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
users.csv	This file contains a line for each user playing the game.	timestamp: when user first played the game. id: the user id assigned to the user. nick: the nickname chosen by the user. twitter: the twitter handle of the user. dob: the date of birth of the user. country: the two-letter country code where the user lives.
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. userSessionId: a unique id for the session. userId: the current user's ID. teamId: the current user's team. assignmentId: the team assignment id for the user to the team. sessionType: whether the event is

		the start or end of a session. teamLevel: the level of the team during this session. platformType: the type of platform of the user during this session.
teams.csv	This file contains a line for each team terminated in the game.	teamid: the id of the team name: the name of the team teamCreationTime: the timestamp when the team was created teamEndTime: the timestamp when the last member left the team strength: a measure of team strength, roughly corresponding to the success of a team 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.	time: when the user joined the team. team: the id of the team userid: the id of the user assignmentid: a unique id for this assignment
ad-clicks.csv	A line is added to this file when a player clicks on an advertisement in the Flamingo	timestamp: when the click

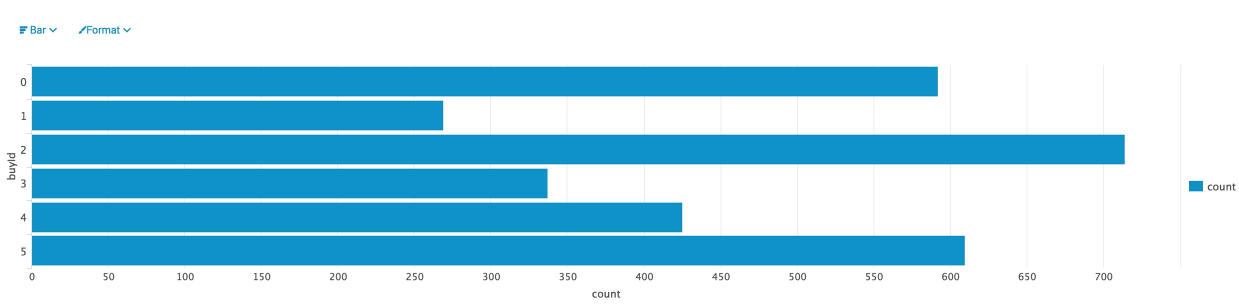
	app.	occurred. txID: a unique id (within ad-clicks.log) for the click userSessionid: the id of the user session for the user who made the click teamid: the current team id of the user who made the click userid: the user id of the user who made the click adID: the id of the ad clicked on adCategory: the category/type of ad clicked on
buy-clicks.csv	A line is added to this file when a player makes an in-app purchase in the Flamingo app.	timestamp: when the purchase was made. txID: a unique id (within buy-clicks.log) for the purchase userSessionid: the id of the user session for the user who made the purchase team: the current team id of the user who made the purchase userid: the user id of the user who made the purchase buyID: the id of the item purchased price: the price of the item purchased

game-clicks.csv	A line is added to this file each time a user performs a click in the game.	time: when the click occurred. clickid: a unique id for the click. userid: the id of the user performing the click. usersessionid: the id of the session of the user when the click is performed. isHit: denotes if the click was on a flamingo (value is 1) or missed the flamingo (value is 0) teamId: the id of the team of the user teamLevel: the current level of the team of the user
level-events.csv	A line is added to this file each time a team starts or finishes a level in the game	time: when the event occurred. eventid: a unique id for the event teamid: the id of the team level: the level started or completed eventType: the type of event, either start or end
<Fill In>	<Fill in short phrase>	<Fill In: Name and describe all fields>

Aggregation

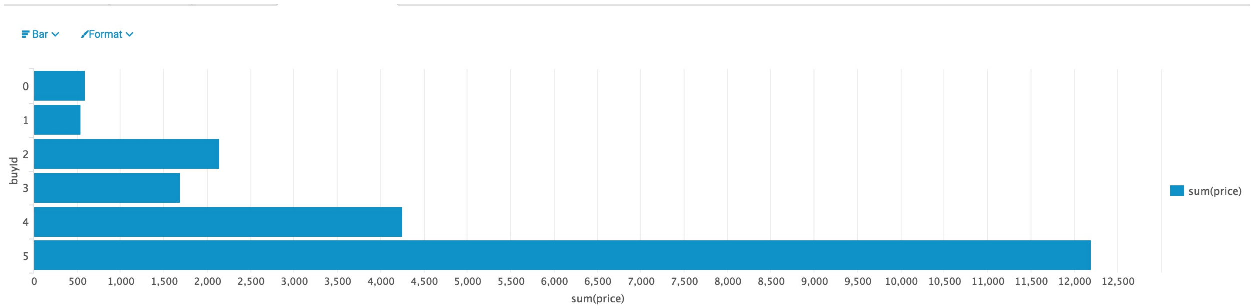
Amount spent buying items	21407.0
# Unique items available to be purchased	6

