Technical Appendix

Catch the Pink Flamingo Analysis

Produced by: Juna Luzi

Acquiring, Exploring and Preparing the Data

Part 1: Data Exploration

Part 1.1: 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	Everytime a user clicks on an add, the information is recorded in this file as a new row	timestamp: when the user clicked on the ad txId: an id for click on an ad userSessionId: the id for the session the user is playing during the time they clicked the ad teamId: the id of the team that the user who clicked on the ad is a part of userId: the id of the user adId: the id of the ad the user clicked on adCategory: the category of the ad that the user clicked on
buy-clicks.csv	Everytime a user buys an item, that information is recorded in this file as a new row	timestamp: when the user bought the item txtId: an id for the click to buy an item userSessionId: the id for the session the user is playing during the time they bought an item team: which team the user is part of userId: the id of the user buyId: the id of the item bought price: the price the user paid for the item
game-clicks.csv	All the clicks for the flamingo game are recorded in this file as a new row	timestamp: when the user clicked on the flamingo game clickId: an id for the click during the game userId: the id of the user userSessionId: the session id when the user made the click in the game

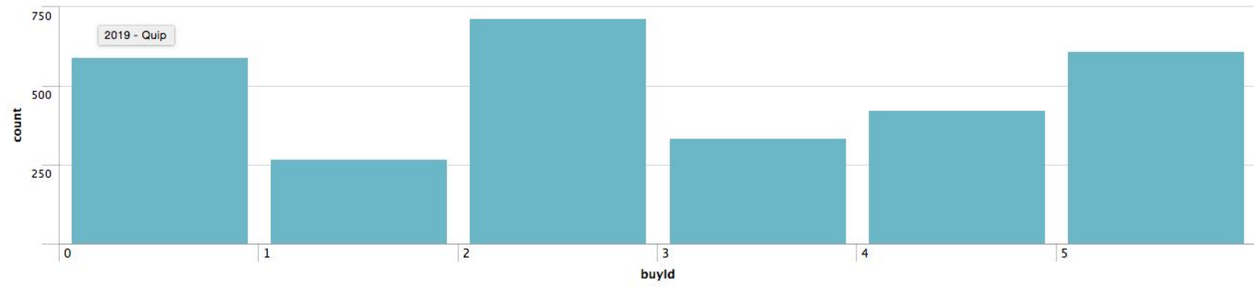
		isHit: whether a flamingo is hit or not (binary value) teamId: the id of the team the user is part of teamLevel: the level of the team in the game
level-events.csv	This file has information regarding teams moving up, down, or staying the same in game levels	timestamp: time when the event happened eventId: a unique id for the event teamId: the id of the team teamLevel: level of the team at that time eventType: the end ot the start of the event
team-assignments.csv	This file has information regarding which users are part of which teams	timestamp: when the user joined the team team: name of the team userId: the id of the user assignmentId: the id when the user is assigned to the team
team.csv	This file has information regarding teams	teamId: the id of the team name: the name of the team teamCreationTime: when the team was formed teamEndTime: time when the last member of the team left the team strength: how well the team is playing currentLevel: the current level of the team in the game
user-session.csv	This file keeps track of when a user starts and stops playing the game	timestamp: time when the user started or stopped playing userSessionId: a unique id for the session userId: user playing that session assignmentId: an assignment id of the team the user is a part of sessionType: the start or end of the event/session teamLevel: the level of the team during that session platformType: on which platform (mobile, web, etc) the user is

		playing
users.csv	This file has information regarding the user themselves	timestamp: when they became a user, signed up for the game userId: their unique id nick: nickname/username twitter: their twitter handler dob: their birthday information country: where they currently reside

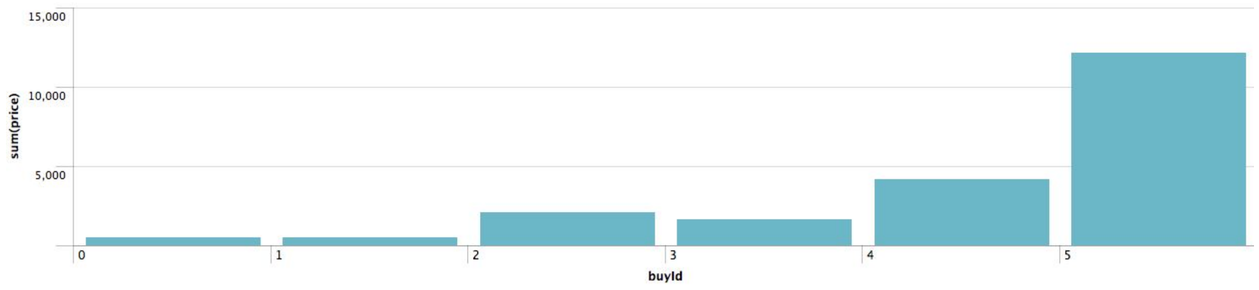
Part 1.2: Aggregation

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

A histogram showing how many times each item is purchased:

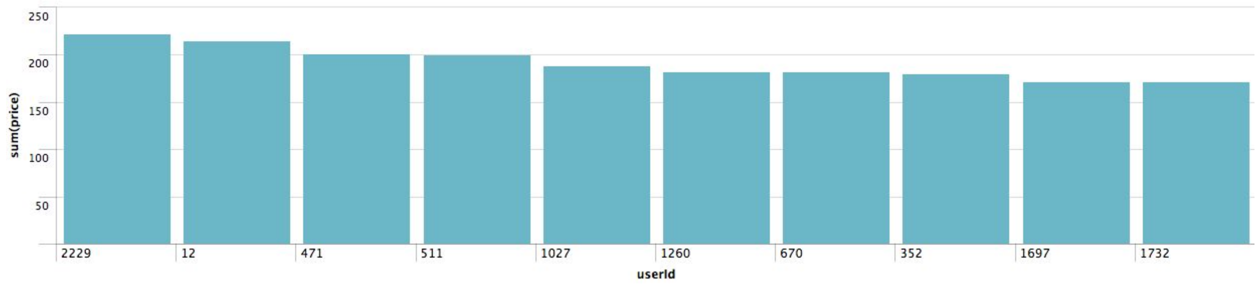


A histogram showing how much money was made from each item:



Part 1.3: 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	11.596958
2	12	iPhone	13.068182
3	471	iPhone	14.503817

