Technical Appendix

Analysis of Catch the Pink Flamingo data | Nyan Lynn Htet | Sep 8 2024

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
ad-clicks.csv	The dataset of clicks on ads	timestamp: when user clicked txId: unique Id (in ad-clicks.log) for the click userSessionId: Id of user session for the user who click teamId: the current team Id of the user who click userId: ID of the user who click

		adId: the id of the ad clicked on adCategoryL the category/ type of ad clicked on
buy-clicks.csv	Database of purchases.	timestamp: when the purchase was made. txId: a unique id (within buyclicks.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 database 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 database of each level event for a team. Level events are recorded when a team ends or	timestamp: when the event occurred.

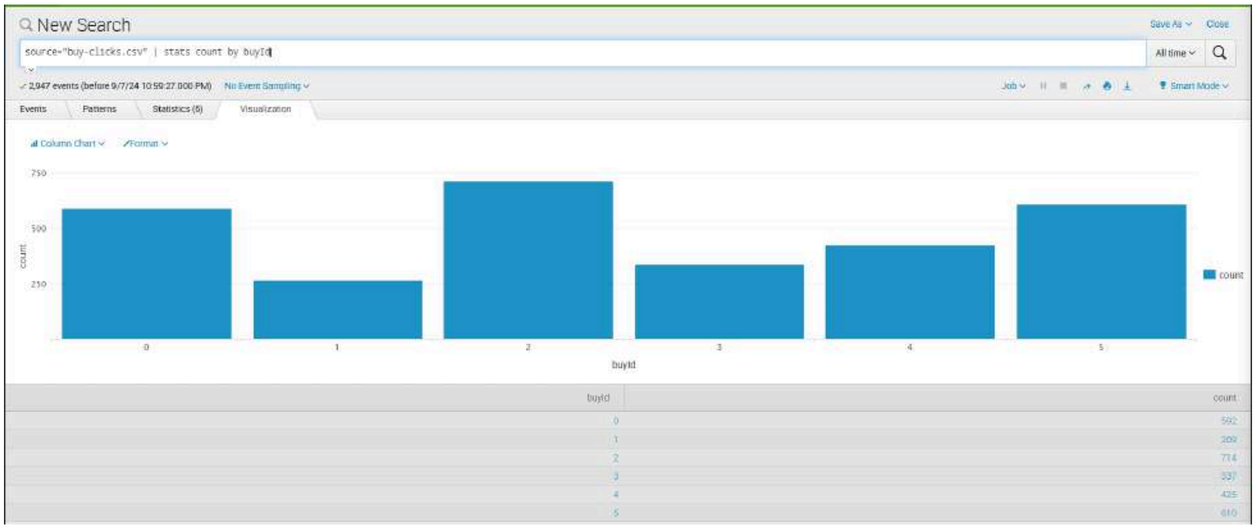
	begins a new level	eventId: a unique id for the event teamId: the id of the team teamLevel: the level started or completed eventType: the type of event, either start or end
team-assignments.csv	A record of each time a user joins a team.	timestamp: when the user joined the team. team: the id of the team 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 database of each session a user plays. When a team levels up, each current user session ends and a new session begins with the 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	$21407
Number of unique items available to be purchased	6

A histogram showing how many times each item is purchased:



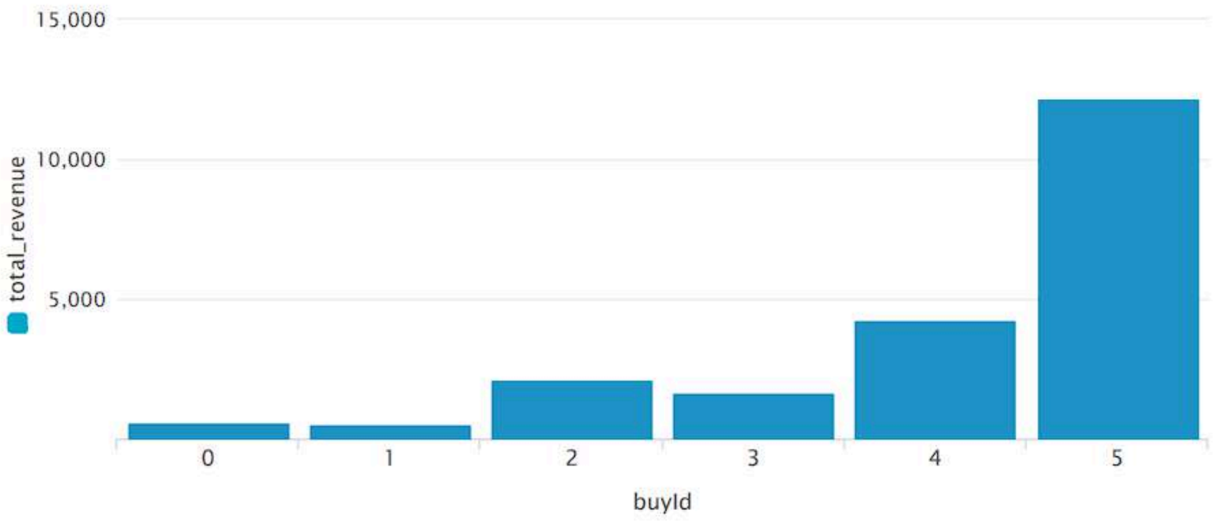
New Search

```
source="buy-clicks.csv" | stats count by buyId
```

✓ 2,947 events (before 9/7/24 10:59:27.000 PM) No Event Sampling

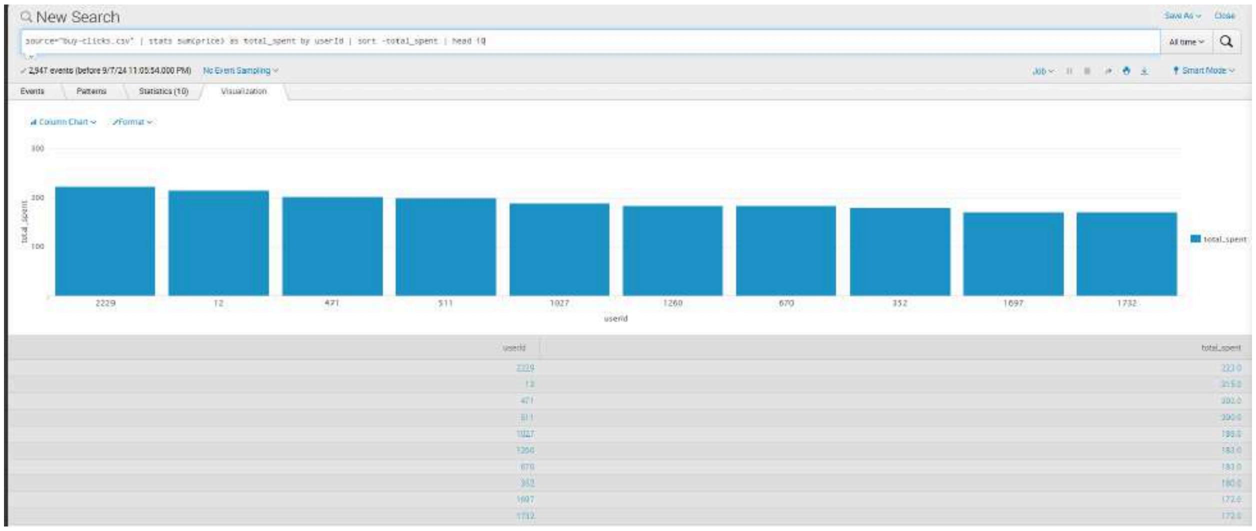
buyId	count
0	592
1	269
2	714
3	337
4	425
5	610