A histogram showing how many times each item is purchased:



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

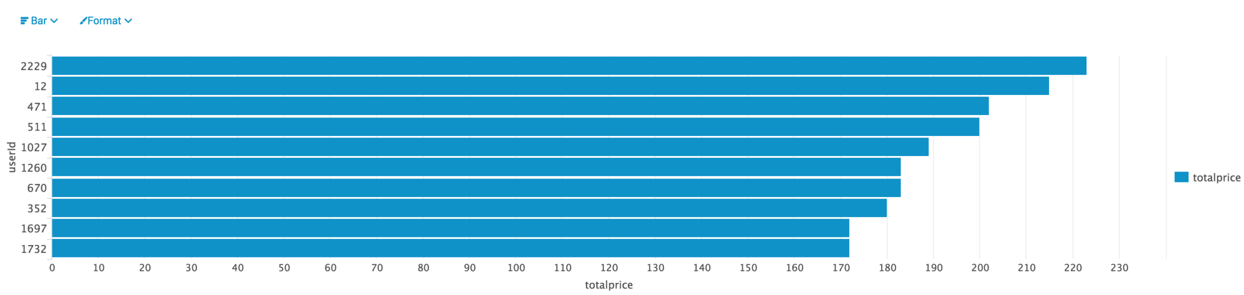
A histogram showing how much money was made from each item:



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

Filtering

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



userId	totalprice
2229	223.0
12	215.0
471	202.0
511	200.0
1027	189.0
1260	183.0
670	183.0
352	180.0
1697	172.0
1732	172.0

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	0.116
2	12	iphone	0.131
3	471	iphone	0.145

Data Preparation

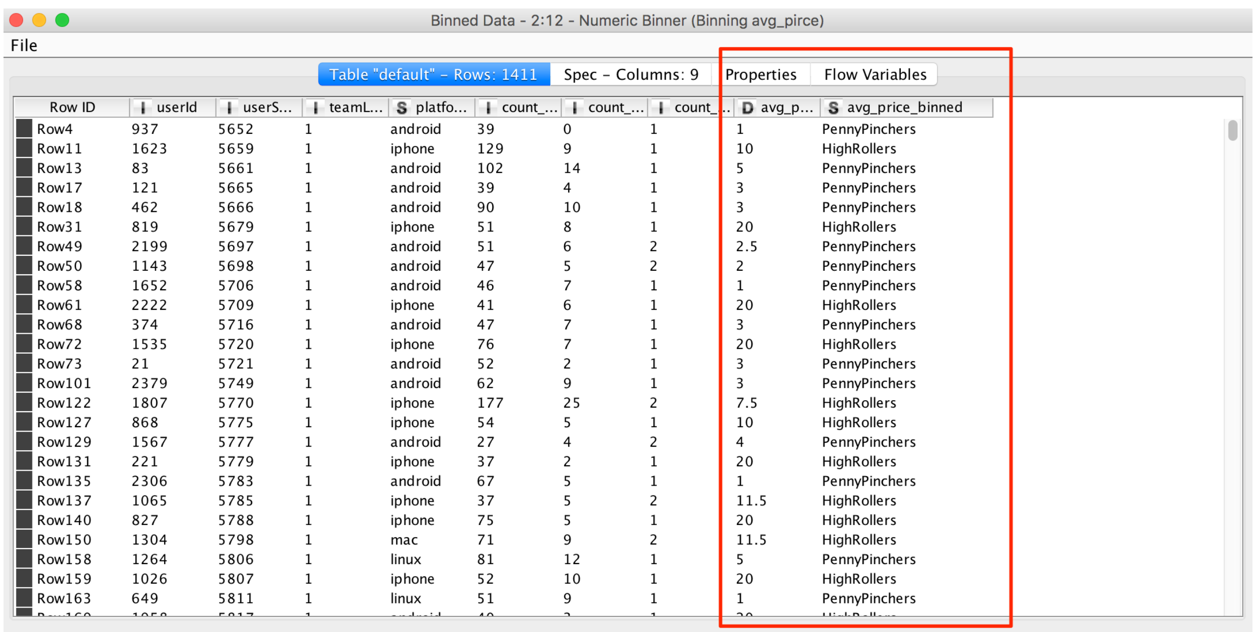
Analysis of combined_data.csv

Sample Selection

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:



Binned Data - 2:12 - Numeric Binner (Binning avg_pirce)

