

Technical Appendix

Catch the Pink Flamingo Analysis

Produced by: Ramana Sonti

Acquiring, Exploring and Preparing the Data

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
add-clicks.csv	ERD table: AdClicks A line is added to this file when a player clicks on an advertisement in the Flamingo app.	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	ERD table: InAppPurchases	timestamp: when the purchase was

	<p>A line is added to this file when a player makes an in-app purchase in the Flamingo app.</p>	<p>made.</p> <p>txId: a unique id (within buy-clicks.log) for the purchase</p> <p>userSessionId: the id of the user session for the user who made the purchase</p> <p>team: the current team id of the user who made the purchase</p> <p>userId: the user id of the user who made the purchase</p> <p>buyId: the id of the item purchased</p> <p>price: the price of the item purchased</p>
users.csv	<p>ERD table: User</p> <p>This file contains a line for each user playing the game.</p>	<p>timestamp: when user first played the game.</p> <p>userId: the user id assigned to the user.</p> <p>nick: the nickname chosen by the user.</p> <p>twitter: the twitter handle of the user.</p> <p>dob: the date of birth of the user.</p> <p>country: the two-letter country code where the user lives.</p>

team.csv	ERD table: Team This file contains a line for each team terminated in the game.	eamId: 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	ERD table: TeamAssignment 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.	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	ERD table: LevelEvent A line is added to this file each time a team starts or finishes a level in the game	timestamp: when the event occurred. eventId: a unique id for the event

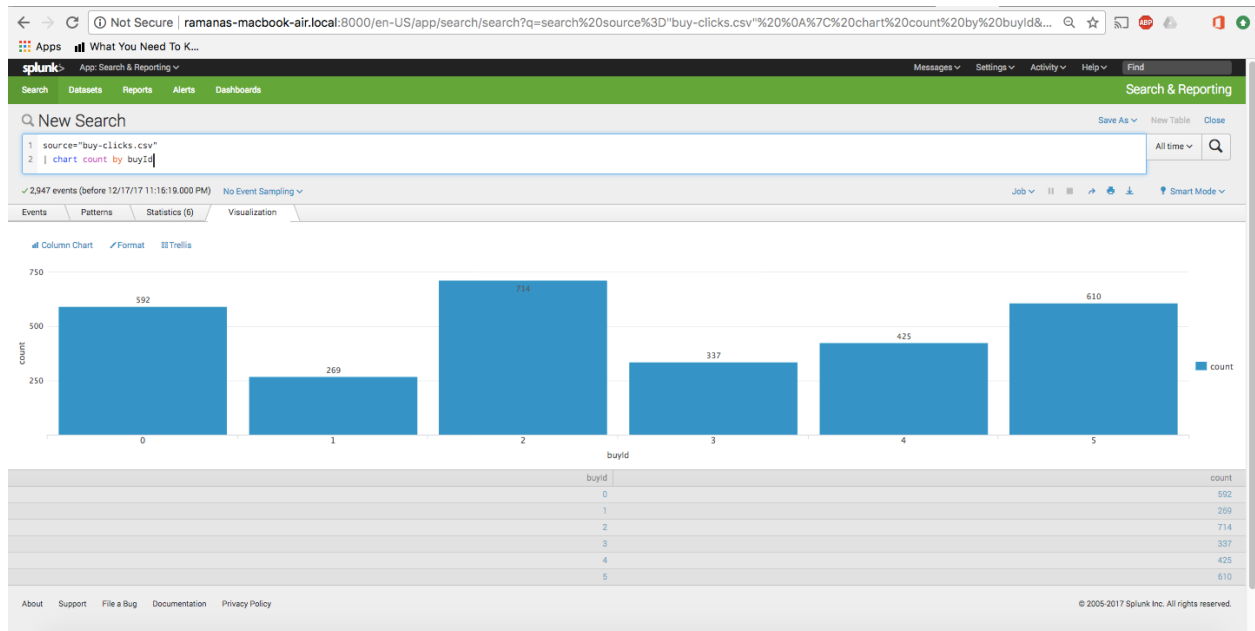
		<p>teamId: the id of the team</p> <p>teamLevel: the level started or completed</p> <p>eventType: the type of event, either start or end</p>
user-session.csv	<p>ERD table: User_Sessions</p> <p>Each line in this file describes a user session, which denotes when a user starts and stops playing the game. 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.</p>	<p>timestamp: a timestamp denoting when the event occurred.</p> <p>userSessionId: a unique id for the session.</p> <p>userId: the current user's ID.</p> <p>teamId: the current user's team.</p> <p>assignmentId: the team assignment id for the user to the team.</p> <p>sessionType: whether the event is the start or end of a session.</p> <p>teamLevel: the level of the team during this session.</p> <p>platformType: the type of platform of the user during this session.</p>
game-clicks.csv	<p>ERD table: GameClicks</p> <p>A line is added to this file each time a user performs a click in the</p>	<p>timestamp: when the click occurred.</p> <p>clickId: a unique id for the click.</p>

	game.	<p>userId: the id of the user performing the click.</p> <p>userSessionId: the id of the session of the user when the click is performed.</p> <p>isHit: denotes if the click was on a flamingo (value is 1) or missed the flamingo (value is 0)</p> <p>teamId: the id of the team of the user</p> <p>teamLevel: the current level of the team of the user</p>
<Fill In>	<Fill in short phrase>	<Fill In: Name and describe all fields>

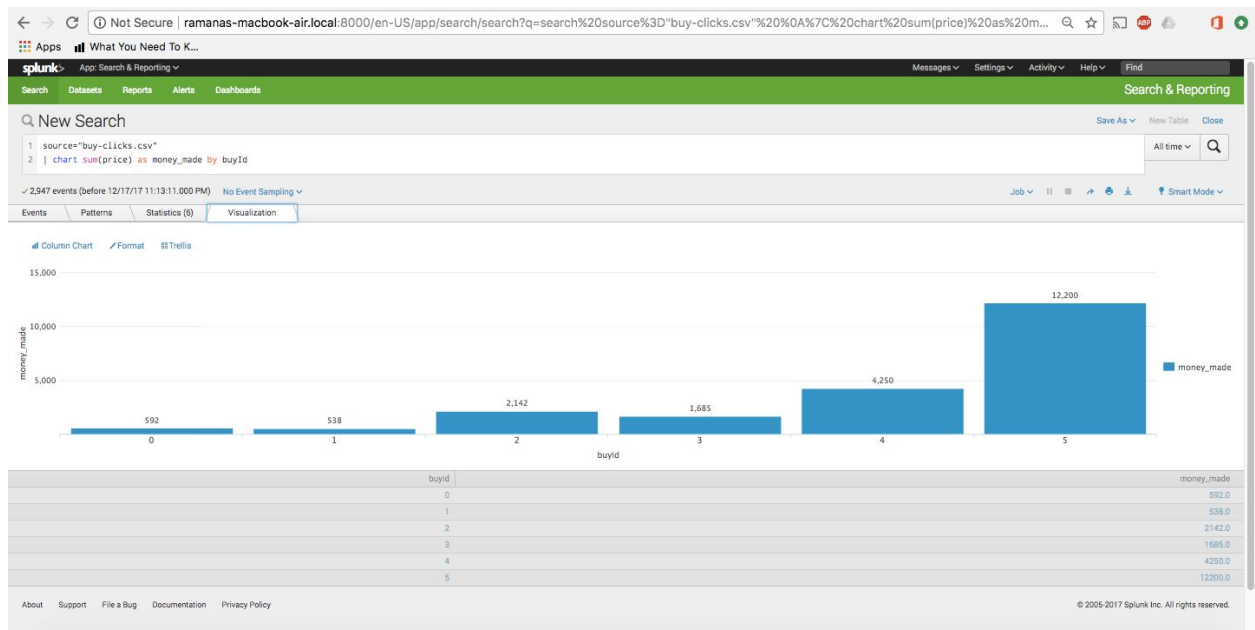
Aggregation

Amount spent buying items	21407.0
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:



