**Technical Appendix**

**Catch the Pink Flamingo Analysis**

**Produced by: Julian Hatwell**

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.

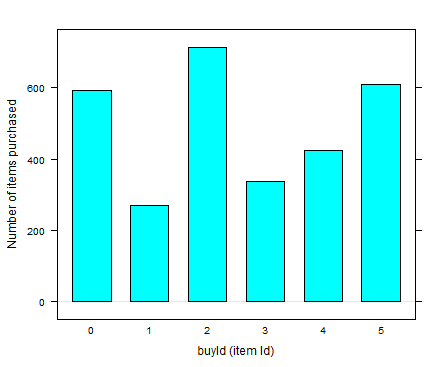
Hello I love you

|  |  |  |
| --- | --- | --- |
| **File Name** | **Description** | **Fields** |
| ad­clicks.csv | This is the data generated when a game user clicks on an ad in the Flamingo app | timestamp: when the click  occurred.  txID: a unique id (within ad­clicks 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 |
| buy­clicks.csv | This data is generated when a player makes an in­app purchase in the Flamingo app | 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 thepurchase    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 |
| users.csv | This is the user demographic data. Each user is uniquely identified in this table | 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 two­letter country  code where the user lives. |
| team.csv | This is the team archive.  It is important to note that the description in the documentation says that “the file contains a line for each team terminated in the game”  It appears as though there are quite a number of teams with a teamEndTime of "9999-12-31 23:59:59"  This not a valid end time.  Further analysis will be done to determine if this value has been used as a default or a proxy for some other information or state. | team: the id of the team  name: the name chosen by the team (presumably by the originator)  teamCreationTime: when the team was created  teamEndTime: when last member left the team.  strength: no information given. It’s value range is 0 < strength < 1  currentLevel: no information given but assumption that it’s the level the team had reached when the last member left unless running teams are included (to be confirmed and verified if this is the case) |
| team­assignments.csv | This data is generated when a user joins a team. From the information given it is also knows that the timestamp when they join is also the timestamp for when they exit their previous 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 |
| user­session.csv | This data is generated when a user starts or completes a game session. Note that this data differs from the description given. It does not contain a start time and end time. It appears to contain two lines for each game session and sessionType can be “start” or “end”.  According to the description, each user on a team will have their respective current sessions ended and new ones started when they level up.  Suggestion to consider wrangling this data to a single line per session with a calculated duration field. | timestamp: when the user session started or ended  userSessionid: a unique id for the session.    userid: the id of the user  teamId: the id of the team  assignmentid: the team assignment id for the user to the team.  sessionType: indicating start or end of session  teamLevel: the level of the team during this session.  platformType: the type of platform of the user during this session. |
| game­clicks.csv | This data is generated whenever a user clicks within 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 |
| level­events.csv | This data is generated whenever a team starts or finishes a level within the game | 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 |
| combined-data.csv | This data combines data from 3 of the log files: user-session.csv, buy-clicks.csv, and game-clicks.csv. | See descriptions above for the following fields:  userId  userSessionId  teamLevel  platformType  in addition are found the following aggregate fields  count\_gameclicks: aggregate number of clicks for user session  count\_hits: aggregate number of hits for user session  count\_buyId: aggregate number of buys for user session  avg\_price: average purchase price for user session  There are NULL values. This results from records not actually having values for those particular variables because of how the aggregation was done. |
| chat\_create\_team\_chat.csv | This data is generated when a player creates a new chat with their team | userid, teamid, TeamChatSessionID, timestamp |
| chat\_item\_team\_chat.csv | Represents nodes labeled ChatItems.  Also create an edge labeled "PartOf" from the ChatItem node to the TeamChatSession node. This edge should also have a timeStamp property using the value from Column 3. | Column 0 is User id  Column 1 is the TeamChatSession id  Column 2 is the ChatItem id (i.e., the id property of the ChatItem node)  Column 3 is the timestamp for an edge labeled "CreateChat". |
| chat\_join\_team\_chat.csv | Creates an edge labeled "Joins" from User to TeamChatSession. | User id  TeamChatSession id  timestamp of the Joins edge. |
| chat\_leave\_team\_chat.csv | Creates an edge labeled "Leaves" from User to TeamChatSession. | User id  TeamChatSession id  timestamp of the Leaves edge. |
| chat\_mention\_team\_chat.csv | Creates an edge labeled "Mentioned". | Column 0 is the id of the ChatItem  Column 1 is the id of the User  Column 2 is the timeStamp of the edge going from the chatItem to the User. |
| chat\_respond\_team\_chat.csv | A line is added to this file when a player responds to a chat post. | chatid1  chatid2  timestamp |

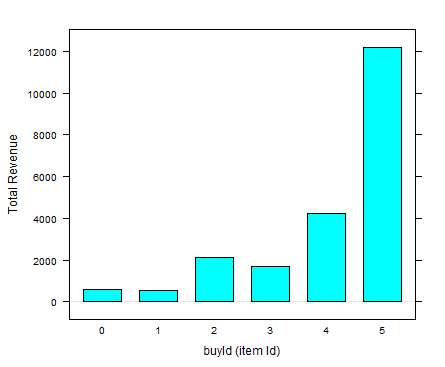
Aggregation

|  |  |
| --- | --- |
| Amount spent buying items | 21407 |
| # Unique items available to be purchased | 6 |

A histogram showing how many times each item is purchased:

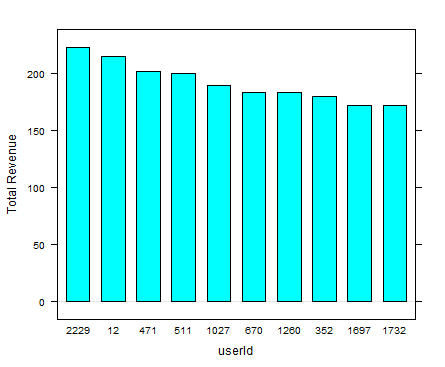


A histogram showing how much money was made from each item:



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 | 11.6 |
| 2 | 12 | iPhone | 13 |
| 3 | 471 | iPhone | 14.5 |