Part 2.1: Data Preparation

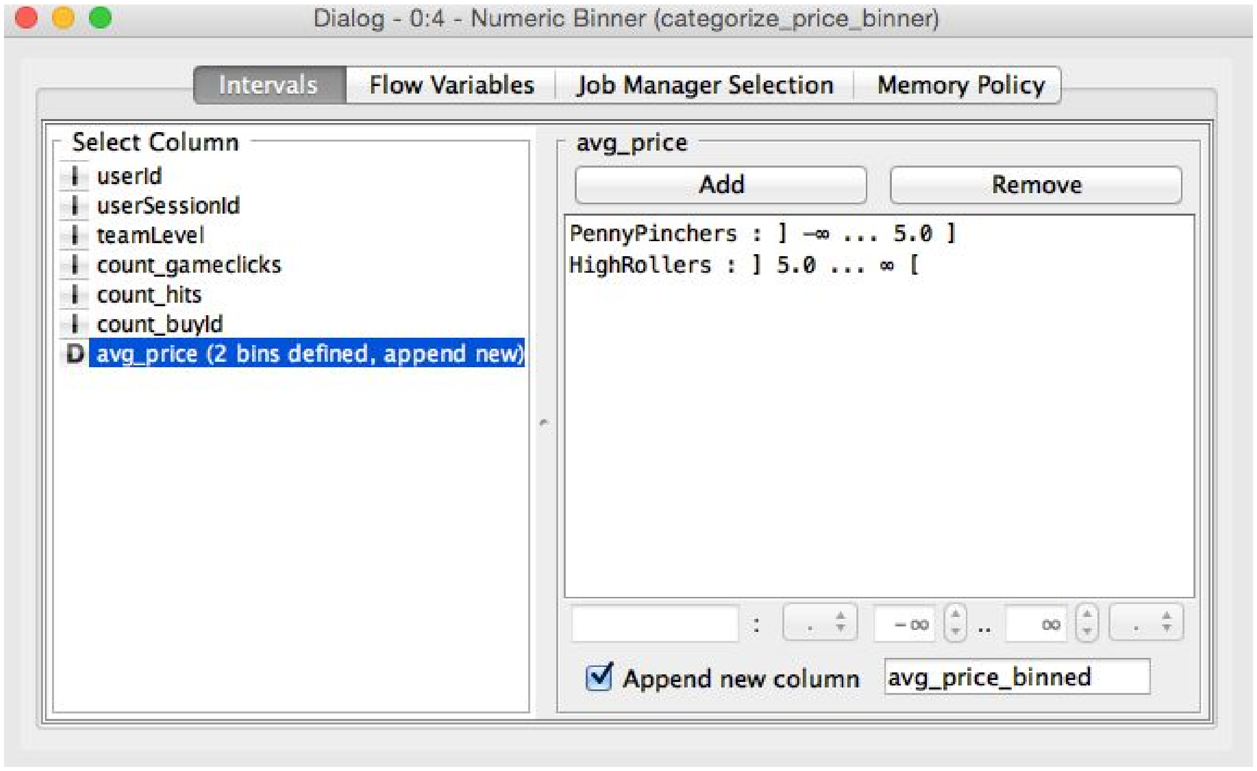
Analysis of combined_data.csv

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:



Dialog - 0:4 - Numeric Binner (categorize_price_binner)

userId
userSessionId
teamLevel
count_gameclicks
count_hits
count_buyId
avg_price (2 bins defined, append new)

avg_price

```
PennyPinchers : ] -∞ ... 5.0 ]
HighRollers : ] 5.0 ... ∞ [
```

☑ Append new column: avg_price_binned

The categorical attribute which I have named *avg_price_binned* will be used as the target during the classification process to categorize the users under two classes: buyers of items that cost more than $5.00 - shown above as HighRollers - and buyers of items that cost $5.00 or less - shown above as PennyPinchers. The table above maps any average price from negative infinity (although a negative price does not make sense so we can assume an item average price start at $0) to an average price of $5.0 including $5.0 as PennyPinchers. The "]" bracket after 5.0 means that the number 5 is included in the PennyPinchers category. Whereas, HighRollers include any average price that is bigger than 5.0. The "]" bracket in this case implies that the value 5.0 is excluded from the HighRollers class. The *avg_price_binned* is a binary categorical variable of either 0/1 or True/False.

The creation of this new categorical attribute was necessary because it will group user behavior under two categories and will allow us to predict with some level of uncertainty whether new users would fall under PennyPinchers or HighRollers. This information is also useful for targeting users who are HighRollers with ads of items that are more expensive which could increase revenues from items.

Attribute Selection

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

Attribute	**Rationale for Filtering**
avg_price	Since this is the information we are trying to predict as a categorical variable, it should not be included in the classification
userId	Since we are trying to predict the users that are more likely to buy items that cost more than $5.0, userId does not affect this
userSessionId	userSessionId does not add any knowledge or value to finding which users are more likely to buy items that cost more than $5.0

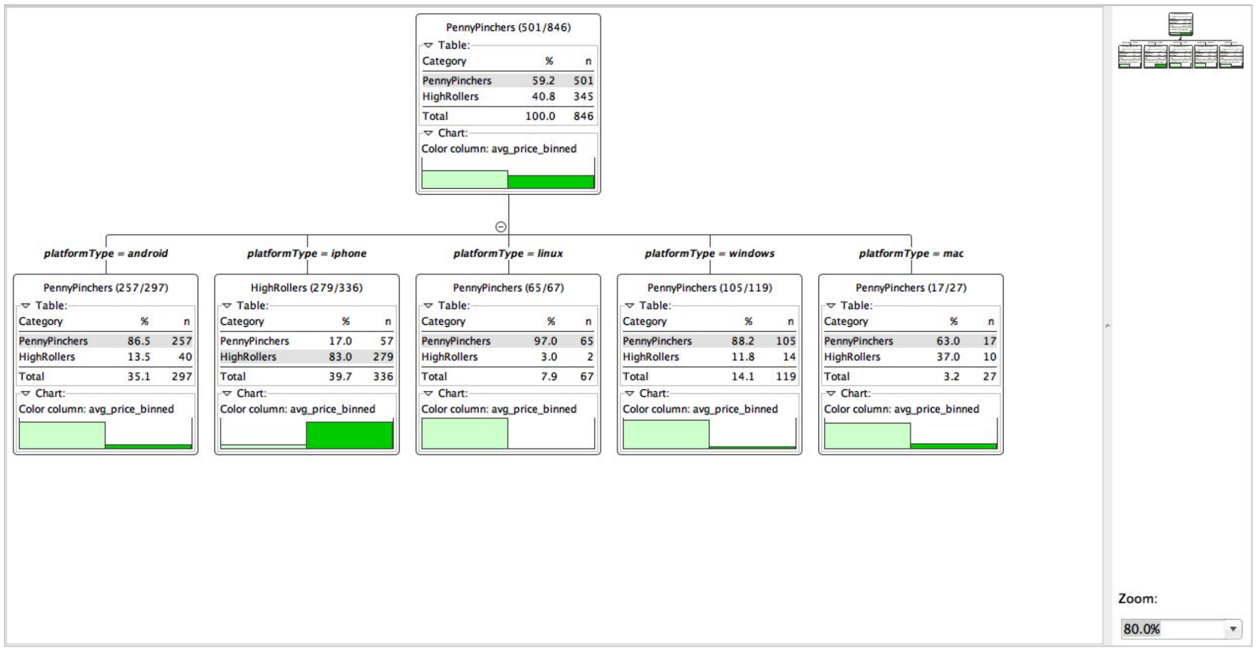
Part 2.2: Data Partitioning and Modeling

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 want to make sure our classifier is not overfitting or underfitting.