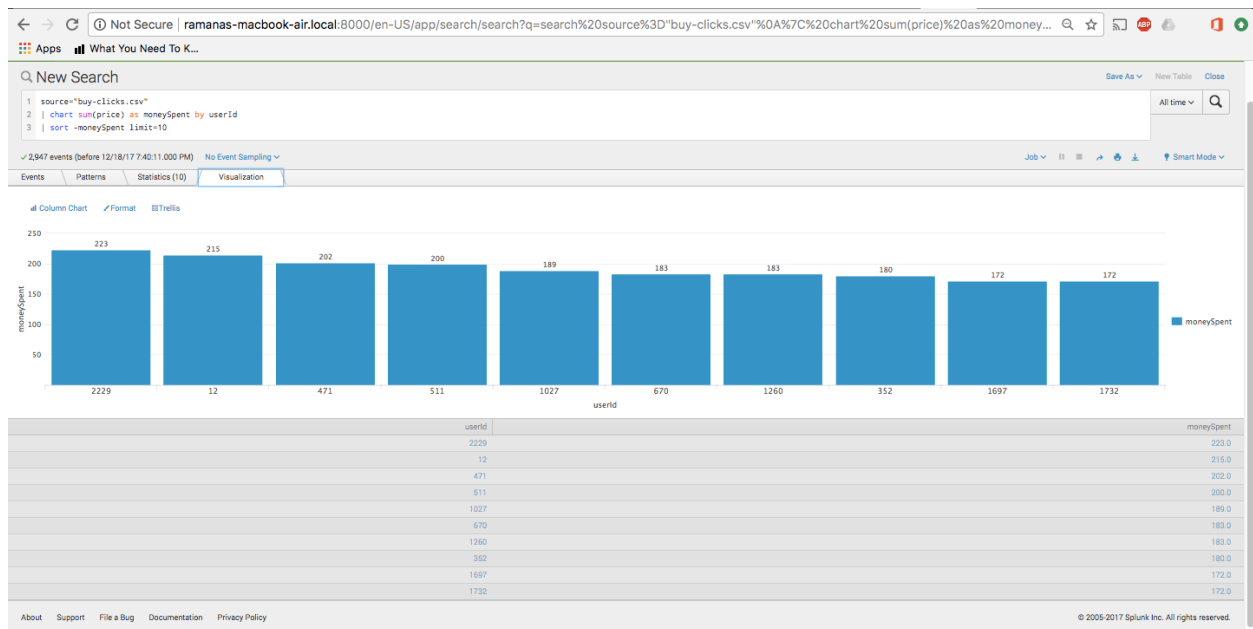
Report on your key findings from your aggregation analysis:

1. Most of the money was made by item 5 (\$12200) which is also the most expensive of all.
2. Item 2 was the top seller in terms of quantity followed by item 5.

- Item 0 was least expensive.
- Item 1 made the least amount of money and sold the least in terms of quantity

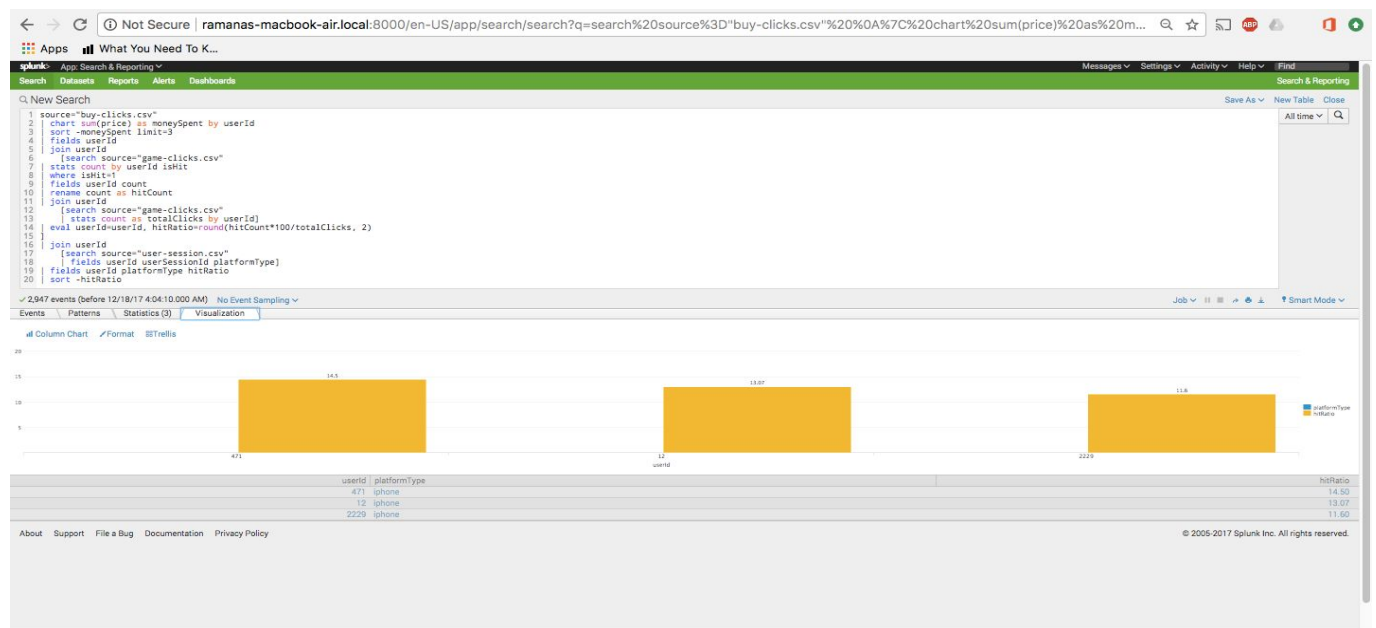
Filtering

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	471	iphone	14.50
2	12	iphone	13.07
3	2229	iphone	11.60



Report on your key findings from your filtering analysis:

1. All top spenders use iphone as a platform
2. Top three spenders spent over \$200 each
3. The best hit ratio from any player (happens to be the one with userId 471) is 14.5%