**Data Preparation**

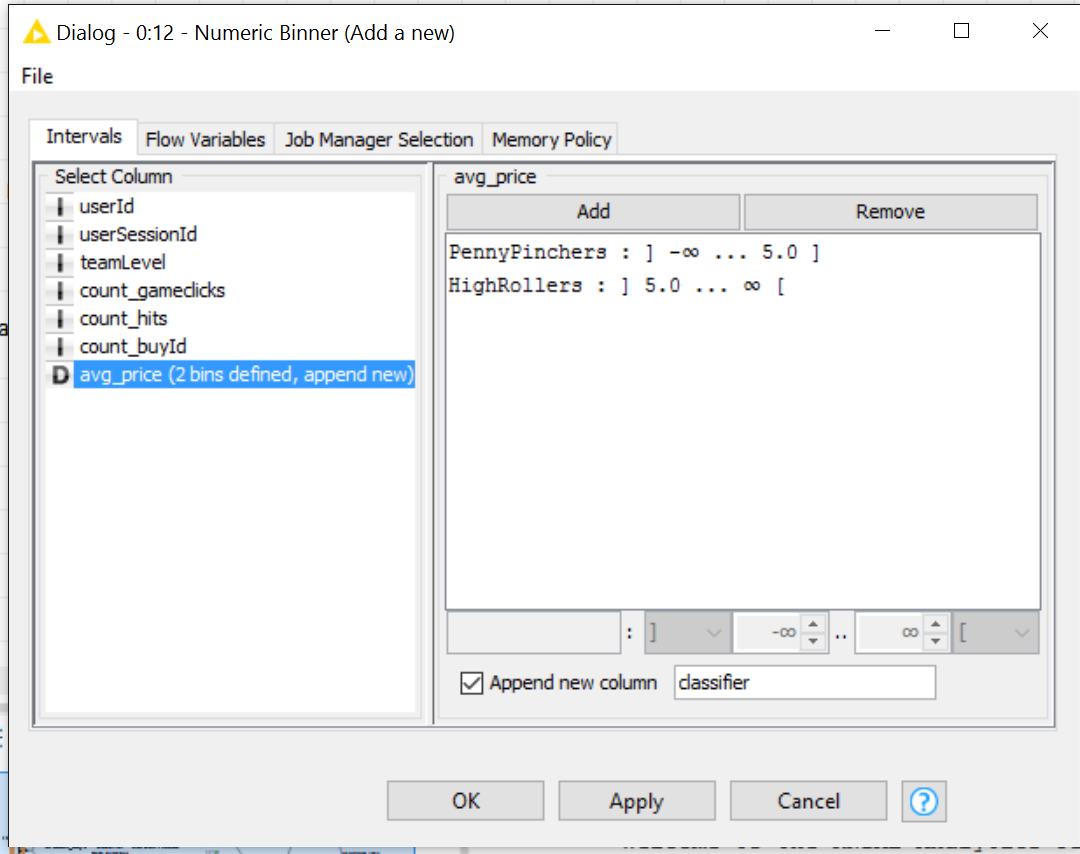
Analysis of combined\_data.csv

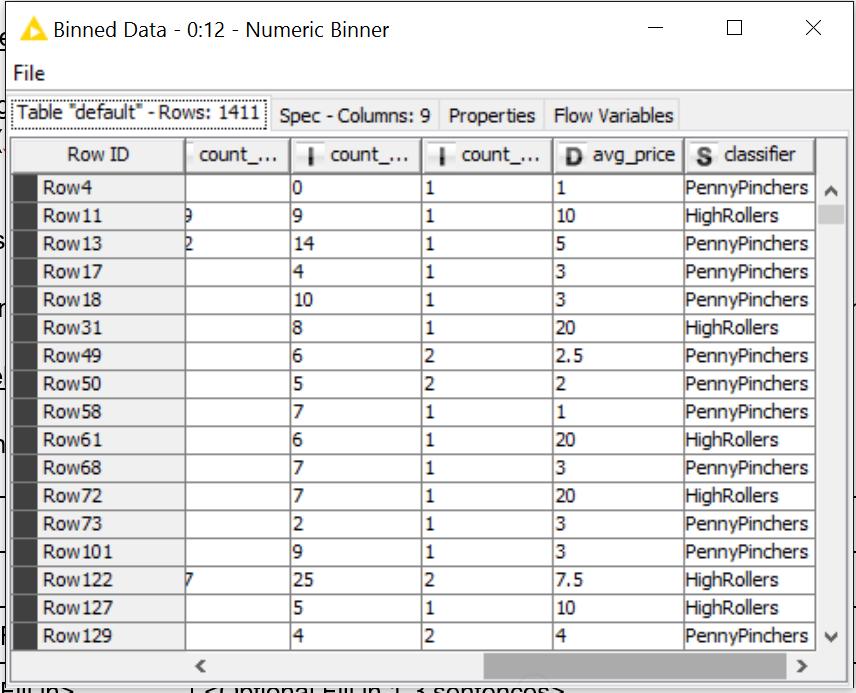
Sample Selection

|  |  |
| --- | --- |
| **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). This was done using the binner Node. A screenshot of the Node and resulting attribute follows:





**Description:** A “numeric binner” node was used which is set to have two named bins. Bin1 is set to range <= 5.0 and renamed “PennyPinchers.” Bin2 is set to range < 5.0 and is renamed “HighRollers.”

The creation of this new categorical attribute was necessary because a decision tree algorithm is a classification learning model. This means it requires discrete categories of data, not numerical scalar data.

Attribute Selection

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

|  |  |
| --- | --- |
| **Attribute** | **Rationale for Filtering** |
| UserId | UserId will uniquely classify each row and cause over-fitting |
| UserSessionId | This will also uniquely classify each row and cause over-fitting |
| avgPrice | This numerical value has already been converted into the categorical classifier and so adds no new information. It is redundant |

**Data Partitioning and Modeling**

The data was partitioned into train and test datasets.