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



Filtering

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



```
source="buy-clicks.csv" | stats sum(price) as total_spent by userId | sort -total_spent | head 10
```

userId	total_spent
2229	223.0
12	215.0
471	202.0
511	200.0
1027	198.0
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	11.5%
2	12	iphone	13%
3	471	iphone	14.5%

Data Preparation (Catch the Pink Flamingo)

Analysis of combined_data.csv

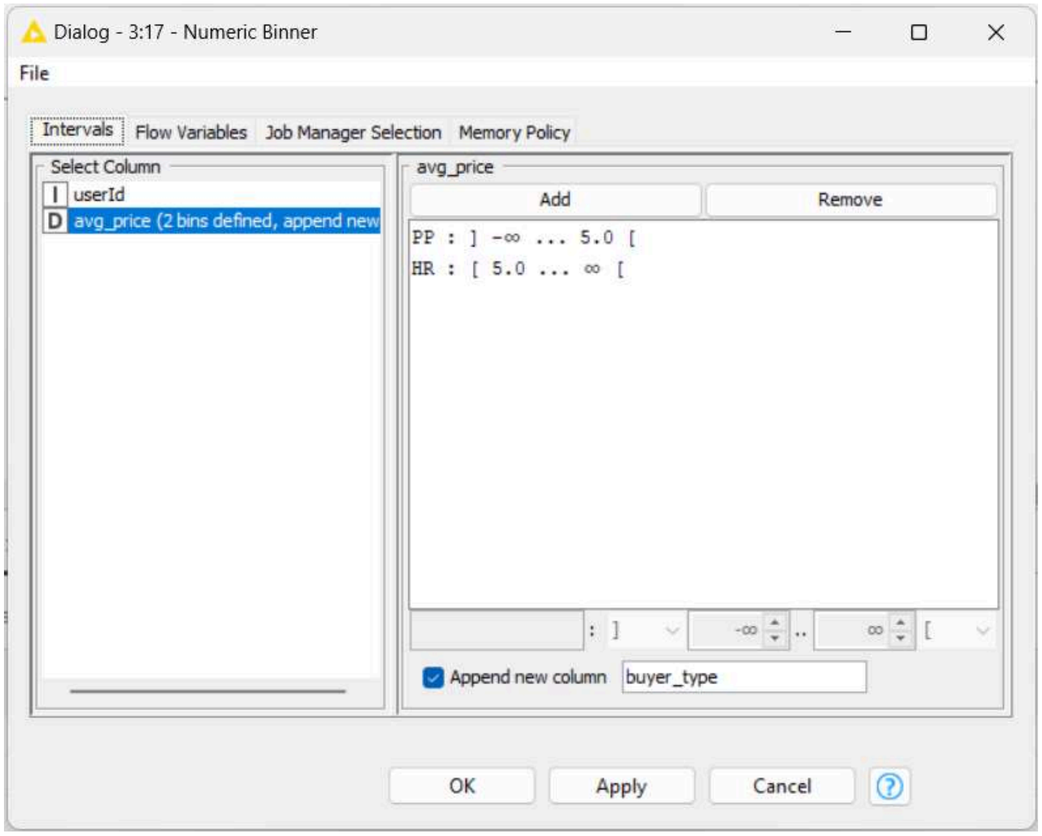
Classifying in KNIME (Catch the Pink Flamingo)

<u>Sample Selection</u>

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

<u>Attribute Creation</u>

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:



```
userId
avg_price (2 bins defined, append new

avg_price
PP : ] -∞ ... 5.0 [
HR : [ 5.0 ... ∞ [

Append new column: buyer_type
```

Note HR = HighRollers, PP = PennyPinchers*

Description: HighRollers were defined those who spent $5.0 (on average price) and over and PennyPinchers are those who spent less than $5.0 (on average price).

The creation of this new categorical attribute was necessary because:
We aim to identify the group of people who spend generous amount and those who spend less. Then make a marketing strategy for the game according to our findings so that we can raise the sales. Also help in the decision making process.

<u>Attribute Selection</u>

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

Attribute	**Rationale for Filtering**
count_buyId	To filter out NULL since their purchase data is not available
userSessionId	Not relevant for the model this time
avg_price	To sort this so that we identify the HighRoller and PennyPinchers