Data Classification Analysis

Data Preparation

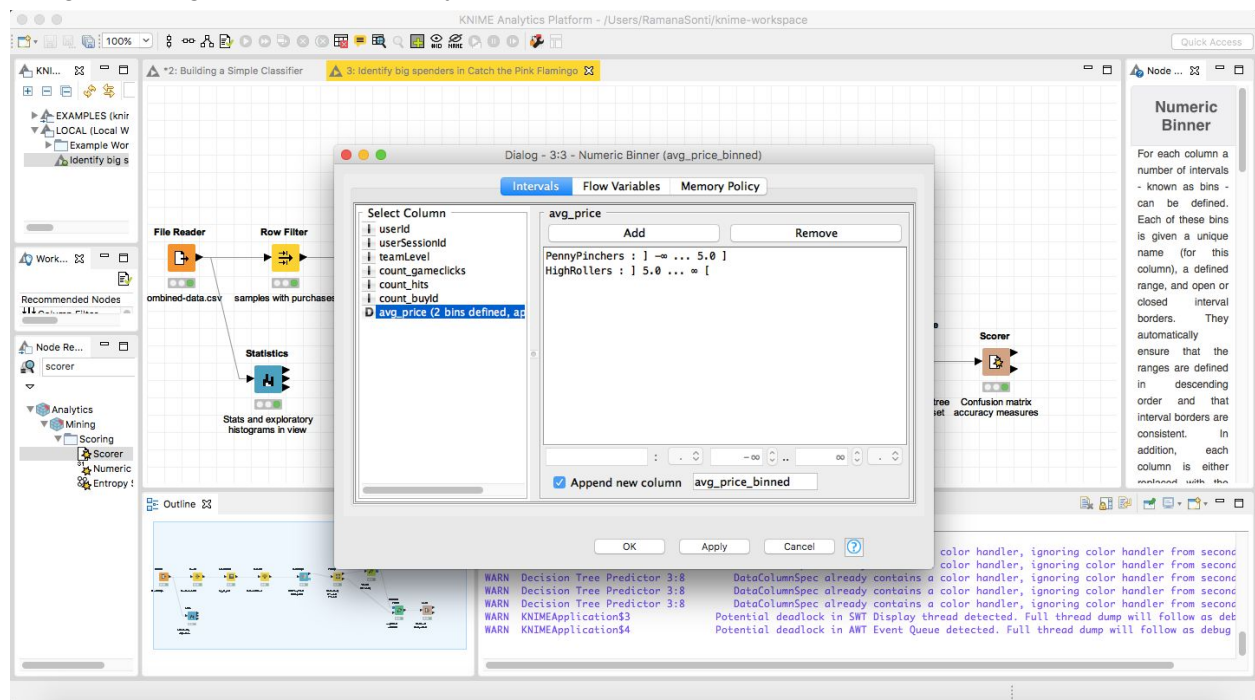
Analysis of combined_data.csv

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:



Design of the attribute:

A numeric binner node was used to create a new variable avg_price_binned with two bins. The range used for HighRollers bin was] 5.0 ... inf [and the range used for PennyPinchers bin was] -inf ... 5.0 [. ']' before 5.0 indicates excluding 5.0 and '[' after 5.0 indicates inclusive of 5.0.

Append new column option was checked in order to avoid overwriting/using the existing column.

The creation of this new categorical attribute was necessary because <Fill in 1-2 sentences>.

The goal is to classify the users as buyers of big ticket items vs buyers of inexpensive items based on avg_price attribute which is a continuous variable ranging from 1 to 20. As the classification involves associating a label or qualifier to the group of users who made the purchases in a given price range. A new variable has to be created to take the value HighRollers for the users whose avg_price was above \$5 and the value PennyPinchers for the users whose avg_price was \$5 or less.

Attribute Selection

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

Attribute	Rationale for Filtering
userID	ID of a user is not expected influence the user's ability or his/her need to influence in-app purchases.
userSessionID	Though the User's session id makes the record unique, it is not expected to influence his/her ability or the need to spend more money on purchases.
avg_price	The categorical variable avg_price_binned was derived from continuous variable avg_price which is no longer needed.

Data Partitioning and Modeling

The data was partitioned into train and test datasets.

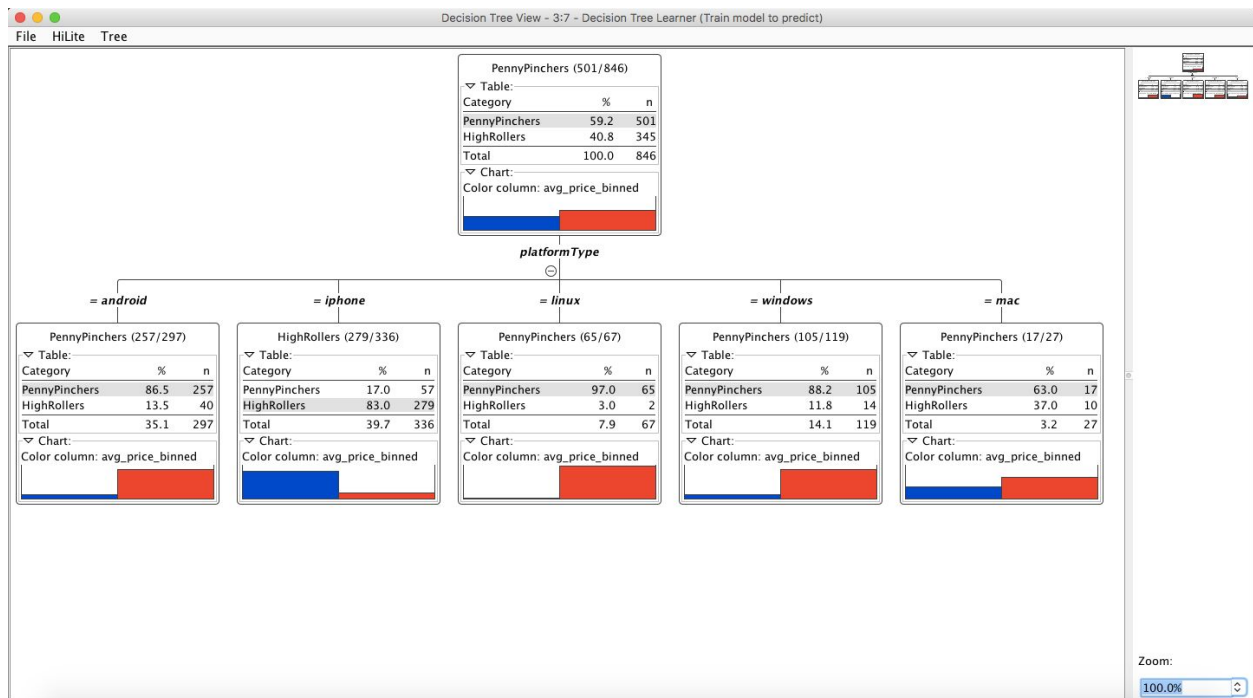
The training (60%) data set was used to create the decision tree model.

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

This is important because you want to test your model on data that was not used to train (i.e., create) it.

