Catch The Pink Flamingo Analysis

Technical Appendix

Produced by Eren Berkay Ünlü

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
ad-click.csv	Database of each click a user performs on an advertisement	timestamp: time when the advertisement click performed userId: id of the user that performed the click userSessionId: id of the session for the user that performed the click txId: id for the click adId: id of the advertisement that is clicked adCategory: category of the advertisement that is clicked
buy-clicks.csv	Database of each click a user performs to purchase	timestamp: time when the purchase made userId: id of the user that made the purchase userSessionId: id of the session for

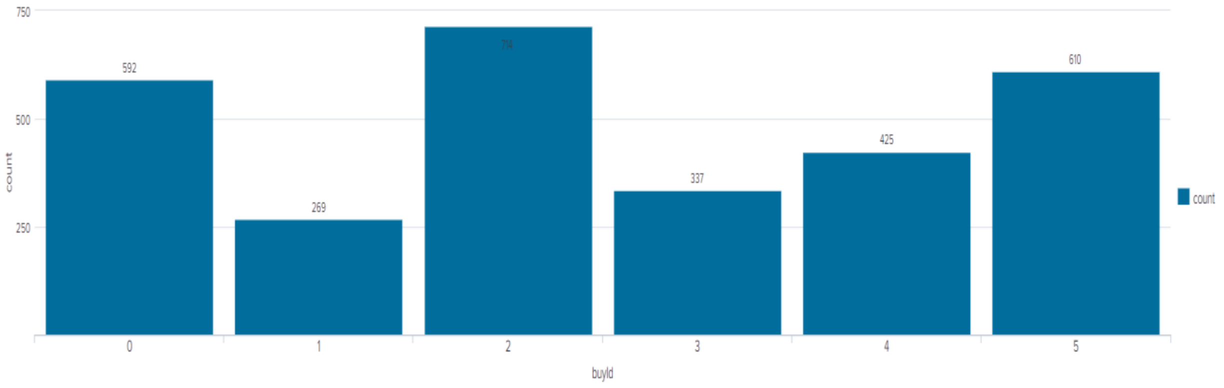
		the user that made the purchase txId: id for the purchase team: id of the team for the user that made the purchase buyId: id of the item that is purchased price: price of the item that is purchased
game-clicks.csv	Database of each click a user performs in game	timestamp: time when the in game click performed userId: id of the user that performed the click userSessionId: id of the session for the user that performed the click clickId: id for the in game click teamId: id of the team for the user that performed the click teamLevel: level of the the team of the user isHit: denotes whether the click was on a flamingo or not
level-events.csv	Database that holds each level event for a team	timestamp: time when the event occured eventId: id of the event eventType: type of the event teamId: id of the team teamLevel: level of the team
team-assignments.csv	Database that holds a record each time a user joins a team	timestamp: time when the user joined the team team: id of the team userId: id of the user assignmentId: id for the assignment
team.csv	Database of each team in the game	teamId: id of the team name: name of the team teamCreationTime: time when the team was created

		teamEndTime: time when the team drop to zero members strength: a measure for the success of the team currentLevel: current level of the team
user-session.csv	Database that holds each session a user is in	timestamp: time when the session is created or finished userId: id of the current user userSessionId: if of the session teamId: id of the team assignmentId: id for the assignment sessionType: whether the session is a start session or an end session teamLevel: level of the the team platformType: type of the platform for user during the session
users.csv	Database of the users	timestamp: time when the user is created userId: id of the user nick: nickname of the user twitter: twitter handler for the user dob: date of birth of the user country: two letter country code where the user lives

