

# I. Data Exploration with Splunk

## 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
<b>ad-clicks.csv</b>	A line is added to this file when a player clicks on an advertisement in the Flamingo app.	<ul style="list-style-type: none"><li>• timestamp: when the click occurred.</li><li>• txId: a unique id (within ad-clicks.log) for the click</li><li>• sessionId: the id of the user session for the user who made the click</li><li>• teamid: the current team id of the user who made the click</li><li>• userid: the user id of the user who made the click</li><li>• adId: the id of the ad clicked on</li><li>• adCategory: the category/type of ad clicked on</li></ul>
<b>buy-clicks.csv</b>	A line is added to this file when a player makes an in-app purchase in the Flamingo app.	<ul style="list-style-type: none"><li>• timestamp: when the purchase was made.</li><li>• txId: a unique id (within buy-clicks.log) for the purchase</li><li>• sessionId: the id of the user session for the user who made the purchase</li><li>• team: the current team id of the user who made the purchase</li><li>• userId: the user id of the user who made the purchase</li><li>• buyId: the id of the item purchased</li><li>• price: the price of the item purchased</li></ul>
<b>users.csv</b>	This file contains a line for each user playing the game.	<ul style="list-style-type: none"><li>• timestamp: when user first played the game.</li><li>• userId: the user id assigned to the user.</li><li>• nick: the nickname chosen by the user.</li></ul>

		<ul style="list-style-type: none"> <li>• twitter: the twitter handle of the user.</li> <li>• dob: the date of birth of the user.</li> <li>• country: the two-letter country code where the user lives.</li> </ul>
<b>Team</b>	This file contains a line for each team terminated in the game	<ul style="list-style-type: none"> <li>• teamId: the id of the team</li> <li>• name: the name of the team</li> <li>• teamCreationTime: the timestamp when the team was created</li> <li>• teamEndTime: the timestamp when the last member left the team</li> <li>• strength: a measure of team strength, roughly corresponding to the success of a team</li> <li>• currentLevel: the current level of the team</li> </ul>
<b>team-assignments.csv</b>	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.	<ul style="list-style-type: none"> <li>• timestamp: when the user joined the team.</li> <li>• team: the id of the team</li> <li>• userId: the id of the user</li> <li>• assignmentId: a unique id for this assignment</li> </ul>
<b>level-events.csv</b>	A line is added to this file each time a team starts or finishes a level in the game	<ul style="list-style-type: none"> <li>• timestamp: when the event occurred.</li> <li>• eventId: a unique id for the event</li> <li>• teamId: the id of the team</li> <li>• teamLevel: the level started or completed</li> <li>• eventType: the type of event, either start or end</li> </ul>
<b>user-session.csv</b>	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	<ul style="list-style-type: none"> <li>• timestamp: a timestamp denoting when the event occurred.</li> <li>• userSessionId: a unique id for the session.</li> <li>• userId: the current user's ID.</li> </ul>

	ended for each user in the team and a new one started.	<ul style="list-style-type: none"> <li>• teamId: the current user's team.</li> <li>• assignmentId: the team assignment id for the user to the team.</li> <li>• sessionType: whether the event is the start or end of a session.</li> <li>• teamLevel: the level of the team during this session.</li> <li>• platformType: the type of platform of the user during this session.</li> </ul>
<b>game-clicks.csv</b>	A line is added to this file each time a user performs a click in the game.	<ul style="list-style-type: none"> <li>• timestamp: when the click occurred.</li> <li>• clickId: a unique id for the click.</li> <li>• userId: the id of the user performing the click.</li> <li>• userSessionId: the id of the session of the user when the click is performed.</li> <li>• isHit: denotes if the click was on a flamingo (value is 1) or missed the flamingo (value is 0)</li> <li>• teamId: the id of the team of the user</li> <li>• teamLevel: the current level of the team of the user</li> </ul>

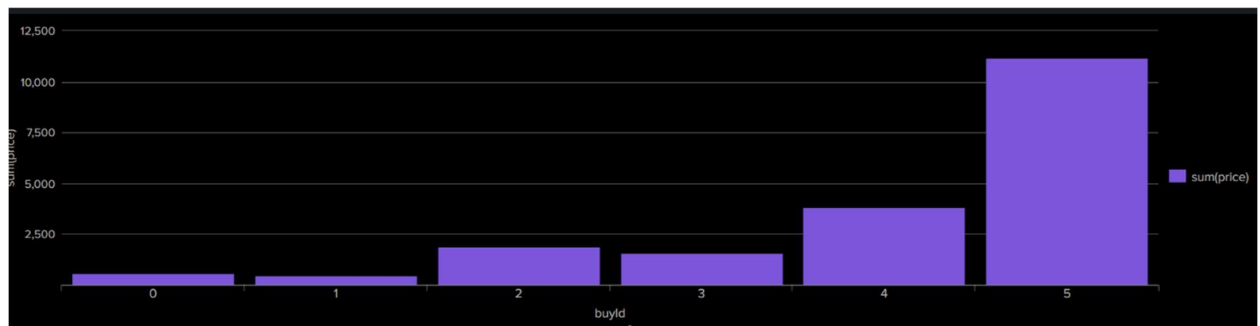
## Aggregation

Amount spent buying items	\$21407.0
Number of unique items available to be purchased	6 unique items [2, 4, 5, 1, 3, 0]

