Technical Appendix
Catch the Pink Flamingo Analysis

Produced by: Markus Jung

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
ad-clicks.csv	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	A line is added to this file when a player makes an in-app purchase in the Flamingo app.	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
users.csv	This file contains a line for each user playing the game.	timestamp: when user first played the game. id: 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.
team.csv	This file contains a line for each team terminated 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
team-assignments.csv	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.	time: 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

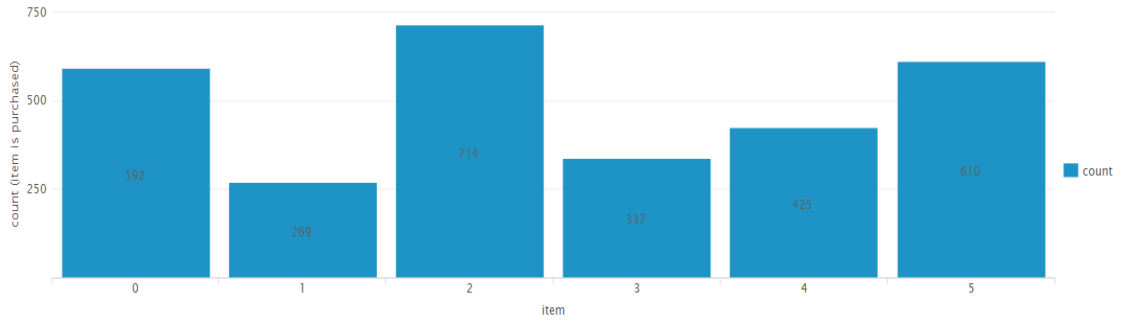
level-events.csv	A line is added to this file each time a team starts or finishes a level in the game	time: when the event occurred. eventid: a unique id for the event teamid: the id of the team level: the level started or completed eventType: the type of event, either start or end
user-session.csv	Each 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.	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.
game-clicks.csv	A line is added to this file each time a user performs a click in the game.	time: 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

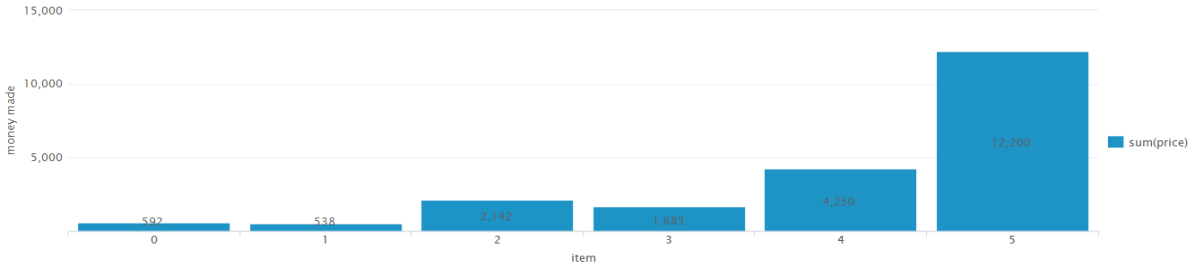
Aggregation

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:

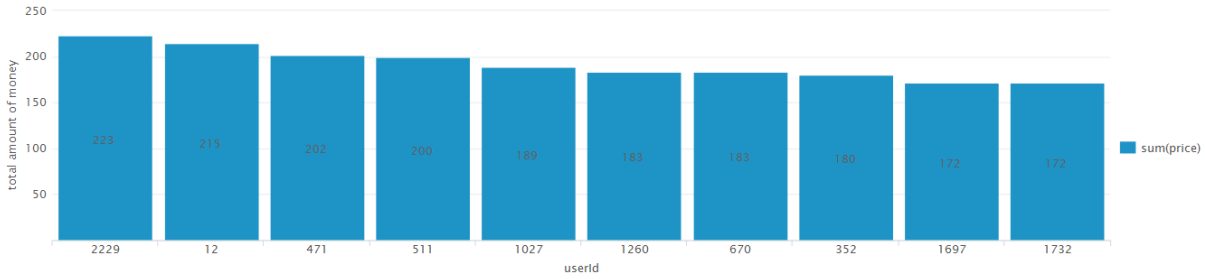


A histogram showing how much money was made from each item:



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	13.1
3	471	iphone	14.5

Data Classification Analysis

Data Preparation

Analysis of combined_data.csv

Sample Selection

Item	Amount
# of Samples	4,619
# of Samples with Purchases	1,411

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:

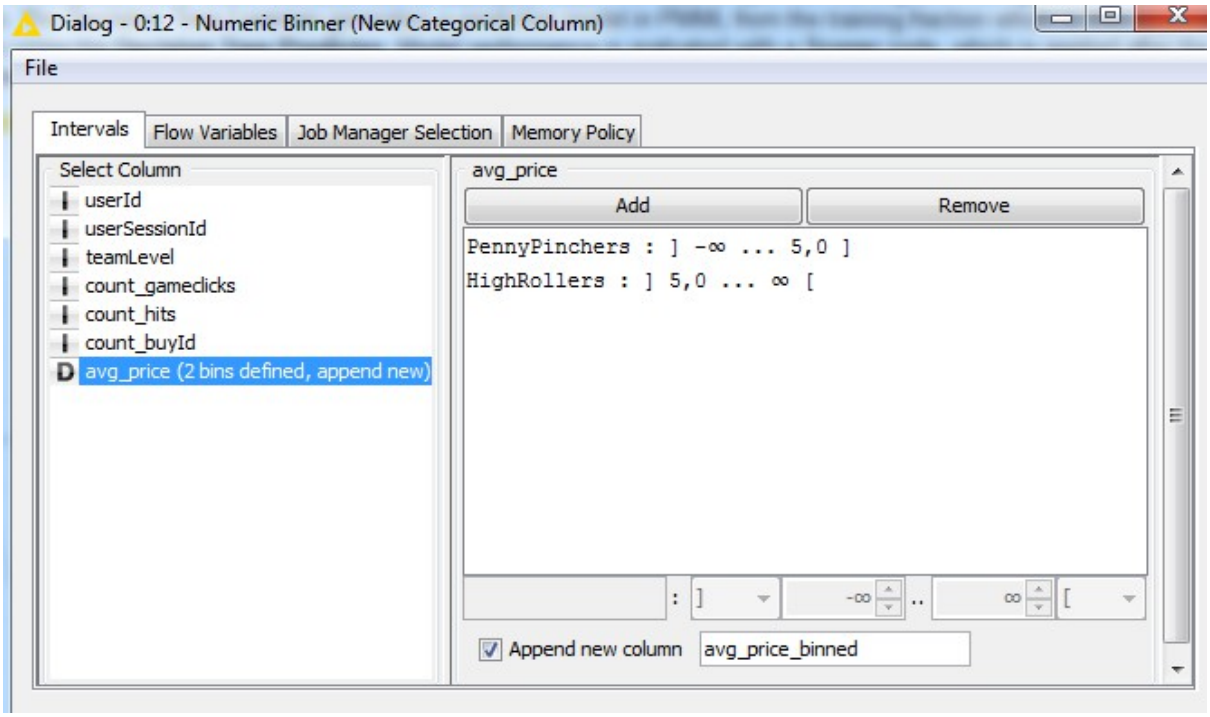


Figure 1: Creation Catigorical Column

The name of the new categorical attribute is “avg_price_binned”.

PennyPinchers (buyers of items that cost $5.00 or less) are defined as byers with the attribute “avg_price” not higher than 5.
HighRollers (buyers of items that cost more than $5.00) are defined as byers with the attribute “avg_price” higher than 5.

The creation of this new categorical attribute was necessary because we want to predict if a user is a penny pincher or a high roller. This is not possible with a nondiscret variable.

Attribute Selection

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

Attribute	**Rationale for Filtering**
userId	The id of the user does not contain any relvant information for the prediction of “HighRollers” and “PennyPinchers”. It is a unique number for the user.
userSessionId	The id for the session does not contain any relvant information for the prediction of “HighRollers” and “PennyPinchers”. It is a unique number for the session.
avg_price	The new categorical attribute “avg_price_binned” was created based on “avg_price”. We want to build the prediction for “avg_price_binned”. Therefore “avg_price” is not useable.

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 we need to know the accuracy of the model.

When partitioning the data using sampling, it is important to set the random seed because this ensures getting the same partitions every time the partions are created. So we get reproducable results.