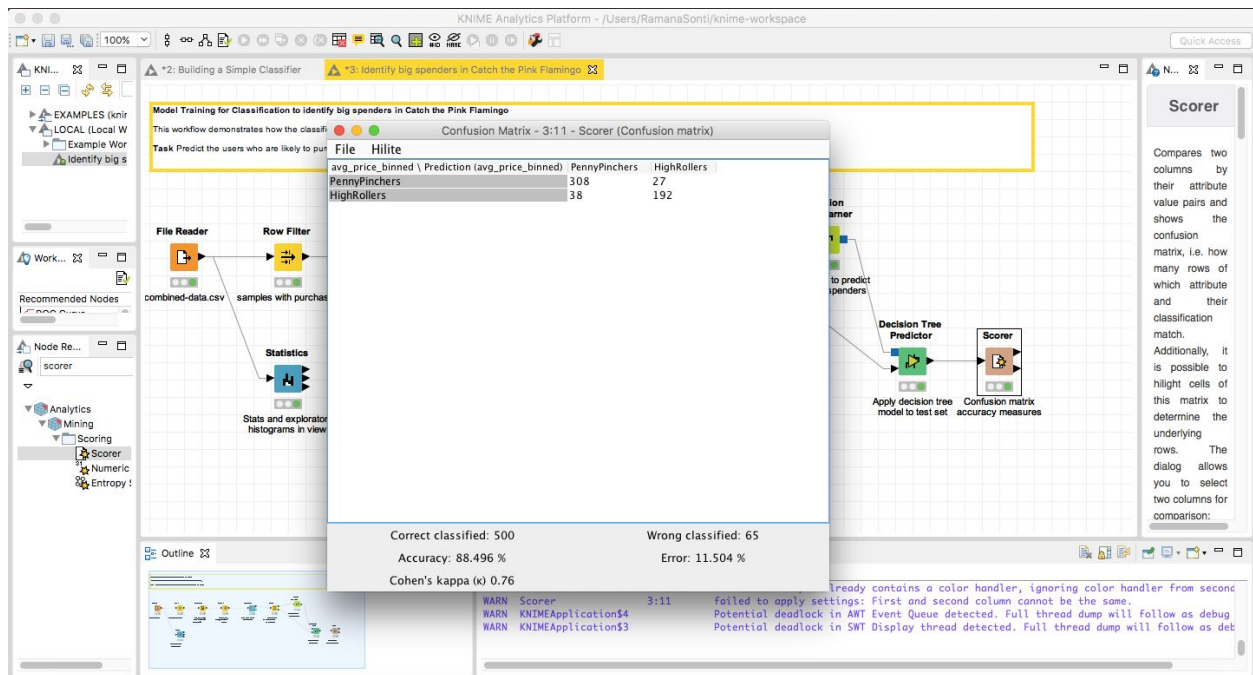
When partitioning the data using sampling, it is important to set the random seed because it is to ensure that we will get the same partitions every time we execute this node. It is important to get reproducible results.

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



Evaluation

A screenshot of the confusion matrix can be seen below:



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

Out of 565 entries in the test set, the model predicted 500 of them correctly and 65 them in correctly.

308 - Number of users correctly predicted by the model as PennyPinchers

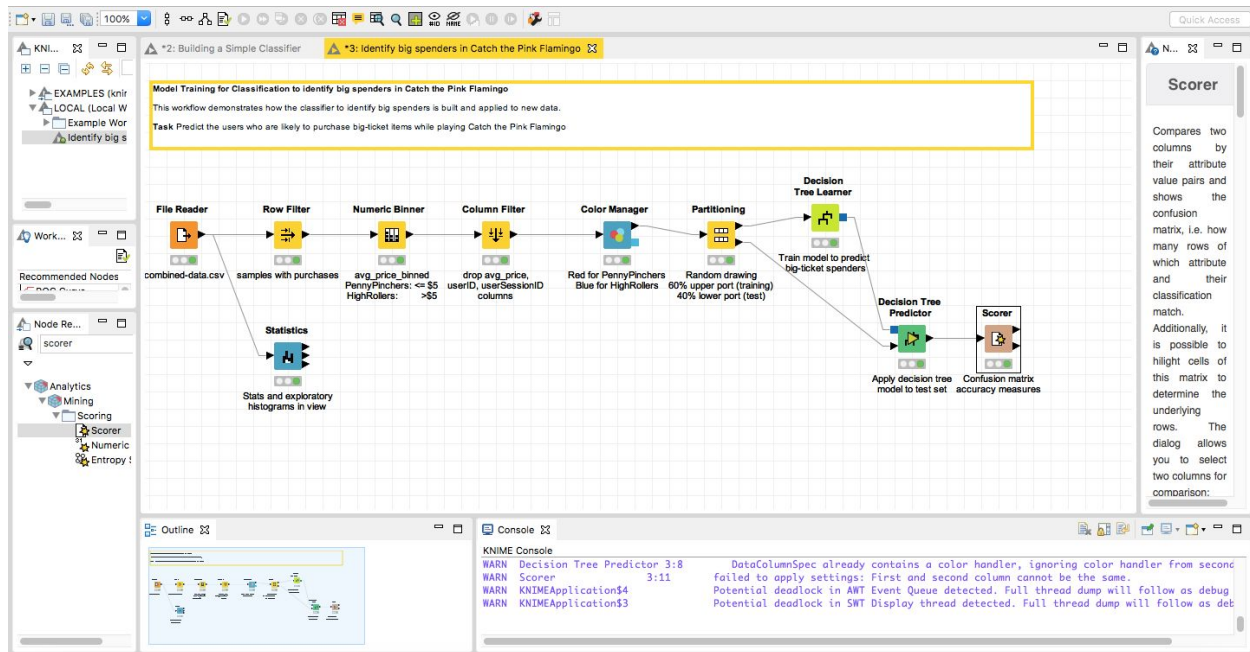
192 - Number of users correctly predicted by the model as HighRollers

38 - Number of HighRollers incorrectly predicted by the model as PennyPinchers (False Positives)

27 - Number of PennyPinchers incorrectly predicted by the model as HighRollers (False Negatives)

Analysis Conclusions

The final KNIME workflow is shown below:



What makes a HighRoller vs. a PennyPincher (based on the insights from analysis)?

As shown in the decision tree, the model predicts that 83% of the users that play from platformType iphone are HighRollers and they are 39.7% of the total test dataset. The rest of the users that use other platformTypes are mostly PennyPinchers.