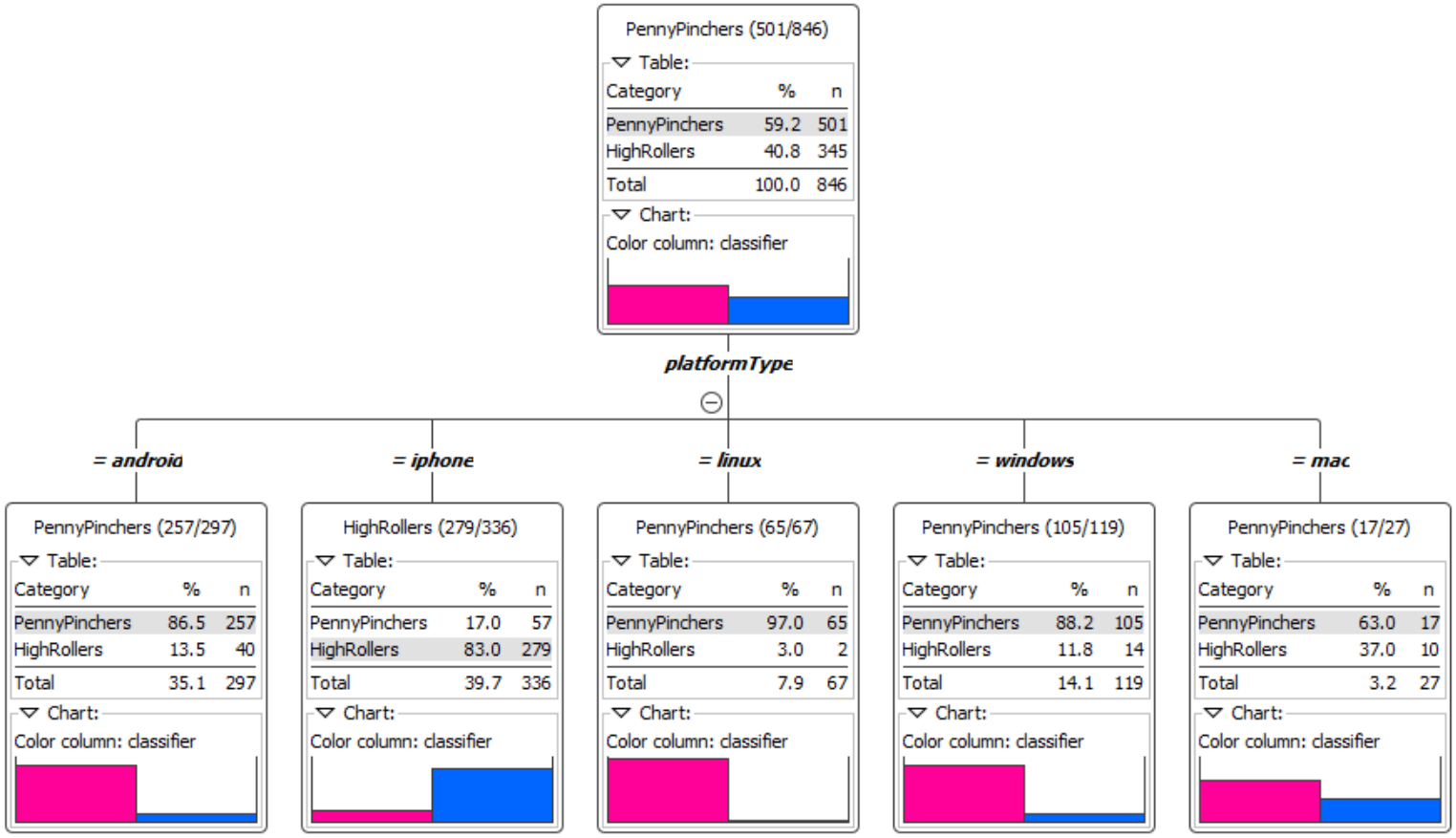
The training data set was used to create the decision tree model.

The trained model was then applied to the test dataset.

This is important because the accuracy of the model on the training data is not a good measure of the model accuracy. The reason for this is that a model can easily over-fit the training data and score as much as 100% accuracy yet it will be poor at predicting outcomes on previously unseen data. A more general model, which is less well fit to the training data will probably make better predictions on new data. This is known as the bias-variance trade off.

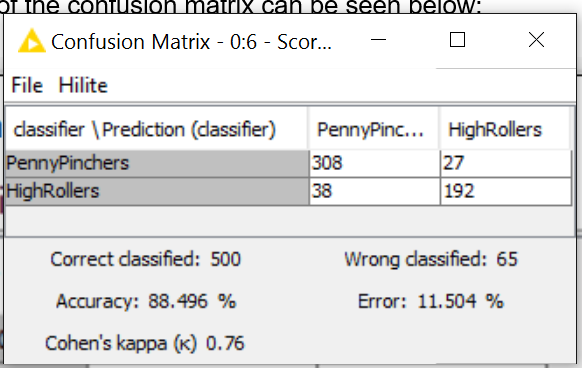
When partitioning the data using sampling, it is important to set the random seed because this will ensure the same partitions are used each time the model is run, allowing the results to be completely reproducible.

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 88.5%

In the top-left the prediction was PennyPincher and the true value was PennyPincher 308 times.

In the top-right the prediction was HighRoller and the true value was PennyPincher 27 times.

In the bottom-left the prediction was PennyPincher and the true value was HighRoller 38 times.

In the bottom-right the prediction was HighRoller and the true value was HighRoller 192 times.

