that the users click on the ads a lot and also like to buy the items.

Cluster 3 is different from the others in that the users play the game very often.

Cluster 4 is different from the others in that the users is not active in playing the games, buying items and clicking the ads.

## Recommended Actions

| Action Recommended | Rationale for the action |
|---|---|
| Launch some promotions targeting at users in cluster 1. | Cluster 1 is formed by those who like clicking on the ads but seldom spend money in the game. We can launch some promotions to see whether it can encourage them to buy. |
| Perform A/B testing on users in cluster 4 when needed. | When new features are built, we usually perform A/B testing to avoid launching the features that most of the people do not like. Cluster 4 is formed by the inactive users. So if they decide to quit the game forever when they see the features they do not like, the loss is relatively small. |

# Graph Analytics

## Modeling Chat Data using a Graph Data Model

There are 4 types of nodes in total: user, team session, team, chat item. And different edge types represent different actions between the nodes:
“CreatesSession” edge represents that a user creates a team chat session.
“OwnedBy” edge represents that a team chat session is owned by a team.
"Joins" edge represents that a user joins a team chat session.
“Leaves” edge represents that a user leaves a team chat session.
“CreateChat” edge represents that a user creates chat item.
“PartOf” edge represents that a chat item is part of a chat session.
“Mentioned” edge represents that a user is mentioned in a chat item.
“ResponseTo” edge represents that a chat item is a response to another chat item.

## 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
   1. File: chat_create_team_chat.csv
      Schema: userid, teamid, TeamChatSessionID, timestamp
   2. File: chat_join_team_chat.csv
      Schema: userid, TeamChatSessionID, teamstamp
   3. File: chat_leave_team_chat.csv

Schema: userid, teamchatsessionid, timestamp

4. File: chat_item_team_chat.csv
   Schema: userid, teamchatsessionid, chatitemid, timestamp
5. File: chat_mention_team_chat.csv
   Schema: userid, teamchatsessionid, chatitemid, timestamp
6. File: chat_respond_team_chat.csv
   Schema: chatitemid1, chatitemid2, timestamp

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

Step 1:
Create the constraints for uniqueness of IDs:

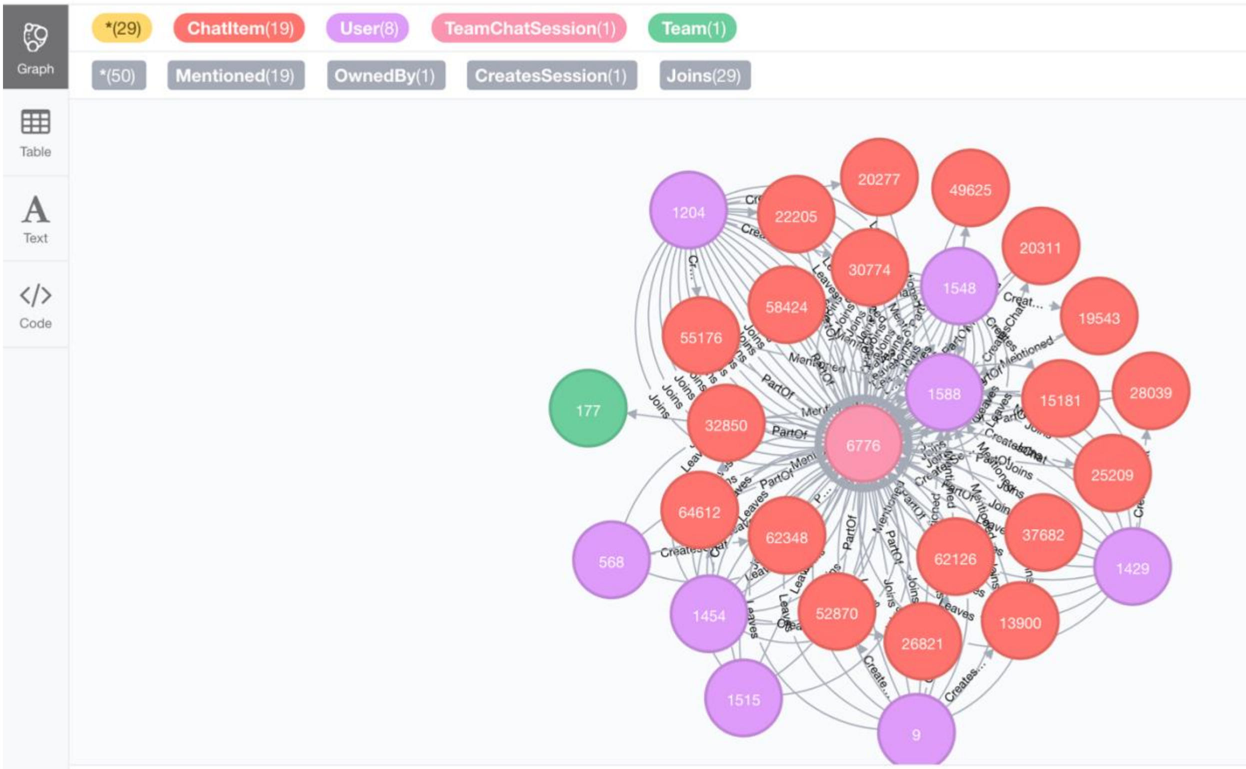
```
CREATE CONSTRAINT ON (u:User) ASSERT u.id IS UNIQUE;
CREATE CONSTRAINT ON (t:Team) ASSERT t.id IS UNIQUE;
CREATE CONSTRAINT ON (c:TeamChatSession) ASSERT c.id IS UNIQUE;
CREATE CONSTRAINT ON (i:ChatItem) ASSERT i.id IS UNIQUE;
```