A histogram showing how many times each item is purchased:

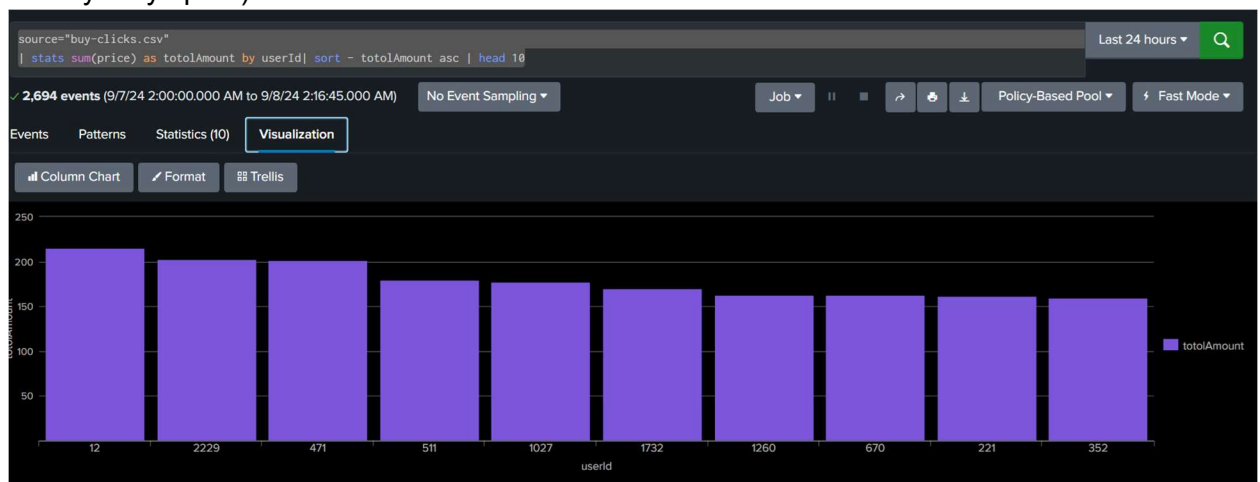


A histogram showing how much money was made from each item:



## 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	12	iphone	13
3	471	iphone	14.5

Table of result

	Rank	User Id	platform	Hit-Ratio (%)
0	1	2229	iphone	11.6
1	2	12	iphone	13.1
2	3	471	iphone	14.5

## II. Data Classification with KNIME

### 1. Data Preparation

Analysis of combined\_data.csv

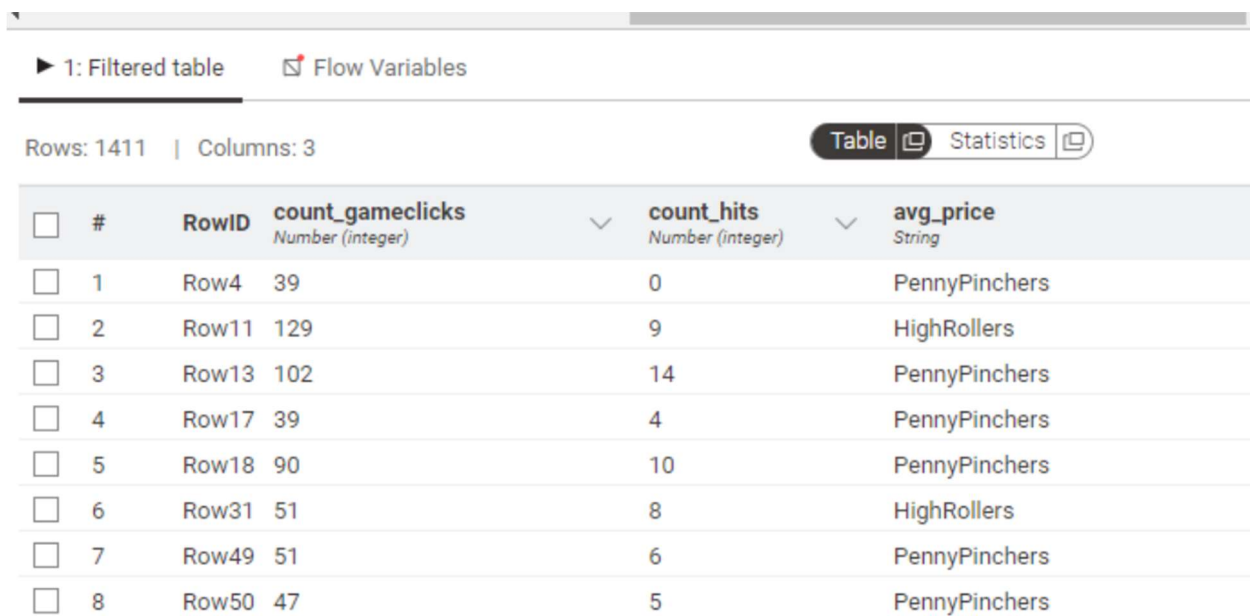
#### Sample Selection

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:



1: Filtered table    Flow Variables

Rows: 1411 | Columns: 3    Table    Statistics

<input type="checkbox"/>	#	RowID	count_gameclicks <i>Number (integer)</i>	count_hits <i>Number (integer)</i>	avg_price <i>String</i>
<input type="checkbox"/>	1	Row4	39	0	PennyPinchers
<input type="checkbox"/>	2	Row11	129	9	HighRollers
<input type="checkbox"/>	3	Row13	102	14	PennyPinchers
<input type="checkbox"/>	4	Row17	39	4	PennyPinchers
<input type="checkbox"/>	5	Row18	90	10	PennyPinchers
<input type="checkbox"/>	6	Row31	51	8	HighRollers
<input type="checkbox"/>	7	Row49	51	6	PennyPinchers
<input type="checkbox"/>	8	Row50	47	5	PennyPinchers

Describe the design of your attribute:

I want to include some fields but it make decision stop at 2. With changing some combine of columns, I see this seemingly better.

- The creation of this new categorical attribute was necessary because:

Avg-price is stored in string, so I need a step to convert it to numeric. We will classify in this column.

### Attribute Selection

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

Attribute	Rationale for Filtering
Avg_price	With this dataset, I have number of samples is bigger than unique user. I can directly analysis or I can make an flow with groupby user and avg of avg_price.
Count_gameclicks	In time of multitask, this is an good attribute to see how long and how much they attend to game, or they can be tracked by our location ads, or an coffee shop working.
Count_hits	Although distraction, they can have high skill or effort or combine action and sense.

## 2. Data Partitioning and Modeling

The data was partitioned into train and test datasets.

The 60% data set was used to create the decision tree model.

The trained model was then applied to the 40% dataset.

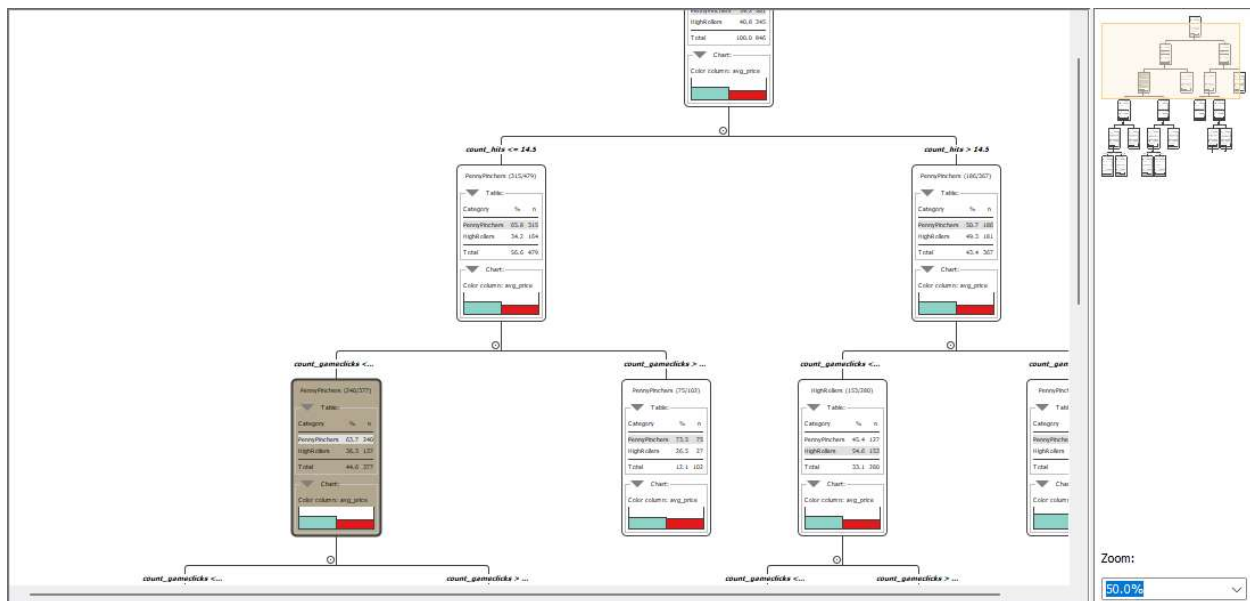
This is important because... you want to test your model on data that was not used to train

When partitioning the data using sampling, it is important to set the random seed because...

This ensures that you will get the same partitions every time you execute this node.

This is important to get reproducible results

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



### 3. Evaluation

A screenshot of the confusion matrix can be seen below:

avg_price ...	PennyPinc...	HighRollers
PennyPinchers	256	79
HighRollers	165	65

Correct classified: 321

Accuracy: 56.814%

Cohen's kappa ( $\kappa$ ): 0.05%

Wrong classified: 244

Error: 43.186%

As seen in the screenshot above, the overall accuracy of the model is 56.814

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

256: PennyPinchers can be correctly predicted.

65: HighRollers is harder.