Specific Recommendations to Increase Revenue

1. Target iphone users more with the ad campaigns as they tend to spend more.
2. Introduce more in-app purchases.

Clustering Analysis

Attribute Selection

Attribute	Rationale for Selection
Number of gameclicks per hour by each user	This metric provides important insight into answering the following questions: Do users with high rate of gameclicks tend to spend more or less on ads? Do users with high rate of gameclicks tend to click more on ads or less?
Total money spent on all ad categories by each user	This metric provides important insight into answering the following questions: What type of users spend more money on ads in general? How do aspects like total ad clicks and the rate of gameclicks relate to total money spent by a user?
Total number of ad clicks per user	This metric provides important insight into answering the following questions: Do users who click more on ads really buy more? Do users with high number of ad clicks tend to have more/less number of game clicks?
<Optional Fill in>	<Optional Fill in 1-3 sentences>

Training Data Set Creation

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

Screenshot of the original spark dataframe prepared before splitting into training and test datasets :

The screenshot shows a Jupyter Notebook interface with the following content:

Code Cell 187: `sparkDF.toPandas().head()`

Output 187:

	userid	money_spent	count_adclicks	gameclicks_per_hour
0	1	21.0	44.0	2.0
1	8	53.0	10.0	5.0
2	9	80.0	37.0	1.0
3	10	11.0	19.0	12.0
4	12	215.0	46.0	2.0

Code Cell 188: `sparkDF.toPandas().shape`

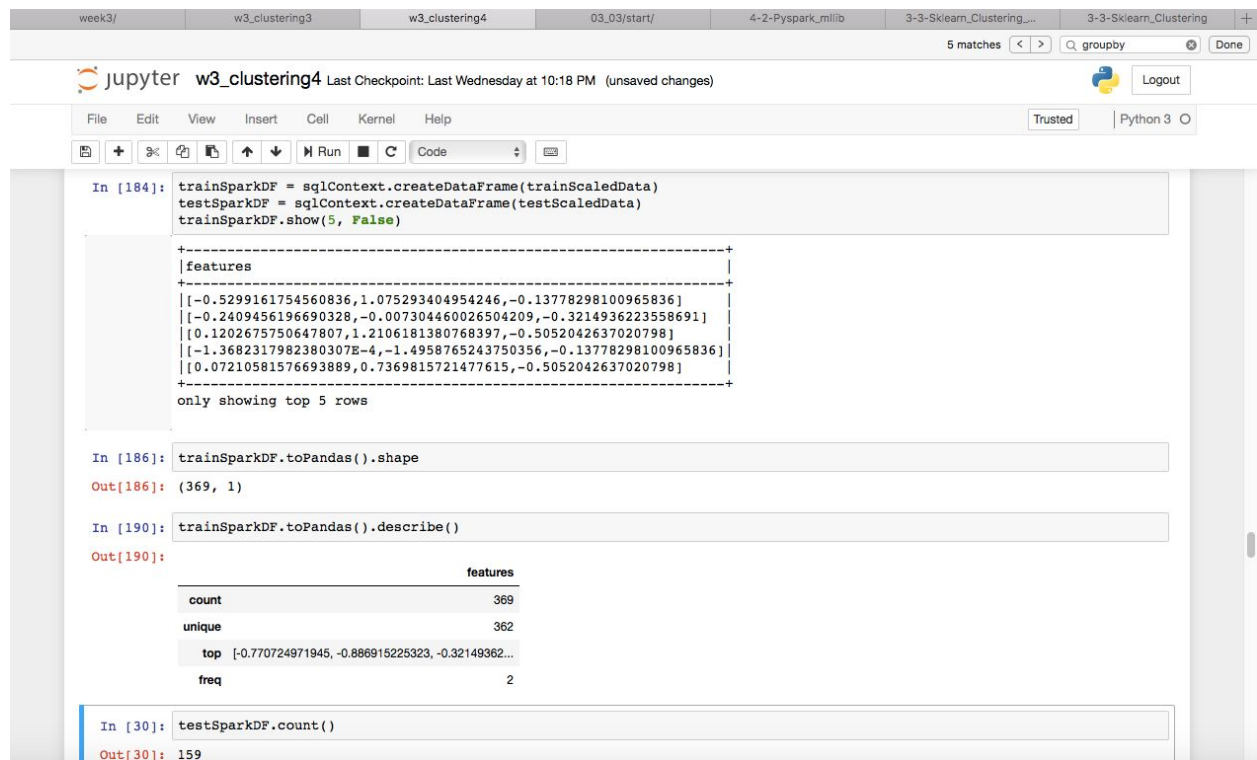
Output 188: `(528, 4)`

Code Cell 189: `sparkDF.toPandas().describe()`

Output 189:

	userid	money_spent	count_adclicks	gameclicks_per_hour
count	528.000000	528.000000	528.000000	528.000000
mean	1219.142045	40.005682	30.107955	3.750000
std	684.995916	41.526722	14.779264	5.443343
min	1.000000	1.000000	1.000000	1.000000
25%	647.500000	10.000000	17.000000	2.000000
50%	1234.500000	25.000000	31.000000	2.000000
75%	1795.000000	55.000000	42.250000	4.000000
max	2387.000000	223.000000	67.000000	60.000000

Screenshot of the training dataframe after the feature extraction. Please note that I had split the original dataset into training and test datasets after the feature extraction.



The screenshot shows a Jupyter Notebook with the following content:

```
In [184]: trainSparkDF = sqlContext.createDataFrame(trainScaledData)
          testSparkDF = sqlContext.createDataFrame(testScaledData)
          trainSparkDF.show(5, False)
```

Output for In [184]:

```
+-----+
|features|
+-----+
|[-0.5299161754560836,1.075293404954246,-0.13778298100965836]|
|[-0.2409456196690328,-0.007304460026504209,-0.3214936223558691]|
|[0.1202675750647807,1.2106181380768397,-0.5052042637020798]|
|[-1.3682317982380307E-4,-1.4958765243750356,-0.13778298100965836]|
|[0.07210581576693889,0.7369815721477615,-0.5052042637020798]|
+-----+
only showing top 5 rows
```

```
In [186]: trainSparkDF.toPandas().shape
Out[186]: (369, 1)
```

```
In [190]: trainSparkDF.toPandas().describe()
```

Output for In [190]:

	features
count	369
unique	362
top	[-0.770724971945, -0.886915225323, -0.32149362...]
freq	2

```
In [30]: testSparkDF.count()
Out[30]: 159
```

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

The original dataset I had was 528 x 4 including the userId column after dropping rows with NaN/inf.

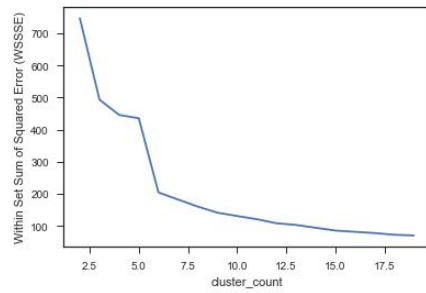
The training dataset I had was 369 x 1 (70% of the original 528 rows). The single column is a vector of three elements money_spent, total_adclicks, and gameclicks_per_hour.

of clusters created: 6

I have tested the training dataset from 2 to 20 clusters and plotted the cluster_count against WSSSE as shown by the elbow curve below. I have selected 6 as the value for number of clusters because the decrease in WSSSE is much more gradual with increasing number of clusters beyond 6.


```
In [194]: plt.plot(wssseDF['cluster_count'], wssseDF['wssse'])
plt.xlabel('cluster_count')
plt.ylabel('Within Set Sum of Squared Error (WSSSE)')
```

```
Out[194]: Text(0,0.5,'Within Set Sum of Squared Error (WSSSE)')
```



Cluster Centers

Cluster #	Cluster Center [money_spent, count_adclicks, gameclicks_per_hour]	Number of users Total: 369 trained
0	[1.04497335, 0.56475009, -0.25802995]	55
1	[-0.41789708, -0.89960192, -0.23455912]	112
2	[-0.39319892, 0.86139302, -0.30223364]	124
3	[-0.61821273, -1.58609301, 8.3741434]	3
4	[2.80158091, 0.76300556, -0.32855942]	26
5	[-0.57218058, -0.95319673, 1.00197283]	49

Please note that running the model on the test set yielded similar results.