Step 2:
Load the csv files into the database according to the schema. For example, when loading chat_join_team_chat.csv, the following command was used:

```
LOAD CSV FROM "file:///chat_join_team_chat.csv" AS row
MERGE (u:User {id: toInt(row[0])})
MERGE (c:TeamChatSession {id: toInt(row[1])})
MERGE (u)-[:Joins{timeStamp: row[2]}]->(c)
```

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.

```
$ MATCH (n)-[r]->(m) RETURN n,r,m LIMIT 50
```



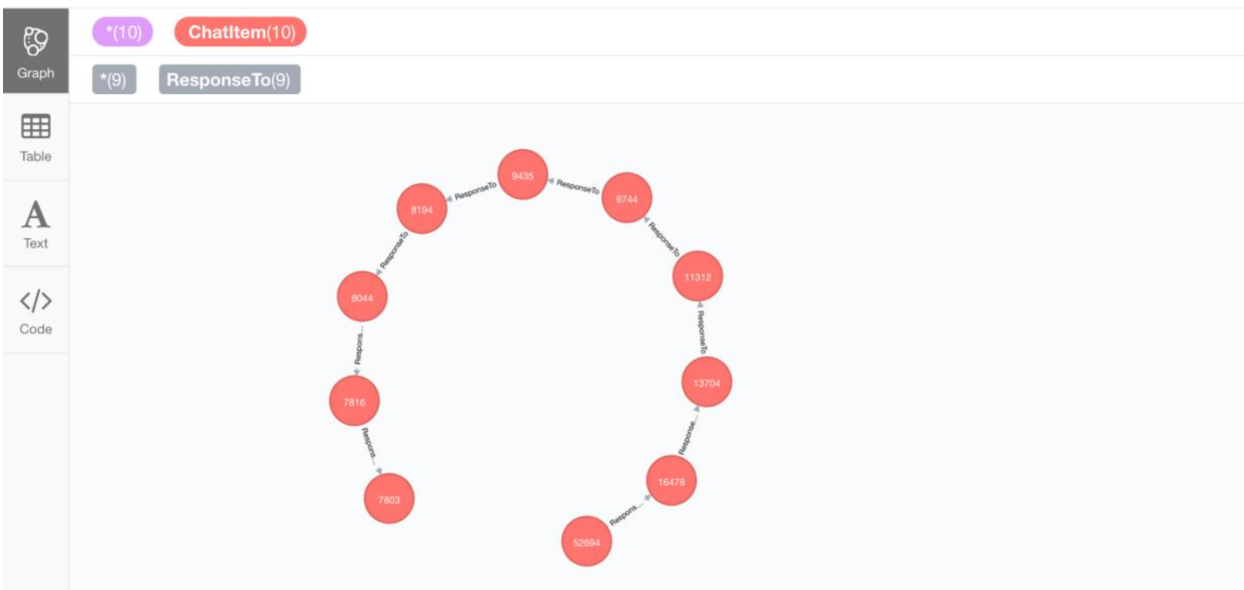