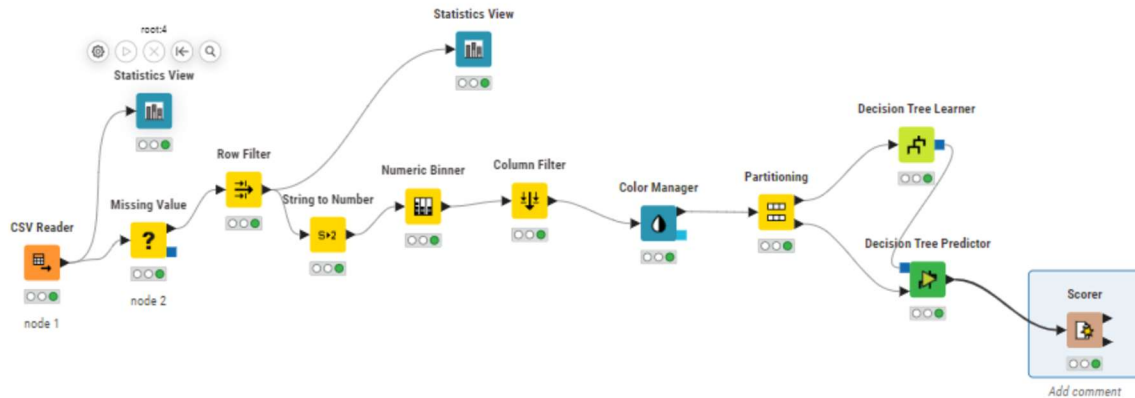
165: HighRollers is easy to be mistake with PennyPinchers. Classifying Negative class is problematic.

79: PennyPinchers hard to be mistaken.



## 4. Analysis Conclusions

The final KNIME workflow is shown below:



What makes a HighRoller vs. a PennyPincher?

I want to make further with average by userId than userSessionId. When take a describe with count\_buyID\*avg\_price then sum with group by userId, I saw that mean is different from 50% and can be say that take 65%. I think this a better guide to make next analysis with changing break point (\$7 for example).

Specific Recommendations to Increase Revenue
1. Focus on PennyPincher, they take a big in population and get good predicted.
2. With HighRoller: if change breakpoint, I believe in a better decision.

### III. Clustering with Spark

#### 1. Attribute Selection

featuresUsed = ['totalAmount', 'adClicks']

With 2 main intake, how can they save their time or stop click ads and pay more?

Attribute	Rationale for Selection
totalAmount	How one ready to pay
Hit-Ratio	I want to see how far it relate to totalAmount and adClick
Platform	What device they choose or ready to change effect to Hit-ratio
adClick	One of attribute relate to our revenue

#### 2. Training Data Set Creation

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

```
df.head(5)
```

✓ 0.0s

	userId	totalAmount	adClicks
0	1	21.0	44.0
1	8	53.0	10.0
2	9	80.0	37.0
3	10	11.0	19.0
4	12	215.0	46.0

Dimensions of the final data set:

```
• df.shape
```

```
✓ 0.0s
```

```
(546, 3)
```

# of clusters created:

```
centers = model.clusterCenters()  
centers
```

```
✓ 0.0s
```

```
[array([-0.21938193,  0.83070563]),  
 array([2.02480264,  0.77616293]),  
 array([-0.42108248, -0.92195285])]
```

### 3. Cluster Centers

The code used in creating cluster centers is given below:

```
from pyspark.sql import SQLContext
from pyspark.ml.clustering import KMeans
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.feature import StandardScaler
from pyspark import SparkContext
```

✓ 1.2s

+ Code

+ Markdown

```
try:
    sc = SparkContext("local", "My App")
except:
    print("sc is existing")
sqlContext = SQLContext(sc)
data = sqlContext.read.load(r"D:\Personality\Nguyen\Bigdata\big_data\big_data.csv",
                             format='com.databricks.spark.csv',
                             header='true',inferSchema='true')
```

```
data.count()
```

✓ 0.4s

546

```
data
```

✓ 0.0s

```
DataFrame[totalAmount: int, adClicks: int]
```

```
data.describe().toPandas().transpose()
```

✓ 0.9s

	0	1	2	3	4
summary	count	mean	stddev	min	max
totalAmount	546	39.20695970695971	41.154656029925626	1	223
adClicks	543	29.373848987108655	15.216343215129637	1	67

```
data = data.na.drop()
```

✓ 0.0s

```
data.columns
```

✓ 0.0s

```
['totalAmount', 'adClicks']
```

```
data
```

✓ 0.0s

```
DataFrame[totalAmount: int, adClicks: int]
```

```
featuresUsed = ['totalAmount', 'adClicks']  
assembler = VectorAssembler(inputCols=featuresUsed, outputCol="features_unscaled")  
assembled = assembler.transform(data)
```

✓ 0.0s

+ Code + Markdown

```
scaler = StandardScaler(inputCol="features_unscaled", outputCol="features", withStd=True, w  
scalerModel = scaler.fit(assembled)  
scaledData = scalerModel.transform(assembled)
```

✓ 0.7s

```
scaledData = scaledData.select("features")  
scaledData.persist()
```

