

How can we increase revenue from Catch the Pink Flamingo

Technical Appendix

by: Michel Truyenque

Index

Index	2
Data Exploration with Splunk	3
Data set overview	3
Aggregation	5
Filtering	6
Data Classification with KNIME	8
Data Preparation	8
Sample Selection	8
Attribute Creation	8
Attribute Selection	9
Data Partition and Modelling	9
Evaluation	10
Analysis and Conclusions	11
Clustering with Spark	13
Attribute Selection	13
Training Data Set Creation	13
Cluster Centers	14
Recommended Actions	14
Graph Analytics of Chat Data with Neo4j	16
Modeling Chat Data using a Graph Data Model	16
Creation of the Graph Database for Chats	16
Longest conversation chain and its participants	18
Analyzing the relationship between top 10 chattiest users and top 10 chattiest teams	19
How Active Are Groups of Users?	21

1. Data Exploration with Splunk

1.1. Data set overview

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

File Name	Description	Fields
ad-clicks.csv	A line is added to this file when a player clicks on an advertisement in the Flamingo app.	<ul style="list-style-type: none">● timestamp: when the click 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.	<ul style="list-style-type: none">● 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

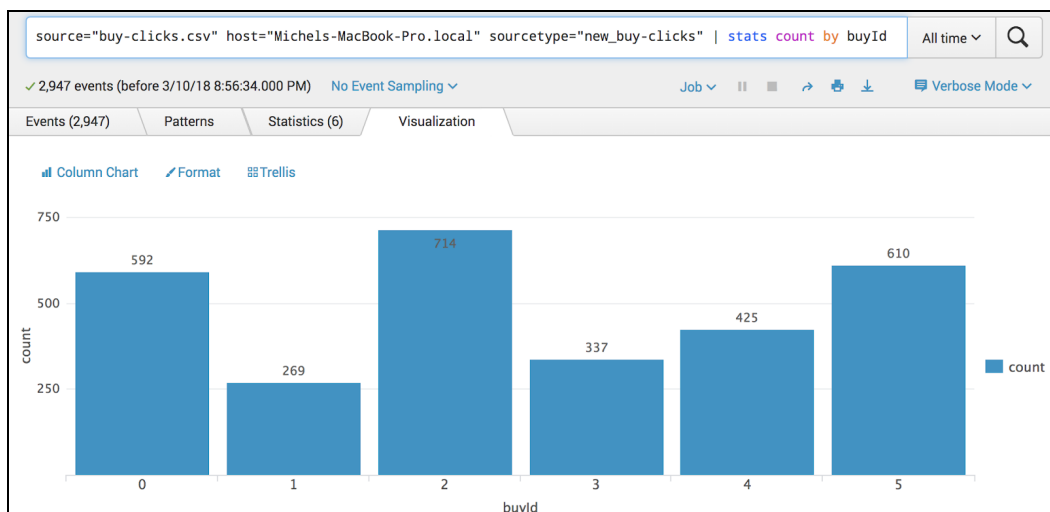
team.csv	This file contains a line for each team terminated in the game.	<ul style="list-style-type: none"> ● teamId: the id of the team ● name: the name of the team ● teamCreationTime: the timestamp when the team was created ● teamEndTime: the timestamp when the last member left the team ● strength: 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.	<ul style="list-style-type: none"> ● timestamp: 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.
level-events.csv	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"> ● timestamp: when the event occurred. ● eventId: a unique id for the event ● teamId: the id of the team ● teamLevel: the level started or completed ● 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.	<ul style="list-style-type: none"> ● 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.

		<ul style="list-style-type: none"> ● platformType: the type of platform of the user during this session.
game-clicks.csv	A line is added to this file each time a user performs a click in the game.	<ul style="list-style-type: none"> ● timestamp: 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