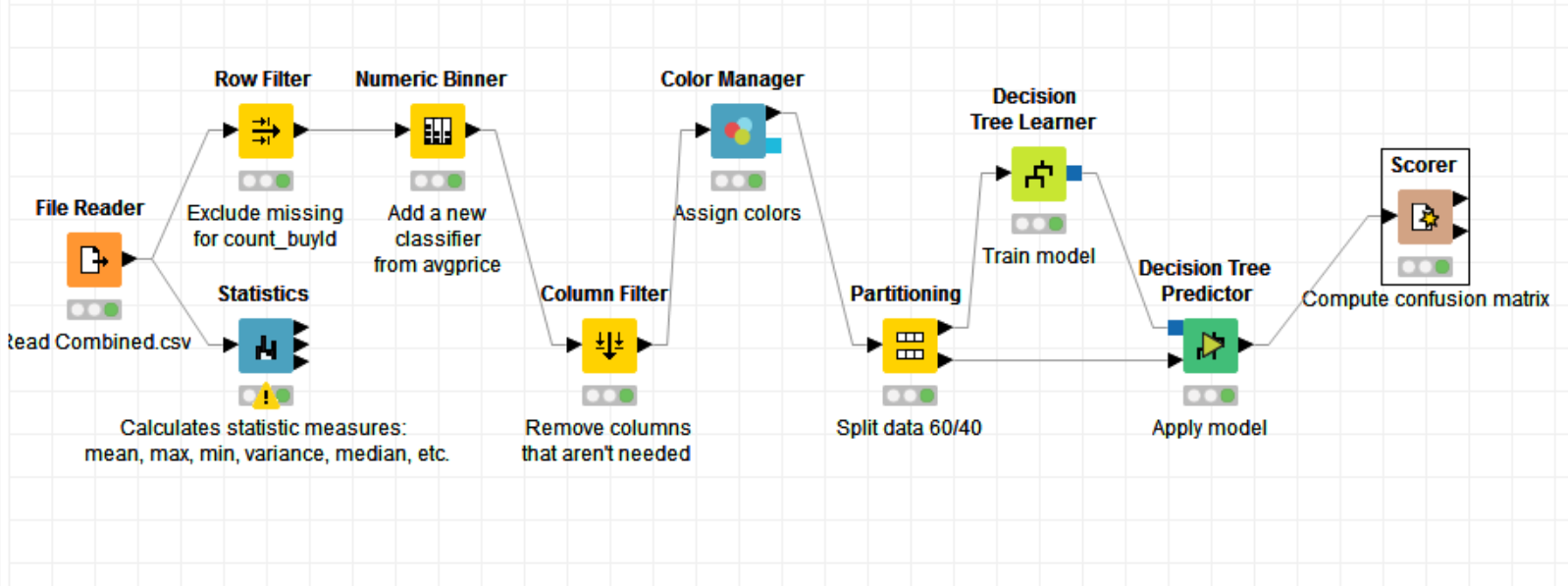
Let HighRoller be a positive outcome and PennyPincher be a negative outcome.

The True Positive Rate = Sensitivity = TP / ( TP + FN ) = 192 / ( 192 + 38 ) = 83.5%

The True Negative Rate = Specificity = TN / ( TN + FP ) = 308 / ( 308 + 27 ) = 91.9%

**Analysis Conclusions**

The final KNIME workflow is shown below:



What makes a HighRoller vs. a PennyPincher?

Based on the decision tree discovered by the model, any iPhone user would be very likely to be a HighRoller. Game players on all other platforms are most likely to be PennyPinchers.

|  |
| --- |
| **Specific Recommendations to Increase Revenue** |
| 1. Increase marketing spend on acquiring new user on the iPhone platform. Decrease marketing spend on other platforms as these users are not making large purchases. |
| 2. Target non-iPhone users with special offers and other incentives to make larger purchases. |

**Attribute Selection**

|  |  |
| --- | --- |
| **Attribute** | **Rationale for Selection** |
| HitRatio | This will be calculated PER USER from the game-clicks data isHit column, as the sum of the column (hit = 1, miss = zero), divided by length of the column for each userId.  The rationale for this is to determine who are the better players in the game. We want find out how player ability relates to other behaviours. |
| TotalTimeSpent | This will be calculated PER USER from the user-session data subtracting the difference between the start time stamp and end time stamp for each userSessionId and summing the resulting duration of each session for each userId  The rationale for this is to determine who are the people who spend the longest time in the game and how this relates to other behaviours |
| TotalMoneySpent | This will be calculated PER USER from the buy clicks data by summing the price for each userId  The rationale for this is to determine who are the people who spends the most money on the game and how this relates to other behaviours |
| numTeams | This will be calculated PER USER as the number of teams in which they have participated.  The rationale for this is to determine who are the people who tend to stick to one team or switch from team to team and see how this behaviour relates to other behaviours |

**Training Data Set Creation**

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

totalMoneySpent totalTimeSpent hitRatio

1: 0 10530.00 0.1055351

2: 21 26256.92 0.1340782

3: 0 19170.00 0.0952381

4: 0 4470.00 0.1059603

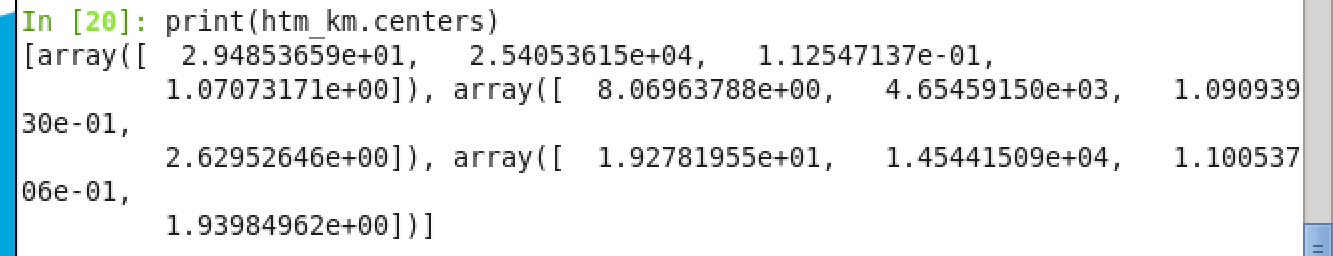
5: 53 4800.00 0.1000000

Dimensions of the training data set (rows x columns) : 1093 x 3

# of clusters created: 3

**Cluster Centers**

Screenshot from PySpark:



|  |  |
| --- | --- |
| Cluster # | Cluster Center |
| 1 | TotalMoneySpent: 8.07, TotalTimeSpent: 4654, HitRatio: 0.1091 numTeams: 2.63 |
| 2 | TotalMoneySpent: 19.28, TotalTimeSpent: 14544, HitRatio: 0.1005, numTeams: 1.94 |
| 3 | TotalMoneySpent: 29.5, TotalTimeSpent: 25405, HitRatio: 0.1125, numTeams: 1.09 |

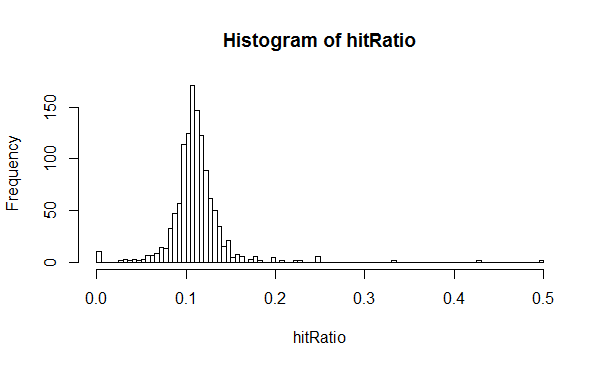
These clusters can be differentiated from each other as follows:

Cluster 1 is different from the others in that they have spend the least time in the game. They spend the least money and have the lowest hit ratio. This cluster shall be name “JustPassingTime.”