File

Table "default" – Rows: 1411 | Spec – Columns: 9 | Properties | Flow Variables

Row ID	userId	userS...	teamL...	platfo...	count_...	count_...	count_...	avg_p...	avg_price_binned
Row4	937	5652	1	android	39	0	1	1	PennyPinchers
Row11	1623	5659	1	iphone	129	9	1	10	HighRollers
Row13	83	5661	1	android	102	14	1	5	PennyPinchers
Row17	121	5665	1	android	39	4	1	3	PennyPinchers
Row18	462	5666	1	android	90	10	1	3	PennyPinchers
Row31	819	5679	1	iphone	51	8	1	20	HighRollers
Row49	2199	5697	1	android	51	6	2	2.5	PennyPinchers
Row50	1143	5698	1	android	47	5	2	2	PennyPinchers
Row58	1652	5706	1	android	46	7	1	1	PennyPinchers
Row61	2222	5709	1	iphone	41	6	1	20	HighRollers
Row68	374	5716	1	android	47	7	1	3	PennyPinchers
Row72	1535	5720	1	iphone	76	7	1	20	HighRollers
Row73	21	5721	1	android	52	2	1	3	PennyPinchers
Row101	2379	5749	1	android	62	9	1	3	PennyPinchers
Row122	1807	5770	1	iphone	177	25	2	7.5	HighRollers
Row127	868	5775	1	iphone	54	5	1	10	HighRollers
Row129	1567	5777	1	android	27	4	2	4	PennyPinchers
Row131	221	5779	1	iphone	37	2	1	20	HighRollers
Row135	2306	5783	1	android	67	5	1	1	PennyPinchers
Row137	1065	5785	1	iphone	37	5	2	11.5	HighRollers
Row140	827	5788	1	iphone	75	5	1	20	HighRollers
Row150	1304	5798	1	mac	71	9	2	11.5	HighRollers
Row158	1264	5806	1	linux	81	12	1	5	PennyPinchers
Row159	1026	5807	1	iphone	52	10	1	20	HighRollers
Row163	649	5811	1	linux	51	9	1	1	PennyPinchers

The rows with average price more than 5$ are assigned the value of "HighRollers", while ones with or less than 5$ are assigned the value of "PennyPinchers".

The creation of this new categorical attribute was necessary because we are having a classification problem. And the label average price is of continuous value type. It's necessary to transform it into categorical one.

Attribute Selection

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

Attribute	**Rationale for Filtering**
userId	The index is useless as the attributes.
userSessionId	The index is useless as the attributes.
avg_price	It is redundant now, since the binned average price has been used as the label.
<Optional Fill in>	<Optional Fill in 1-3 sentences>

Data Partitioning and Modeling

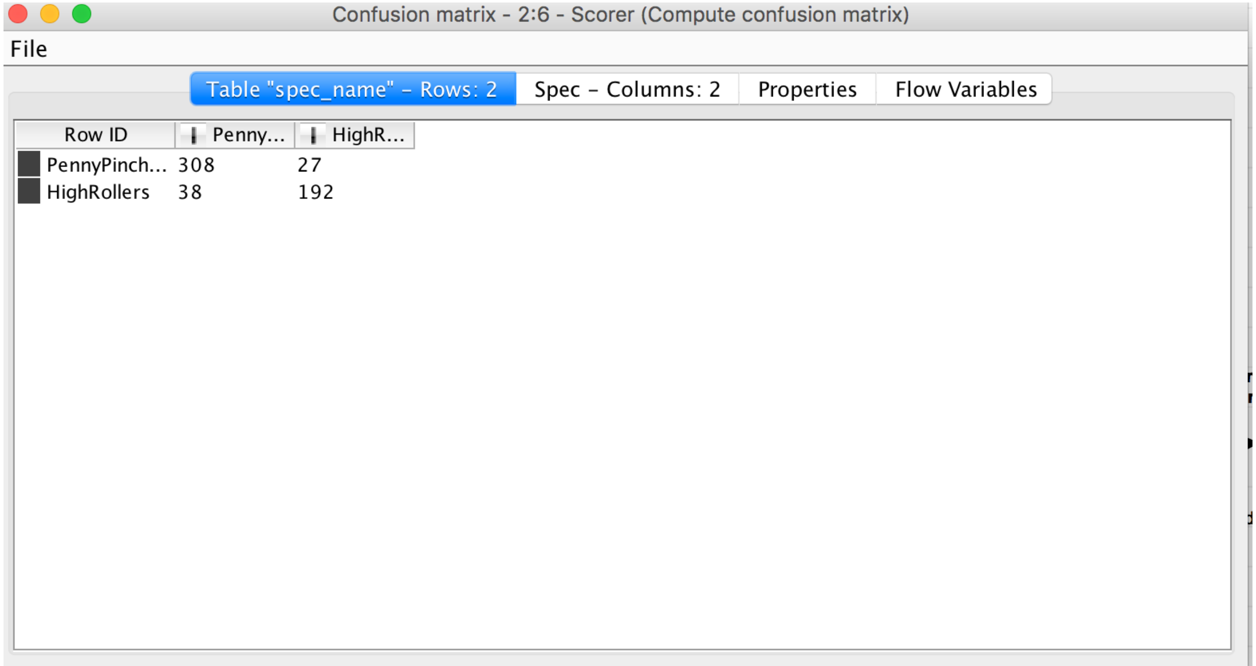
The data was partitioned into train and test datasets.
The training data set was used to create the decision tree model.
The trained model was then applied to the test dataset.
This is important because it could avoid overfitting. If we train all data to create the model and test it using the same data. The model is to memorize the data and unable to handle the unknown situation, which leads to overfitting.