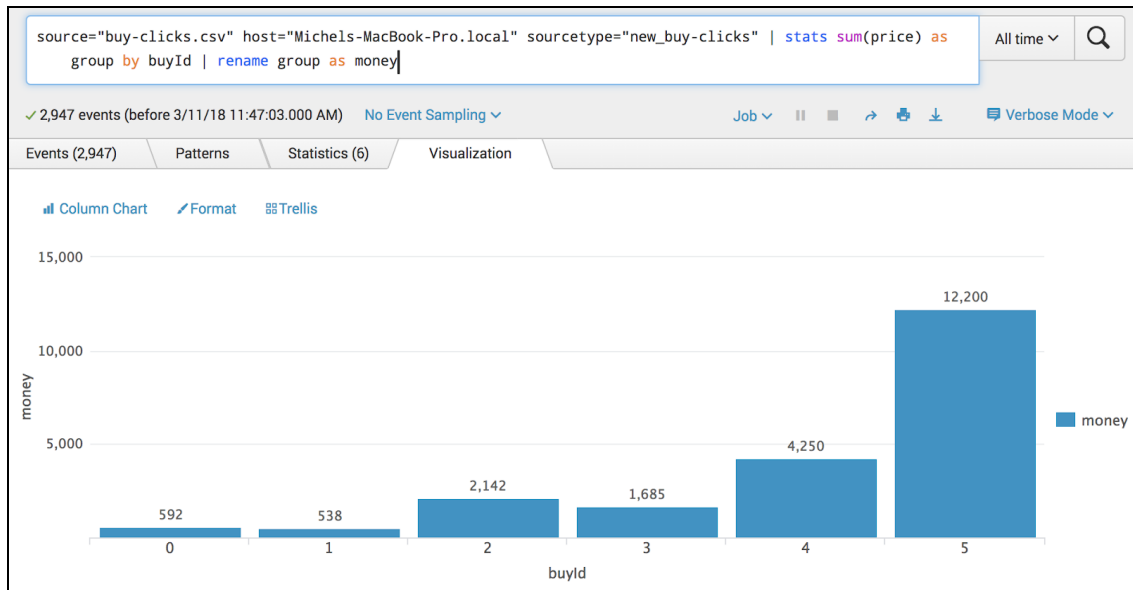
1.2. Aggregation

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

A histogram showing how many times each item is purchased:

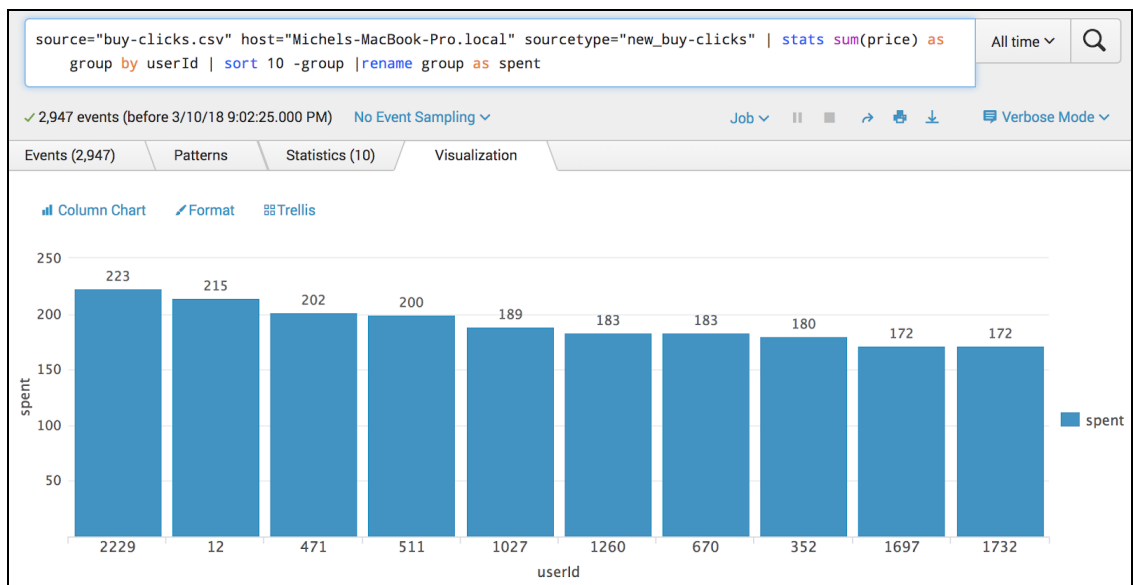


A histogram showing how much money was made from each item:



1.3. 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	11.59
2	12	iphone	13.06
3	471	iphone	14.50