When partitioning the data using sampling, it is important to set the random seed because we want to obtain the same training and test datasets to train and test the decision tree algorithm throughout the participants in the course.

The data was partitioned into 60% train and 40% test datasets. The training data is used to build the classification algorithm, whereas the test data is used to evaluate the classifier on new, unseen data to make sure our classification is not overfitting or underfitting. For this project, we selected the stratified sampling method which divides the datasets into separate groups referred to as strata and a probability sample is drawn from each group. When partitioning the data, it is important to set the random seed to a certain number which in this case is 1466016757670 so that we obtain the same training and test datasets to train and test the decision tree algorithm.

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



PennyPinchers (501/846)

Table:

Category	%	n
PennyPinchers	59.2	501
HighRollers	40.8	345
Total	100.0	846

Chart:
Color column: avg_price_binned

platformType = android

PennyPinchers (257/297)

Category	%	n
PennyPinchers	86.5	257
HighRollers	13.5	40
Total	35.1	297

Color column: avg_price_binned

platformType = iphone

HighRollers (279/336)

Category	%	n
PennyPinchers	17.0	57
HighRollers	83.0	279
Total	39.7	336

Color column: avg_price_binned

platformType = linux

PennyPinchers (65/67)

Category	%	n
PennyPinchers	97.0	65
HighRollers	3.0	2
Total	7.9	67

Color column: avg_price_binned

platformType = windows

PennyPinchers (105/119)

Category	%	n
PennyPinchers	88.2	105
HighRollers	11.8	14
Total	14.1	119

Color column: avg_price_binned

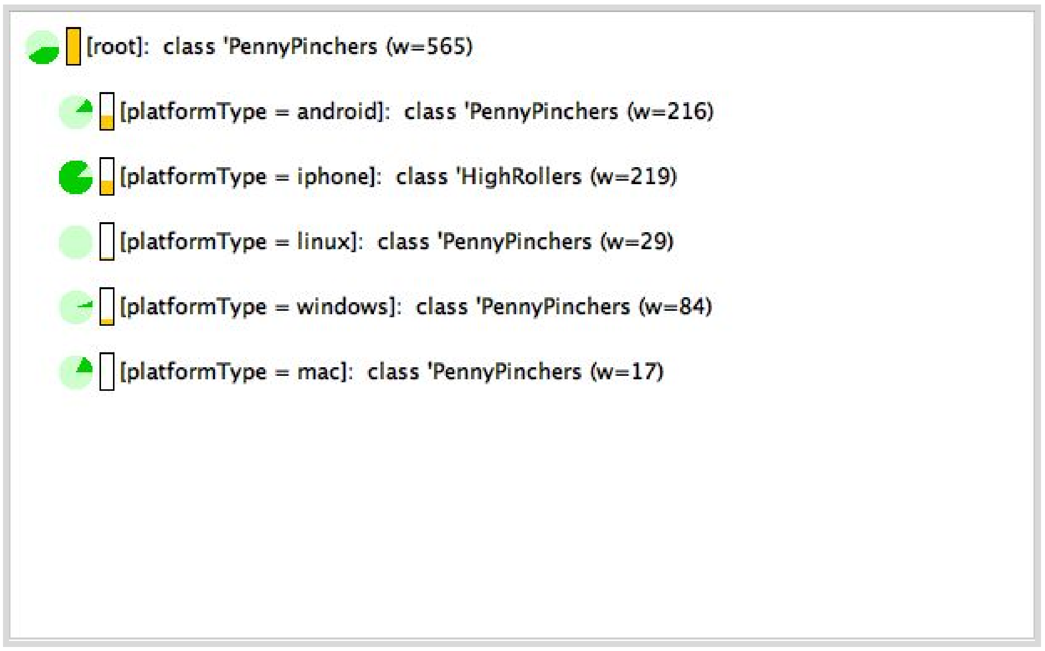
platformType = mac

PennyPinchers (17/27)

Category	%	n
PennyPinchers	63.0	17
HighRollers	37.0	10
Total	3.2	27

Color column: avg_price_binned

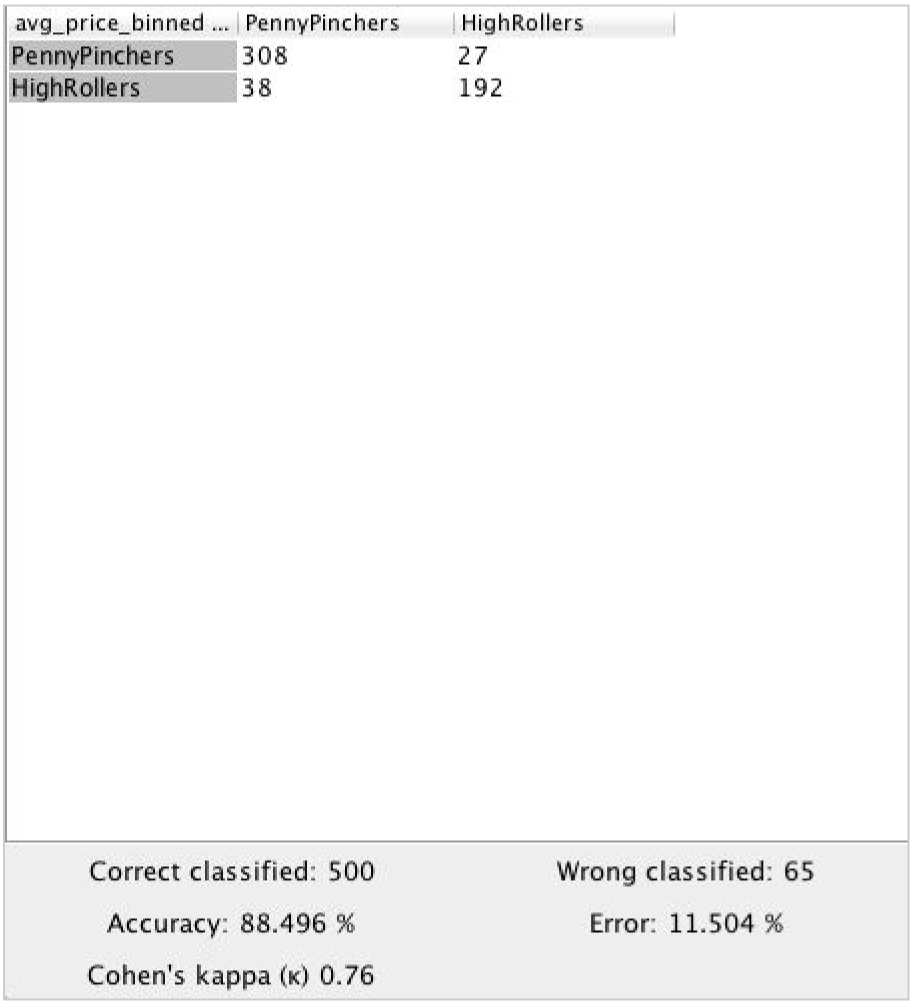
Zoom: 80.0%



```
[root]:  class 'PennyPinchers (w=565)
    [platformType = android]:  class 'PennyPinchers (w=216)
    [platformType = iphone]:  class 'HighRollers (w=219)
    [platformType = linux]:  class 'PennyPinchers (w=29)
    [platformType = windows]:  class 'PennyPinchers (w=84)
    [platformType = mac]:  class 'PennyPinchers (w=17)
```