platformType	To identify the platform used by the HighRoller and PennyPinchers

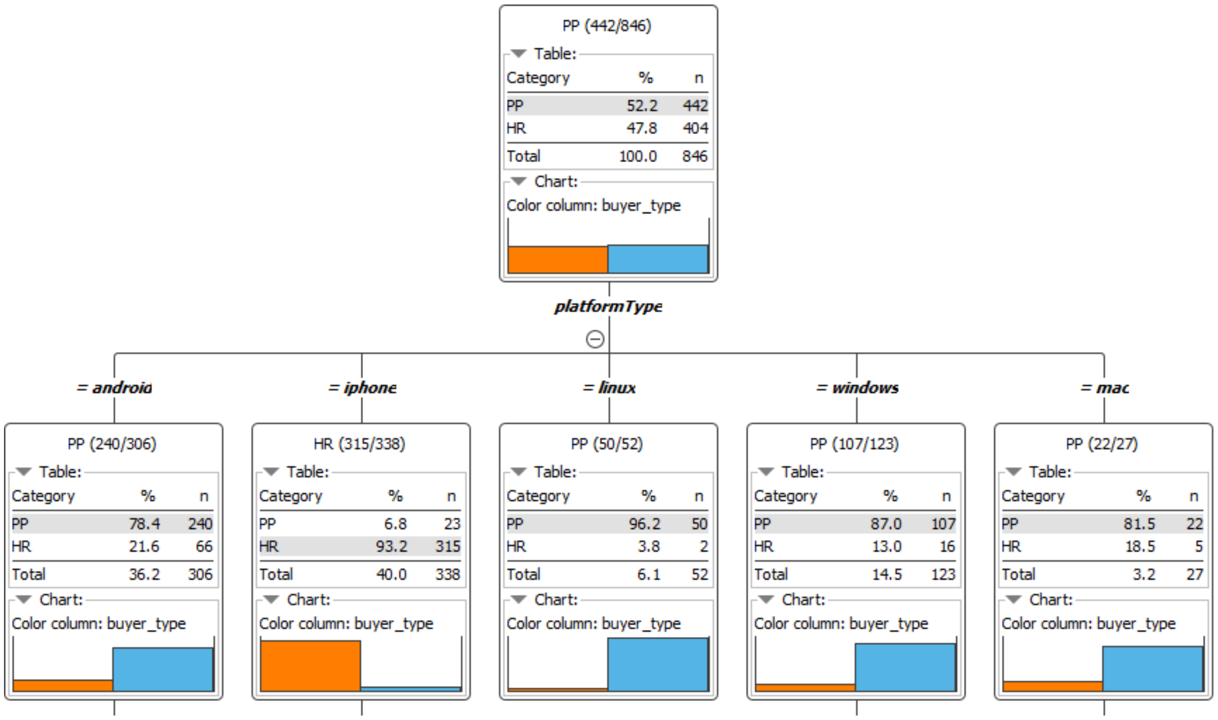
Data Partitioning and Modeling (Catch the Pink Flamingo)

Analysis of combined_data.csv

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 data** dataset.
This is important because partitioning the data set into **training** and **test data** allows me to verify the output accuracy of the trained model.

When partitioning the data using sampling, it is important to set the random seed becauseit ensures that you get the same results every time you perform the partitioning.

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



PP (442/846)

Category	%	n
PP	52.2	442
HR	47.8	404
Total	100.0	846

Color column: buyer_type

= android: PP (240/306)

Category	%	n
PP	78.4	240
HR	21.6	66
Total	36.2	306

Color column: buyer_type

= iphone: HR (315/338)

Category	%	n
PP	6.8	23
HR	93.2	315
Total	40.0	338

Color column: buyer_type

= linux: PP (50/52)

Category	%	n
PP	96.2	50
HR	3.8	2
Total	6.1	52

Color column: buyer_type

= windows: PP (107/123)

Category	%	n
PP	87.0	107
HR	13.0	16
Total	14.5	123

Color column: buyer_type

= mac: PP (22/27)

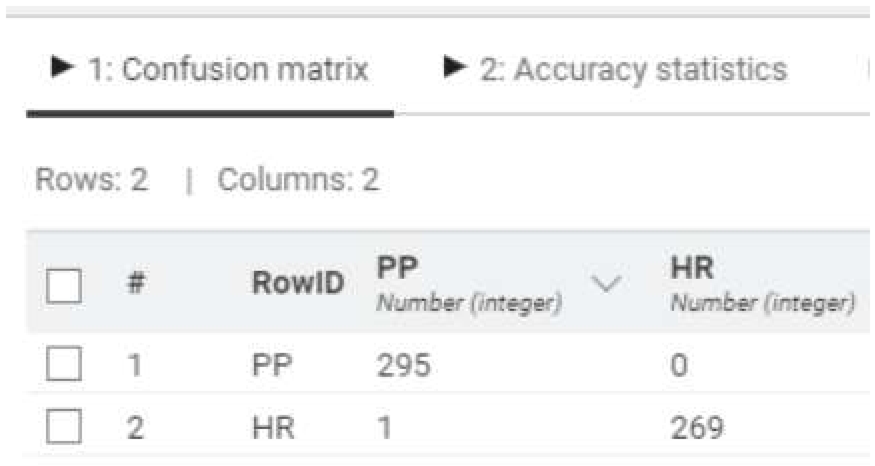
Category	%	n
PP	81.5	22
HR	18.5	5
Total	3.2	27

Color column: buyer_type

Evaluation (Catch the Pink Flamingo)

Analysis of combined_data.csv

A screenshot of the confusion matrix can be seen below:



#	RowID	PP Number (integer)	HR Number (integer)
1	PP	295	0
2	HR	1	269

Correctly classified: 295
Wrongly classified: 1
Cohen's kappa(k): 0.996
Accuracy: 99.8%
Error: 0.02%

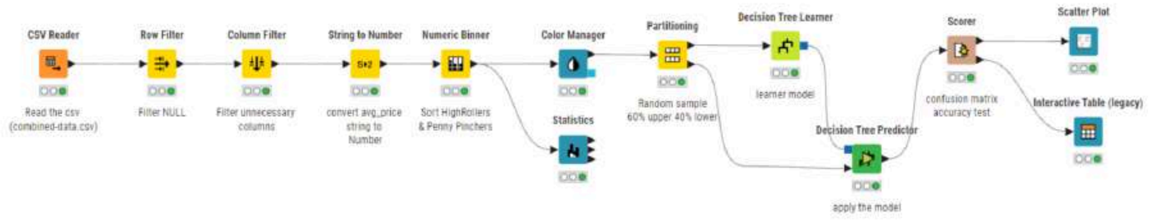
As seen in the screenshot above, the overall accuracy of the model is 99.8%

After running the model, it predicted correctly (295) times and incorrectly (0) times for the PP(PennyPincher) type; and predicted correctly (269) times and incorrectly (1) times for the HR(HighRoller) type.

Analysis Conclusions (Catch the Pink Flamingo)

Analysis of combined_data.csv

The final KNIME workflow is shown below:


CSV Reader
Read the csv (combined-data.csv)
Row Filter
Filter NULL
Column Filter
Filter unnecessary columns
String to Number
convert avg_price string to Number
Numeric Binner
Sort HighRollers & Penny Pinchers
Color Manager
Statistics
Partitioning
Random sample 60% upper 40% lower
Decision Tree Learner
learner model
Decision Tree Predictor
apply the model
Scorer
confusion matrix accuracy test
Scatter Plot
Interactive Table (legacy)

What makes a HighRoller vs. a PennyPincher?
The device and OS used by the players. According to the analysis, most of the HR(HighRollers) are using iOS which is apple device, specifically Iphones, Ipads. And PP(PennyPinchers) use the other OS and devices.