These clusters can be differentiated from each other as follows:

Cluster 0 is different from the others in that

- . it ranks 2nd in the average money spent per user.
- . the users in this cluster rank 3rd in the number of ad-clicks per user (average).
- . the average number of game clicks per user in this groups ranks 5th

Cluster 1 is different from the others in that

- . The activity from the users in this group in all three categories below average
- . The game-clicks per hour count per user is the least for this cluster compared to other clusters.

Cluster 2 is different from the others in that

- . The amount of money spent per user is the lowest for this cluster compared to other clusters
- . Interestingly, this cluster ranks first for number ad-clicks per user
- . game-clicks per hour per user is below average

Cluster 3 is different from the others in that

- . this cluster has highest number of game-clicks per hour per user.
- . the amount of money spent per user on ads in this cluster is the lowest
- . the number of ad-clicks per user is also the lowest

Cluster 4 is different from the others in that

- . this cluster has the users with the most amount of money spent per user
- . it ranks 2nd for number of ad-clicks per user
- . the number of game-clicks per hour per user is the lowest for this cluster

Cluster 5 is different from the others in that

- . amount of money spent per user is much below average if not the lowest
- . number of ad-clicks per user is much below average if not the lowest
- . it ranks 2nd for the number of game-clicks per hour per user

Recommended Actions

Action Recommended	Rationale for the action
Charge Ad sponsors more	Cluster 2 (3rd cluster) has the highest number of users (about $\frac{1}{3}$ of the sample set). They rank first in number of ad-clicks per person though they spent much less per person towards in-app purchases. The revenue should increase if Eglence charge more for displaying ads to the users.
Provide incentives to users to click on Ads	Cluster 1 (2nd cluster) has almost $\frac{1}{3}$ of the total number of users in the training sample set. Their tendency to click on ads is not that great. They also have the least number of game-clicks per hour per user. Since this is a large segment of the user base , Eglence should come up with a plan to incentivise clicking on ads.
<Optional Fill in>	<Optional Fill in 1-3 sentences>
<Optional Fill in>	<Optional Fill in 1-3 sentences>

Graph Analytics Analysis

Modeling Chat Data using a Graph Data Model

Prompt: (Describe the graph model for chats in a few sentences. Try to be clear and complete.)

Graph model for chats models the chatting activity of the active users. The four entity types User, Team, TeamChatSession, and ChatItem correspond to four node labels. Team chat sessions are created when a user creates a new chat session for a new team. One team will have only one chat window at any given time. A chat item is the actual chat with an optional ability to mention another user. The message part of the text is excluded as the goal is not to analyze the message text itself.

The edges of the graph can be represented by different actions taken by the user. A user can create a session for a team, create a new chat in the session, may join or leave a chat session and may mention another user. The users can send a response to a previous chat item. This creates multiple threads of conversation. A chat session is owned by a team and a specific chat item will be part of that session. Each user's action has timestamp which is recorded as seconds elapsed from a specific point in time.

Creation of the Graph Database for Chats

Schema of CSV files:

1. chat_create_team_chat.csv

Label	Entity Type	Property	Data Type
User	Node	id	Integer
Team	Node	id	Integer
TeamChatSession	Node	id	Integer
CreatesSession	Edge (User->TeamChatSession)	timeStamp	Float
OwnedBy	Edge (TeamChatSession->Team)	timeStamp	Float

2. chat_item_team_chat.csv

Label	Entity Type	Property	Data Type
User	Node	id	Integer
TeamChatSession	Node	id	Integer
ChatItem	Node	id	Integer
CreateChat	Edge (User->ChatItem)	timeStamp	Float

PartOf	Edge (ChatItem->TeamChatSession)	timeStamp	Float
--------	----------------------------------	-----------	-------

3. chat_join_team_chat.csv

Label	Entity Type	Property	Data Type
User	Node	id	Integer
TeamChatSession	Node	id	Integer
Join	Edge (User->TeamChatSession)	timeStamp	Float

4. chat_leave_team_chat.csv

Label	Entity Type	Property	Data Type
User	Node	id	Integer
TeamChatSession	Node	id	Integer
Leaves	Edge (User->TeamChatSession)	timeStamp	Float

5. chat_mention_team_chat.csv

Label	Entity Type	Property	Data Type
ChatItem	Node	id	Integer
User	Node	id	Integer
Mentioned	Edge (CHatItem->User)	timeStamp	Float

6. chat_respond_team_chat.csv

Label	Entity Type	Property	Data Type
ChatItem	Node	id	Integer
ChatItem	Node	id	Integer
ResponseTo	Edge (ChatItem->ChatItem)	timeStamp	Float

Loading Process:

1. Copied all 6 csv files to /var/lib/neo4j/import folder on my linux system running ubuntu with neo4j installed from the command prompt.
2. Started neo4j service using the following command

```
sudo service neo4j start
```

3. Executed the following command to create constraints

```
cat constraints.cql | cypher-shell -u neo4j -p mypassword --format plain
```

4. I have created a file with load statements for loading all 6 csv files and executed the following command to load the data

```
cat load_csv.cql | cypher-shell -u neo4j -p mypassword --format plain
```

5. I have used localhost://7474/browser for interactive querying

6. The following is the example load command to load the data from chat_create_team_chat.csv

```
LOAD CSV FROM "file:/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])})
```

