EGLENCE INC.

A Case Study

Aryan Rastogi

September 2020

Part 1: Data Exploration

Software used: 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
ad-clicks.csv	Database of clicks on ads	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	Database of purchases	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 record of each click a user performed during the game	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
level-events.csv	A record of each level event for a team. Level events are recorded when a team ends or begins a new level	timestamp: when the event occurred. eventId: a unique id for the event teamId: the id of the team teamLevel: the level started or completed

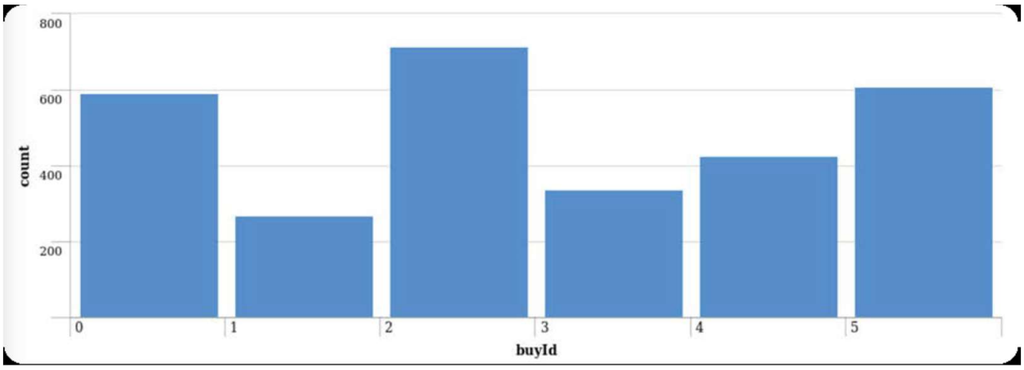
		eventType: the type of event, either start or end
team-assignments.csv	A record of each time a user joins a team.	timestamp: when the user joined the team. team: the id of the team userId: the id of the user assignmentId: a unique id for this assignment
team.csv	A record of each team in the game	teamId: 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
user-session.csv	A record of each session a user plays. When a team levels up, each current user session ends and a new session begins with a new level.	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.
users.csv	Database of the game users	timestamp: when user first played the game. userId: 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.

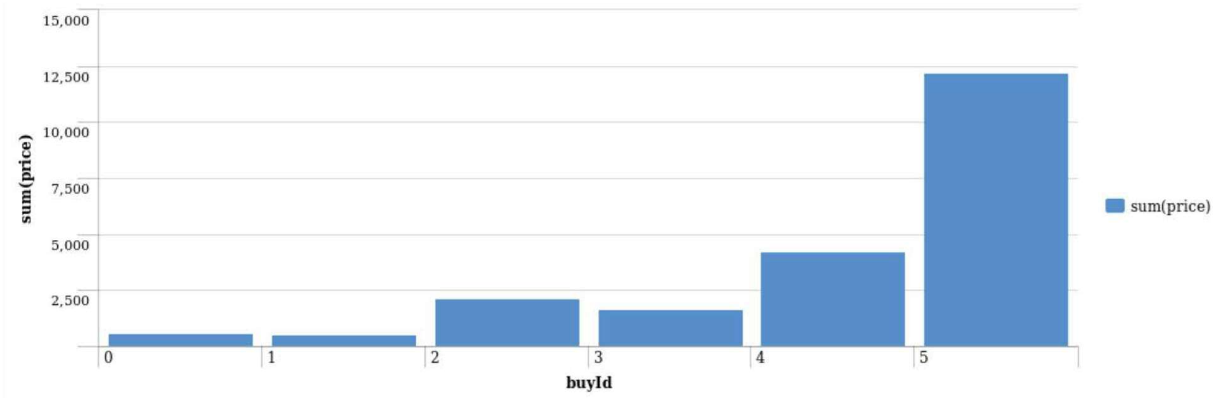
- **Aggregation**

Amount spent buying items	$21,407
Number of unique items available to be purchased	6

A histogram showing how many times each item is purchased:

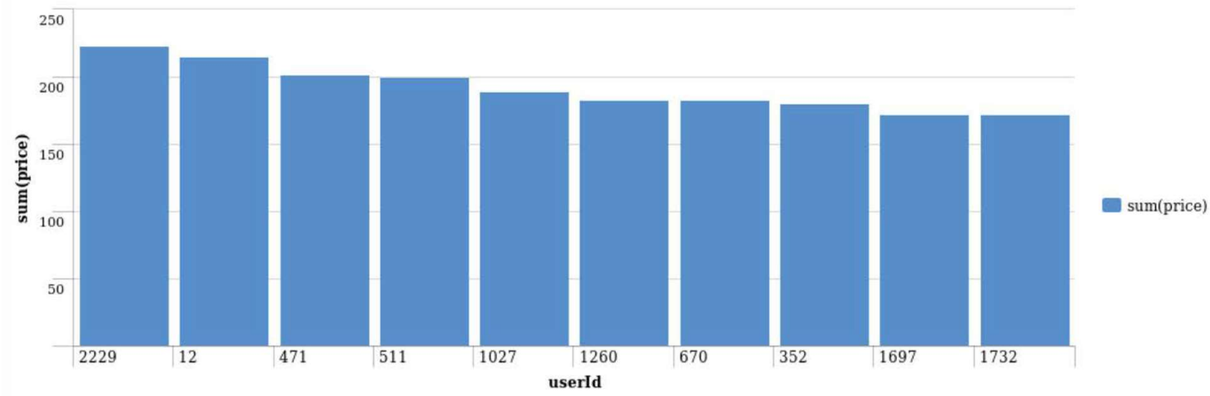


A histogram showing how much money was made from each item:



- ## 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	11.5
2	12	iPhone	13
3	471	iPhone	14.5

Part 2: Classification

Aim: Predicting which user is likely to purchase big-ticket items while playing Catch the Pink Flamingo is valuable knowledge to have for Eglence since in-app purchases are a major source of revenue.

Software used: KNIME Workflows

- **Data Preparation**

Analysis of combined_data.csv

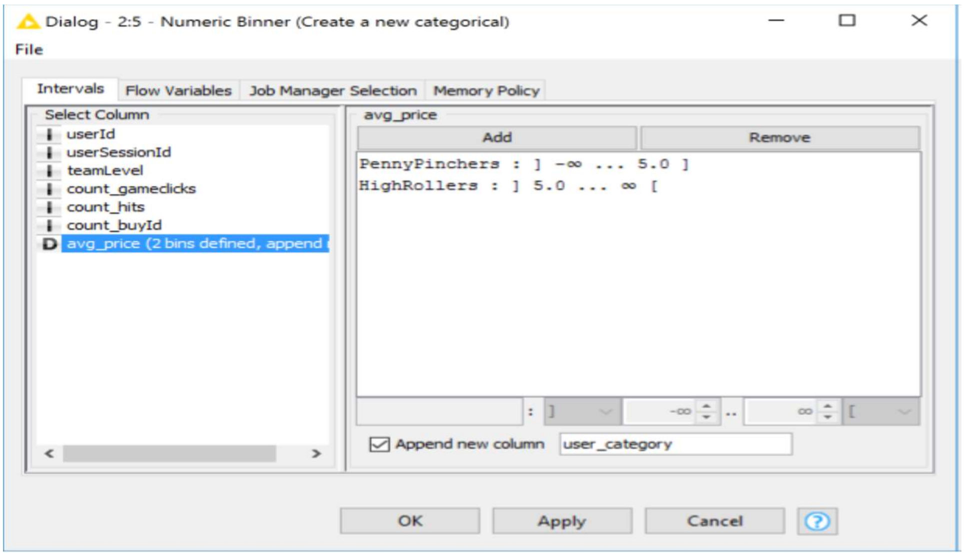
1. Sample Selection

Item	Amount
# of Samples	4619

# of Samples with Purchases	1411

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:



A new categorical attribute, named “user_category”, is created by the Numeric Binner node. As presented in the instruction, we need to define two categories for price which we will use to distinguish between HighRollers(buyers of items that cost more than $5.00) and PennyPinchers (buyers of items that cost $5.00 or less), so seen in the screenshot above, the user who costs $5.00 or less is placed in the category of “PennyPinchers”, the user who costs more than $5.00 is placed in the category of “HighRollers”.