Specific Recommendations to Increase Revenue
1. We can make campaigns and events to attract android and other platform users to spend more. (and may be regional competitions to promote game purchases)
2. Make special events and targeted campaigns to the ios users to praise the behavior of high roller spending.

Attribute Selection (Catch the Pink Flamingo)

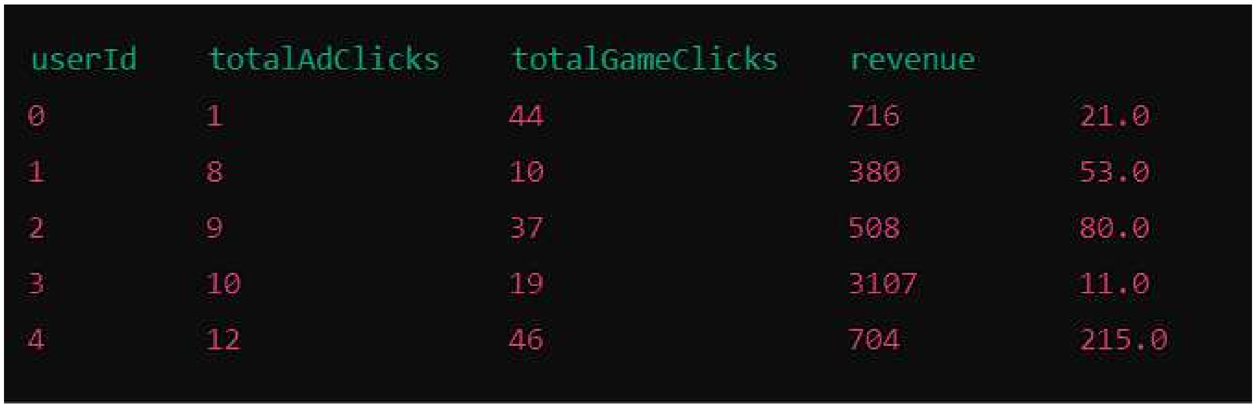
Attribute	Rationale for Selection
totalGameClicks	The frequency with which users play the game indicates their level of engagement.
totalAdClicks	The number of times users click on ads reflects their potential to be monetized.
revenue	The amount users spend in the game provides the monetary value of each customer.

Training Data Set Creation (Catch the Pink Flamingo)

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

```
combined_df = ads_per_user.merge(game_clicks_per_user, on='userId')
#userId, adCount, clickCount
combined_df = combined_df.merge(revenue_per_user, on='userId') #userId,
adCount, clickCount, price
```

```
combined_df.head(5) #display how the merged table looks
```



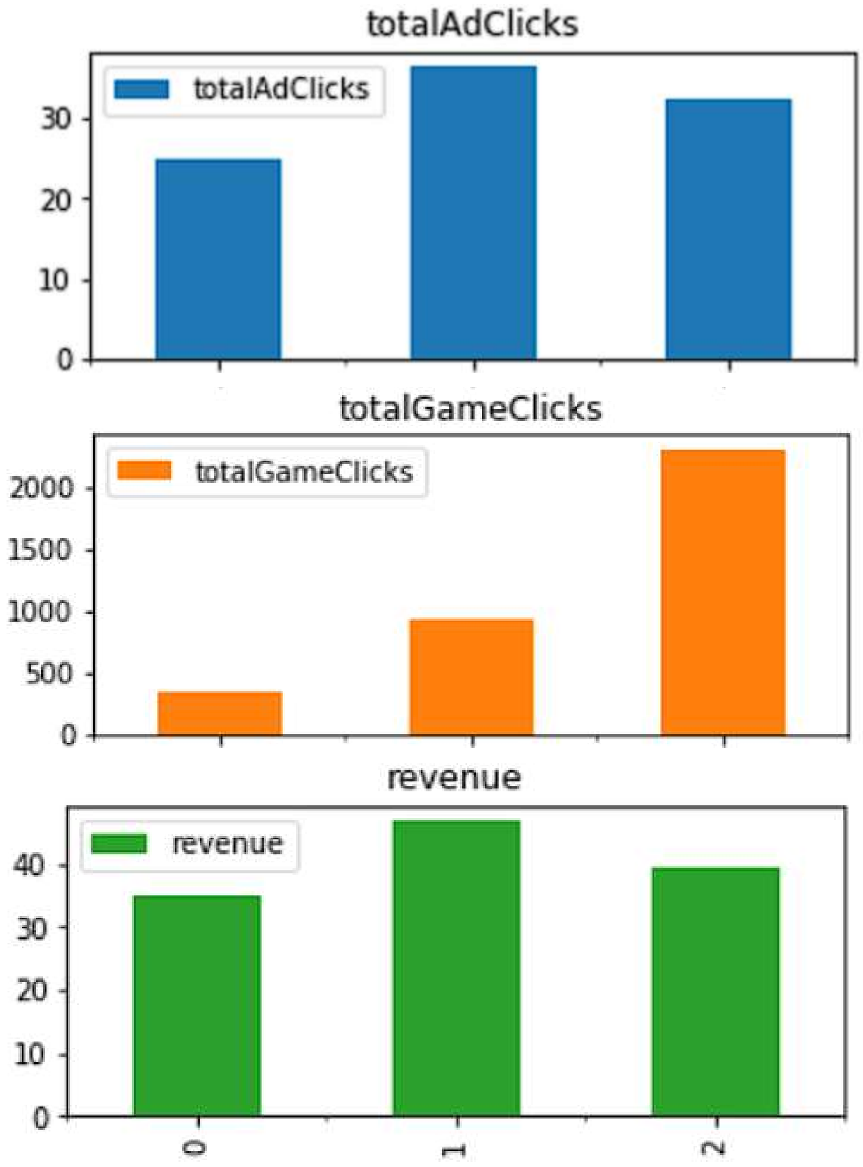
```
userId    totalAdClicks    totalGameClicks    revenue
0         1                44                 716          21.0
1         8                10                 380          53.0
2         9                37                 508          80.0
3         10               19                 3107         11.0
4         12               46                 704          215.0
```

Dimensions of the final data set: 543*3
# of clusters created: 3 (THREE)

Cluster Centers (Catch the Pink Flamingo)

Cluster centers formed are given in the table below

Cluster #	Center: totalAdClicks , totalGameClicks , revenue
1	24.99 , 357.96 , 35.07
2	36.44 , 926.12 , 46.97
3	32.36 , 2310.64 , 39.42



These clusters can be differentiated from each other as follows:

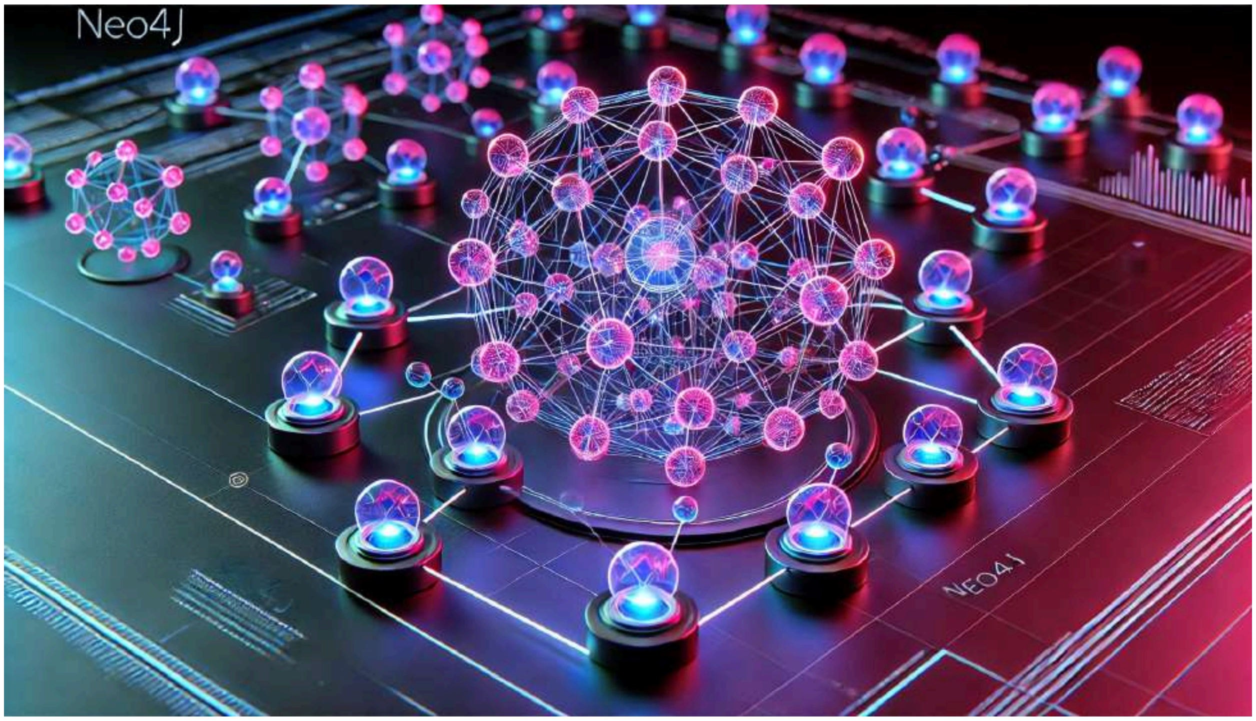
Cluster 0: Customers that don’t play much but still generate some revenue

Cluster 1: Customers that are very cash convertible with ads even if they play a moderate amount, generating a high amount of revenue

Cluster 2: Customers that are very active and generate a moderate amount of revenue

Recommended Actions (Catch the Pink Flamingo)

Action Recommended	**Rationale for the action**
Target specific lower cost promo/discounts for Cluster 2.	Cluster 2: Users that are very active and generate a moderate amount of revenue. If we could make them buy a bit more with cheaper prices, we could take advantage of their behavior.
Target specific higher cost premium features for Cluster 1.	Cluster 1: Users that are very cash convertible with ads even if they play a moderate amount, generating a high amount of revenue. These guys like to spend their money, even if they aren’t playing all the time. They are probably older and have a day job. We should target premium features and personalize ads so that they are aware and can spend their cash.



Graph Analytics with Neo4J (Catch the Pink Flamingo Game)

Modeling Chat Data using a Graph Data Model

The graph data model is utilized to represent user interactions through chat data. Users can initiate chat sessions and add messages within those sessions. A user may be mentioned in a chat message, and one message can reply to another. Additionally, users have the ability to join or exit existing team chat sessions.

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
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.

schema of the 6 CSV files

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 was 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 joined 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 left 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 was 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

Explain the loading process and include a sample LOAD command

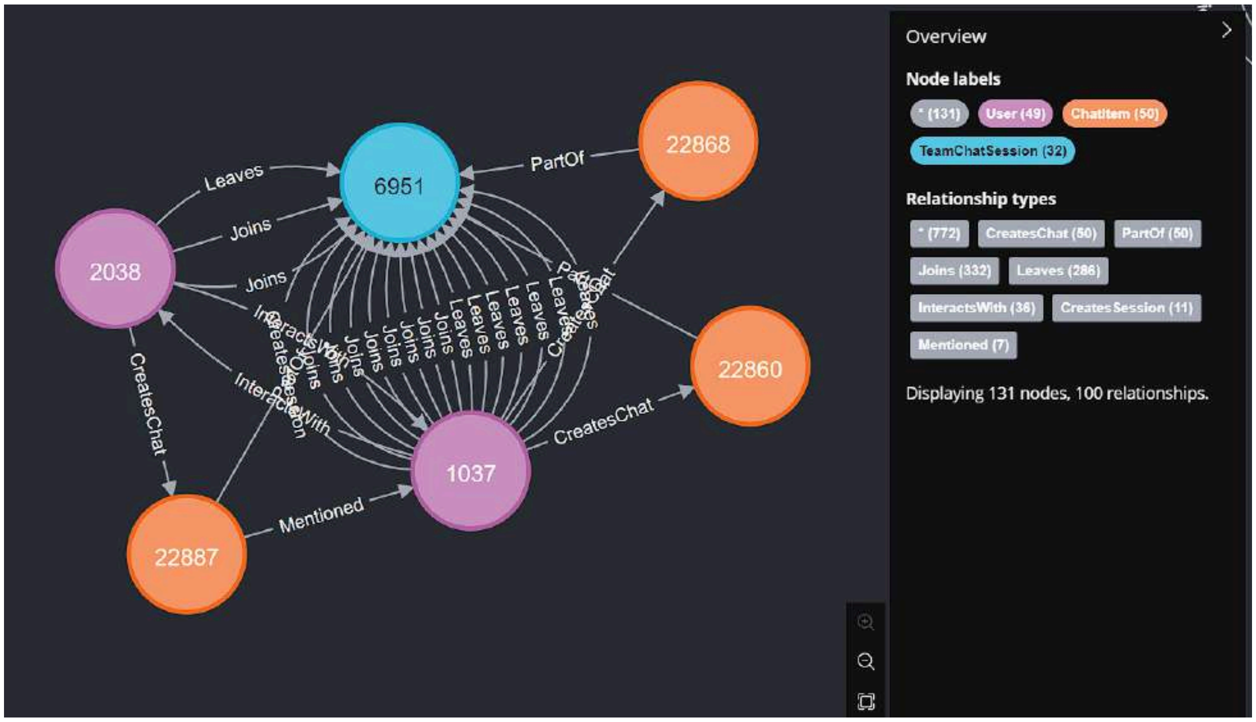
The first line loads the CSV file from a designated location, processing one row at a time. Lines two through four generate nodes for User, Team, and TeamChatSession, with a specific column converted to an integer, which is then assigned to the id attribute. Lines five and six establish the CreatesSession and OwnedBy edges, connecting the previously created nodes. These edges include a timestamp property, which is populated from the fourth column of the schema.