Cluster 2 is different from the others in that it spends 3 times as long in the game, 2.5 times as much money on in app purchases and yet their scores are barely any better than the JustPassingTime cluster. They spend less time and money and their scores are very significantly poorer than cluster 3 (“SeriousGamers”). This cluster shall be named “Wannabes.”

Cluster 3 is different from the others in that the spend the most amount in the game at more than 5 times as much as the “JustPassingTime” cluster. They spend about 3.5 times as much money as “JustPassingTime” and 1.5 times as much money as the “Wannabes.” Their scores are much higher than the other clusters at around 1.5 percentage points better hit rate. Cluster 3 shall be called “SeriousGamers”

*Note that the hit ratio for all players is centred around 0.1 (see screenshot below) and we’re taking a shift of plus or minus 0.005 as significant between players. 0.01 difference is very significant between players.*



**Recommended Actions**

|  |  |
| --- | --- |
| **Action Recommended** | **Rationale for the action** |
| Increase time spent in the game – use teasers to keep people playing longer. Use cliff hangers and regular notifications to remind them to come back to the game. | Money spent in game is clearly linked to time spent in the game as we can see from the pattern in the 3 clusters. |
| Create more options for in app purchases that help users increase their scores. Make sure the messages are clear that these purchases help boost scores. | Money spent on the game is linked to higher scores and people with higher scores spend more time and money in the game. It stands to reason that a good incentive to keep people playing is to offer them little boosts to their scores. |
| Discover ways to convert “JustPassingTime” users into “Wannabes” | There is not much difference in the score between these two groups, but “Wannabes” are playing more and spending more. Eglence needs to come up with a new approach to get more of the casual gamers hooked on the game. |

**Graph Analytics**

**Modeling Chat Data using a Graph Data Model**

An initial graph model was loaded up from the csv files given plus some additional scripts which were run to enhance the model.

The initial graph model consists of the following Nodes. Each type of node has a unique {id} attribute:

* User – each unique user
* Team – each unique team
* TeamChatSession – A unique container for a Team’s ChatItems
* ChatItem – each unique Chat Item added by a User

In the initial model, these Nodes have relationships that are described using the notation as follows

(Node)-[Edge]->(Node) – this indicates a directional relationship

toward the (From)-[]->(To)

(Node)-[Edge]-(Node) – This indicates a relationship where direction is not considered.

Each Edge has a {timestamp} attribute indicating when it was created. The following patterns are modelled:

* (User)-[CreatesSession]->(TeamChatSession)-[OwnedBy]->(Team)
* (User)-[Joins]->(TeamChatSession)
* (User)-[Leaves]->(TeamChatSession)
* (User)-[CreateChat]->(ChatItem)-[PartOf]->(TeamChatSession)
* (ChatItem)-[Mentioned]->(User)
* (ChatItem)-[ResponseTo]->(ChatItem) – The two ChatItems never have the same {id}

**Creation of the Graph Database for Chats**

Here are the steps taken to create the graph:

1. Importing the 6 csv files which have the following schema. *Please note that this is slightly different to the documentation given, which has been discussed as having material errors in the Course Forum:*
   * File: chat\_create\_team\_chat.csv
     + userid, teamid, timestamp
   * File: chat\_item\_team\_chat.csv
     + userid, TeamChatSessionID, teamid, timestamp
   * chat\_join\_team\_chat.csv
     + userid, TeamChatSessionID, timestamp
   * chat\_leave\_team\_chat.csv
     + userid, TeamChatSessionID, timestamp
   * chat\_mention\_team\_chat.csv
     + ChatItem, userid, timestamp
   * chat\_respond\_team\_chat.csv
     + 1, ChatItem 2
2. This was done using the load command, to model all the relationships based on the patterns described in the introducation. One example is given below:

LOAD CSV FROM "file:///chat-data/chat\_create\_team\_chat.csv" AS row

MERGE (u:User {id: toInt(row[0])})