## Finding the longest conversation chain and its participants

To find the longest conversation chain, I used the command:

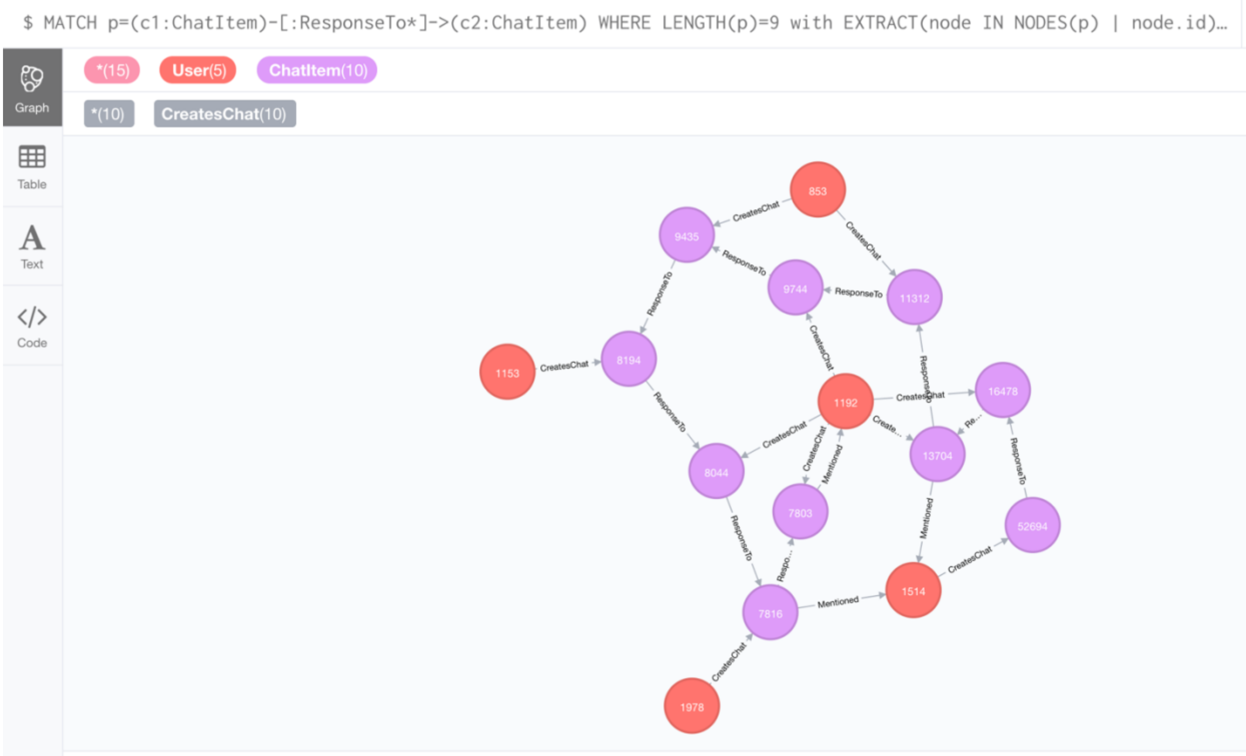
match p=(c1:ChatItem)-[:ResponseTo*]->(c2:ChatItem) return length(p) as l, p order by l desc limit 1

The result is shown below. We can see that the length of the longest path is 9.

```
$ match p=(c1:ChatItem)-[:ResponseTo*]->(c2:ChatItem)  return length(p) as l, p order by l desc limit 1
```