2. Data Classification with KNIME

2.1. Data Preparation

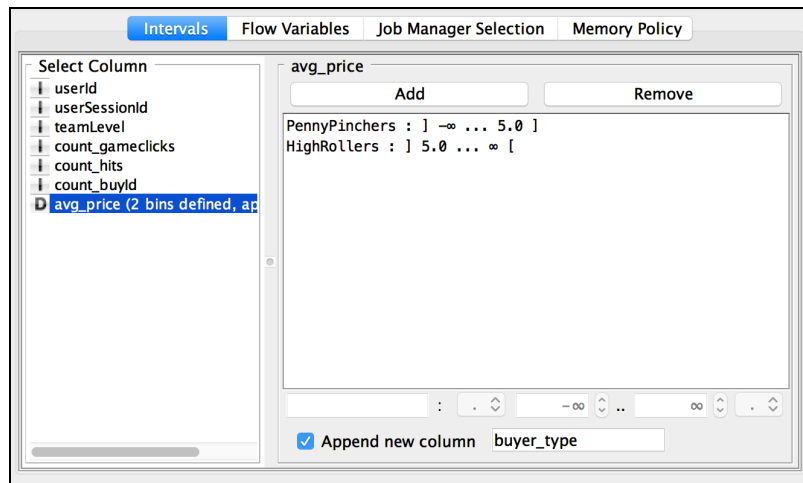
Analysis of combined_data.csv

2.1.1. Sample Selection

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

2.1.2. 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:



It was created a new categorical variable named “buyer_type”. It is derived directly from the avg_price attribute and generates two bins (categories). PennyPinchers for those with purchases ≤ 5.0 and HighRollers for those with purchases > 5.0 . This categorical variable is appended as a new column as shown in the creation dialog.

The creation of this new categorical attribute was necessary to categorize (to label) current rows considering the purchased amount, and map this value into a desired types of users (PennyPinchers and HighRollers) in our case study.

2.1.3. Attribute Selection

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

Attribute	Rationale for Filtering
avg_price	The categorical variable buyer_type has been generated based in this attribute, so considering it in our classification process would be redundant.
userId	This attribute identify the user saying who is the user (It can be repeated), but it does not influence the category a user could be considered based on their purchases.
sessionId	This attribute identify the session. Sessions do not influence how the user performs and also their purchase behavior.

2.2. Data Partition and Modelling

The data was partitioned into train and test datasets.

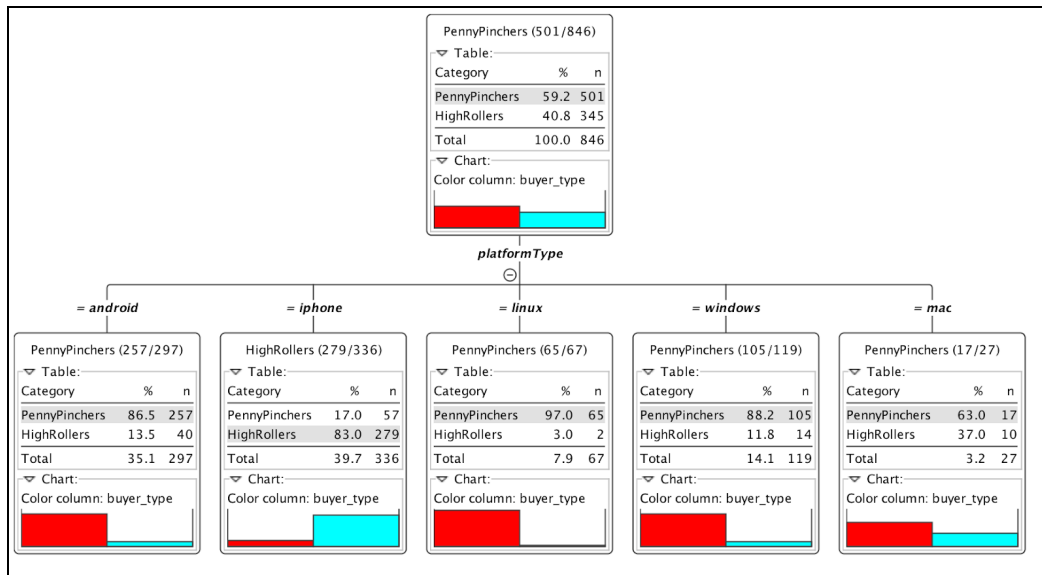
The 60% training data set was used to create the decision tree model.

The decision tree model was then applied to the remaining 40% test dataset.

To split the data into a training and test datasets is important because it allows us to build a model based on the training data and then apply this model to the remaining rows (test data set) to measure the accuracy of classification of our model.

When partitioning the data using sampling, it is important to set the same random seed because we can have consistent (same) data partitions when partition node is executed.

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



- [root]: class 'PennyPinchers' (501 of 846)
- [platformType = android]: class 'PennyPinchers' (257 of 297)
- [platformType = iphone]: class 'HighRollers' (279 of 336)
- [platformType = linux]: class 'PennyPinchers' (65 of 67)
- [platformType = windows]: class 'PennyPinchers' (105 of 119)
- [platformType = mac]: class 'PennyPinchers' (17 of 27)

2.3. Evaluation

A screenshot of the confusion matrix can be seen below:

Confusion Matrix - 2:11 - Scorer		
File	Hilite	
buyer_type \ Prediction...	PennyPinchers	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%**.