Part 2.3: Evaluation

A screenshot of the confusion matrix can be seen below:



avg_price_binned ...	PennyPinchers	HighRollers
PennyPinchers	308	27
HighRollers	38	192

Correct classified: 500

Wrong classified: 65

Accuracy: 88.496 %

Error: 11.504 %

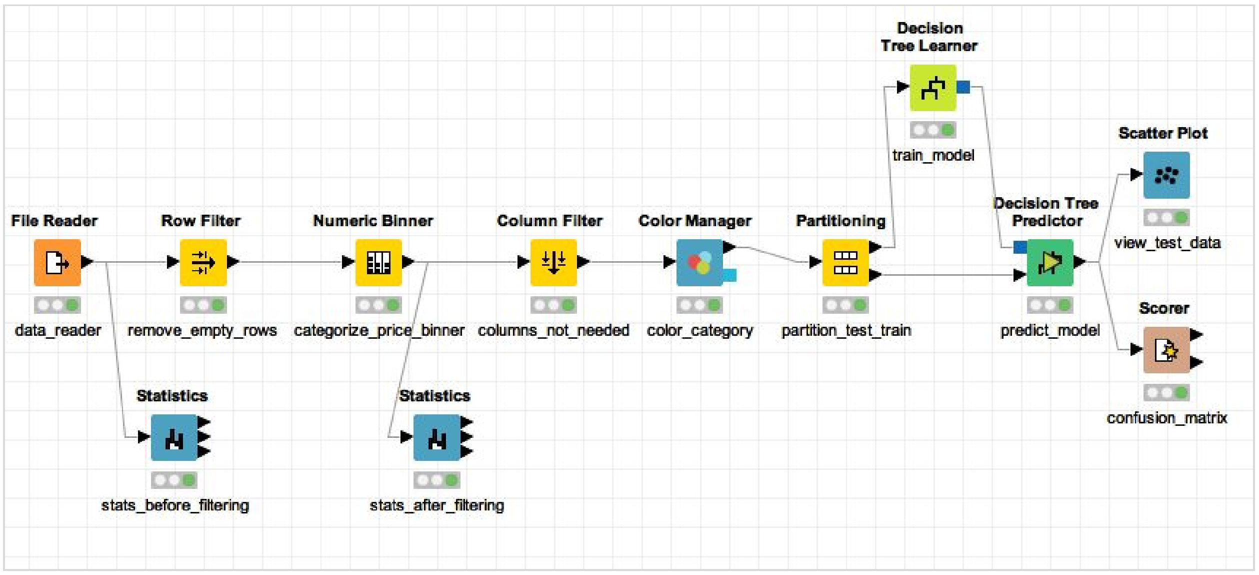
Cohen's kappa (κ) 0.76

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

The number of PennyPinchers that were categorized correctly as PennyPinchers is 308. The number of PennyPinchers that were categorized as HighRollers is 27. This means that 92% of all PennyPinchers were categorized correctly.
The number of HighRollers that were categorized correctly as HighRollers is 192. The number of HighRollers that were categorized as PennyPinchers is 38. This means that 83% of all the HighRollers were categorized correctly.
Overall, approximately 88% of all the data is categorized correctly which is also indicated above under the accuracy score.

Part 2.4: Analysis Conclusions

The final KNIME workflow is shown below:



What makes a HighRoller vs. a PennyPincher?

From the decision tree view where the users are grouped by *platformType*, it seems obvious that PennyPinchers are more likely to be users on platforms such as android, linux, windows and mac; whereas HighRollers are more likely to be iPhone users.

Specific Recommendations to Increase Revenue
1. Target iPhone users with high price items since they are more likely to be HighRollers
2. Increase the number of low price items for Android, Linux, Windows and Mac users since they are more likely to be PennyPinchers

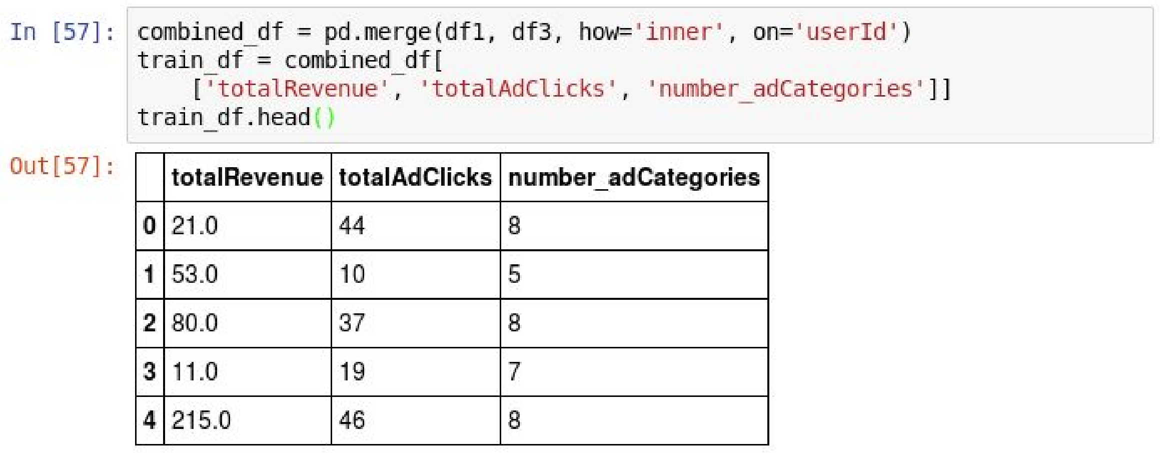
Part 3: Clustering Analysis

Part 3.1: Attribute Selection

Attribute	Rationale for Selection
Total ad clicks per user	This attribute would allow us to see if there are groups of users that are high ad-click users or low ad-click users. This provides information on user ad click behavior.
Sum of money spent by users	This attribute would allow us to see if there are varying degrees on user preferences since we can capture the total cost per user
Number of ad categories per user	This attribute would allow us to see if users prefer varying ad categories or if they focus on a few since we can capture the number of ad categories per user