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

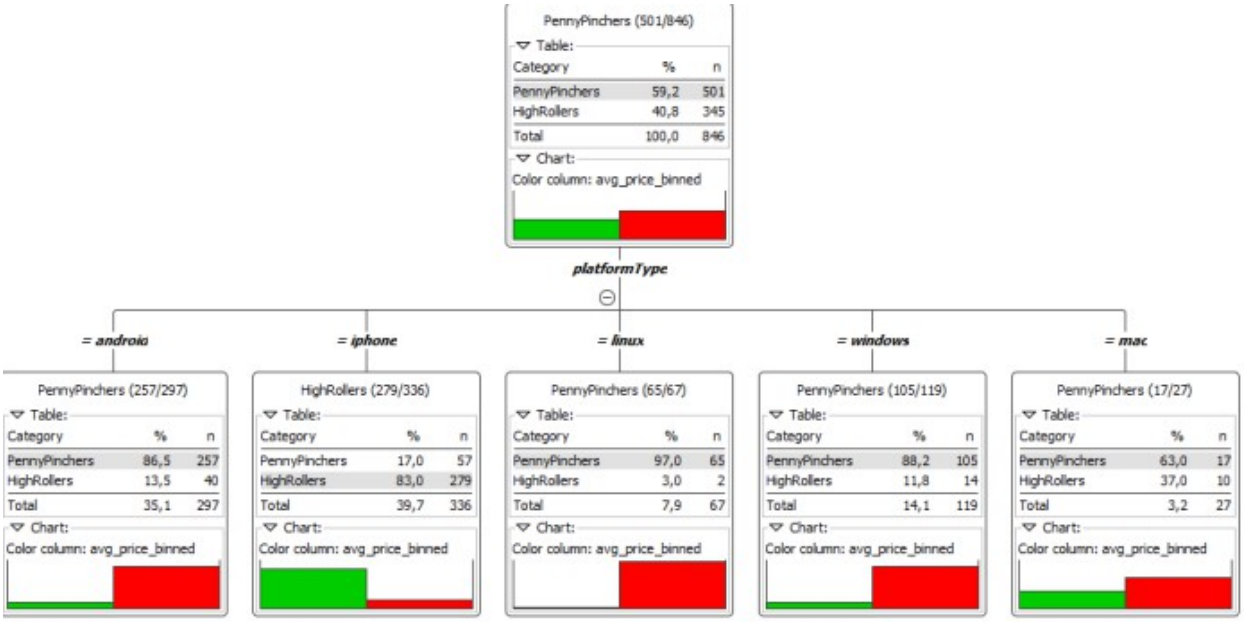


Figure 2: Decision Tree

Evaluation

A screenshot of the confusion matrix can be seen below:

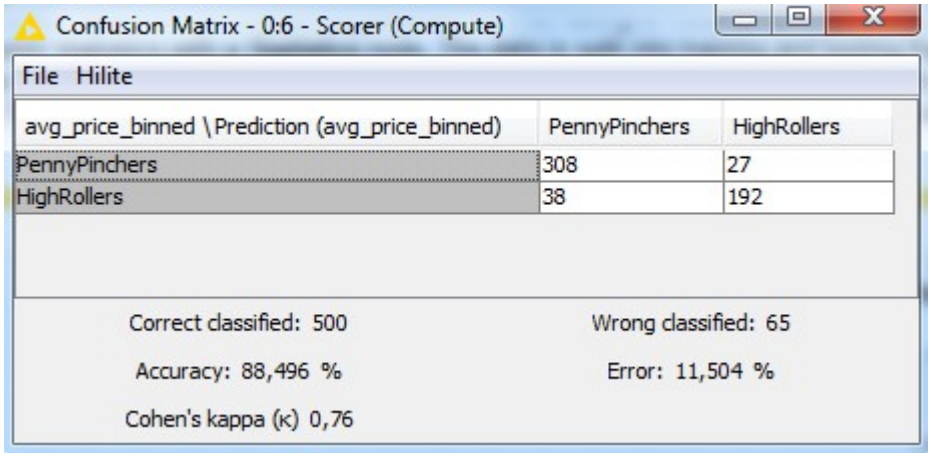


Figure 3: Confusion Matrix

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

When we consider HighRollers as “Positives” we see from the matrix the following:

- True Positives: 192
- False Positives: 38
- True Negatives: 308

- False Negatives: 27

Correctly predicted with this decision tree (correct classified) are the “True Positives” and the “True Negatives”, in sum 500.