From the confusion matrix we can see that from all 565 users (rows) used as test dataset (308+38) were classified as PennyPinchers and (27+ 192) were classified as HighRollers.

It is important to validate our model in order to verify for each user, if the classification model predicts its category accurately. As the accuracy of our model is not 100% it suggest that some users were classified with the wrong category.

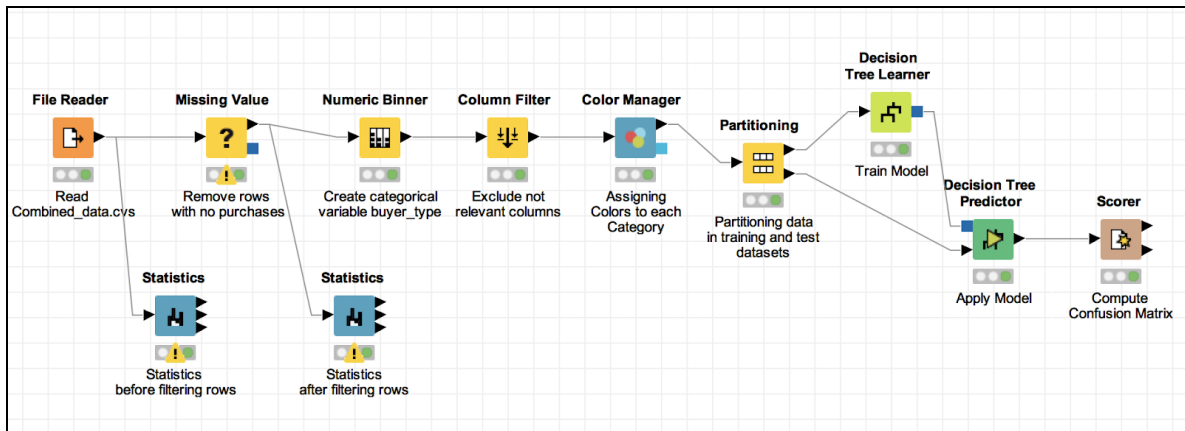
Indeed, for PennyPinchers category it shows that 308 users were classified correctly; but 38 users (originally HighRollers) were wrongly classified as PennyPinchers.

For HighRollers category it shows that 192 users were classified correctly; but 27 users (originally PennyPinchers) were wrongly classified as HighRollers.

Notice that the number of wrong classified users in both categories ($38 + 27 = 65$) represent exactly the 11.50% error of our model.

2.4. Analysis and Conclusions

The final KNIME workflow is shown below:



What makes a HighRoller vs. a PennyPincher?

According to the resulting model, we infer that `user_category` is highly affected by the type of platform the user is gaming. PennyPinchers are highly correlated to users that use android, linux, windows and mac platforms. While HighRollers category is highly correlated to Iphone platform users. Certainly there are a set of game features in iphone platform that makes users to be attracted more to the game and purchase more products as they advance in the game levels.

Specific Recommendations to Increase Revenue
1. As there is a high concentration of PennyPinchers on linux, windows, android and mac platforms. We can offer some promotions/rewards for these specific platforms in order to increase the game engagement and purchases. For instance, maybe it is possible to reproduce the same Iphone gaming experience and features in a Mac platform, so we can increase our HighRoller user % in this platform.
2. For Iphone users, we could increase the number of products offered in this platform. Also as Iphone platform is a high revenue platform, we can open a kind of incentive and rewards for iphone users who recommend the game to friends using an Iphone platform as well.

3. Clustering with Spark

3.1. Attribute Selection

Attribute	Rationale for Selection
Total ads clicks per user	According this attribute, we can compare users based on total amount of ads clicks they made. It captures users ads clicks behavior for all ads categories.
Total game clicks per user	According this attribute, we can compare users also based on total amount of game clicks they made in all interaction with the game. It captures user game clicks behavior for all time users interacted with the game.
Total revenue per user	According this attribute, we can capture total sum of money spent by users in all categories.