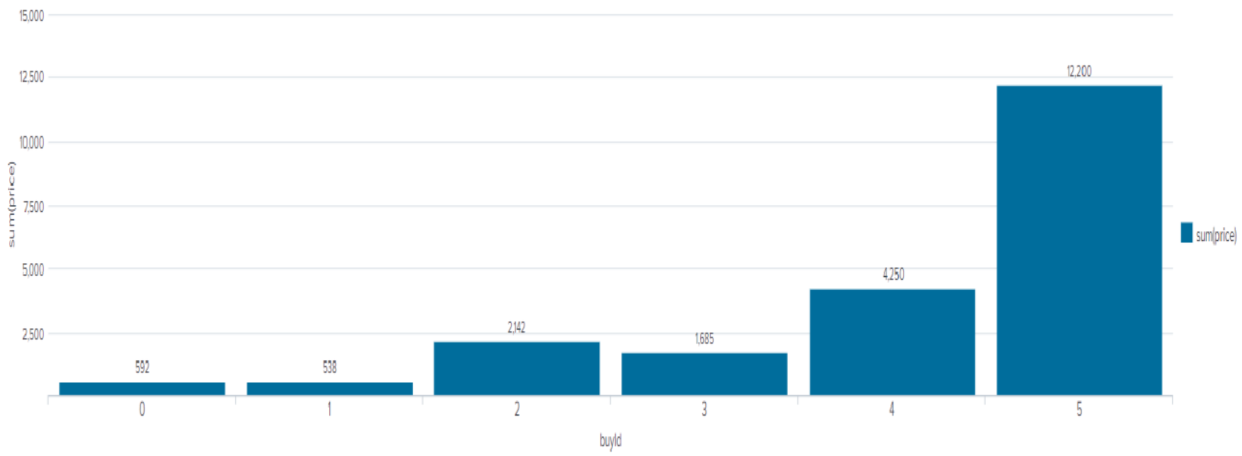
Aggregation

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

A histogram showing how many times each item is purchased:

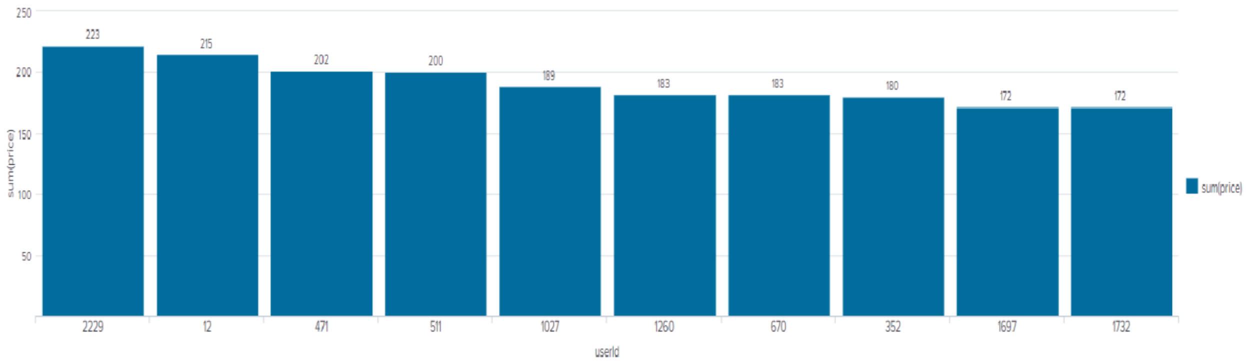


A histogram showing how much money was made from each item:



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	11.597
2	12	iPhone	13.068
3	471	iPhone	14.504