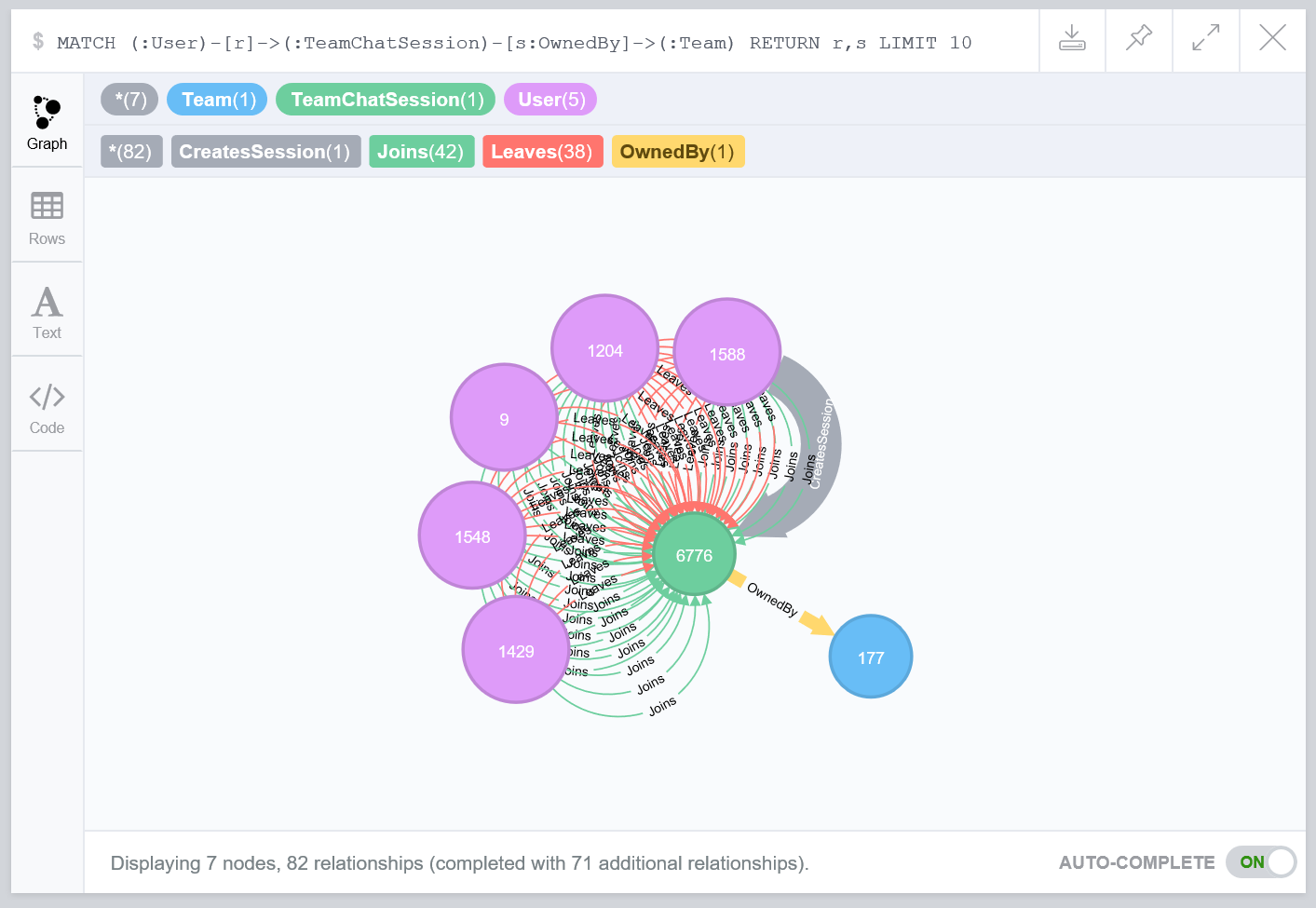
MERGE (t:Team {id: toInt(row[1])})

MERGE (c:TeamChatSession {id: toInt(row[2])})

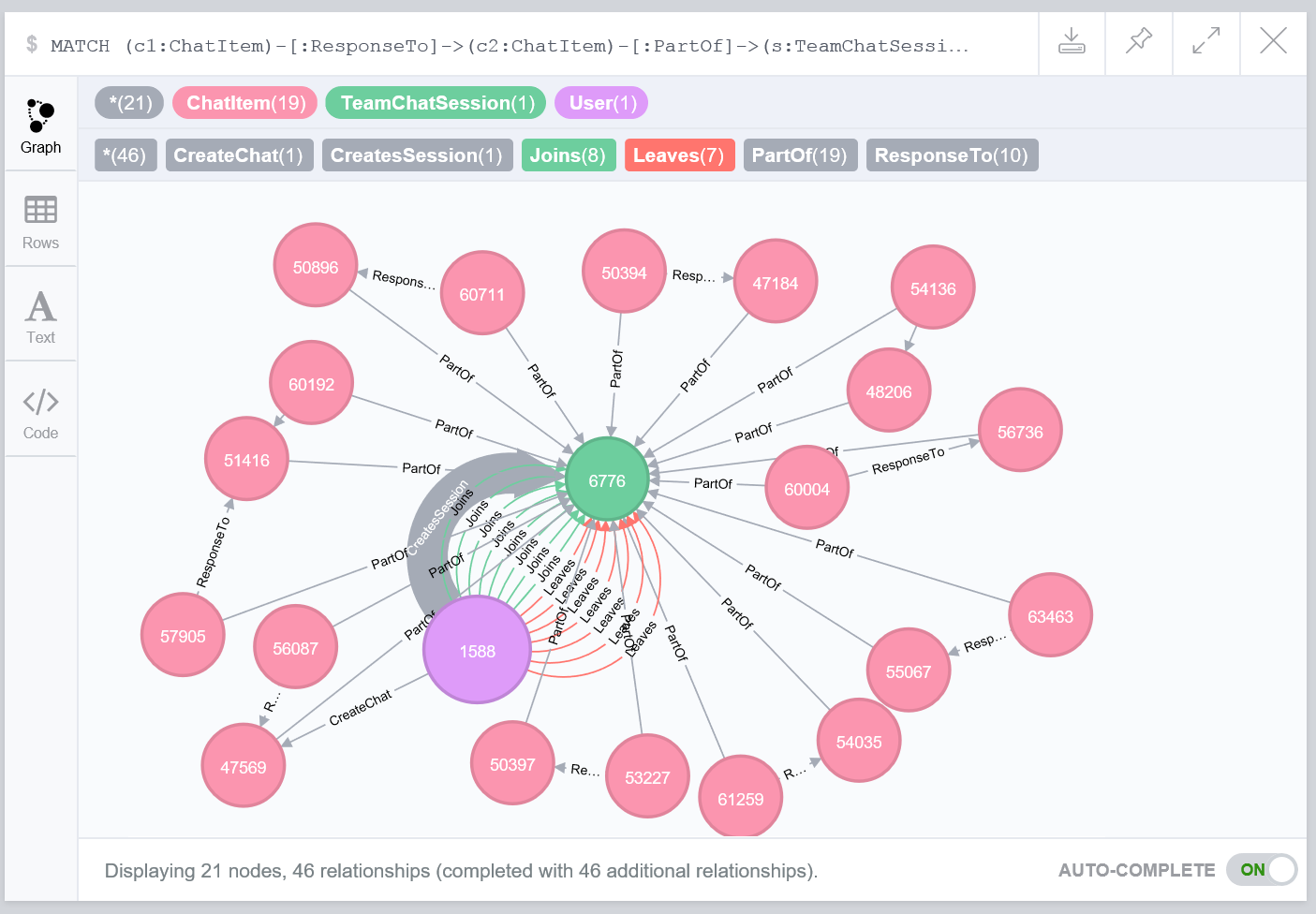
MERGE (u)-[:CreatesSession{timeStamp: row[3]}]->(c)

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

1. A couple of screen shots of the graph are presented here:
   * Figure 1. Showing the relationship between users and the TeamChatSession. One user CreatesSession which is OwnedBy Team. Other Users Join and Leave the TeamChatSession many times.