The creation of this new categorical attribute was necessary because it can facilitate the classification of users and contribute to the following steps.

3. Attribute Selection

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

Attribute	**Rationale for Filtering**
userId	The userID attribute plays no role in deciding/ classifying users as HighRollers or PennyPinchers

usersSessionId	The ID of the game session plays no role in deciding the purchasing activities.
avg_price	Is already denoted by user_category

• Data Partitioning and Modelling

The data was partitioned into train and test datasets.

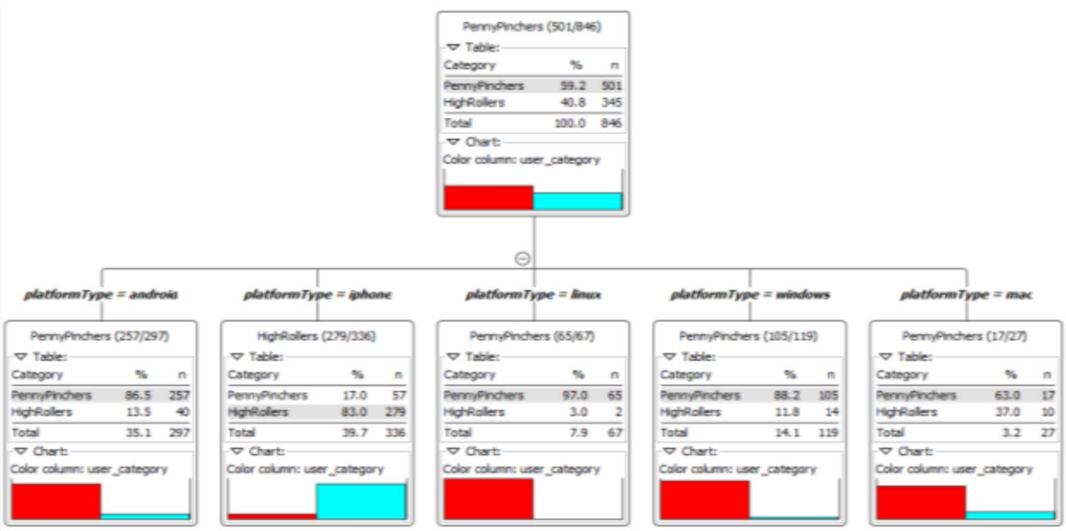
The training data set was used to create the decision tree model.

The trained model was then applied to the test dataset.

This is important because train data set is used in training the decision tree model, and the test data, comprising of samples separate from the training data, is used to evaluate the performance of the model, i.e., observe whether the trained model fits adequately to new data samples.

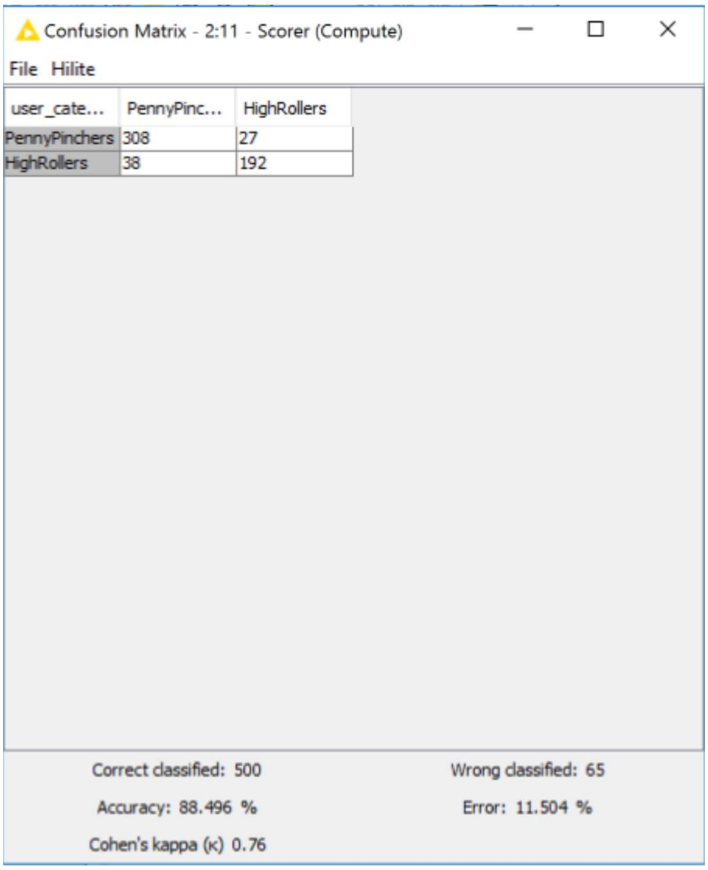
When partitioning the data using sampling, it is important to set the random seed because otherwise, the model will get the same training and test data partitions each time the node is run, leading to overfitting.