Classification with KNIME

Data Preparation

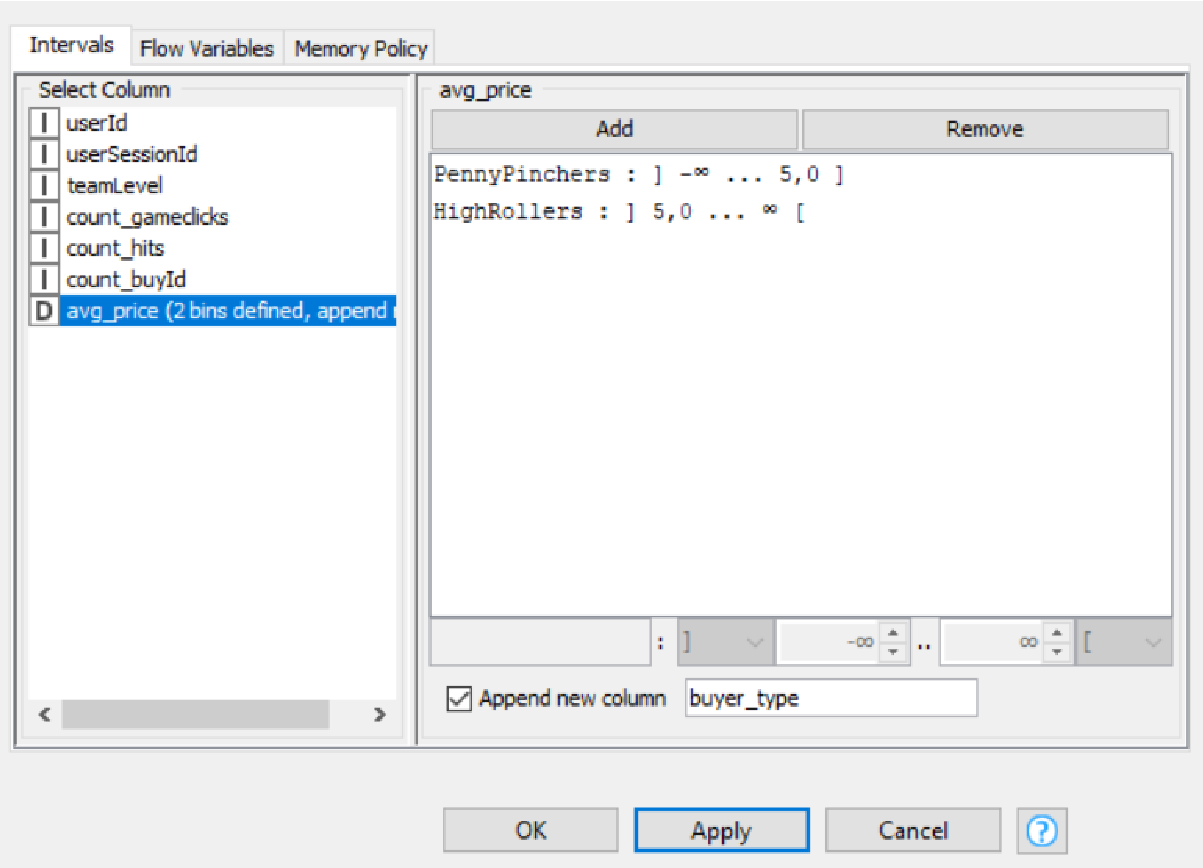
Analysis of combined_data.csv

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:



A new categorical attribute "buyer_type" is created. We defined two categories namely HighRollers (buyers of items that cost more than $5.00) and PennyPinchers (buyers of items that cost less than $5.00).

The creation of this new categorical attribute was necessary because it simplifies the classification of users and it is the base that we are going to use to build our decision tree.