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

LOAD CSV FROM "file:///D:/chat_csv/chat_create_team_chat.csv" AS row
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)
```

Present a screenshot of some part of the graph you have generated. The graphs must include clearly visible examples of most node and edge types.

```
$ MATCH (n)-[r]->(m) RETURN n, r, m  LIMIT 100
```



Overview

Node labels

* (131) User (49) ChatItem (50) TeamChatSession (32)

Relationship types

* (772) CreatesChat (50) PartOf (50) Joins (332) Leaves (286) InteractsWith (38) CreatesSession (11) Mentioned (7)

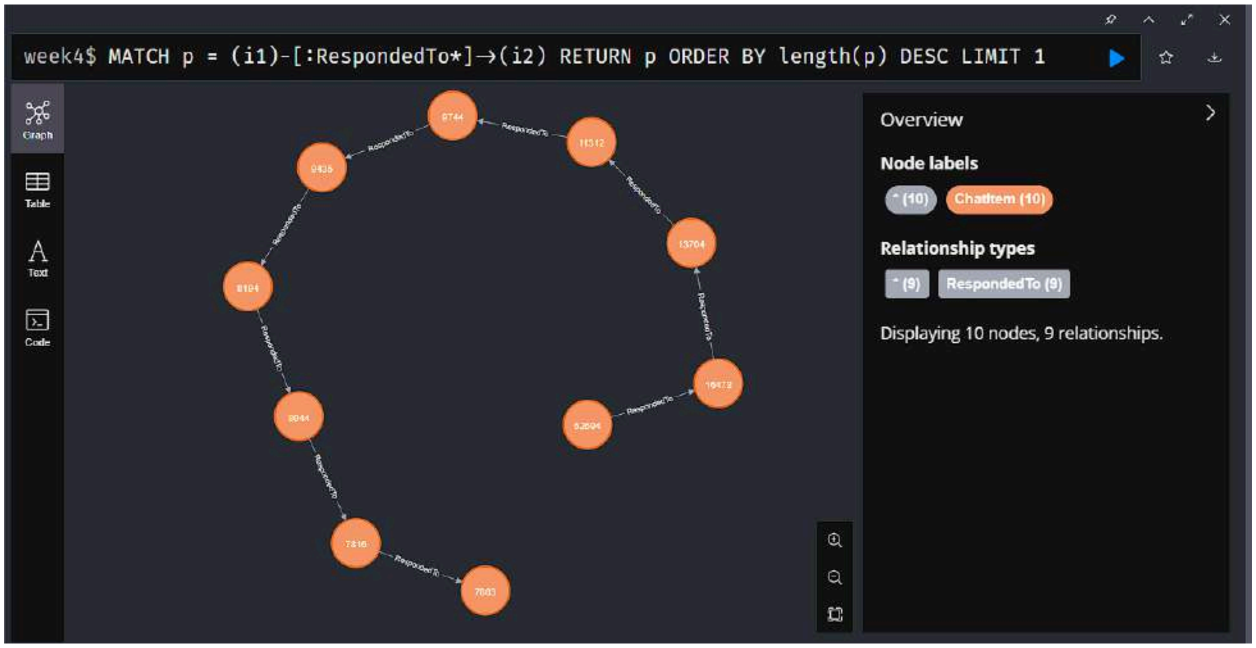
Displaying 131 nodes, 100 relationships.

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.

the length of the conversation (path length)

```
MATCH p = (i1)-[:RespondedTo*]->(i2)
RETURN p ORDER BY length(p) DESC LIMIT 1
```



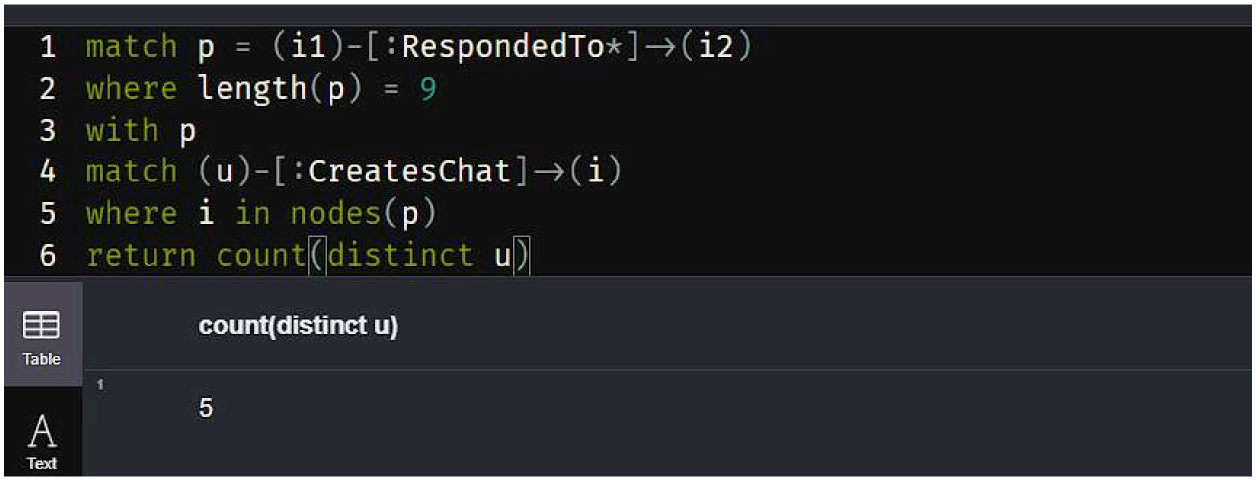
```
week4$ MATCH p = (i1)-[:RespondedTo*]→(i2) RETURN p ORDER BY length(p) DESC LIMIT 1
```

Displaying 10 nodes, 9 relationships.

The longest conversation chain in the chat data has path length 9, therefore 10 chats are involved in it.

how many unique users were part of the conversation chain.