3.2. Training Data Set Creation

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

```
In [33]: trainingDF = trainingDF[['totalAdClicks', 'totalGameClicks', 'totalPurchase']]
trainingDF.show(5)
trainingDF.count()
```

```
+-----+-----+-----+
|totalAdClicks|totalGameClicks|totalPurchase|
+-----+-----+-----+
|          19|          365|          63.0|
|          39|          683|          20.0|
|          37|          287|          28.0|
|          34|          794|          95.0|
|          41|          644|          53.0|
+-----+-----+-----+
only showing top 5 rows
```

```
Out[33]: 543
```

Dimensions of the training data set (rows x columns) : **543 x 3**

of clusters created: **2**

3.3. Cluster Centers

Cluster #	Cluster Center
1	array([-0.81771878, -0.37749641, -0.4354828])
2	array([0.84528233, 0.39022101, 0.45016199])

These clusters can be differentiated from each other as follows:

The first number (**field1**) in each array refers to the scaled version of total number of ad_clicks per user; the second number (**field2**) refers to the scaled version of total game_clicks per user. The third number (**field 3**) refers to the scaled version of amount of money revenue per user.

Clustering our training set into two clusters, allowed to us to identify two big user groups. As expected they have big differences on all values and we could infer that the amount of revenue per user is directly influenced by the amount of clicks (on game and ads). Also we can see that low values in game clicking normally influence to low values in ads clicking. High number of game_clicks normally influence into high values on ads clicking.

Cluster 1 is different from the other, as players in this group have a low number of game clicks, wich traduce in less often ads clicking behavior reducing the revenue we have per user in this group.

Cluster 2 is different from the other, as players in this group have a high number of game clicks. As we can see, it potentially traduce into a more often ads clicking behavior increasing the revenue we have per each user in this group.

3.4. Recommended Actions

Action Recommended	Rationale for the action
Engage "low spending users" to spend more time gaming	As revenue is highly affected by the amount of game clicks user does during the game. We can provide some fixed pay packages or promotions that encourage them to increasing the time in gaming. Thus we potentially increase the number of game clicks and also the number of ads-clicks user does, increasing our revenue.

<p>Increase prices of ads shown to “high spending users”</p>	<p>As we get paid for showing ads, we could increase our revenue by building a price mechanism that would increase ads fees when shown to frequent clickers. As we have users clicking profile we can charge more to ads shown to top x% users, for instance.</p>
<p>Provide more purchase items to “high spending users”</p>	<p>We can diversify and also increase the purchase items offer to frequent clickers, so we potentially increase our revenue with additional products.</p>

4. Graph Analytics of Chat Data with Neo4j

4.1. Modeling Chat Data using a Graph Data Model

In the graph model for the game, a user in a team can create a chat session for the team he belongs to. Each team member can also create a new chat post (chatItem) for the specific session, and as the chat session extends, user can respond or also can be mentioned for other chat post from other users in same team. Users at any point in time can join or leave teams as well.

4.2. Creation of the Graph Database for Chats

1.The database for the graph analytics exercise consist of 6 .csv files as we detailed in the table below.

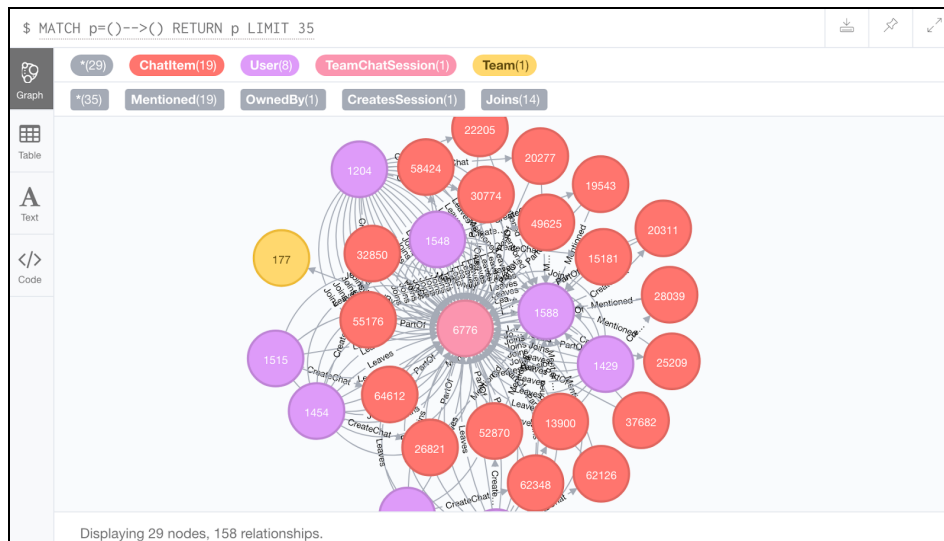
Files	Description	Attributes
chat_create_team_chat.csv	A line is added to this table when a user creates a new chat session for the team.	userId : user id who created the session. teamid : the id of the team. teamChatDataSessionId : unique id for the chat session. timestamp : time session was created.
chat_item_team_chat.csv	A line is added to this table when a user creates a new chat post for a specific team chat session.	userId : user id who created the post. teamChatDataSessionId : id for the chat session. chatItemId : unique id for chat post. timestamp : time post was created.
chat_join_team_chat.csv	A line is added to this table when a user joins to a team chat session	userId : user id joins the session. teamChatDataSessionId : id for the chat session. timestamp : time user joins session.
chat_leave_team_chat.csv	A line is added to this table when a user leaves from a team chat session.	userId : user id leaves the session. teamChatDataSessionId : id for the chat session. timestamp : time user leaves session.
chat_mention_team_chat.csv	A line is added to this table when a user is mentioned in a chat post.	userId : user id is mentioned. chatItemId : id for chat post. timestamp : time user was mentioned.
chat_respond_team_chat.csv	A line is added to this table when a chat post is in response to another chat post	chatItemId : id of chat item responses. chatItemId : id of chat item timestamp : time when chat post responded to the another.