Attribute Selection

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

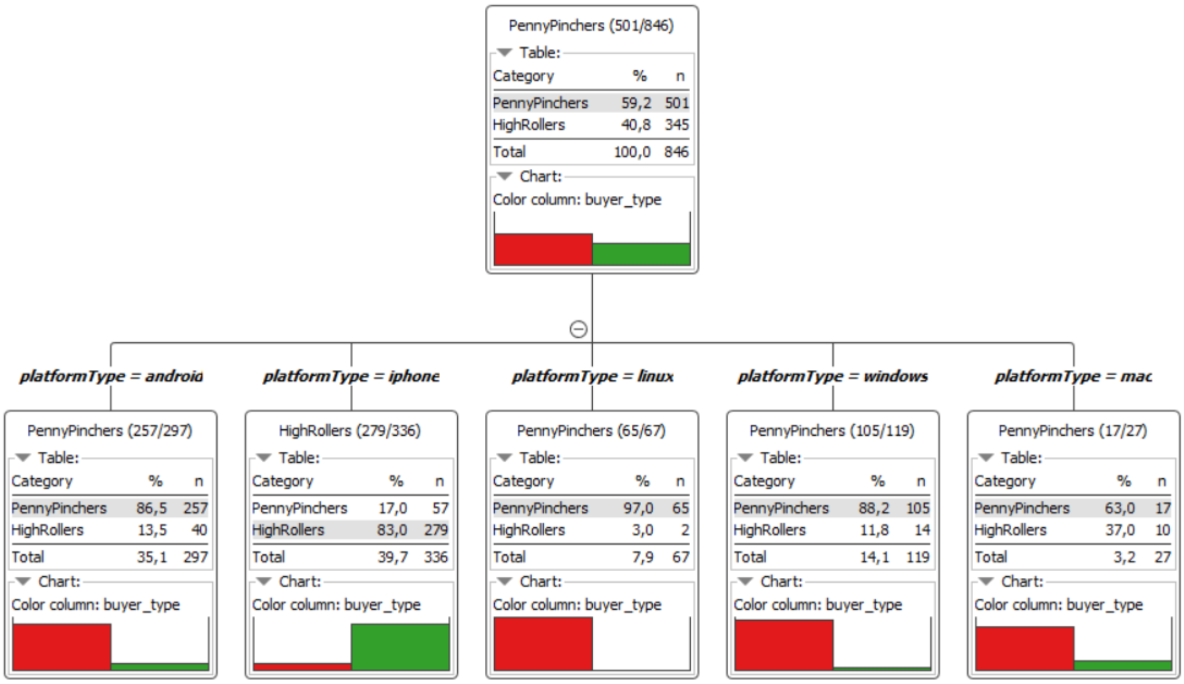
Attribute	**Rationale for Filtering**
userId	It is filtered since this attribute has no effect on what we are looking for.
userSessionId	It is filtered since this attribute has no effect on what we are looking for.
Avg_price	It is filtered since a new categorical attribute "buyer_type" is created which made this attributte not necessary anymore.

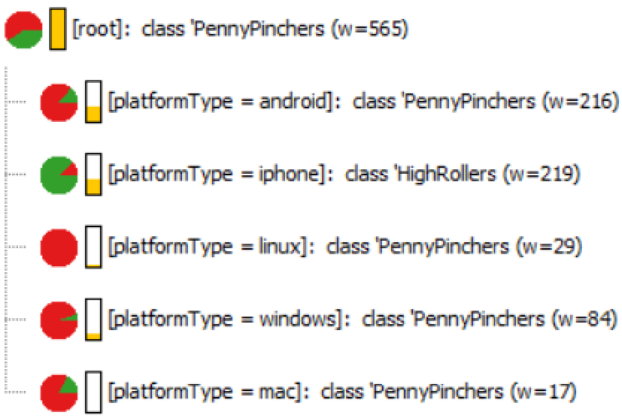
Data Partitioning and Modeling

The data was partitioned into train and test datasets. The train 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 for creating the decision tree, test data allows us to see the accuracy of the model.

When partitioning the data using sampling, it is important to set the random seed because we can get the same data every time we execute the model.

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





Evaluation

A screenshot of the confusion matrix can be seen below:

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

Correct classified: 500 Wrong classified: 65

Accuracy: 88,496 % Error: 11,504 %

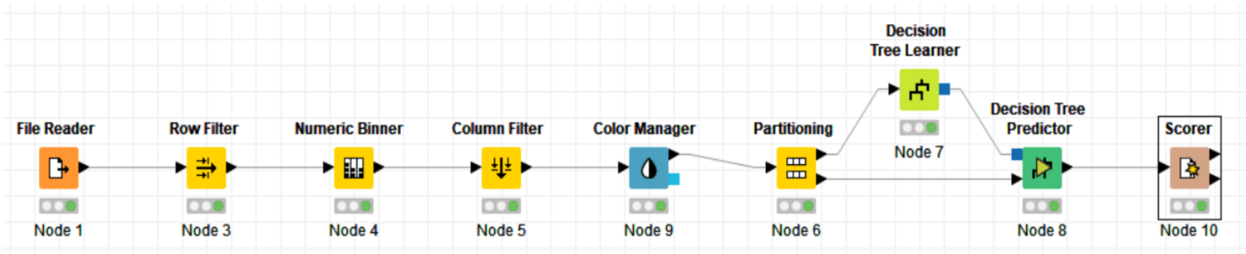
Cohen's kappa (κ) 0,76

As seen in the screenshot above, the overall accuracy of the model is **88,496 %**

For “buyer_type” PennyPinchers our model classified correctly 308 and incorrectly 27 of them.
For “buyer_type” HighRollers our model classified correctly 192 and incorrectly 38 of them.

Analysis Conclusions

The final KNIME workflow is shown below:



What makes a HighRoller vs. a PennyPincher?

Users who are HighRollers seem to use iPhone. On the other hand, in toher platforms most of the users are PennyPinchers.

Specific Recommendations to Increase Revenue
1. Target suggestions and promotions especially for iOS users.
2. A lot of cheap offers for other platforms so that they purchass more and more.

Clustering Analysis with Spark MLlib

Attribute Selection

features_used = ["revenue", “totalAdClicks”]

Attribute	Rationale for Selection
totalRevenue	how much the users spend in the game will give us a monetary value of that user
totalBuyClicks	how often the users click to buy will show the engagement
totalAdClicks	how often the users click on the ad will show how likely they will spend money

Training Data Set Creation

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

```
trainingDF = combinedDF[['totalAdClicks','totalBuyClicks','totalRevenue']]
trainingDF.head(n=5)
```

	totalAdClicks	totalBuyClicks	totalRevenue
0	44	9	21.0
1	10	5	53.0
2	37	6	80.0
3	19	10	11.0
4	46	13	215.0

Dimensions of the final data set: 543 rows x 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)
```

Cluster centers formed are given in the table below:

Cluster #	Center
1	36.44134078, 926.11731844, 46.96648045
2	24.98746082, 357.95924765, 35.06583072
3	32.35555556, 2310.64444444, 39.42222222

These clusters can be differentiated from each other as follows:

Cluster 1 is different from the others in that users with the highest revenue and adclick are not the ones who are playing the most but an intermediate result in game clicks.
Cluster 2 is different from the others in that the users who play the less also produces the less ad revenue and click count
Cluster 3 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

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

	0	1	2	3	4
summary	count	mean	stddev	min	max
totalAdClicks	709	32.208744710860366	16.38412081704256	1	73
revenue	709	44.64880112834979	44.9445287652853	1.0	278.0