When partitioning the data using sampling, it is important to set the random seed because we need the modeling results to be reproducible so that our conclusion could be persuasive.

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

Evaluation

A screenshot of the confusion matrix can be seen below:



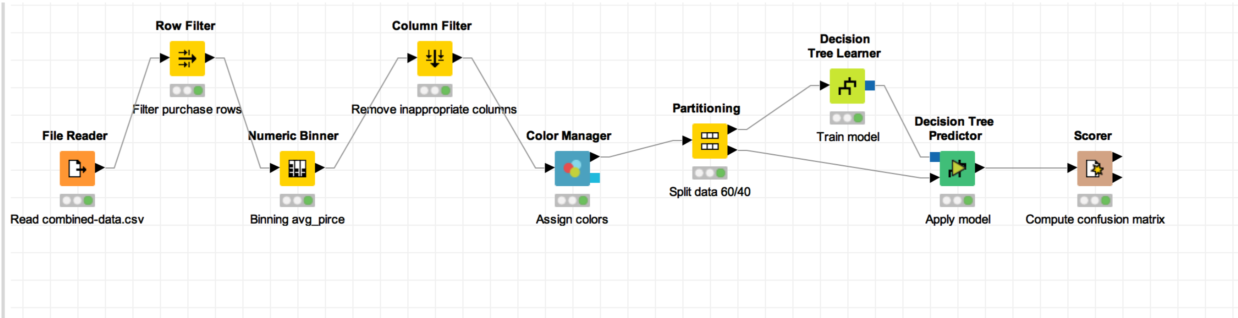
As seen in the screenshot above, the overall accuracy of the model is 88.5%

<Fill In: Write one sentence for each of the values of the confusion matrix indicating what has been correctly or incorrectly predicted.>

308: 308 true PennyPincher are correctly predicted as PennyPinchers
27: 27 true PennyPincher are incorrectly predicted as HighRollers
38: 38 true Highroller are incorrectly predicted as PennyPinchers
192: 192 true Highrollers are correctly predicted as HighRollers

Analysis Conclusions

The final KNIME workflow is shown below:



What makes a HighRoller vs. a PennyPincher?
<Fill In 2-3 sentences answering this question based on insights from your analysis.>

1. The platformType makes the type of buyer type, and mobile platforms contributes more than PC platforms.
2. In mobile platform, iphone players are more likely to be HighRoller(83%), while android players tend to be PennyPinchers(86.5%)
3. In PC platform, players are generally PennyPinchers, but part of mac users are HighRollers(37%).

Specific Recommendations to Increase Revenue
1. Android and Windows are two big user group to develop more HighRollers.
2. Considering the potential HighRoller group in mac platform, it’s worth investing to attract more players from mac platform.

Attribute Selection

Attribute	**Rationale for Selection**
totalAdClicks	Ad clicks relate to the revenue that brings profit to the company
total**Revenue**	Revenue show the purchase power of users
hitsPerHour	Hits reflect the performance of plays that might affect the purchase activities
<Optional Fill in>	<Optional Fill in 1-3 sentences>

Training Data Set Creation

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

jupyter week3 Last Checkpoint: 26 minutes ago (unsaved changes)

File Edit View Insert Cell Kernel Help | Python 2

Out[115]:

	totalAdClicks	revenue	hitsPerHour
0	44	21.0	0.219371
1	10	53.0	0.475000
2	37	80.0	0.164894
3	19	11.0	1.333333
4	46	215.0	0.221420

```
In [116]: trainingDF.shape
Out[116]: (529, 3)

In [118]: pDF = sqlContext.createDataFrame(trainingDF)

In [119]: parsedData = pDF.rdd.map(lambda line: array([line[0], line[1], line[2]])) #totalAdClicks, revenue
```