✓ 0.0s

```
DataFrame[features: vector]
```

▷ ▾

```
kmeans = KMeans(k=3, seed=1)  
model = kmeans.fit(scaledData)  
transformed = model.transform(scaledData)
```

[16] ✓ 1.3s

+ Code + Markdown

```
centers = model.clusterCenters()
centers
```

✓ 0.0s

```
[array([-0.21938193,  0.83070563]),
 array([2.02480264,  0.77616293]),
 array([-0.42108248, -0.92195285])]
```

Cluster centers formed are given in the table below

Cluster #	Center
1	[-0.21938193, 0.83070563]
2	[2.02480264, 0.77616293]
3	[-0.42108248, -0.92195285]

These clusters can be differentiated from each other as follows:

Cluster 1 is different from the others in that the users who play the most are not the ones who produce the most revenue, the revenue in the middle along with the ad click count

Cluster 2 is different from the others in that users with the high revenue and adclick are not the ones who are playing the most but an intermediate result in game clicks.

Cluster 3 is different from the others in that the users who play the less also produces the less ad revenue and click count

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

```
data.describe().toPandas().transpose()
```

✓ 0.9s

	0	1	2	3	4
summary	count	mean	stddev	min	max
totalAmount	546	39.20695970695971	41.154656029925626	1	223
adClicks	543	29.373848987108655	15.216343215129637	1	67

#### 4. Recommended Actions

Action Recommended	Rationale for the action
isHit have a same, adjust	willing pay will increase bag so fast
Device and their upgrade	Battery and connection, where they will stop to catch
adClicks in position	Where they love to click ad and buy something in real life

## IV. Graph Analytics

### Graph Analytics

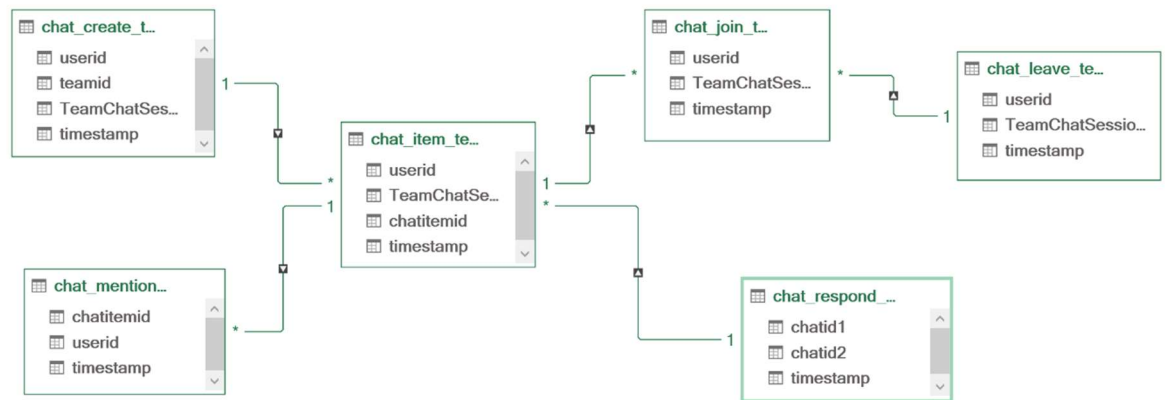
#### Modeling Chat Data using a Graph Data Model

Some created a chat and wrote session to chat\_create\_team\_chat and chat\_item\_team\_chat. Another can join to a TeamChatSession then leave. Some can be mentioned. Respond can start from two chatitemid.

#### 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



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

Since they loop around these nodes and properties. For my version, commands were bit different from source code on-for and assert-require

```
1 // loading process
2 CREATE CONSTRAINT for (u:User) require u.id IS UNIQUE;
3 CREATE CONSTRAINT for (t:Team) require t.id IS UNIQUE;
4 CREATE CONSTRAINT for (c:TeamChatSession) require c.id IS UNIQUE;
5 CREATE CONSTRAINT for (i:ChatItem) require i.id IS UNIQUE;
6
```