```
print(km_model.centers)
```

[array([36.44134078, 926.11731844, 46.96648045]), array([24.98746082, 357.95924765, 35.06583072]), array([32.35555556, 2310.64444444, 39.42222222])]

Recommended Actions

Action Recommended	Rationale for the action
Increase ads to users who play a lot	It was seen that users who play a lot are also the users who spend less and click less on ads. If we increase ads to users who play a lot, it will promote these users to spend more and therefore increase the revenue
Show higher price ads to users who spend more	If we show higher price ads to users who spend more, we can increase the revenue faster.The users who spend the more also do not play too much, thus by showing them the more valuable ads first, we can increase the revenue faster

Graph Analytics with Neo4j

Modeling Chat Data using a Graph Data Model

The graph model is a network based on the chat interactions between users. A user in a team can create a chat session and then create chat item in that session. It is possible for a user to mention another user. Edges signify timestamp for the chat items.

Creation of the Graph Database for Chats

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

i) The schema of the 6 CSV files

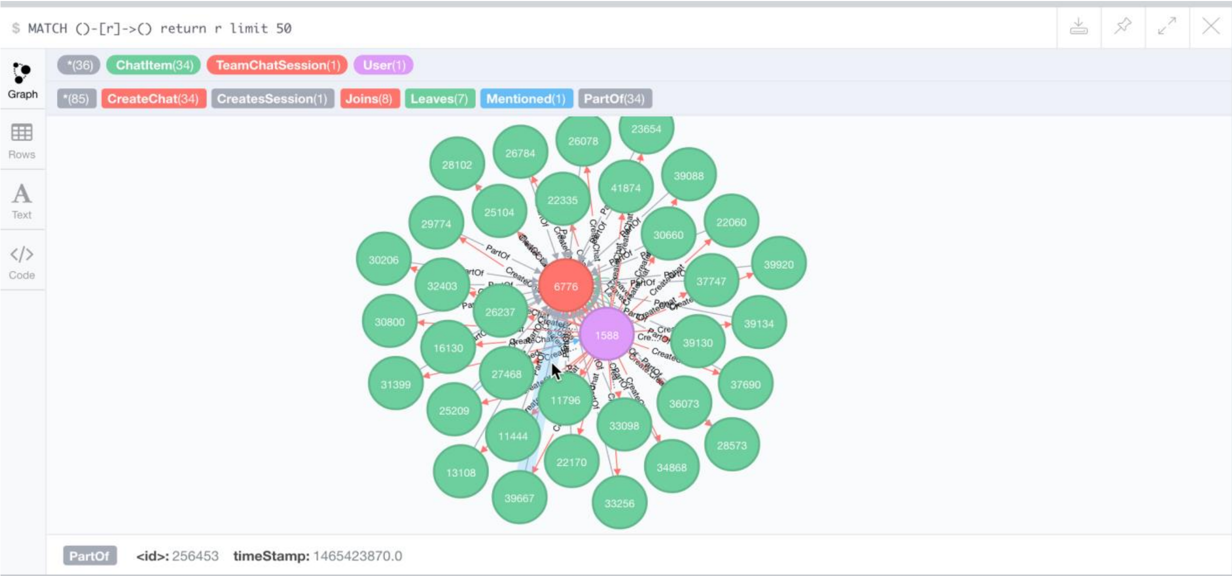
File name	Fields
chat_create_team_chat.csv	userID teamID teamChatSessionID timestamp
chat_join_team_chat.csv	userID teamChatSessionID timestamp
chat_leave_team_chat.csv	userID teamChatSessionID timestamp
chat_item_team_chat.csv	userID teamChatSessionID chatItemID timestamp
chat_mention_team_chat.csv	userID chatItemID

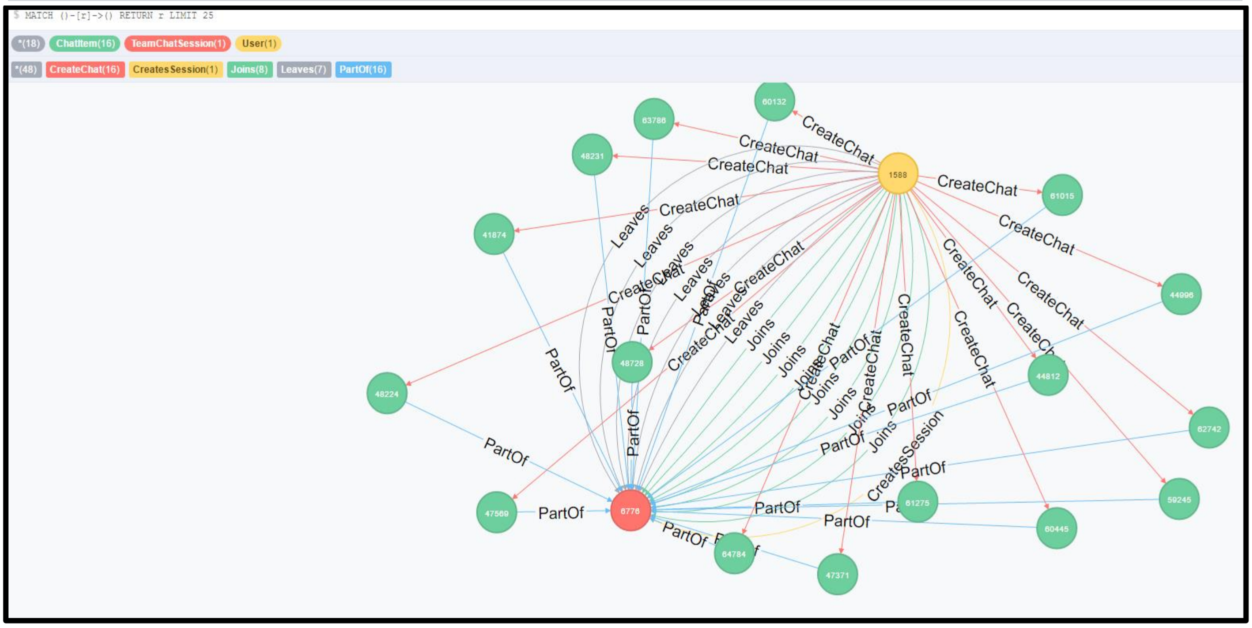
	timestamp
chat_respond_team_chat.csv	chatItemID_1 chatItemID_2 timestamp

ii) The loading process and a sample LOAD command