Not Correctly predicted with this decision tree (wrong classified) are the “False Positives” and the “False Negatives”, in sum 65.

Analysis Conclusions

The final KNIME workflow is shown below:

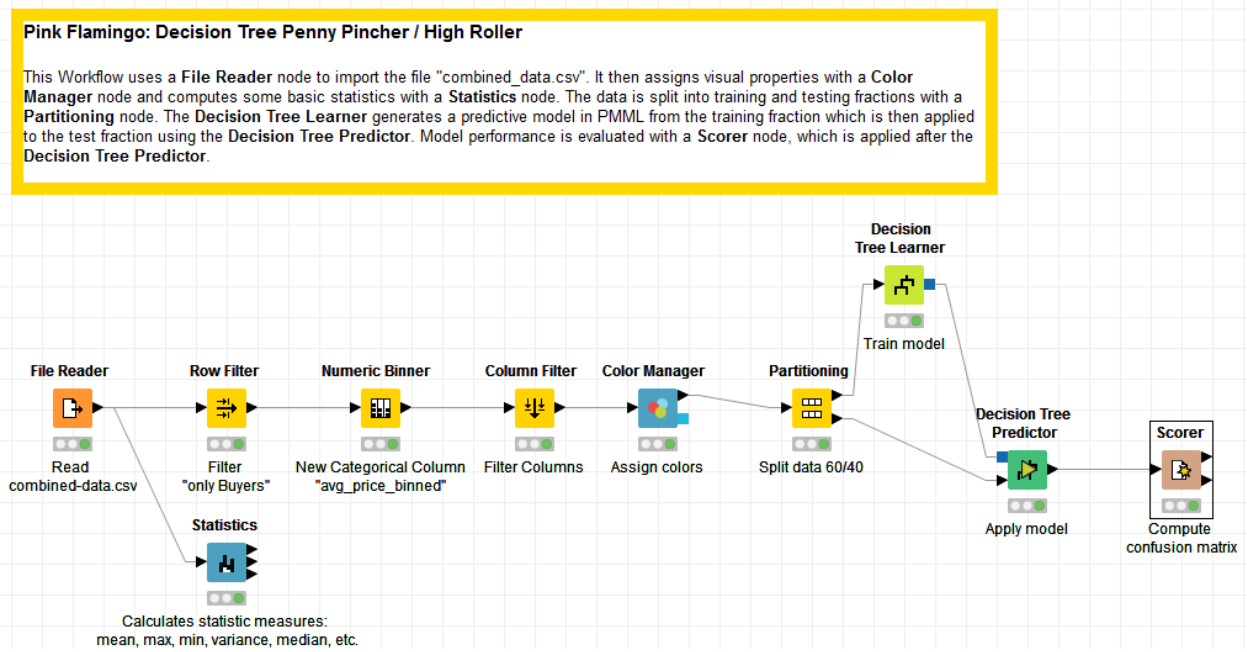


Figure 4: KNIME Workflow

What makes a HighRoller vs. a PennyPincher?

- About 40 % of the users who pruchase are HighRoller
- About 83 % of the users with iphone as type of platform are HighRoller
- Only 3 % of the users with linux as type of platform are Highroller
- Relevant for the prediction if a user is a HighRoller or a PennyPincher based on the sample data is only the platformtype of the user during a session.

Specific Recommendations to Increase Revenue
1. Better offers for users with platformtype not equal iphone.
2. Do some markt analysis what offers are interesting for users with platformtype not equal iphone, specially for “linux” user.

Clustering Analysis

Attribute Selection

Attribute	Rationale for Selection
„price“ from buy-clicks.csv	Attribute that captures the purchasing behaviour of users.
„timestamp“ from user-session.csv	Hour of a day when user's „enter“ the website (enter the „shop“). Therefore only rows with session = 'start' are interesting. From the timestamp of the Start we extract the hour of the day. With the „starttime“ and „price“ we can see when advertising should cost more money or when to put special offers on the platform. The Join Attribut from buy-clicks.csv and user-session.csv is „userSessionId“.
“userSessionId” from buy-clicks.csv/user-session.csv	When starts a session, what “price” in a session. Join Attribute for the two datasets, but not needed for the final dataset (Training Data).