Train KMeans model

```
In [123]: my_kmmodel = KMeans.train(parsedData, 3, maxIterations=10, runs=10, initializationMode="random")

In [124]: print(my_kmmodel.centers)
[array([ 26.93714286,  16.68      ,   0.46511615]), array([  41.        ,  142.10204082,    0.3021974 ]), array([ 34.
35384615,  64.4       ,   0.39701866])]
```

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

# of clusters created: 3

Cluster Centers

Cluster #	**Cluster Center** [totalAdClicks, revenue, hitsPerHour]
1	`[41.06666667, 145.51111111, 0.30275967]`
2	`[27.14044944, 17.07022472, 0.46141664]`
3	`[34.3203125 , 66.78125, 0.40095501]`

These clusters can be differentiated from each other as follows:

Cluster 1 is different from the others in that it has the highest totalAdClicks and revenue and lowest hitsPerHour.

Cluster 2 is different from the others in that it has the highest hitsPerHour but lowest totalAdClicks and revenue

Cluster 3 is different from the others in that it has all the attributes in the middle of the clusters.
Note: Copy and fill in if you selected more than 3 clusters.

Recommended Actions

Action Recommended	Rationale for the action
Limit easy level games to advanced players	Since players with highest hitsPerHour dislike to purchase and click ads, the game difficulty could increase gradually for advanced players to reduce their hit rates.
Target ads to plyers with lower hit rates	Lowest hitsPerhour comes with the hightest totalAdClicks and revenue, player groups with lower rates tend to purchase more.
<Optional Fill in>	<Optional Fill in 1-3 sentences>
<Optional Fill in>	<Optional Fill in 1-3 sentences>

Graph Analytics

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 graph describes the following network. When one User creates a TeamChatSession, it is then owned by team. Users can join and leave the TeamChatSession. In TeamChatSession, users can create ChatItem that is part of TeamChatSession. ChatItem could also be mentioned by Users. And User could respond to User as well. All the relationships are recorded with timestamp.
Four types of nodes:

1. User
2. Team
3. TeamChatSession
4. ChatItem.

Seven types of relationships:

1. User creates TeamChatSession with timestamp
2. Team owns TeamChatSession with timestamp
3. User joins TeamChatSession with timestamp
4. User leaves TeamChatSession with timestamp
5. User creates ChatItem with timestamp
6. ChatItem is part of TeamChatSession with timestamp
7. ChatItem is mentioned by User with timestamp
8. ChatItem responses to ChatItem with timestamp

Creation of the Graph Database for Chats

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

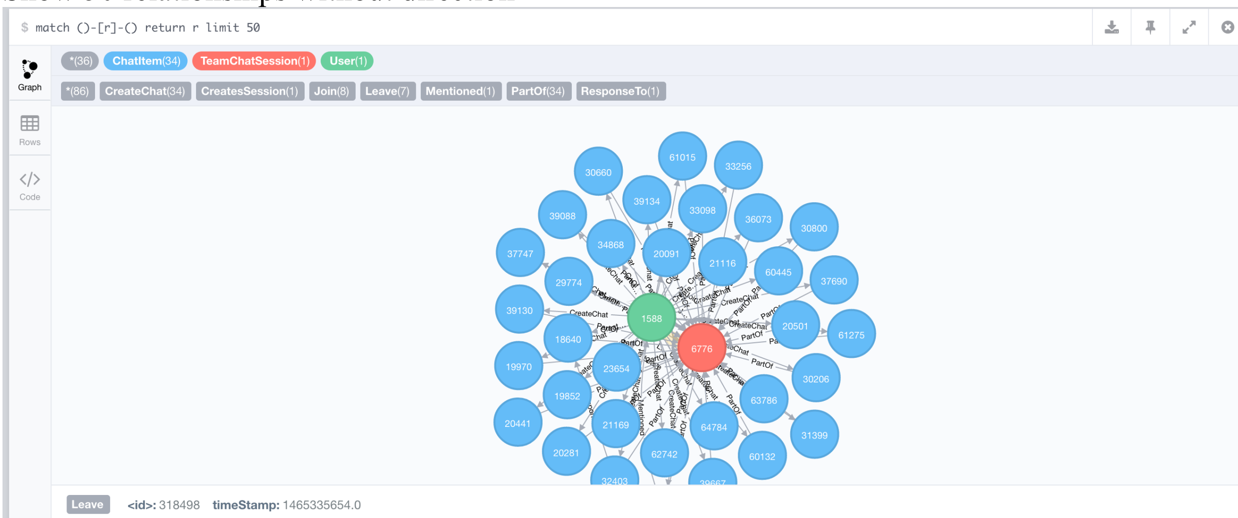
i) Write the schema of the 6 CSV files
1. chat_create_team_chat.csv
userid, teamid, TeamChatSessionID, timestamp
2. chat_item_team_chat.csv
userid, TeamChatSessionID, chatitemid, timestamp
3. chat_join_team_chat.csv
userid, TeamChatSessionID, timestamp
4. chat_leave_team_chat.csv
userid, TeamChatSessionID, timestamp
5. chat_mention_team_chat.csv
chatitemid, userid, timestamp
6. chat_respond_team_chat.csv
chatitemid1, chatitemid2,timestamp

ii) Explain the loading process and include a sample LOAD command
clear out the database
MATCH (n)