Part 3.2: Training Data Set Creation

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

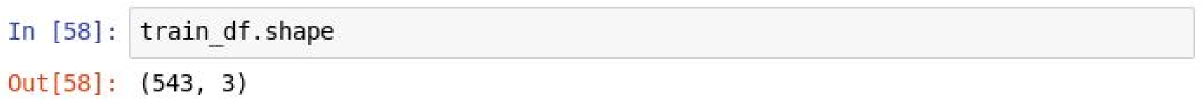


```
In [57]: combined_df = pd.merge(df1, df3, how='inner', on='userId')
         train_df = combined_df[
             ['totalRevenue', 'totalAdClicks', 'number_adCategories']]
         train_df.head()
```

Out[57]:

	totalRevenue	totalAdClicks	number_adCategories
0	21.0	44	8
1	53.0	10	5
2	80.0	37	8
3	11.0	19	7
4	215.0	46	8

Dimensions of the training data set (rows x columns) : (543 * 3)



```
In [58]: train_df.shape

Out[58]: (543, 3)
```

# of clusters created: 3

Part 3.3: Cluster Centers

Cluster #	Center
1	[67.448, 34.144, 8.128]
2	[145.51111111, 41.06666667, 8.48888889]
3	[17.12600536, 26.36461126, 7.36729223]

The first index of all the arrays above shows the totalRevenue; the second index of all the arrays above shows totalAdClicks; and the last index of all the arrays above shows the number_adCategories.

These clusters can be differentiated from each other as follows:

Cluster 1 is different from the others in that they are not necessarily the highest or lowest spenders from the two clusters, their total ad clicks fall in between the ranges of clusters 2 and 3, and the number of ad categories that they also click on is between the ranges of the other two clusters. It seems like these users are on-the-margin users.

Cluster 2 is different from the others in that it seems to be the cluster with users that buy the most items, have higher total ad clicks and click on a higher variety of ad categories (although the magnitude is not substantially different from the other two clusters, it is still a bit higher). We can categorize these users are high spending users.

Cluster 3 is different from the others in that the total revenue, total ad clicks per user and number of ad categories per user are all less than the rest of the cluster; which indicates that these users are low level spending users. Their ad-click ratio is smaller than the rest of the users and their number of ad categories is less than the other users -although not by much - while it looks like the cost is a lot less for cluster 3 users than the other two clusters.

Part 3.4: Recommended Actions

Action Recommended	Rationale for the action
Provide promotional offers for cluster 1 users	Target these users with promotional offers to bring the users to buy more items and click more ads since they are users on the margin. Figuring out what incentivizes these users to become as “active” users as cluster 2 users could change their behavior and eventually shift them to cluster 2 users.
Provide cluster 2 users	Target these users with more products since they are high

with more products (efficient resource allocation)	spenders and have higher ad-click ratio and their ad category preference is higher. Maybe diverting some of the resources from cluster 3 users to cluster 2 users could benefit more since we will be allocating resources to higher spending users.

Part 4: Graph Analytics Analysis

Modeling Chat Data using a Graph Data Model

In the graph model for chats, a user can create, join or leave a chat session while being a member of a team. In that chat session, a user can create chat items and be a part of a chat item. A user can also be mentioned in a chat item. Lastly, a chat item can respond to another chat item, which represents the communication between users.

Part 4.1: Creation of the Graph Database for Chats

Part 4.1.1: Schema

The schema of the 6 CSV files which were used for creating the graph model is as follows:

chat_create_team_chat.csv	UserId	Contains the id of the user; format of the entries are all int
	TeamId	Contains the id of the team that the user is part of; format of the entries are all int
	TeamChatSessionId	Contains the id of the chat session the user is a part of; format of the entries are all int
	timestamp	Contains the timestamp that the user created the chat
chat_join_team_chat.csv	UserId	Contains the id of the user; format of the entries are all int
	TeamChatSessionId	Contains the id of the chat session the user is a part of; format of the entries are all int
	timestamp	Contains the timestamp that the user joined the team
chat_leave_team_chat.csv	UserId	Contains the id of the user; format of the entries are all int
	TeamChatSessionId	Contains the id of the chat session the user is a part of; format of the entries are all int
	timestamp	Contains the timestamp that the user left the team

chat_item_team_chat.csv	UserId	Contains the id of the user; format of the entries are all int
	TeamChatSessionId	Contains the id of the chat session the user is a part of; format of the entries are all int
	ChatItemId	Contains the id of the chat item; format of the entries are all int
	timestamp	Contains the timestamp the user created the chat item; can also be used as timestamp that denotes the time the user became a part of the chat item
chat_mention_team_chat.csv	ChatItemId	Contains the id of the chat item; format of the entries are all int
	UserId	Contains the id of the user; format of the entries are all int
	timestamp	Contains the timestamp the user was mentioned in the chat item
chat_respond_team_chat.csv	ChatItemId	Contains the id of the chat item; format of the entries are all int
	ChatItemId	Contains the id of the chat item; format of the entries are all int
	timestamp	Contains the timestamp one chat item responded to a chat item