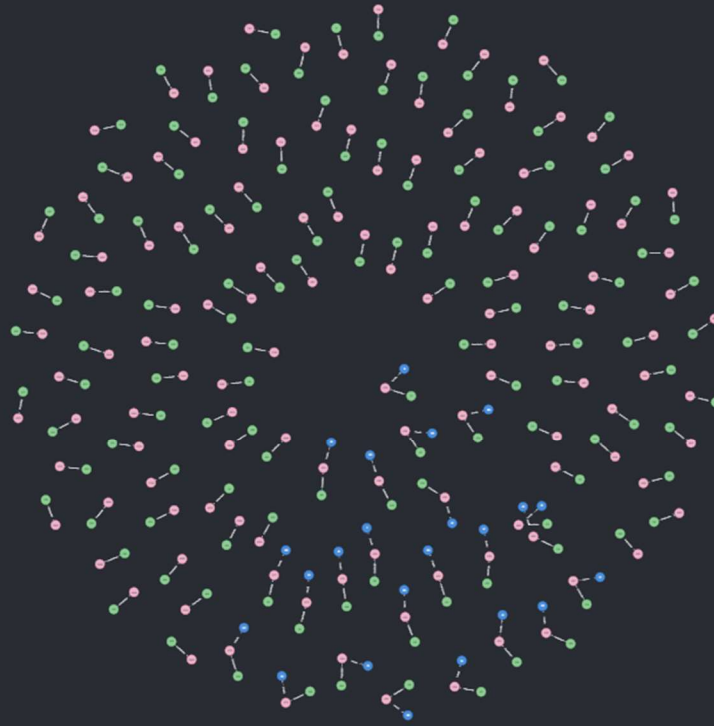
I added nodes and edges from create-file

```
1 LOAD CSV FROM "http://localhost:11001/project-39cd2376-844c-444d-b44a-d80aba8f6c5d/chat_create_team_chat.csv" AS row
2 MERGE (u:User {id: toInteger(row[0])}) MERGE (t:Team {id: toInteger(row[1])})
3 MERGE (c:TeamChatSession {id: toInteger(row[2])})
4 MERGE (u)-[:CreatesSession{timeStamp: row[3]}]-(c)
5 MERGE (c)-[:OwnedBy{timeStamp: row[3]}]-(t)
```

As expectation from unique values, we could see in graph



```
j$ MATCH (n)-[r]-(m) RETURN n,r,m limit 300
```



Sample load command for others

```
// sample for load create_team_chat
LOAD CSV FROM "http://localhost:11001/project-39cd2376-844c-444d-b44a-d80aba8f6c5d/chat_cr
MERGE (u:User {id: toInteger(row[0])}) MERGE (t:Team {id: toInteger(row[1])})
MERGE (c:TeamChatSession {id: toInteger(row[2])})
MERGE (u)-[:CreatesSession{timeStamp: row[3]}]->(c)
MERGE (c)-[:OwnedBy{timeStamp: row[3]}]->(t)
```

```
// sample for load join_team_chat
LOAD CSV FROM "http://localhost:11001/project-39cd2376-844c-444d-b44a-d80aba8f6c5d/chat_join
merge (u:User {id: toInteger(row[0])})
merge (c:TeamChatSession {id: toInteger(row[1])})
MERGE (u)-[:Join{timeStamp: row[2]}]->(c)
```