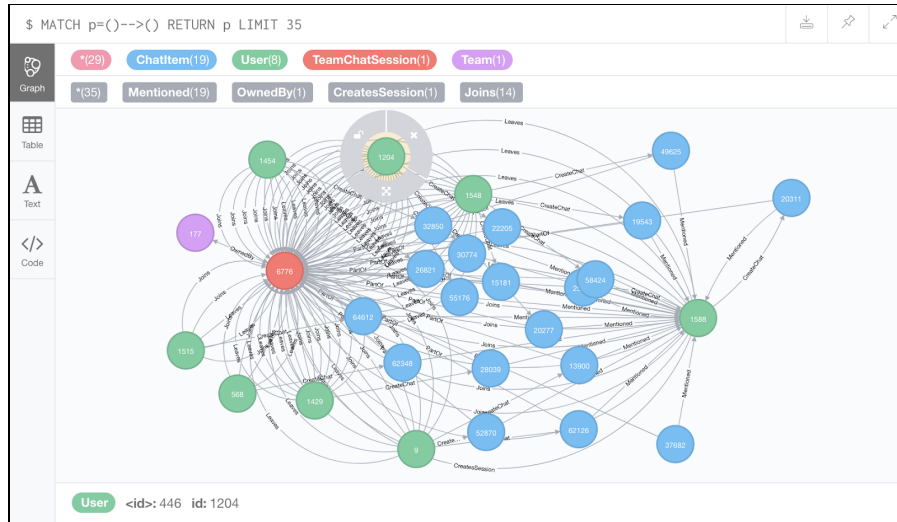
2. For the loading process we are gonna use Cypher language in order to read the .csv files with the LOAD command row by row. Then, we will use the MERGE command in order to create the specific nodes and edges are represented in the file. Let explain the process with the following example.

```
1 LOAD CSV FROM "file:///chat-data/chat_item_team_chat.csv" AS row
2 MERGE (u:User {id: toInteger(row[0])})
3 MERGE (c:TeamChatSession {id: toInteger(row[1])})
4 MERGE (t:ChatItem{id: toInteger(row[2])})
5 MERGE (u)-[:CreateChat{timeStamp: row[3]}]->(t)
6 MERGE (t)-[:PartOf{timeStamp: row[3]}]->(c)
```

The LOAD command reads the file row by row. For each row it creates a **User** node using as id attribute the column 0; a **TeamChatSession** node using as id attribute the column 1; and a **ChatItem** node using as id attribute the column 2. The MERGE command on line 5 crates a new edge (called “**CreateChat**”) between **User** and **TeamChatSession** nodes and sets the timeStamp value with the value of column 3. The MERGE command on line 6 crates a new edge (called “**PartOf**”) between **ChatItem** and **TeamChatSession** nodes and sets the timeStamp value with the value of column 4. Similar queries would be used to load the other files and merge commands will be modified to create nodes and edges as expected by the exercise.