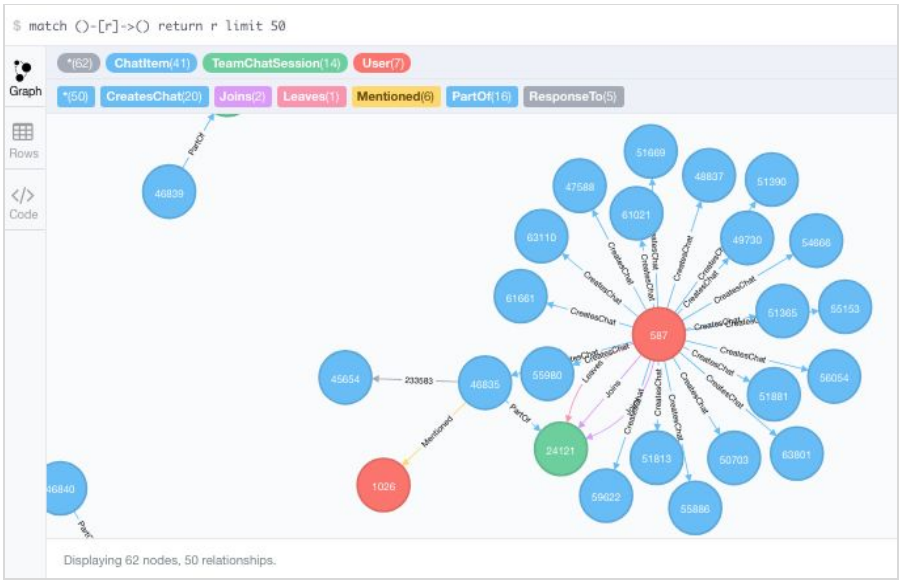
Part 4.1.2: Loading Process

To load the datasets we use the cypher language. Below is an example of how to load the file chat_create_team_chat.csv.

```
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);
```

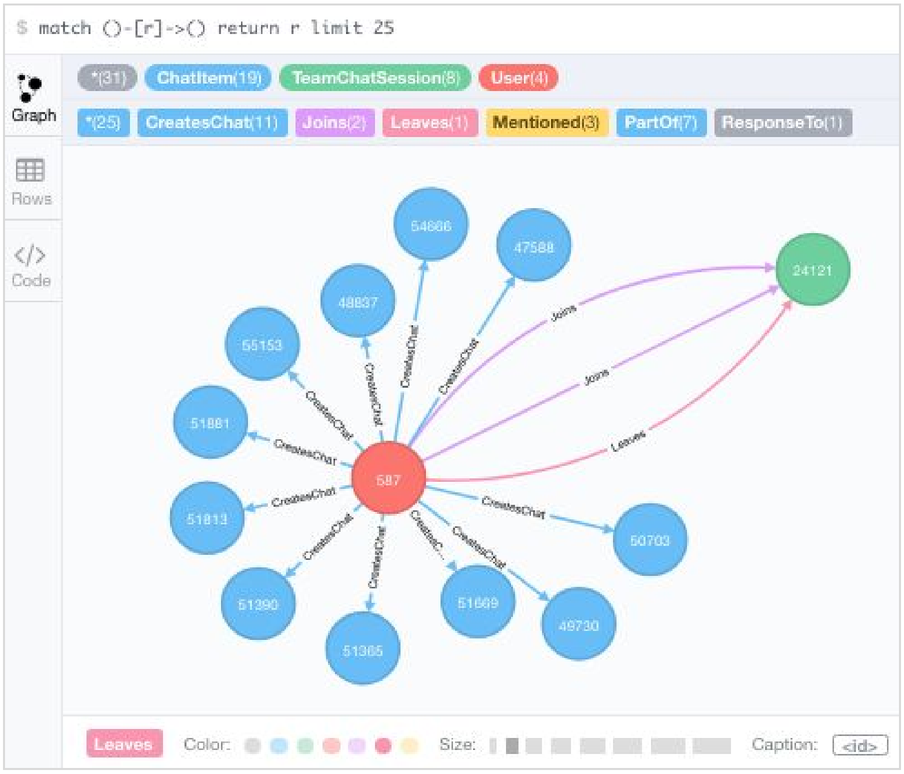
The first row loads the csv file from the path under which the data files are saved and iterates through every row of the csv file as “row”. It then creates the graph data by creating the nodes “User”, “Team”, and “TeamChatSession”, and converting their type from String to Int. The indexing of the rows indicates that the first item (index zero) is the “User” node information, the second index (index one) is the “Team” node information, and the third index (index two) is the “TeamChatSession” information. Then, the edges are created by identifying which nodes they connect and including any additional attributes for each edge. In the example above, we create edges “CreatesSession” with a “timeStamp” attribute which is stored in the fourth index (index three) of the “row”. These edges connect nodes “User” and “TeamChatSession”. Lastly, we create the edges “OwnedBy” with the same “timeStamp” attribute as the edge “CreatesSession”, but these edges connect nodes “TeamChatSession” and “Team”.

Part 4.1.3: Screenshots of the Graph



```
$ match ()-[r]->() return r limit 50
```

```
Displaying 62 nodes, 50 relationships.
```

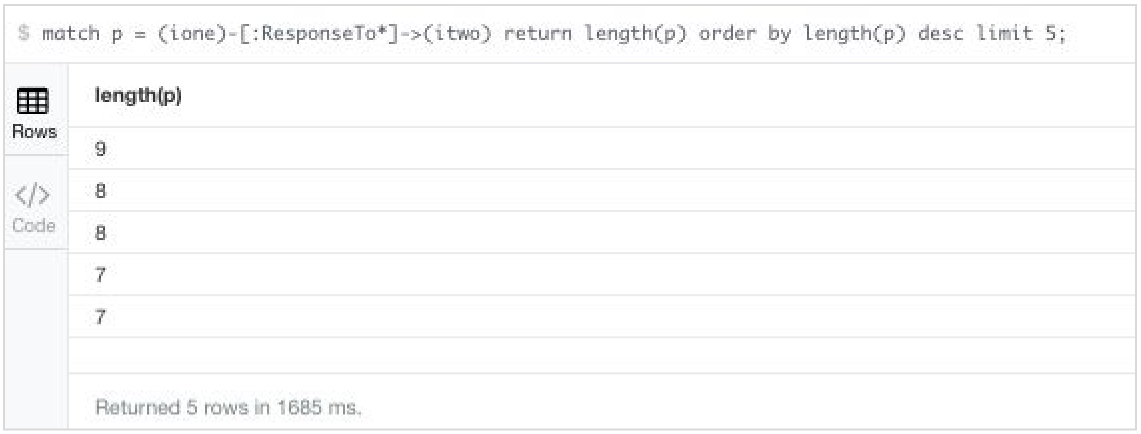

```
$ match ()-[r]->() return r limit 25
```

Part 4.2: Finding the longest conversation chain and its participants

Part 4.2.1: Longest Conversation Chain

To find the longest conversation chain in the chat data using the "ResponseTo" edge label we can use the cypher script below to find the length of the edge chains with label "ResponseTo" and sort them by descending order so that the longest conversation chains are shown. In my script, I have limited the results to the top 5 to make sure that there are no two conversation chains with the same length.

```
match p = (ione)-[:ResponseTo*]->(itwo)
return length(p)
order by length(p) desc limit 5;
```



```
$ match p = (ione)-[:ResponseTo*]->(itwo) return length(p) order by length(p) desc limit 5;
```