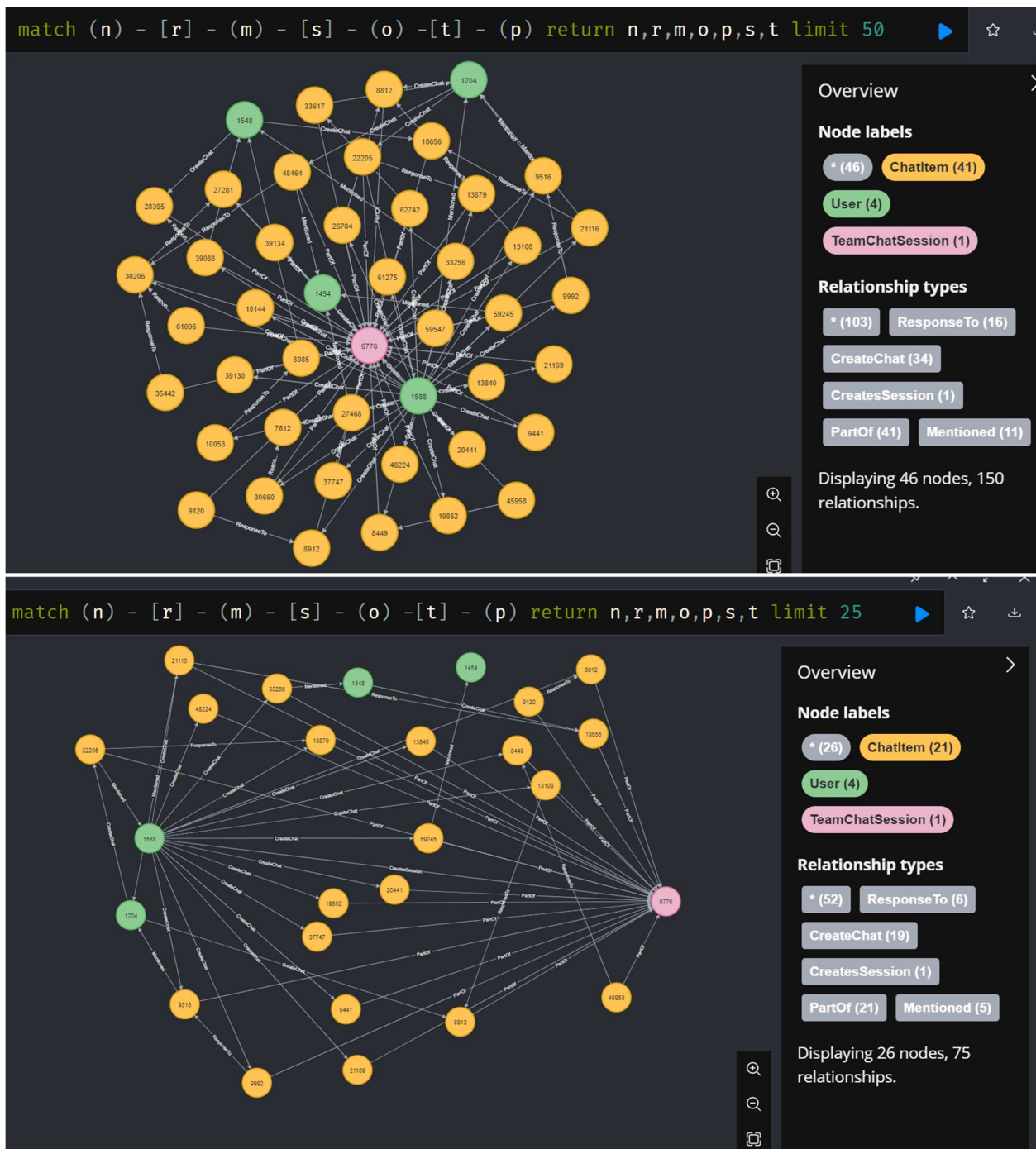
```
// sample for load leave_team_chat
LOAD CSV FROM "http://localhost:11001/project-39cd2376-844c-444d-b44a-d80aba8f6c5d/chat_leav
MERGE (u:User {id: toInteger(row[0])})
MERGE (c:TeamChatSession {id: toInteger(row[1])})
MERGE (u)-[:Leaves{timeStamp: row[2]}]->(c)
```

```
// sample for load item_team_chat
LOAD CSV FROM "http://localhost:11001/project-39cd2376-844c-444d-b44a-d80aba8f6c5d/chat_iter
MERGE (u:User {id: toInteger(row[0])})
MERGE (i:ChatItem {id: toInteger(row[2])})
MERGE (c:TeamChatSession {id: toInteger(row[1])})
MERGE (i)-[:PartOf{timeStamp: row[3]}]->(c)
MERGE (u)-[:CreateChat{timeStamp: row[3]}]->(i)
```

```
// sample for load mention_team_chat
LOAD CSV FROM "http://localhost:11001/project-39cd2376-844c-444d-b44a-d80aba8f6c5d/chat_ment
MERGE (i:ChatItem {id: toInteger(row[0])})
MERGE (u:User {id: toInteger(row[1])})
MERGE (i)-[:Mentioned{timeStamp: row[2]}]->(u)
```

```
// sample for load respond_team_chat
LOAD CSV FROM "http://localhost:11001/project-39cd2376-844c-444d-b44a-d80aba8f6c5d/chat_resp
MERGE (i0:ChatItem {id: toInteger(row[0])})
MERGE (i1:ChatItem {id: toInteger(row[1])})
MERGE (i0)-[:ResponseTo{timeStamp: row[2]}]->(i1)
```

- 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.



## 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.

```
match p=(n:ChatItem)-[r:ResponseTo*]-(m:ChatItem)
return length(p)
order by length(p) desc limit 1
```