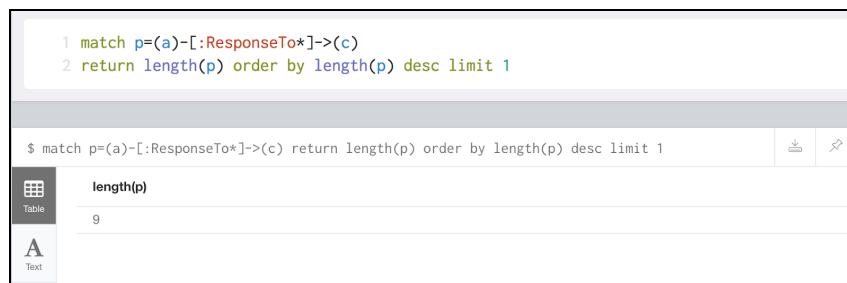
3. Following screen shots show the resulting graph we obtained after loading all 6 files in our Neo4j environment. The first example is a rendered in the default Neo4j distribution, the second has had some nodes moved to expose the edges more clearly.



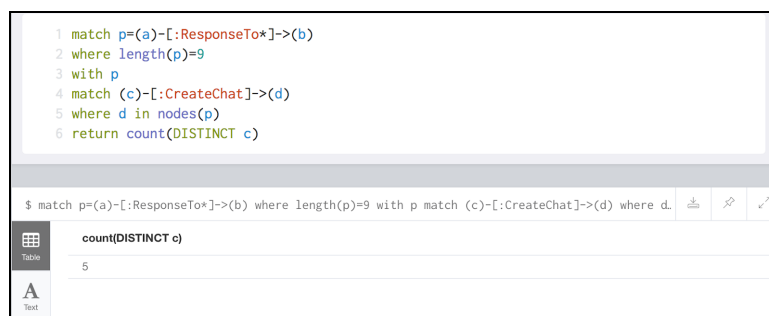


4.3. Longest conversation chain and its participants

1. The longest conversation chain can be found analyzing the connectivity of **ChatItem** nodes through the **ResponseTo** edges. The following query will find the longest conversation chain in the graph, which is **9**.



2. In order to get the number of unique users were part of the conversation chain. We can find all users are linked to **ChatItem** nodes in the longest chain. Via **CreateChat** edges, we can count the distinct users have created a post is part of longest conversation chain. The following query will give the value of **5** unique users.

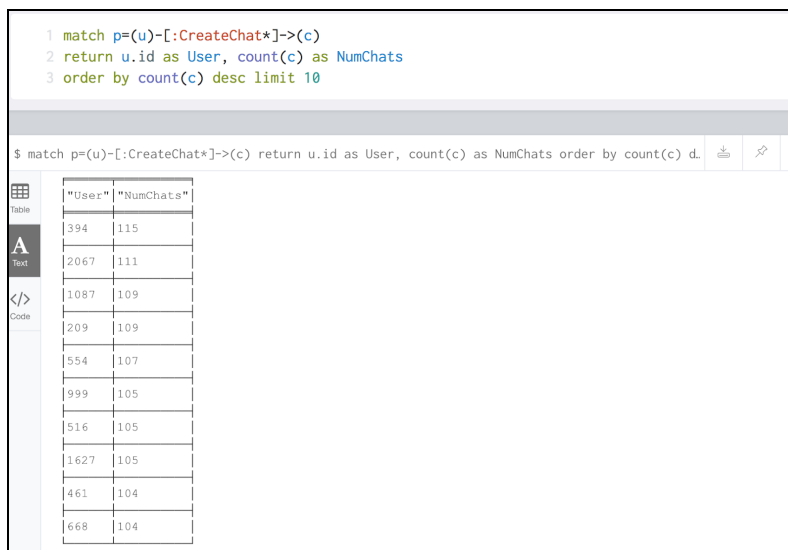


4.4. Analyzing the relationship between top 10 chattiest users and top 10 chattiest teams

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

We can find the top 10 chattiest **User** nodes using the **CreateChat** edge. Thus, we can count the number of **ChatItem** nodes each user has created. The following query will find the top 10 chattiest users.



```
1 match p=(u)-[:CreateChat*]->(c)
2 return u.id as User, count(c) as NumChats
3 order by count(c) desc limit 10
```

\$ match p=(u)-[:CreateChat*]->(c) return u.id as User, count(c) as NumChats order by count(c) d. [Icons]

"User"	"NumChats"
394	115
2067	111
1087	109
209	109
554	107
999	105
516	105
1627	105
461	104
668	104

Users	Number of Chats
394	115
2067	111
209 & 1087	109

Chattiest Teams

We can find the top 10 chattiest **Team** nodes using, as suggested in the exercise, all **ChatItem** nodes that has a **PartOf** edge connecting to a **TeamChatSession** node. Then, we can count the number of **TeamChatSession** nodes we find for each team. The following query will find top 10 chattiest teams.

```
1 match p= (c)-[:PartOf*]->(s)-[:OwnedBy*]->(t)
2 return t.id as Team, count(s) as NumChatSessions
3 order by count(s) desc limit 10
```

\$ match p= (c)-[:PartOf*]->(s)-[:OwnedBy*]->(t) return t.id as Team, count(s) as NumChatSession.