```
MATCH p = (i1)-[:RespondedTo*]->(i2)
WHERE length(p) = 9
WITH p
MATCH (u)-[:CreatesChat]->(i)
WHERE i IN nodes(p)
RETURN count(distinct u)
```



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

	count(distinct u)
1	5

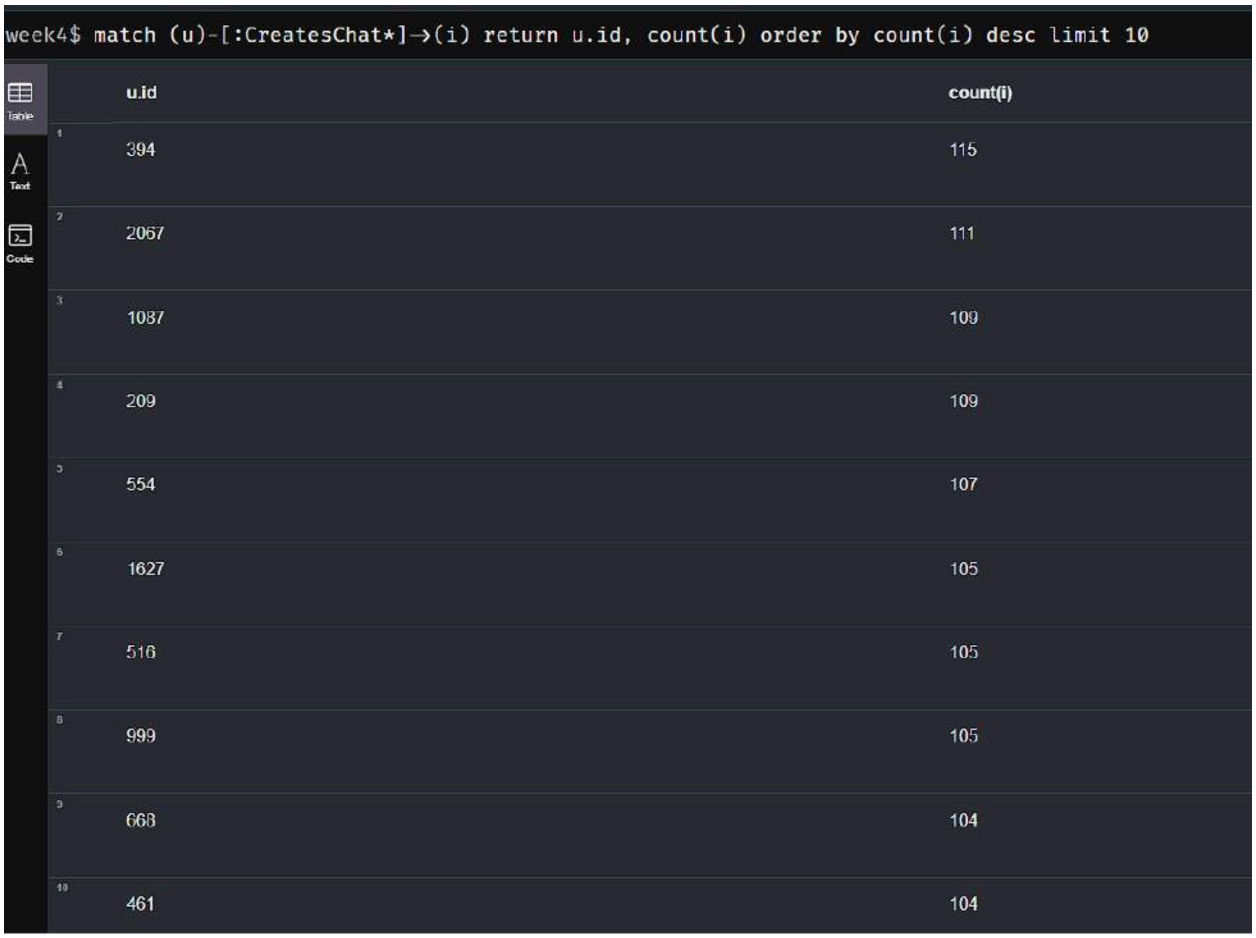
The unique user count is 5

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

Chattiest Users

Query the number of chats created by a user from the CreateChat edge

```
MATCH (u)-[:CreatesChat*]->(i)
RETURN u.id, count(i)
ORDER BY count(i) desc limit 10
```



```
week4$ match (u)-[:CreatesChat*]→(i) return u.id, count(i) order by count(i) desc limit 10
```

```
   u.id      count(i)
1  394       115
2  2067      111
3  1087      109
4  209       109
5  554       107
6  1627      105
7  516       105
8  999       105
9  668       104
10 461       104
```

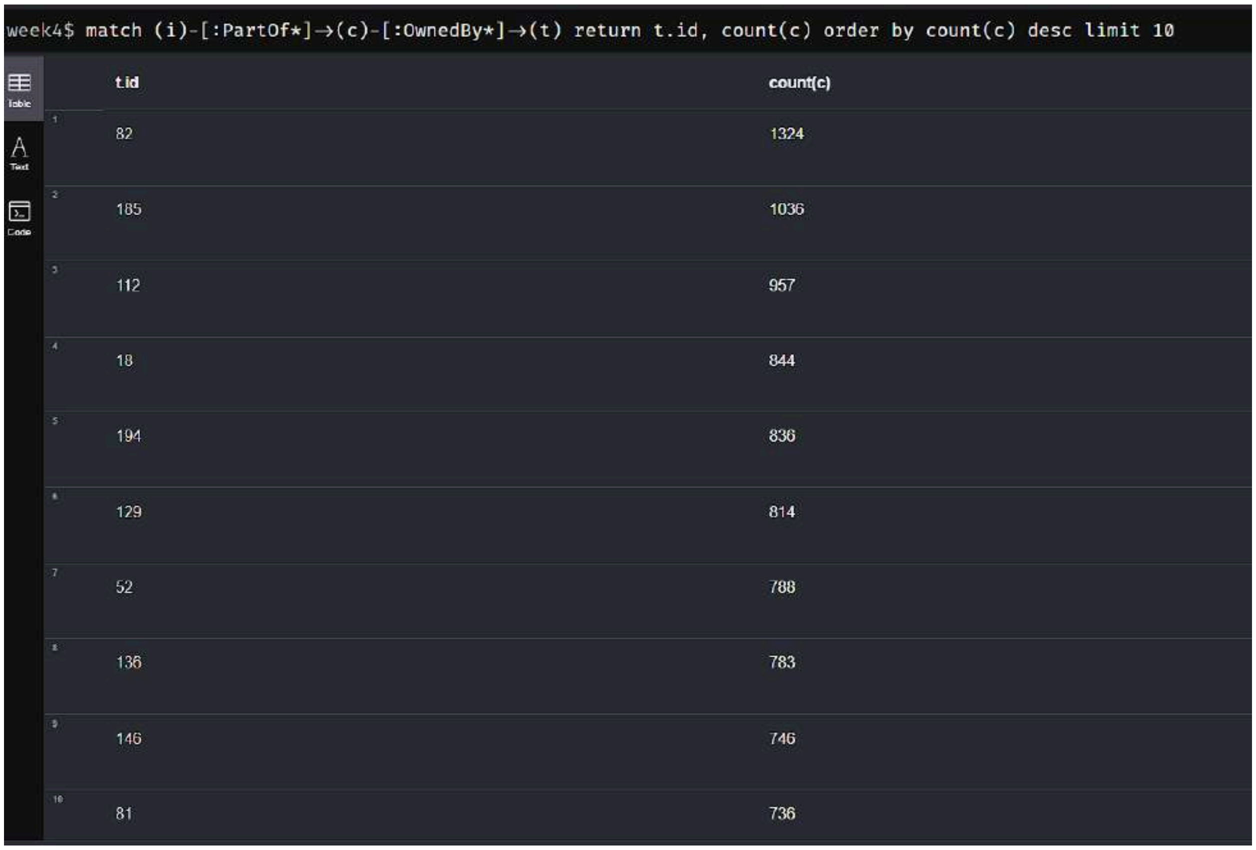
Chattiest Users

Users	Number of Chats
394	115
2067	111
1087	109

.

Chattiest Teams

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