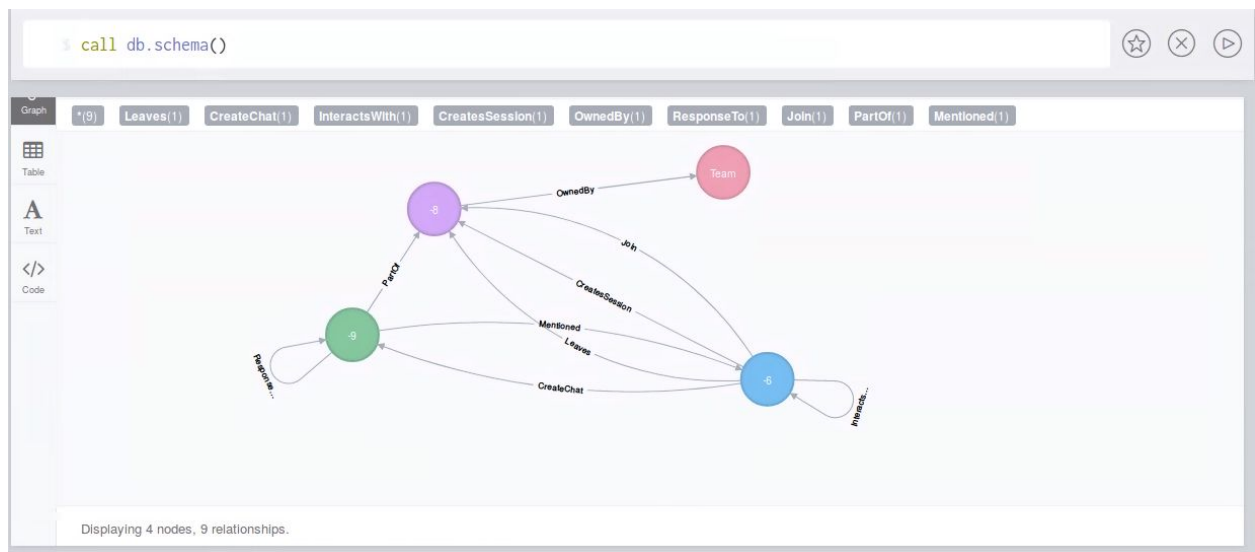
```
MERGE (u)-[:CreatesSession{timeStamp: row[3]}]->(c)
```

```
MERGE (c)-[:OwnedBy{timeStamp: row[3]}]->(t)
```

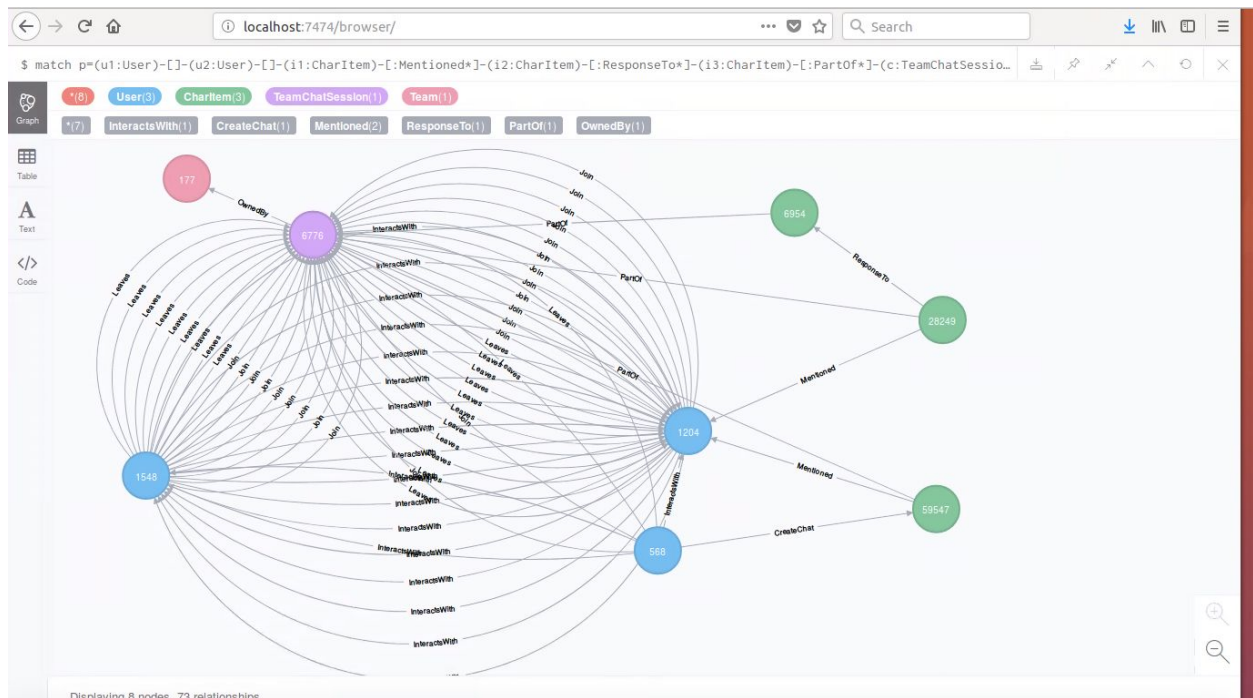
```
;
```

NOTE: Please note that I had used CharItem instead of ChatItem in the database. I had a typo while building the load statements, something I noticed after loading the data. As loading the data takes considerable time, I did not go back and change/rename the label it in the interest of time. It did not impact anything as I had used CharItem everywhere in the database including the constraint creation. Also, I used Join instead of Joins in the load statement for chat_join_team_chat.

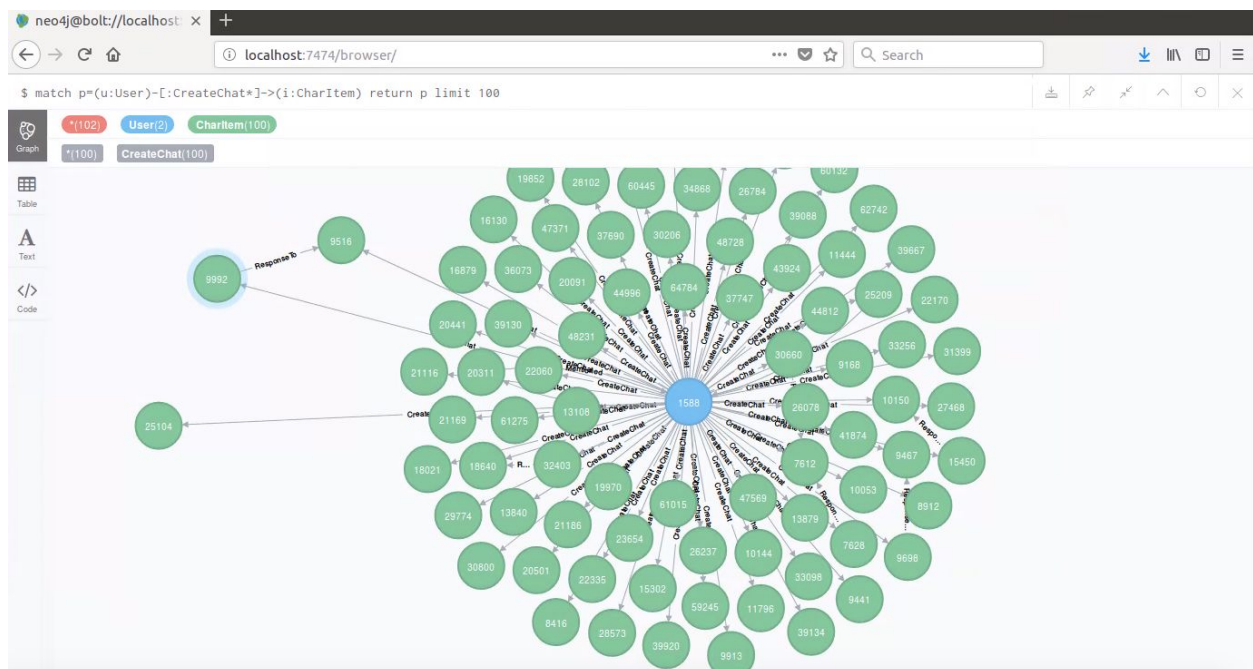
Schema in the database:



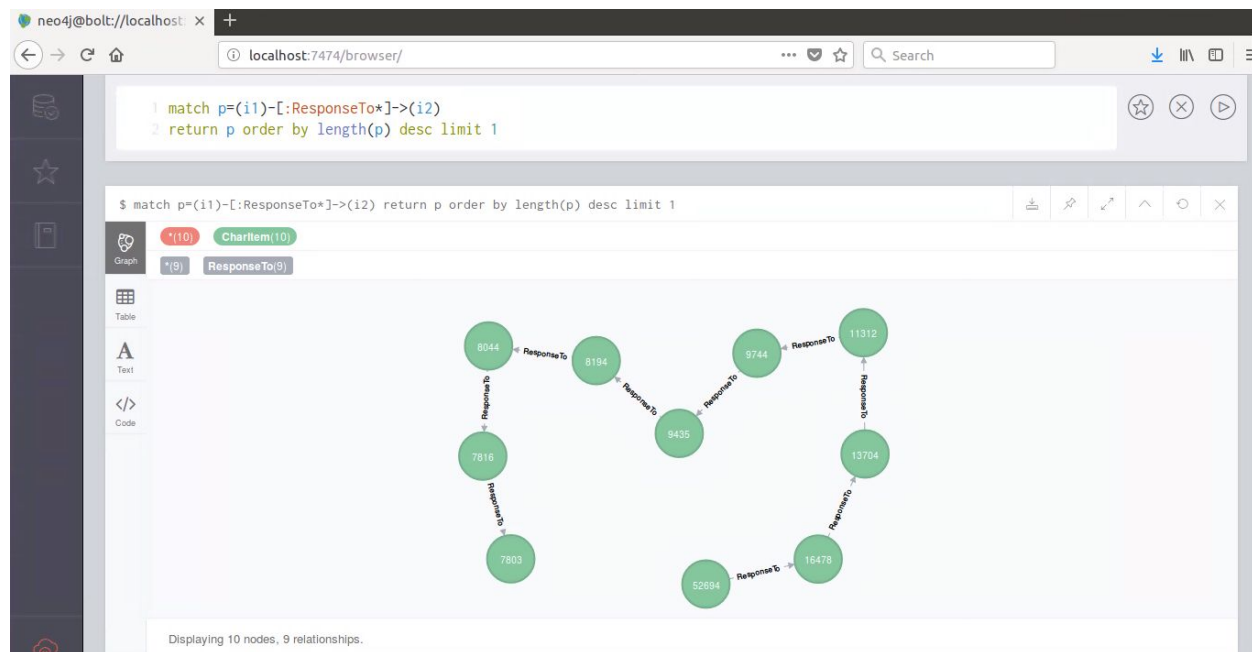
Screenshot 1: With all types of nodes and edges.



Screenshot 2:



Screenshot 3:



Finding the longest conversation chain and its participants

Prompt: 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 path **length of the longest conversation chain is 9** and **there are 5 unique users** participated in the conversation.

I have matched on all CharItems connected connected by ResponseTo and arranged them by their length in descending order to get the first one on the list of paths that would give me longest conversation path. Then I matched on users who created chat items that are part of the longest path obtained in the first part of the query (match+with). Then I returned distinct user ids, distinct user count, and the length of the longest conversation path.

```
match p=(i1:CharItem)-[:ResponseTo*]->(i2:CharItem)
```

```
with p order by length(p) desc limit 1
```

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

```
where i in (nodes(p))
```

```
return collect(distinct u.id) as users, count(distinct u) as user_count, length(p) as path_count
```

"users"	"user_count"	"path_count"
[1978,1514,1192,1153,853]	5	9

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

Prompt: 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.

Top 10 chattiest users:

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

```
return u.id as Users, count(u) as Chat_Count order by Chat_Count desc limit 10
```

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

Chattiest Users

Users	Number of Chats
394	115
2067	111
1087	109

209 also has the same number of chats 109.

Top 10 chattiest teams:

```
match (i:CharItem)-[:PartOf*]->(c:TeamChatSession)-[:OwnedBy*]->(n)
return n.id as Team, count(*) as Chat_Count order by Chat_Count desc limit 10
```

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

Chattiest Teams

Teams	Number of Chats
82	1324
185	1036
112	957

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

User 999 of top 10 chattiest users part of team 52 which is one of the top 10 chattiest teams. The following query returns the team ids for the top 10 chattiest users. Only team id 52 appears in the top 10 chattiest teams above.

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

```
WITH u.id as uid, count(u) as count order by count desc limit 10
```

```
match (u:User)-[:OwnedBy*]-(t:Team)
```

where uid=u.id

return collect(distinct uid) as Users, t.id as Team

"Users"	"Team"
[461]	104
[1087]	77
[668]	89
[554]	181
[999]	52
[2067,209,516,1627]	7
[394]	63

How Active Are Groups of Users?

Prompt: 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.

1. Executed the following query to create InteractsWith relationship when onne user mentioned another user in a chat

```
match (u1:User)-[:CreateChat]->(i:CharItem)-[:Mentioned]->(u2:User)
```

```
create (u1)-[:InteractsWith]->(u2)
```

reated 11084 relationships, completed after 216 ms.

2. Executed the following query to create InteractsWith relationship when one user created a charItem in response to another user's charItem

```
match (u1:User)-[:CreateChat]->(i1:CharItem)-[:ResponseTo]->(i2:CharItem)<-[:CreateChat]-(u2:User)
```

```
create (u1)-[:InteractsWith]->(u2)
```

Created 11073 relationships, completed after 299 ms.

3. Executed the following to delete self loops

Match (u1)-[r:InteractsWith]->(u1) delete r

Deleted 4377 relationships, completed after 153 ms.

4. Executed the following query for each user id from top 10 chattiest users to calculate the clustering coefficients. With 209 user id, the below query returns 0.952 for clustering coeff.

```
match (u:User {id:209})-[:InteractsWith]-(b)
with count(distinct b.id) as neighbor_count, collect(distinct b.id) as neighbors
match (n)-[r:InteractsWith]-(m)
where (n.id in (neighbors))
and (m.id in (neighbors))
with n, m, neighbor_count, count(distinct r) as degree,
case when (n)-->(m) then 1 else 0 end as value
return neighbor_count, count(value) as edge_count, toFloat(count(value))/toFloat(neighbor_count *
(neighbor_count-1)) as CC
```

"neighbor_count"	"edge_count"	"CC"
7	40	0.9523809523809523

5. Executed the query from step 4 for user id 554 to get the following result

"neighbor_count"	"edge_count"	"CC"
7	38	0.9047619047619048

6. Executed the query from step 4 for user 1087 to get the following result.

"neighbor_count"	"edge_count"	"CC"
6	24	0.8

Most Active Users (based on Cluster Coefficients)

User ID	Coefficient
209	0.9523809523809523
554	0.9047619047619048
1087	0.8