* + Figure 2. Showing the relationship between ChatItems being PartOf the TeamChatSession and possibly responses to other ChatItems. The User who created the session is also shown.

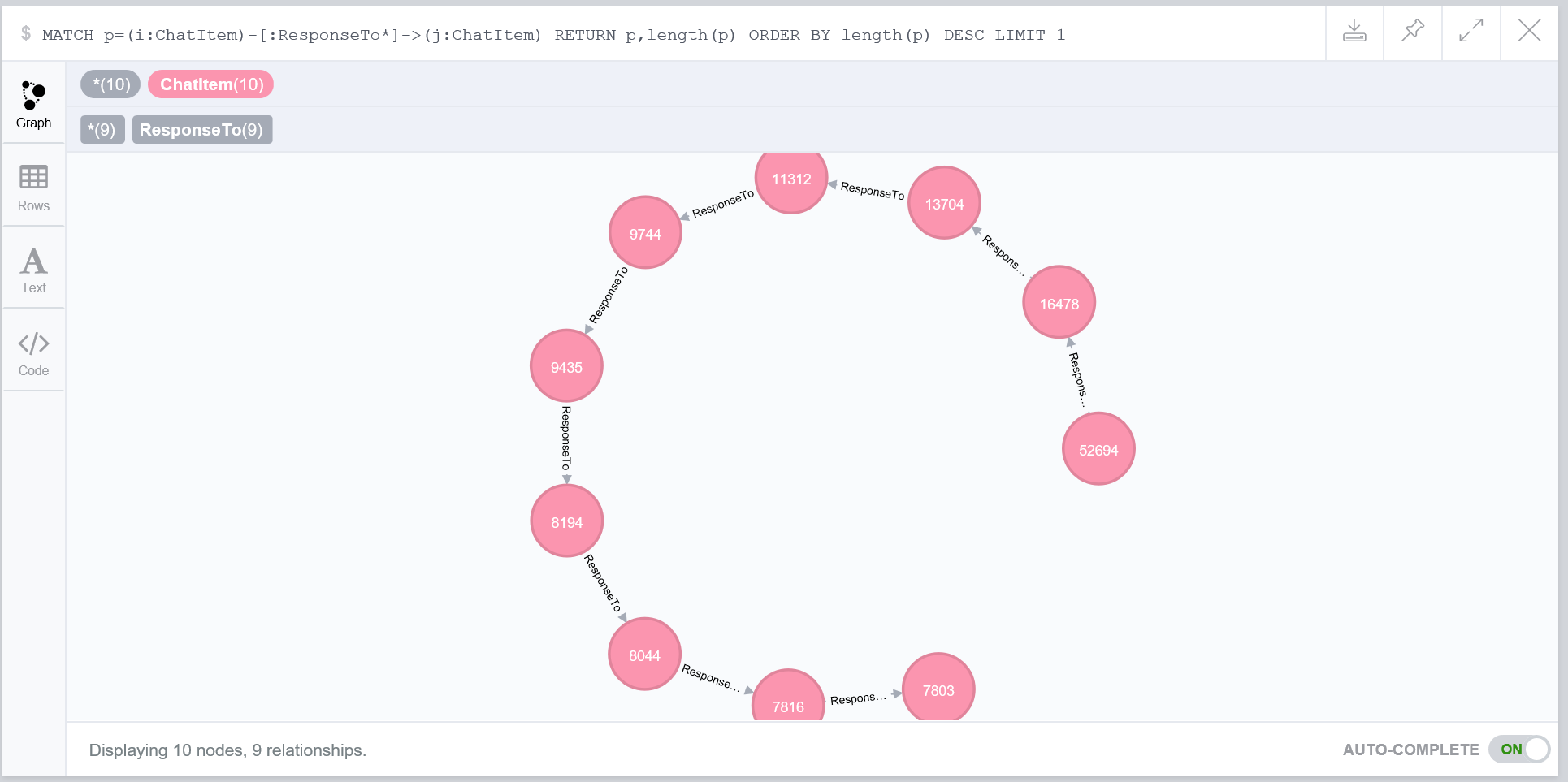


**Finding the longest conversation chain and its participants**

The length of the longest chat chain is 10 ChatItem Nodes or 9 ResponseTo Edges. This can be shown by executing the following query:

MATCH p=(i:ChatItem)-[:ResponseTo\*]->(j:ChatItem) RETURN p,length(p) ORDER BY length(p) DESC LIMIT 1

This screenshot shows the result:



The number of unique users involved in this chain can be determined by the following query:

MATCH p=(i:ChatItem)-[:ResponseTo\*]->(j:ChatItem)