Training Data Set Creation

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

```
In [136]: trainingDF.head(5)
Out[136]:
   StartHour  price
0         14    1.0
1         14    2.0
2         14    2.0
3         14    1.0
4         14    3.0
```

Dimensions of the training data set (rows x columns) : [2375 rows x 2 columns]

# of clusters created: 3

Cluster Centers

Cluster #	Cluster Center
1	12.4825462, 20.
2	6.05505762, 3.77976953
3	17.70280036, 3.88166215

Screenshot:

```
In [146]: print(clusters.centers)#
[array([ 12.4825462,  20.       ]), array([ 6.05505762,  3.77976953]), array([ 17.70280036,   3.88166215])]
```

These clusters can be differentiated from each other as follows:

Cluster 1 is different from the others in that the center of the StartHours is ~12.48, which means between 12:00 and 13:00 o'clock. The price is centered by 20.

Cluster 2 is different from the others in that the center of the StartHours is ~6.05, which means short time after six o'clock am. The price is centered by ~3.78.

Cluster 3 is different from the others in that the center of the StartHours is ~17.70, which means short time before 6 o'clock pm. The price is centered by ~3.88.

Recommended Actions

Action Recommended	Rationale for the action
Advertising costs more during midday break	The center of the start time of a userSession is between 12 and 13 o'clock. So advertising can be priced with a model that determines the Starttime when users spend most money.
Special Offers for early birds and late players.	When users enter the paltform in the morning and in the late afternoon they do not spend much money. So it would be a good idea to place special offers during this time to animate these users to buy.
Mailing just before midday	Advertising E-Mails should be send just before the users who spend most money start there sessions.

Graph Analytics Analysis

Modeling Chat Data using a Graph Data Model

The Graph model for Chats has the following four nodes: User, Team, TeamChatSession and ChatItem.

There are eight relationships defined between these nodes:

- CreateChat: A User creates a ChatItem
- CreateSession: A User creates a TeamChatSession
- Join: A User joins a TeamChatSession
- Leaves: A User leaves a TeamChatSession
- Mentioned: A ChatItem mention a User
- OwnedBy: A TeamChatSession is owned by a Team
- PartOf: A ChatItem is part of a TeamChatSession
- ResponseTo: A ChatItem is a Response to an other ChatItem

Creation of the Graph Database for Chats

Theses are the steps for creating the graph database with Neo4j:

i) Analyzing the csv-files to load:

a. chat_create_team_chat.csv

- Rows: Team, TeamChatSession, timeStamp
- Relationships:
  - OwnedBy: TeamChatSession → Team
  - CreatesSession: User → TeamChatSession

b. chat_join_team_chat.csv

- Rows: User, TeamChatSession, timeStamp
- Relationships:
  - Join: User → TeamChatSession

c. chat_leave_team_chat.csv

- Rows: User, TeamChatSession, timeStamp

- Relationships:
  - Leaves: User → TeamChatSession

d. chat_item_team_chat.csv

- Rows: User, TeamChatSession, ChatItem, timeStamp
- Relationships:
  - CreateChat: User → ChatItem
  - PartOf: ChatItem → TeamChatSession

e. chat_mention_team_chat.csv

- Rows: ChatItem, User, timeStamp
- Relationships:
  - Mentioned: ChatItem → User

f. chat_respond_team_chat.csv

- Rows: ChatItem (1), ChatItem (2), timeStamp
- Relationships:
  - ResponseTo: ChatItem (1) → ChatItem (2)