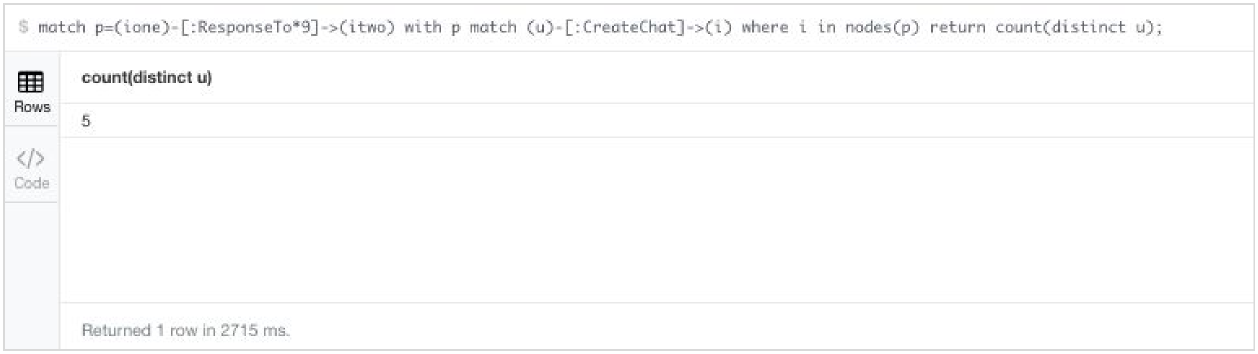
```
length(p)
9
8
8
7
7

Returned 5 rows in 1685 ms.
```

Part 4.2.2: Longest Conversation Chain Participants

The answer to the query, as seen above, is 9. This information helps to find out the participants of the longest query. To find the participants, we use the beginning of the query as in the example above and filter where the length of the edge chains is 9.

```
match p=(ione)-[:ResponseTo*9]->(itwo)
with p
match (u)-[:CreateChat]->(i)
where i in nodes(p)
return count(distinct u);
```

The nodes and edges information is saved in the variable "p". We then find all the users that have created a chat item where the chat item is part of the nodes that have been saved in variable p (the item chats that are part of the longest conversation chain). To get the number of participants, we return the count of the unique user nodes. Results can be seen in the image below:



```
$ match p=(ione)-[:ResponseTo*9]->(itwo) with p match (u)-[:CreateChat]->(i) where i in nodes(p) return count(distinct u);
```

```
count(distinct u)
5

Returned 1 row in 2715 ms.
```

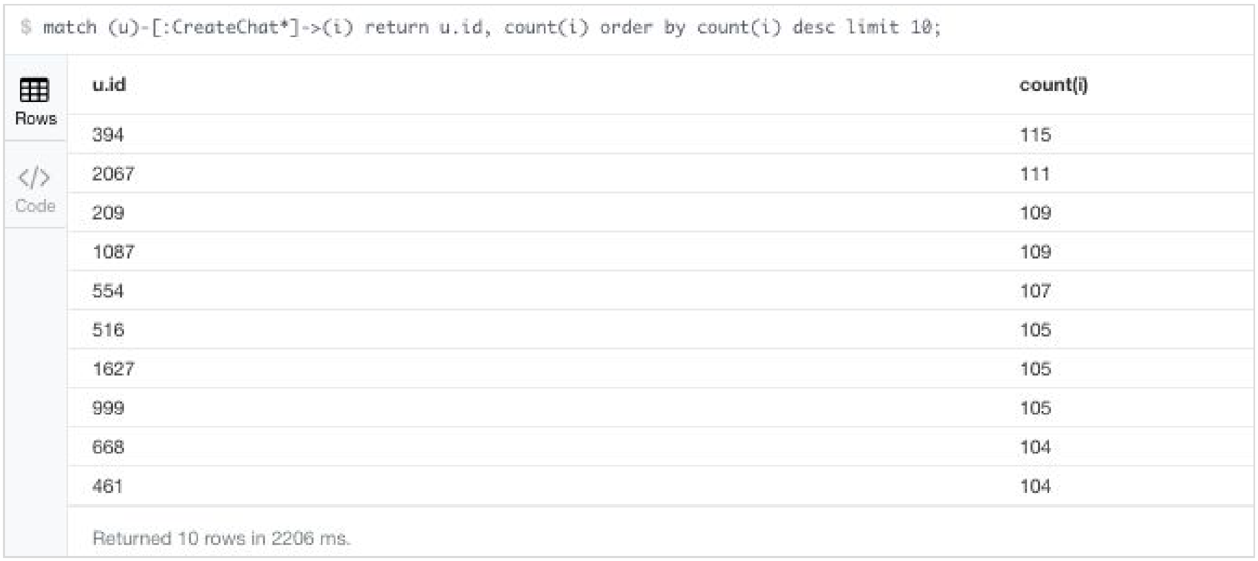
Part 4.3: Analyzing the relationship between top 10 chattiest users and top 10 chattiest teams

Part 4.3.1: Top 10 Chattiest Users

First we find the top 10 chattiest users with the following cypher query:

```
match (u)-[:CreateChat*]->(i)
return u.id, count(i)
order by count(i) desc limit 10;
```

We find all the user nodes that have created a chat item, therefore are connected to a chat item and return the user id attribute and count the item chats each user has created. We then order the results by showing the users that have higher numbers of chat items created and limiting the results to top 10. The table below shows the results.



```
$ match (u)-[:CreateChat*]->(i) return u.id, count(i) order by count(i) desc limit 10;
u.id      count(i)
394       115
2067      111
209       109
1087      109
554       107
516       105
1627      105
999       105
668       104
461       104
Returned 10 rows in 2206 ms.
```