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



- **Evaluation**

A screenshot of the confusion matrix can be seen below:


Confusion Matrix - 2:11 - Scorer (Compute)

File Hilite

user_cate...	PennyPinc...	HighRollers
PennyPinchers	308	27
HighRollers	38	192

Correct classified: 500 Wrong classified: 65

Accuracy: 88.496 % Error: 11.504 %

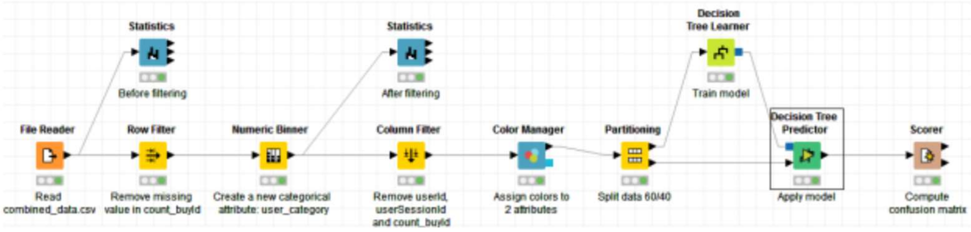
Cohen's kappa (κ) 0.76

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

- **Value** “308”: Users *correctly* predicted as PennyPinchers.
- **Value** “38”: Users *incorrectly* predicted as PennyPinchers.
- **Value** “192”: Users *correctly* predicted as HighRollers
- **Value** “27”: Users *incorrectly* predicted as HighRollers.

- **Analysis Conclusions**

The final KNIME workflow is shown below:



What makes a HighRoller vs. a PennyPincher?

According to the resulting decision tree, it obviously shows that the predicted user_category is different in various platforms, the users on the platforms Android, Linux, Windows and Mac are mostly PennyPinchers, however, most users which on the platform iPhone are HighRollers.

Specific Recommendations to Increase Revenue
1. Offer more products to iPhone users. Higher priced product combinations can also be offered to them.
2. Incentivize users of other platforms by offering them discounts, targeted sales, and maybe some power-ups, based on the nature of purchases.
3. Optionally, users who have not contributed to the revenue at all, may be attracted with level gains and better team recommendations.

Part 3: Clustering

Software used: Spark MLLib

- ## Attribute Selection

Attribute	Rationale for Selection
totalAdClicks	Total of ad-clicks per user. This attribute is important in serving personalized advertisements to groups of people and thus increase advertising revenue.
totalGameClicks	Represents number of game clicks per user. It may be important to capture users' play behavior.
totalRevenue	Represents net in-app spending per user. Helps in deciding targeted sales and price of newer products.

- ## Training Data Set Creation

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

Create the final training dataset

Our training data set is almost ready. At this stage we can remove the 'userId' from each row, since 'userId' is a computer generated random number assigned to each user. It does not capture any behavioral aspect of a user. One way to drop the 'userId', is to select the other two columns.

In [37]:
```
training_df = combined_df[['totalAdClicks', 'totalGameClicks','revenue']]
training_df.head(5)
```

Out[37]:

	totalAdClicks	totalGameClicks	revenue
0	44	716	21.0
1	10	380	53.0
2	37	508	80.0
3	19	3107	11.0
4	46	704	215.0

Display the dimensions of the training dataset

Display the dimension of the training data set. To display the dimensions of the training_df, simply add .shape as a suffix and hit enter.