```
week4$ match (i)-[:PartOf*]->(c)-[:OwnedBy*]->(t) return t.id, count(c) order by count(c) desc limit 10
```

	t.id	count(c)
1	82	1324
2	185	1036
3	112	957
4	18	844
5	194	836
6	129	814
7	52	788
8	136	783
9	146	746
10	81	736

Chattiest Teams

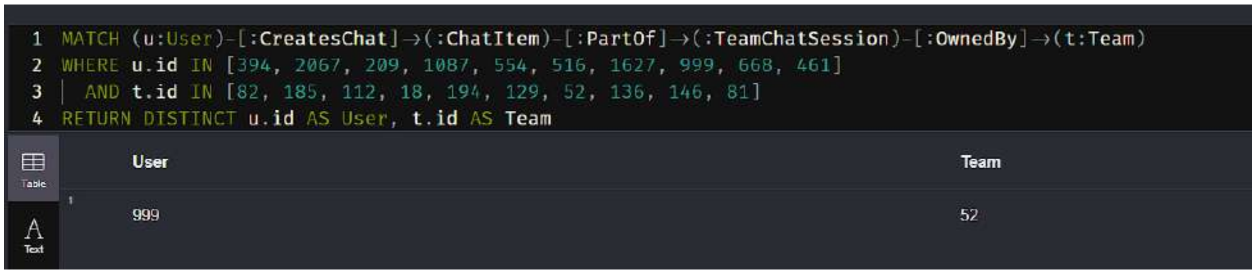
Match all ChatItem with a PartOd edge and connect them with a TeamChatSession node that have an OwnedBy edge connection them with any other node.

Teams	Number of Chats
82	1324
185	1036
112	957

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

```
MATCH
(u:User)-[:CreatesChat]->(:ChatItem)-[:PartOf]->(:TeamChatSession)-[:OwnedB
y]->(t:Team)
WHERE u.id IN [394, 2067, 209, 1087, 554, 516, 1627, 999, 668, 461]
```

```
AND t.id IN [82, 185, 112, 18, 194, 129, 52, 136, 146, 81]
RETURN DISTINCT u.id AS User, t.id AS Team
```



```
MATCH (u:User)-[:CreatesChat]→(:ChatItem)-[:PartOf]→(:TeamChatSession)-[:OwnedBy]→(t:Team)
WHERE u.id IN [394, 2067, 209, 1087, 554, 516, 1627, 999, 668, 461]
  AND t.id IN [82, 185, 112, 18, 194, 129, 52, 136, 146, 81]
RETURN DISTINCT u.id AS User, t.id AS Team
```

User	Team
999	52

Use this query to investigate if the most chattiest user are part of any chattiest team and it return as: userID 999 is part of teamID 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.

a. Connect mentioned users

```
MATCH (u1:User)-[:CreatesChat]->(:ChatItem)-[:Mentioned]->(u2:User)
MERGE (u1)-[:InteractsWith]->(u2)
```

b. Connect users responses with the chat creator

```
MATCH
(u1:User)-[:CreatesChat]->(:ChatItem)-[:RespondedTo]->(:ChatItem)<-[:Create
sChat]-(u2:User)
MERGE (u1)-[:InteractsWith]->(u2)
```

c. Eliminate all self interaction

```
MATCH (u1)-[r:InteractsWith]->(u1)
DELETE r
```

d. Calculate the cluster coefficient

```
MATCH (u1:User {id: 394})-[:InteractsWith]->(u2:User)
WITH collect(u2.id) AS neighbours, count(u2) AS k
```

```
MATCH (u3:User)-[iw:InteractsWith]->(u4:User)
WHERE u3.id IN neighbours AND u4.id IN neighbours
WITH count(iw) AS triangleCount, k
RETURN CASE WHEN k > 1 THEN triangleCount / (k * (k - 1) * 1.0) ELSE 0 END
AS Clustering_Coefficient

MATCH (u1:User {id: 2067})-[:InteractsWith]->(u2:User)
WITH collect(u2.id) AS neighbours, count(u2) AS k
MATCH (u3:User)-[iw:InteractsWith]->(u4:User)
WHERE u3.id IN neighbours AND u4.id IN neighbours
WITH count(iw) AS triangleCount, k
RETURN CASE WHEN k > 1 THEN triangleCount / (k * (k - 1) * 1.0) ELSE 0 END
AS Clustering_Coefficient

MATCH (u1:User {id: 209})-[:InteractsWith]->(u2:User)
WITH collect(u2.id) AS neighbours, count(u2) AS k
MATCH (u3:User)-[iw:InteractsWith]->(u4:User)
WHERE u3.id IN neighbours AND u4.id IN neighbours
WITH count(iw) AS triangleCount, k
RETURN CASE WHEN k > 1 THEN triangleCount / (k * (k - 1) * 1.0) ELSE 0 END
AS Clustering_Coefficient
```

Most Active Users (based on Cluster Coefficients)

User ID	Coefficient
394	0.750
2067	0.933
209	0.999

THE END of technical appendix.