```
LOAD CSV FROM "file:///chat-data/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)
```

iii) A screenshot of some part of the graph you have generated. The graphs include clearly visible 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=(a)-[:ResponseTo*]->(b)
RETURN p, length(p)
ORDER BY length(p) desc limit 1
```

The longest conversation chain in the chat data has a path lenght of 9 and there are 10 chats in this path.

```
match p=(c:ChatItem)-[:ResponseTo*]->(j:ChatItem)
where length(p)=9
with p
match q=(u:User)-[:CreateChat]-(c:ChatItem)
where (c IN NODES(p))
return count(distinct u)
```

Count the number of distinct users in the longest path. The result is 5.

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

Chattiest Users

Users	Number of Chats
394	115
2067	111
209	109

```
match (u:User)-[:CreateChat]-(i:ChatItem)
return u.id as Users, count(u.id) as Num_Chats
order by count(u.id) desc limit 10
```

Users	Num_Chats
394	115
2067	111
209	109
1087	109
554	107
516	105
1627	105
999	105
668	104
461	104

Chattiest Teams

Teams	Number of Chats
82	1324
185	1036
112	957

```
match (:ChatItem)-[:PartOf]->(:TeamChatSession)-[:OwnedBy]->(t:Team)
return t.id as Teams, count(t.id) as Num_Chats
order by count(t.id) desc limit 10
```

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

```
match (u:User)-[:CreateChat]->(: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
```

This query is used to investigate if the most chattiest users are part of any of the chattiest teams. It returns one result as the user with the userID 999 is part of the team with teamID 52.

How Active Are Groups of Users?

```
match (u1:User)-[:CreateChat]->(:ChatItem)-[:Mentioned]->(u2:User)
merge (u1)-[:InteractsWith]->(u2)
```

```
match (u1:User)-[:CreateChat]->(:ChatItem)-[:ResponseTo]-(:ChatItem)<-[:CreateChat]-(u2:User)
merge (u1)-[:InteractsWith]->(u2)
```

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

```
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))
return count(iw)/(k * (k - 1) * 1.0) as clusteringCoefficient
```

We first connected the mentioned users. Then connected the user responses with the chat creator. Eliminated all self interactions. Finally calculated the cluster coefficients.

Most Active Users (based on Cluster Coefficients)

User ID	Coefficient
394	0.9167
2067	0.7679
209	0.9524