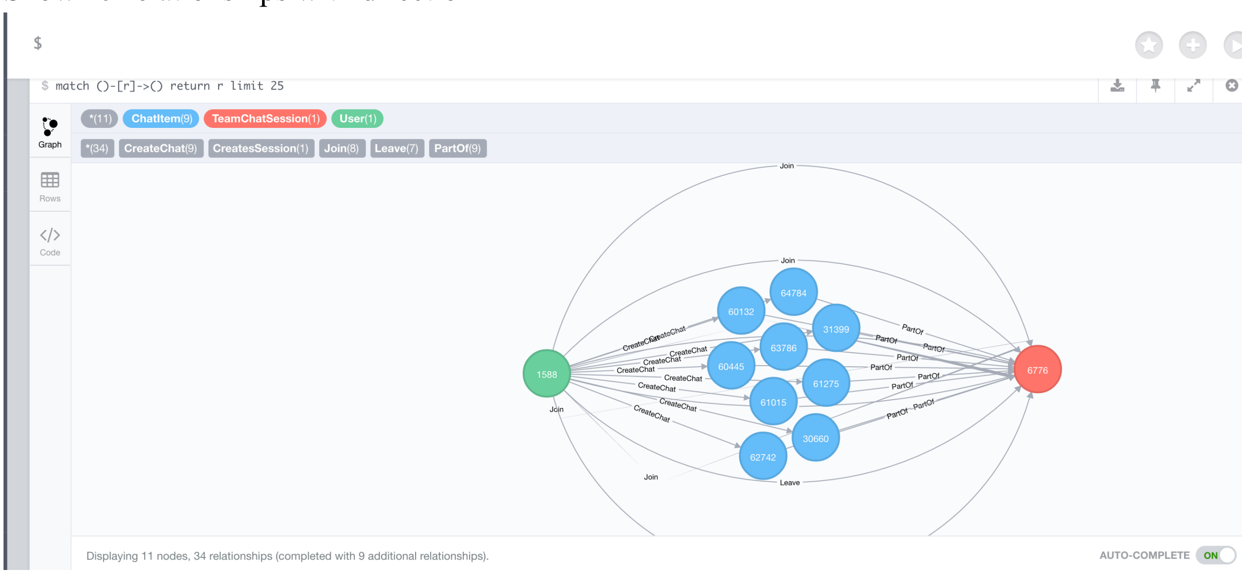
```
OPTIONAL MATCH (n)-[r]-()
DELETE n,r
 # create the constraint on nodes' primary key
CREATE CONSTRAINT ON (u:User) ASSERT u.id IS UNIQUE;
CREATE CONSTRAINT ON (t:Team) ASSERT t.id IS UNIQUE;
CREATE CONSTRAINT ON (c:TeamChatSession) ASSERT c.id IS UNIQUE;
CREATE CONSTRAINT ON (i:ChatItem) ASSERT i.id IS UNIQUE;
# load chat_create_team_chat.csv
LOAD CSV FROM "file:///Users/iBowen/Desktop/chat-
data/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)

# load chat_join_team_chat.csv
LOAD CSV FROM "file:///Users/iBowen/Desktop/chat-
data/chat_join_team_chat.csv" AS row MERGE (u:User {id: toInt(row[0])}) MERGE
(c:TeamChatSession {id: toInt(row[1])}) MERGE (u)-[:Join{timeStamp: row[2]}]-
>(c)

# load chat_leave_team_chat.csv
LOAD CSV FROM "file:///Users/iBowen/Desktop/chat-
data/chat_leave_team_chat.csv" AS row MERGE (u:User {id: toInt(row[0])})
MERGE (c:TeamChatSession {id: toInt(row[1])}) MERGE (u)-[:Leave{timeStamp:
row[2]}]->(c)

# chat_item_team_chat.csv
LOAD CSV FROM "file:///Users/iBowen/Desktop/chat-
data/chat_item_team_chat.csv" AS row MERGE (u:User {id: toInt(row[0])})
MERGE (c:TeamChatSession {id: toInt(row[1])}) MERGE (i:ChatItem {id:
toInt(row[2])}) MERGE (u)-[:CreateChat{timeStamp: row[3]}]->(i) MERGE (i)-
[:PartOf{timeStamp: row[3]}]->(c)

# chat_mention_team_chat.csv
LOAD CSV FROM "file:///Users/iBowen/Desktop/chat-
data/chat_mention_team_chat.csv" AS row MERGE (i:ChatItem {id: toInt(row[0])})
MERGE (u:User {id: toInt(row[1])}) MERGE (i)-[:Mentioned {timeStamp:
row[2]}]->(u)



# chat_respond_team_chat.csv
LOAD CSV FROM "file:///Users/iBowen/Desktop/chat-
data/chat_respond_team_chat.csv" AS row MERGE (i:ChatItem {id: toInt(row[0])})
MERGE (j:ChatItem {id: toInt(row[1])}) MERGE (i)-[:ResponseTo {timeStamp:
row[2]}]->(j)
```