In [38]:
```
training_df.shape
```

Out[38]: (543, 3)

Dimensions of the final data set: 543 rows * 3 columns
# of clusters created: 3

- ## Cluster Centers

The code used in creating cluster centers is given below:

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

Cluster centers formed are given in the table below

Cluster #	Center
1	25.12037037, 362.50308642, 35.35802469
2	32.05, 2393.95, 41.2
3	36.47486034, 953.82122905, 46.16201117

The first index corresponds to cluster center for Ad clicks.
The second index corresponds to cluster center for Game clicks.
The third index corresponds to the cluster center for Total Revenue.

These clusters can be differentiated from each other as follows:

Cluster 1 is different from the others in that the users' ad-clicks, game-clicks and purchases are all less than others, these kind of users can be called "low level spending users".

Cluster 2 is different from the others in that the ad-clicks is not the least, game-clicks is the most but their purchases are not the most, these kind of users can be called "neutral users".

Cluster 3 is different from the others in that the users' ad-clicks, game-clicks and purchases are all more than others, these kind of users can be called "high level spending users".

Below given is the summary of the train data set:

```
print(centers)
```

```
[array([  25.12037037,   362.50308642,   35.35802469]), array([  32.05,    2393.95,    41.2]), array([
36.47486034,   953.82122905,   46.16201117])]
```

- **Recommended Actions**

Action Recommended	Rationale for the action
Provide more products to “High-Spending” users	Since the majority of revenue is generated from these users, we can offer them higher priced products in the form of exclusive content and targeted sales.
Incentivise the “Neutral” users through discounts and power ups	This will ensure that the Neutral Spending users are attracted towards buying products frequently.
Hold some special events and contests	This will majorly stimulate the interests among “Low-Spending” users.

Part 4: Graph Analytics

Software used: Neo4j

Modelling Chat Data using a Graph Data Model

A Graph Data Model is used to illustrate the chatting interaction among users with Chat Data. A user in a team can create a chat session and then create chat (i.e. chat item) in the chat session. Otherwise, a user could be mentioned by a chat item, and a chat item can response to another chat item, which represent the communication among the users in the same team. Moreover, a user can also join in an existed team chat session or leave it.

- **Creation of the Graph Database for Chats**

CSV Files:

File Name	Description	Fields
chat_create_team_chat.csv	userid	the user id assigned to the user
	teamid	the id of the team
	TeamChatSessionID	a unique id for the chat session
	timestamp	a timestamp denoting when the chat session created
chat_item_team_chat.csv	userid	the user id assigned to the user
	teamchatsessionid	a unique id for the chat session
	chatitemid	a unique id for the chat item
	timestamp	a timestamp denoting when the chat item created
chat_join_team_chat.csv	userid	the user id assigned to the user
	TeamChatSessionID	a unique id for the chat session
	timestamp	a timestamp denoting when the user join in a chat session
chat_leave_team_chat.csv	userid	the user id assigned to the user
	teamchatsessionid	a unique id for the chat session
	timestamp	a timestamp denoting when the user leave a chat session
chat_mention_team_chat.csv	ChatItemId	the id of the ChatItem
	userid	the user id assigned to the user
	timeStamp	a timestamp denoting when the user mentioned by a chat item
chat_respond_team_chat.csv	chatid1	the id of the chat post 1
	chatid2	the id of the chat post 2
	timestamp	a timestamp denoting when the chat post 1 responds to the chat post 2

- ## Loading Procedure

Using Cypher Query Language to load the CSV data into neo4j, each row of script is parsed for refine the nodes, the edges and its timestamp. Let's consult the following script as an example:

Query:

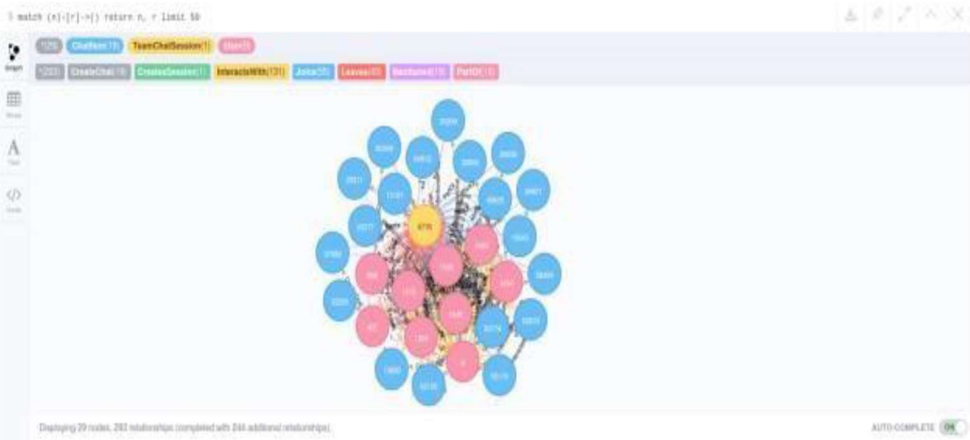
```
MERGE (u:User {id: toInt(row[0])})
MERGE (c:TeamChatSession {id: toInt(row[1])})
MERGE (i:ChatItem {id: toInt(row[2])})
MERGE (u)-[:CreateChat{timeStamp: row[3]}]->(i)
MERGE (i)-[:PartOf{timeStamp: row[3]}]->(c)
```

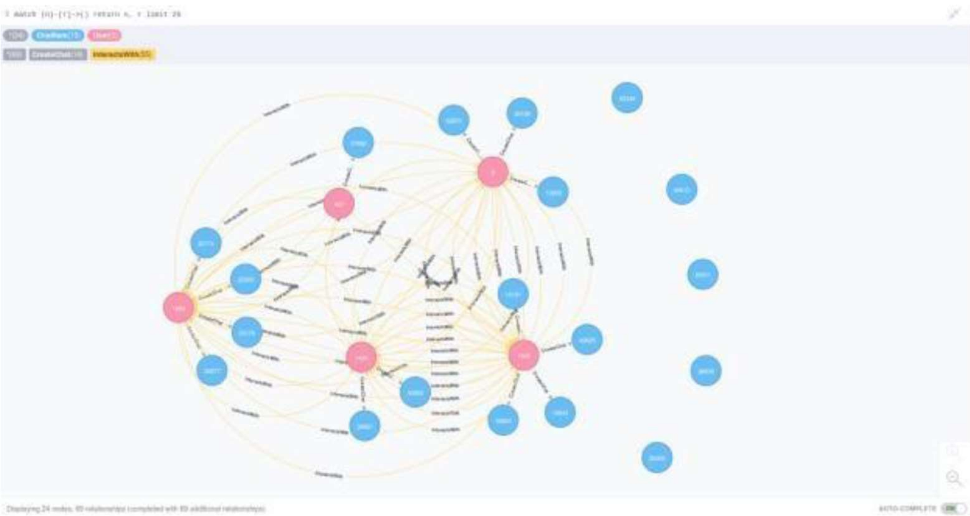
- The first line gives the path of the file, this command reads the chat_item_team_chat.csv at a time and create user nodes. The 0^{th} column value is converted to an integer and is used to populate the id attribute. Similarly, the other nodes are created.

- **Line 5**, *`MERGE (u)-[:CreateChat{timeStamp: row[3]}]->(i)`* creates an edge labeled "`CreateChat`" between the User node `u` and the ChatItem node `i`. This edge has a property called `timeStamp`. This property is filled by the content of column 3 of the same row.

- **Line 6**, *`MERGE (i)-[:PartOf{timeStamp: row[3]}]->(c)`* creates an edge labeled "`PartOf`" between the ChatItem node `i` and the TeamChatSession node `c`. This edge has a property called `timeStamp`. This property is filled by the content of column 3 of the same row.

Partial Screenshot of the Graph



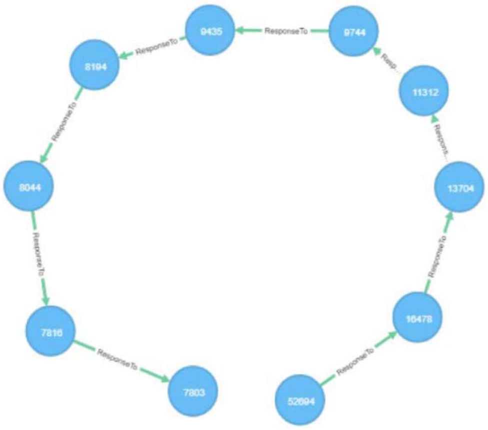


- ## Finding the longest conversation chain and its participants

The longest conversation chain is traced via `ChatItem` nodes which are connected by `ResponseTo` edges, order the length and find the longest one. Running the following query, we get the longest conversation chain has path length of **9**, i.e. the longest conversation has **10 chats**.