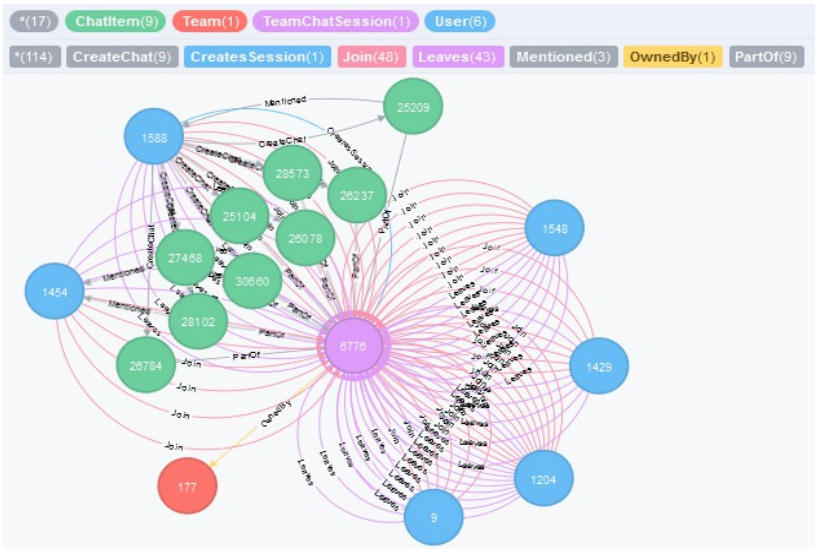
ii) Loading the files

After starting Neo4j open the following adress in a internet browser: http://localhost:7474
Neo4j Browser is a command driven client, like a web-based shell environment.
For Loading the datafiles we use "`LOAD CSV`" command in the command line of the browser environment. We have to define the path of the datafiles (here the files are located on a local path on the server), the columns we want to load and the relationships between the columns. Within the Load command we transform the string values into integer values and define the attribute "timestamp" to the relationsships.

Example LOAD command:

```
LOAD CSV FROM "file:///C:/coursera/data/chat_mention_team_chat.csv" AS row
MERGE (i: ChatItem {id: toInt(row[0])})
MERGE (u:User {id: toInt(row[1])})
MERGE (i)-[:Mentioned{timeStamp: row[2]}]->(u)
```

iii) Screenshot of some part of the graph



Finding the longest conversation chain and its participants

To find the longest converstion chain we use the following comands:

For calculating the five longest conversations chains we use a descending order for all conversation chains. An other possibility would be to use the max -funcion.

Command:

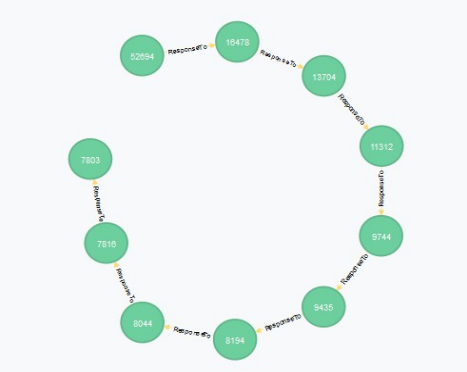
```
match r=(n)-[:ResponseTo*]->(m)
return length(r) as myCount
order by myCount desc limit 5
```

Result:

myCount
9
8
8
7
7

So the longest conversation chain has 9 relationships and 10 nodes (ChatItems).

Graph:



Distinct Users:

Five Users are involved in the Chain.

Notice:

Here we choose the directed paths (“->”), not the undirected. By using the undirected paths (like in the example of the course) we would get 11 relationships and 12 nodes as the longest path.

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

To find the top 10 chattiest users we count the ChatItems that are created by a user.

And sort the list descending.

Command:

```
MATCH (n:User)-[r:CreateChat]-(ChatItem)
WITH n, count(r) as ChatCount
RETURN n, ChatCount order by ChatCount desc limit 10
```

Result:

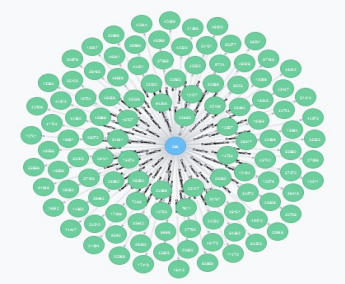
n	ChatCount
{id: 394}	115
{id: 2067}	111
{id: 209}	109
{id: 1087}	109
{id: 554}	107
{id: 516}	105
{id: 1627}	105
{id: 999}	105
{id: 668}	104
{id: 461}	104

Chattiest Users

Users	Number of Chats
394	115
2067	111
209	109

Just to verify it for one User, here User 394:

MATCH p=(u:User)-[r:CreateChat]->() where u.id = 394 RETURN p



To find the Chattiest Teams we first look for the TeamChatSessions with the most ChatItems (“PartOf”).