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 a 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.

Show 50 relationships without direction



Show 25 relationships with direction



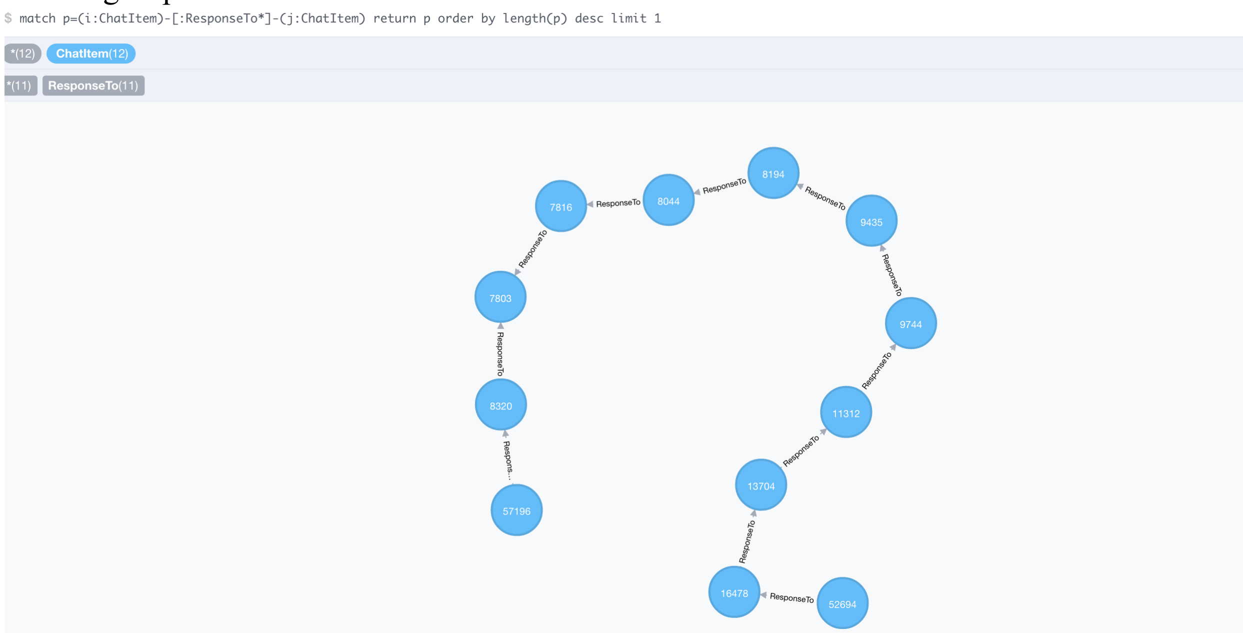
Finding the longest conversation chain and its participants

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.

step 1, find the longest path with edge of "ResponseTo"

match p=(i:ChatItem)-[:ResponseTo*]-(j:ChatItem) return length(p) order by length(p) desc limit 1
step 2, count the number distinct users who create ChatItem in the longest path
match p=(i:ChatItem)-[:ResponseTo*]-(j:ChatItem)
with p order by length(p) desc limit 1
match (u:User)-[:CreateChat]-(i)
where i in nodes(p)
return count(distinct u)

The longest path:



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.