Query:

```
match p = (i1)-[:ResponseTo*]->(i2) return length(p)
order by length(p) desc limit 1
```



The number of users who participated in this chain can be found out by the number of distinct users in the path, which is given by the following query:

Query:

```
match p = (i1)-[:ResponseTo*]->(i2) where length(p) = 9 with p match (u)-
[:CreateChat]->(i) where i in nodes(p) return count(distinct u)
```

The result is returned as **5 Distinct Users**.

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

- ## Chattiest Users:

We firstly `match` the `CreateChat` edge from User node to ChatItem node, then return the ChatItem amount per user, and order by the amount in descending order.

Query:

```
match (u)-[:CreateChat*]->(i) return u.id, count(i)
order by count(i) desc limit 10
```

Chattiest Users

User	Number of Chats
394	115
2067	111
209 ; 1087	109

- ## Chattiest Teams:

We firstly `match` the `PartOf` edge from ChatItem node to TeamChatSession node, `match` the `OwnedBy` edge from TeamChatSession node to Team node, then return the TeamChatSession amount per team, and order by the amount in descending order.

Query:

```
match (i)-[:PartOf*]->(c)-[:OwnedBy*]->(t) return t.id, count(c) order by
count(c) desc limit 10
```

$ match (i)-[:PartOf*]->(c)-[:OwnedBy*]->(t) return t.id, count(c) order by count(c) desc limit 10

"t.id"	"count(c)"
"82"	"1324"
"185"	"1036"
"112"	"957"
"18"	"844"
"194"	"836"
"129"	"814"
"52"	"788"
"136"	"783"
"146"	"746"
"81"	"736"

Started streaming 10 records after 316 ms and completed after 316 ms.

MAX COLUMN WIDTH:

Chattiest Teams

Teams	Number of Chats
82	1324
185	1036
112	957

Combining the above two queries:

Query:

```
match (u)-[:CreateChat*]->(i)-[:PartOf*]->(c)-[:OwnedBy*]->(t) return u.id,
t.id, count(c)
order by count(c) desc limit 10
```

$ match (u)-[:CreateChat*]->(i)-[:PartOf*]->(c)-[:OwnedBy*]->(t) return u.id, t.id, count(c) order by count(c) desc l...

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

Started streaming 10 records after 457 ms and completed after 457 ms.

MAX COLUMN WIDTH:

As result shows, the user 999, which in the team 52 is part of the top 10 chattiest teams, but other 9 users are not part of the top 10 chattiest teams. This states that most of the chattiest users are not in the chattiest teams.

• 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 neighborhoods for direct advertising. We will do this in a series of steps.

Query:

- Creating neighborhood of users

```
match (u1:User)-[:CreateChat]->(i)-[:Mentioned]->(u2:User) create (u1)-
[:InteractsWith]->(u2)
match (u1:User)-[:CreateChat]->(i1:ChatItem)-[:ResponseTo]-(i2:ChatItem) with
u1, i1, i2 match (u2)-[:CreateChat]-(i2) create (u1)-[:InteractsWith]->(u2)
```

- Removing self-loops

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

- Applying concept of Cluster Coefficients

```
match (u1:User)-[r1:InteractsWith]->(u2:User) where u1.id <> u2.id with u1,
collect(u2.id) as neighbors, count(distinct(u2)) as neighborAmount match
(u3:User)-[r2:InteractsWith]->(u4:User) where (u3.id in neighbors)
AND (u4.id in neighbors) AND (u3.id <> u4.id) with u1, u3, u4,
neighborAmount, case when (u3)-->(u4) then 1 else 0 end as value return u1,
sum(value)*1.0/(neighborAmount*(neighborAmount-1)) as coeff order by coeff
desc limit 10
```

Most Active Users (based on Cluster Coefficients)

User ID	Coefficient
209	0.9524
554	0.9048
1087	0.8

Thank You