WITH max(length(p)) AS maxChain

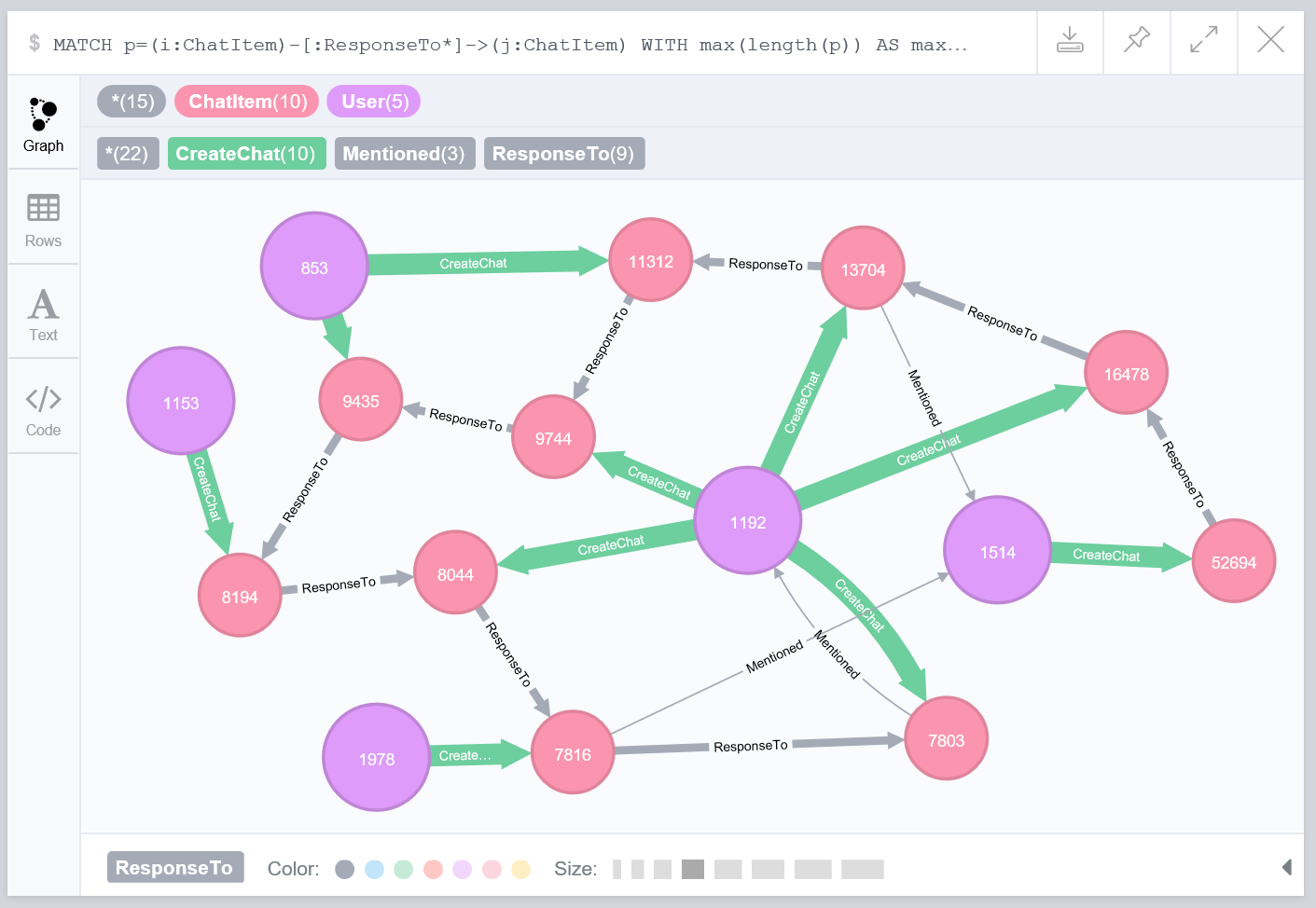
MATCH p=(i:ChatItem)-[:ResponseTo\*]->(j:ChatItem) WHERE length(p) = maxChain

WITH extract(n in nodes(p)|n.id) AS ChatItems

MATCH (u:User)-[c:CreateChat]->(k:ChatItem) WHERE k.id IN ChatItems

RETURN distinct(u), count(distinct(u)),c

The following screenshot shows the result:



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

To find the chattiest users, the following query was executed:

MATCH (u:User)-[:CreateChat]->(i:ChatItem)

RETURN u, count(u) ORDER BY count(u) DESC LIMIT 10

To find the chattiest team, the following query was run:

MATCH (i:ChatItem)-[:PartOf]->(c:TeamChatSession)-[:OwnedBy]->(t:Team)

RETURN t, count(t) ORDER BY count(t) DESC LIMIT 10

Only the top 3 for each are included in the tables below.

**Chattiest Users**

|  |  |
| --- | --- |
| **Users** | **Number of Chats** |
| 395 | 115 |
| 2067 | 111 |
| 1087 | 109 |

**Chattiest Teams**

|  |  |
| --- | --- |
| **Teams** | **Number of Chats** |
| 82 | 1324 |
| 185 | 1036 |
| 112 | 957 |

To identify whether any of the chattiest users are in the chattiest teams, the following query was run:

MATCH (i:ChatItem)-[:PartOf]->(c:TeamChatSession)-[:OwnedBy]->(t:Team)

WITH t AS Teams, count(t) AS numTeamChats WHERE numTeamChats > 735 // smaller than the 10th highest team

MATCH (v:User)-[:CreateChat]->(ChatItem)-[:PartOf]->(:TeamChatSession)-[:OwnedBy]->(w:Team) WHERE w.id IN Teams.id

WITH collect(distinct v.id) AS Users

MATCH (u:User)-[:CreateChat]->(i:ChatItem) WHERE u.id IN Users

WITH u AS FinalUsers, count(u) AS numUserChats WHERE numUserChats > 103 // smaller than the 10th highest user

WITH collect(distinct FinalUsers.id) AS FinalUsersIds

MATCH (v:User)-[:CreateChat]->(ChatItem)-[:PartOf]->(:TeamChatSession)-[:OwnedBy]->(w:Team) WHERE v.id IN FinalUsersIds

RETURN v, w

Only user 999 in team 52 is both a top 10 chattiest user and a user in a top 10 chattiest team.

**How Active Are Groups of Users?**

The initial model was enhanced by adding a new Edge called “InteractsWith” when two non-identical users were connected by either of the following patterns:

* (User)-[CreateChat]->(ChatItem)-[Mentioned]->(User)
* (User)-[CreateChat]->(ChatItem)-[ResponseTo]->(ChatItem)<-[CreateChat]-(User)

The new relationship is represented as:

* (User)-[InteractsWith]->(User)

These steps created self-loops were removed.

For each of the top 10 chattiest users, the list and count (n) of neighbours was retrieved.

The edges between these list of neighbours were discovered and aggregated to either 1 (one or more edge) or 0 (zero edges between a pair of neighbours). This new value is summed to determine the number of connected neighbours (k)

The sum of the number of total possible connections among the neighbours are calculated as n\*(n-1). The clustering coefficient is calculated as k/(n\*(n-1))

**Most Active Users (based on Cluster Coefficients)**

|  |  |
| --- | --- |
| **User ID** | **Coefficient** |
| 209 | .95 |
| 554 | .90 |
| 1078 | .8 |