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

Teams	Number of Chats
82	1324
185	1036
112	957

To respond whether or not any of the chattiest user are part of any of the chattiest teams. We can list in a new query (below) the team where each top 10 chattiest users belongs. Thus, correlating this new table with the top 10 chattiest teams table (found previously); we can verify that just the user **999** belongs to one of chattiest teams, for instance team **52**. The following query will show the team each top 10 chattiest user belongs to.

```
1 match p=(u)-[:CreateChat*]->(c)-[:PartOf*]->(s)-[:OwnedBy*]->(t)
2 return u.id as User, t.id as Team, count(c) as NumChats
3 order by count(c) desc limit 10
```

\$ match p=(u)-[:CreateChat*]->(c)-[:PartOf*]->(s)-[:OwnedBy*]->(t) return u.id as User, t.id as

User	Team	NumChats
394	63	115
2067	7	111
209	7	109
1087	77	109
554	181	107
516	7	105
1627	7	105
999	52	105
461	104	104
668	89	104

4.5. How Active Are Groups of Users?

In this question, we will compute an estimate of how “dense” the neighborhood of a node is. In the context of chat that translates to how mutually interactive a certain group of users are. If we can identify these highly interactive neighborhoods, we can potentially target some members of the neighborhood for direct advertising. As suggested by the exercise we can follow the basic steps described below.

- a) We will construct a neighborhood of users creating new edges (called “*InteractsWith*”) between them. Following the queries that creates new edges for: users are mentioned for another users chat posts.

```
1 match (u1:User)-[:CreateChat]->(c)-[:Mentioned]->(u2:User)
2 create (u1)-[:InteractsWith]->(u2)
```

And when a user creates a new chat post in response to another user’s chat post. Here, for each user we are collecting all chat post created in response to chat items he created. Then, we are creating a new edge with the users owners of these chat post.

```
1 match (u1:User)-[:CreateChat]->(c:ChatItem)-[:ResponseTo]->(c1:ChatItem)
2 with u1, c, c1
3 match (u2:User)-[:CreateChat]->(c1)
4 create (u1)-[:InteractsWith]->(u2)
```

- b) As mentioned by the exercise, after these two queries it is necessary to delete some unnecessary load nodes may be created executing the previous queries.

```
$ match (u1)-[:InteractsWith]->(u1) delete r
```

- c) In order to calculate the clustering coefficient we can follow the steps as suggested by the exercise. Notice that same query can be used for all user nodes; but in our case we are just interested in the top 10 chattiest users. With help of the following query we will explain in more detail each step we do to obtain final coefficients.

First, we need to collect all neighbor nodes “*InteractsWith*” a specific chattiest user node (line 1). We collect nodes have different ids (line 4) in a list called “*neighbors*” and also count the number of different nodes as “*neighborCount*”. Second, for a list of “*neighbors*”, we collect the number of “*InteractsWith*” edges between them (lines 4,5,6). If any pair of neighbors nodes have more than one edge, we count it just as 1 (lines 7,8,9). Thus, *sum(hasedge)* (line 11) will have at the end the number of edges between neighbor nodes. Finally (in line 11), we calculate the clustering coefficient as indicates in the exercise.

```

1 match (u1:User)-[r1:InteractsWith]->(u2:User)
2 where u1.id <> u2.id
3 AND u1.id in [394,2067,1087,209,554,999,516,1627,461,668]
4 with u1, collect(u2.id) as neighbors, count(distinct(u2)) as neighborCount
5 match (u3:User)-[r2:InteractsWith]->(u4:User)
6 where (u3.id in neighbors) AND (u4.id in neighbors) AND (u3.id <> u4.id)
7 with u1, u3, u4, neighborCount,
8 case when count(r2) > 0 then 1
9 else 0
10 end as hasedge
11 return u1.id, sum(hasedge)*1.0/(neighborCount*(neighborCount-1)) as cCoeff order by
    cCoeff desc

```

Finally, report the top 3 most active users in the table below.

Most Active Users (based on Cluster Coefficients)

User ID	Coefficient
209	0.9523
554	0.9047
1087	0.8