Chattiest Users

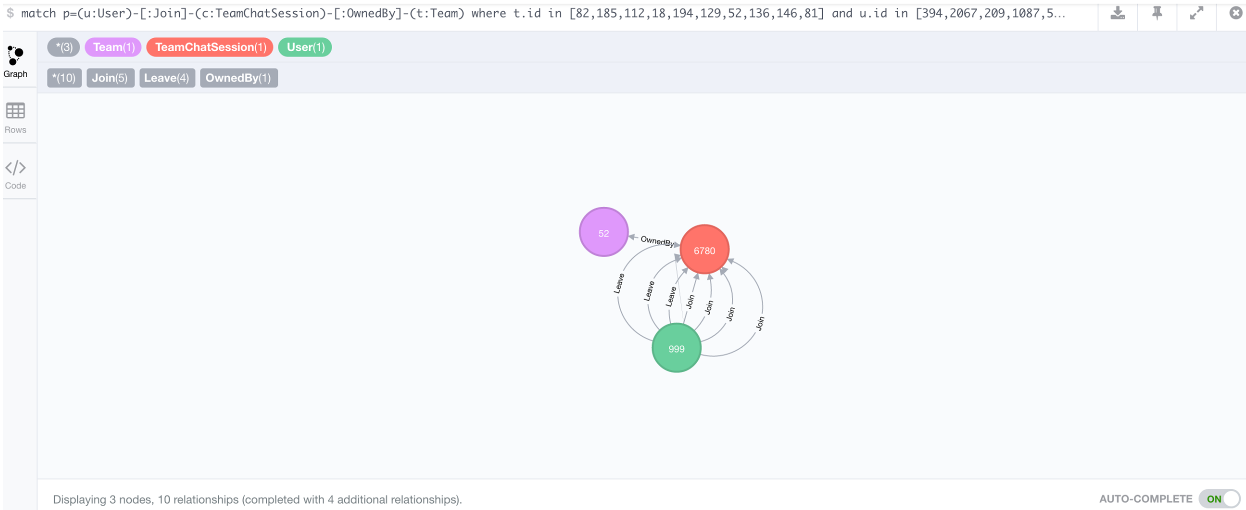
Users	Number of Chats
394	115
2067	111
209	109

Chattiest Teams

Teams	Number of Chats
82	1324
185	1036
112	957

Finally, present your answer.

match p=(u:User)-[:Join]-(c:TeamChatSession)-[:OwnedBy]-(t:Team) where t.id in [82,185,112,18,194,129,52,136,146,81] and u.id in [394,2067,209,1087,554,516,1627,999,668,461] return p
There is one chattest user id 999 who was part of one chattest team id 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.

Most Active Users (based on Cluster Coefficients)

UserId	Coefficient
461	1
209	0.9523809523809523
516	0.9523809523809523

1. connect two users if One mentioned another user in a chat
   Match (u1:User)-[:CreateChat]->(i:ChatItem)-[:Mentioned]->(u2:User) merge (u1)-[:InteractsWith]->(u2)
2. connect two users if One created a chatItem in response to another user's chatItem
   Match (u1:User)-[:CreateChat]->(i1:ChatItem)-[:ResponseTo]-(i2:ChatItem)<-[:CreateChat]-(u2:User) merge (u1)-[:InteractsWith]->(u2)
3. delete self-loop InteractsWith relationship
   match (u1)-[r:InteractsWith]->(u1) delete r
4. get neighbors and set degree based on the number of neighbors on the "InteractsWith" edge.
   match (u1:User)-[r:InteractsWith]-(u2:User) with u1, count(distinct u2) as degree
   set u1.deg = degree

return u1.id, u1.degreturn u1.id, u1.deg
5. get the number of links amongst neighbors, it includes the following steps:
find the neighbors users of one user node, and collect all distinct neighbor ids.
find all links amongst neighbors
count 1 if there are one or more relationships between two neighbors.

```
# show the user degree and add all links together
# calculate the clustering efficient

match (u1:User)-[:InteractsWith]-(u2:User)
with u1, collect(distinct u2.id) as neighbors
match (n:User)-[r:InteractsWith]->(m:User)
where (n.id in neighbors) and (m.id in neighbors)
with u1, case when (n)-->(m) then 1
else 0 end as value
with u1, u1.deg as deg, sum(value) as links
set u1.cc = toFloat(links) / (deg * (deg - 1))
return u1.id, u1.deg, u1.cc

match (u:User) where u.id in [394, 2067, 209, 1087, 554, 516, 1627, 999, 668, 461]
with u.id as id, u.deg as degree, u.cc as cluser_coefficient order by cluser_coefficient desc
return id, degree, cluser_coefficient
```

```
$ match (u:User) where u.id in [394, 2067, 209, 1087, 554, 516, 1627, 999, 668, 461] with u.id as id, u.deg as degree, u.cc as cluser_coefficient order...
```

Rows

Code

id	degree	cluser_coefficient
461	3	1
209	7	0.9523809523809523
516	7	0.9523809523809523
394	4	0.9166666666666666
999	10	0.8333333333333334
554	7	0.8095238095238095
2067	8	0.7678571428571429
1627	8	0.7678571428571429
1087	6	0.7666666666666667
668	5	0.7

Returned 10 rows in 47 ms.