To get the number of distinct users involved, the following command is used:
MATCH p=(c1:ChatItem)-[:ResponseTo*]->(c2:ChatItem) WHERE LENGTH(p)=9 WITH EXTRACT(node IN NODES(p) | node.id) AS ids
MATCH (u:User)-[r:CreatesChat]-(c:ChatItem) WHERE c.id in ids RETURN u,r,c;




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

To find the chattiest users, the following command is used:
MATCH (u:User)-[r:CreatesChat]->() RETURN u.id, COUNT(u) ORDER BY COUNT(u) DESC LIMIT 10;

**Chattiest Users**

| Users | Number of Chats |
|---|---|
| 394 | 115 |
| 2067 | 111 |
| 209 | 109 |

To find the chattiest teams, the following command is used:
MATCH (ci:ChatItem)-[p:PartOf]->(tc:TeamChatSession)-[o:OwnedBy]->(t:Team) RETURN t.id, COUNT(t) ORDER BY COUNT(t) DESC LIMIT 10;

**Chattiest Teams**

| Teams | Number of Chats |
|---|---|
| 82 | 1324 |
| 185 | 1036 |
| 112 | 957 |

To determine whether the chattiest users belong to the chattiest team:
MATCH (u:User)-[cc:CreatesChat]->(ci:ChatItem)-[p:PartOf]->(tc:TeamChatSession)-[o:OwnedBy]->(t:Team) RETURN u.id,  t.id, COUNT(t) ORDER BY COUNT(t) DESC LIMIT 10;

We can see that user 999 is one of the chattiest users, and it belongs to one of the chattiest team, team 52.

```
$ MATCH (u:User)-[cc:CreatesChat]->(ci:ChatItem)-[p:PartOf]->(tc:TeamChatSession)-[o:OwnedBy]->(t:Team) RETURN u…
```

| u.id | t.id | COUNT(t) |
|---|---|---|
| 394 | 63 | 115 |
| 2067 | 7 | 111 |
| 1087 | 77 | 109 |
| 209 | 7 | 109 |
| 554 | 181 | 107 |
| 1627 | 7 | 105 |
| 999 | 52 | 105 |
| 516 | 7 | 105 |
| 461 | 104 | 104 |
| 668 | 89 | 104 |

## 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.

Step 1:

Create the “InteractsWith” relation:

MATCH (u1:User)-[r:CreatesChat]->(ci1:ChatItem)-[m:Mentioned]->(u2:User) CREATE (u1)-[:InteractsWith]->(u2)

MATCH (u1:User)-[:CreatesChat]->(ci1:ChatItem)-[r:ResponseTo]->(ci2:ChatItem)<-[:CreatesChat]-(u2:User) CREATE (u1)-[:InteractsWith]->(u2)

Match (u1)-[r:InteractsWith]->(u1) delete r

Step 2:

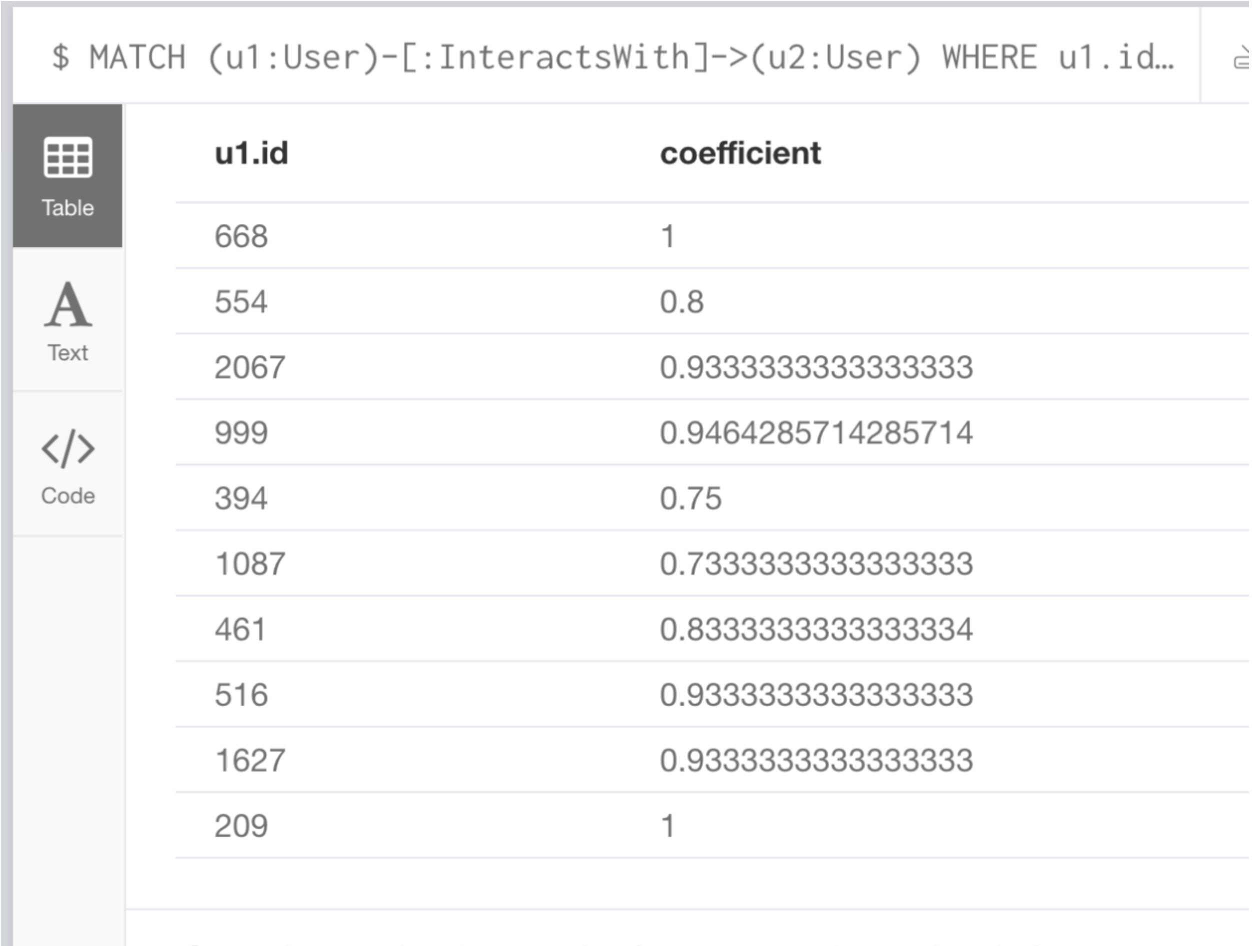
Calculate the coefficients:

MATCH (u1:User)-[:InteractsWith]->(u2:User) WHERE u1.id IN [394, 2067, 209, 1087, 554, 516, 1627, 999, 668, 461] WITH u1, collect(distinct u2.id) AS neighbors

MATCH (u3:User), (u4:User) WHERE u3.id IN neighbors AND u4.id IN neighbors AND u3<>u4

WITH u1, length(neighbors) AS k, sum(case when (u3)-[:InteractsWith]->(u4) then 1 else 0 end) as numEdge

RETURN u1.id, 1.0*numEdge/(k*(k-1)) as coefficient



```
$ MATCH (u1:User)-[:InteractsWith]->(u2:User) WHERE u1.id…
```



| u1.id | coefficient |
|---|---|
| 668 | 1 |
| 554 | 0.8 |
| 2067 | 0.9333333333333333 |
| 999 | 0.9464285714285714 |
| 394 | 0.75 |
| 1087 | 0.7333333333333333 |
| 461 | 0.8333333333333334 |
| 516 | 0.9333333333333333 |
| 1627 | 0.9333333333333333 |
| 209 | 1 |

**Most Active Users (based on Cluster Coefficients)**

| User ID | Coefficient |
|---|---|
| 209 | 1 |
| 668 | 1 |
| 2067 | 0.946 |