	length(p)
1	11

Image iii.1: path length of longest path

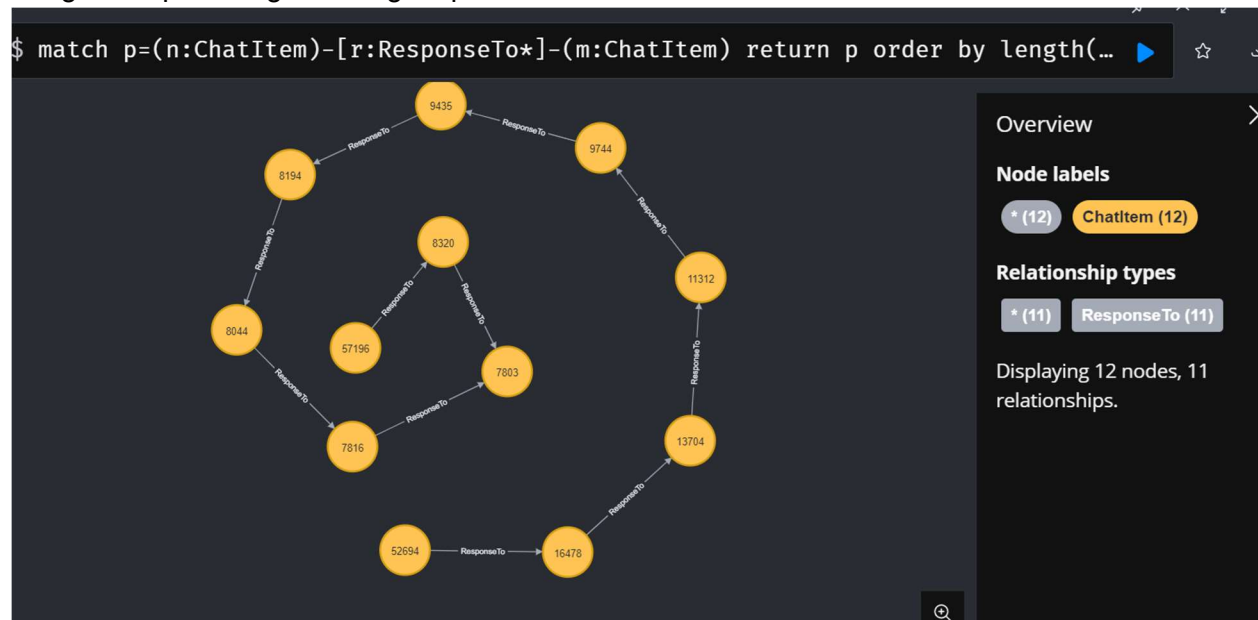


image 2:result including the length of the conversation

```
match p=(n:ChatItem)-[r:ResponseTo*]-(m:ChatItem)
where n.id=52694 and m.id=57196
RETURN [n IN NODES(p) | n.id] AS Paths
```

	Paths
1	[52694, 16478, 13704, 11312, 9744, 9435, 8194, 8044, 7816, 7803, 8320, 57196]

image3: 12 unique users were part of the conversation chain

## 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

```
// you need to
// MATCH all Users with a CreateChat edge,
match (n:User) - [r>CreateChat] - ()
// return the Users (their ID)
// and the count of the Users
return n.id, count(n.id) as userCount
// and sort them in descending order
order by userCount desc
// and limit the results to 10.
limit 10
```

	n.id	userCount
1	394	115
2	2067	111
3	1087	109

Users	Number of Chats
394	115
2067	111
1087	109

### Chattiest Teams

```
1 match (n:ChatItem) -[r:PartOf] → (m:TeamChatSession) - [s:OwnedBy] → (o)
2 return m.id, count(m.id) as teamCount
3 order by teamCount desc
4 limit 10
```

	m.id	teamCount
1	6792	1324
2	6783	1036
3	6925	957

Teams	Number of Chats
6792	1324
6783	1036
6925	957



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

```
teamList = [6792,6783,6925,6791,6974,6889,6780,6819,6850,6778]
✓ 0.0s

teamWithUser = [6875, 6820, 6784, 6781, 6787]
✓ 0.0s

for i in teamWithUser:
| print(i in teamList)
✓ 0.0s

False
False
False
False
False
```

I didn't find a cross between chattiest users in chattiest team. -

## 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.

Collect neighbors

```
// collect neighbor
unwind [394,2067,1087,209,554,1627,999,516,668,461] as userid
match (n1:User{id:userid})-[r1:InteractsWith]->(n2:User)
with n1.id as userid, collect(distinct n2.id) as neighbors
match (n:User{id:userid}) set n.neighbors = neighbors
```

Then write k to each in chattiest user

```
// write k to node
match (n:ChattiestItem)
unwind n.chattiestUser as userid
match (m:User{id:userid})
unwind m.neighbors as neighbor
with m.id as id, count(neighbor) as edges
match (m2:User{id:id})
set m2.edges = edges
return m2.id, m2.edges
```

I want to test to see how to get edges between neighbors

```
// test for 1 neighbor
unwind [1782, 1997, 1012, 2011] as id
match (n:User{id:id}) - [r] -> (m:User)
where m.id in [1782, 1997, 1012, 2011] and m.id <> id
with distinct n.id as n_id, m.id as m_id, case when (n) --> (m) then 1 else 0 end as value
return count(value)
```

And then collect them all

```
// collect clustering coefficient
unwind [394,2067,1087,209,554,1627,999,516,668,461] as userid
match (n1:User{id:userid})-[r1:InteractsWith]->(n2:User)
unwind n1.neighbors as neighbor
match (n:User{id:neighbor}) - [r] -> (m:User)
where m.id in n1.neighbors
with distinct n1.id as id, n.id as n_id, m.id as m_id, case when (n)-->(m) then 1 else 0 end
with id as id, count(value) as newValue
match (n:User{id:id})
// newValue: number of edge between neighbors
// edges: number of neighbors or k
set n.clustering_coefficient = newValue*1.0/(n.edges*(n.edges-1))
return n.clustering_coefficient
```

Let see it in cypher

neo4j\$

```
5 where m.id in n1.neighbors
6 with distinct n1.id as id, n.id as n_id, m.id as m_id, case when (n)→(m) then
  1 else 0 end as value
7 with id as id, count(value) as newValue
8 match (n:User{id:id})
9 // newValue: number of edge between neighbors
10 // edges: number of neighbors or k
11 set n.clustering_coefficient = newValue*1.0/(n.edges*(n.edges-1))
12 return n.clustering_coefficient
```

	n.clustering_coefficient
1	0.75
2	0.9333333333333333
3	0.7333333333333333

Finally, we can easy to extract Cluster Coefficients because I wrote in in each chat member

```

1 match (n:ChattiestItem)
2 unwind n.chattiestUser as userid
3 match (m:User{id:userid})
4 return m.id as top3, m.clustering_coefficient as Coefficient
5 order by Coefficient desc
6 limit 3

```

Table

Text

Code

	top3	Coefficient
1	668	1.0
2	209	1.0
3	999	0.9464285714285714

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

User ID	Coefficient
668	1.0
209	1.0
999	0.95