Command:

```
MATCH (n:ChatItem)-[r:PartOf]-(t:TeamChatSession)
WITH t, count(n) as ChatCount
RETURN t, ChatCount order by ChatCount desc limit 10
```

Result:

t	ChatCount
{id: 6792}	1324
{id: 6783}	1036
{id: 6925}	957
{id: 6791}	844
{id: 6974}	836
{id: 6889}	814
{id: 6780}	788
{id: 6819}	783
{id: 6850}	746
{id: 6778}	736

With the “Owned By” relationship we get the teams related to the TeamChatSessions.

Command:

```
MATCH p=(u:TeamChatSession)-[r:OwnedBy]->(t:Team)
where u.id IN [6792, 6783,  6925,  6791,  6974, 6889, 6780, 6819, 6850, 6778]
RETURN u.id,t.id
```

Result:

u.id	t.id
6819	136
6791	18

6780	52
6792	82
6974	194
6783	185
6850	146
6925	112
6889	129
6778	81

Chattiest Teams

Teams	Number of Chats
82	1,327
185	1.036
112	957

Finally we answer the question whether or not any of the chattiest users are part of any of the chattiest teams.

Command:

```
MATCH p=(u:User)-[r:Join]->(t:TeamChatSession)
WHERE
    u.id IN [394, 2067, 209, 1087, 554, 516, 1627, 999, 668, 461]  AND
    t.id IN [6792, 6783,  6925,  6791,  6974, 6889, 6780, 6819, 6850, 6778]
RETURN DISTINCT u.id,t.id
```

Result:

u.id	t.id
999	6780

We see that only one chattiest user belongs to a chattiest team.

It is User 999 who belongs to team 52. The other teams are more “homogen”.

How Active Are Groups of Users?

First we construct the neighborhood of users. In this neighborhood, we connect two users if a user mentioned an other user in a chat or if a user created a ChatItem in response to an other user's chatitem.

- create a new edge called “InteractsWith”

  1. A User has mentioned an other user in a chat:

  MATCH p=(u1:User)-[r:CreateChat]-(c:ChatItem)-[d:Mentioned]->(u2:User)

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

  Result:

  Created 11084 relationships, statement executed in 1272 ms.

  2. A User created a ChatiItem in response to an other user:

  MATCH p=(u1:User)-[r1:CreateChat]-(c1:ChatItem)-[d:ResponseTo]

  ->(c2:ChatItem)-[r2:CreateChat]- (u2:User) create (u1)-[:InteractsWith]->(u2)

  Result:

  Created 11073 relationships, statement executed in 922 ms.

- Delete relationships

  Because there are relationships from a user to himself we delete these relationships:

  match s=(n:User)-[r:InteractsWith]-(n:User) delete r

  Result:

  Deleted 4377 relationships, statement executed in 505 ms.

With this data preparation we can run the following query to find th most active users based on Cluster Coefficients.

```
Match (u:User)-[:CreateChat]-() with u, count(u) as count order by count desc limit 10
with u match (u)-[r:InteractsWith]-(u2)
with u, collect(distinct u2) as neighbors, count(distinct u2) as n_count with u,
neighbors, n_count
match (u1)-[r:InteractsWith]-(u2) where u1 in neighbors and u2 in neighbors with u,
count(distinct [u1,u2]) as c_count, n_count ,count(distinct
[u1,u2])*1.0/(n_count*(n_count -1)) as coe
return u.id, coe order by coe desc
```

Explanation:

- query the top 10 most active users
- use “InteractsWith” edge to find the number of neighbors for each users and save it to “n_count”
- count unique connections between the members of neighbors and save it to “c_count”
- calculate cluster coefficient using c_count*1.0/(n_count*(n_count -1)) and save it to “coe”.
- return “uid” and “coe” in descent order

Most Active Users (based on Cluster Coefficients)

User ID	Coefficient
461	1
394	1
516	0.9523809523809523
209	0.9523809523809523
554	0.9047619047619048
999	0.8666666666666667
1087	0.8
1627	0.7857142857142857
2067	0.7857142857142857
668	0.7