Chattiest Users

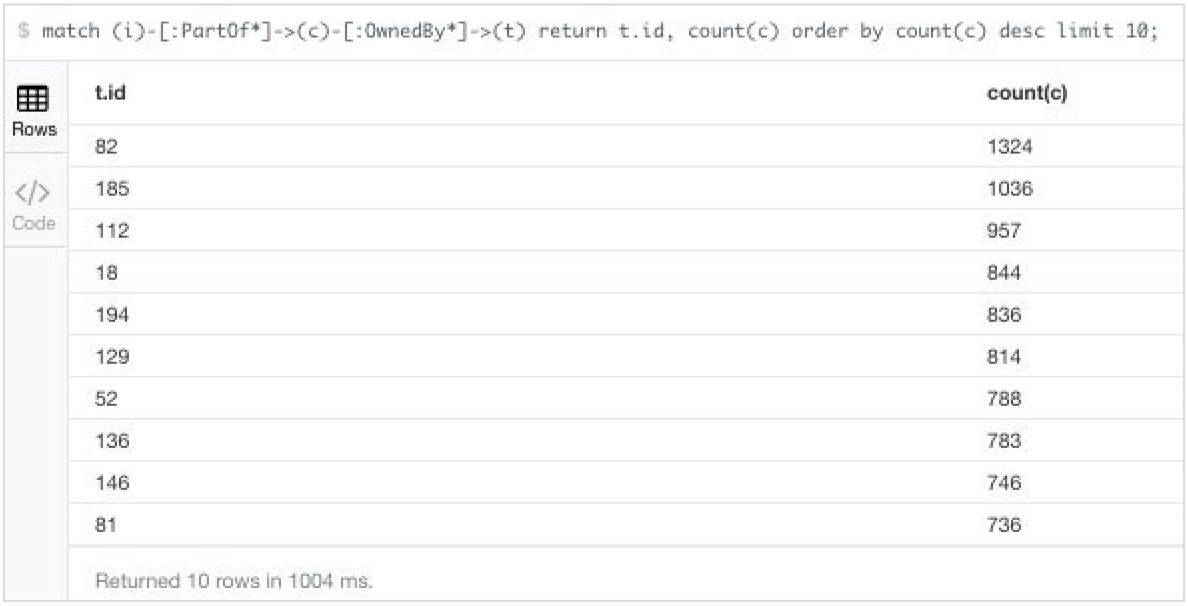
Users	Number of Chats
394	115
2067	111
209	109
1087	109

Part 4.3.2: Top 10 Chattiest Teams

We then proceed to find the top 10 chattiest teams with the following cypher query:

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

We find all the item chats that are part of team chat sessions that are owned by teams, order them by count of team chat sessions so that the highest team chat sessions will show up at the top and limit the display to top 10. The table below shows the results of the query:


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

t.id	count(c)
82	1324
185	1036
112	957
18	844
194	836
129	814
52	788
136	783
146	746
81	736

Returned 10 rows in 1004 ms.

Chattiest Teams

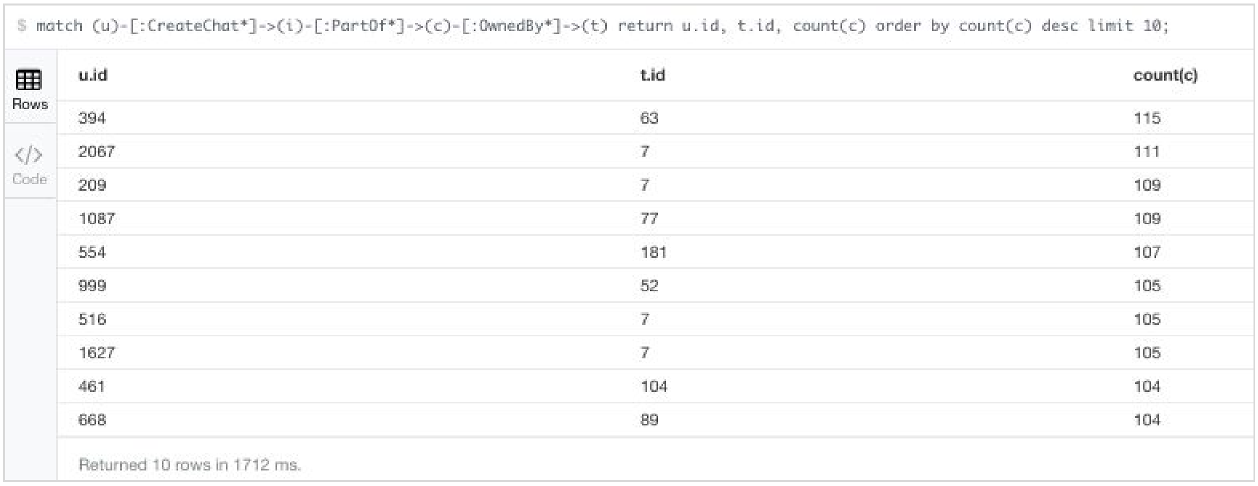
Teams	Number of Chats
82	1324
185	1036
112	957

Part 4.3.3: Results

To find whether the top 10 chattiest users belong to the top 10 chattiest teams, we can combine the two queries to find the teams the top 10 chattiest users belong to, and order the results by highest team chat sessions and only display top 10.

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

The table below shows the results for this query:



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

u.id	t.id	count(c)
394	63	115
2067	7	111
209	7	109
1087	77	109
554	181	107
999	52	105
516	7	105
1627	7	105
461	104	104
668	89	104

Returned 10 rows in 1712 ms.

It is evident that the only user that is one of the top 10 chattiest users who also belongs to the top 10 chattiest teams is user with id 999 who belongs to team with id 52. The rest of the users are not part of the top 10 chattiest teams.

Part 4.4: How Active Are Groups of Users?

To answer this question, we look at how dense each neighborhood of group users is and use that information to capture how active the groups of users are.

Part 4.4.1: Create Edges "InteractsWith"

To identify the neighborhoods of group users we need to create a connection which will be named "InteractsWith" between users if:

1) One user mentioned another user in a chat item

The following query will create the edge with attribute "InteractsWith" if one user mentioned another used in a chat item:

```
match (u1:User)-[:CreateChat]->(i)-[:Mentioned]->(u2:User)
create (u1)-[:InteractsWith]->(u2);
```

2) One user created a chat item in response to another user's chat item

The following query will create the edge with attribute "InteractsWith" if one user created a chat item in response to another user's chat item:

```
match (u1:User)-[:CreateChat]->(i1:ChatItem)-[:ResponseTo]-(i2:ChatItem)
with u1, i1, i2
match (u2)-[:CreateChat]-(i2)
create (u1)-[:InteractsWith]->(u2);
```

The above query creates an undesirable side effect if a user has responded to their own chat item, because it will create a self loop between two users. So after, the query above, we will need to eliminate all self loops involving the edge “InteractsWith”. This can be done through this query:

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

Part 4.4.2: Creating the Clustering Coefficient

To create the clustering coefficient -following the query below - we need to find all the users that are connected through the edge “InteractsWith” (line 1), that are not the same user (line 2) and that are part of the top chattiest users (line 3) and collect the user node information in a list called “neighbors” and the number of distinct nodes as “neighborCount” (line 4). Then, for the list of “neighbors”, we collect number of edges “InteractsWith” (lines 5 and 6). For any pairs of neighbor nodes have more than one edge, we count them as just 1, while for any neighbor nodes that have no edges, we count it as 0 (lines 7, 8, and 9). Therefore, the command *sum(hasedge)* will have the total number of edges between neighbor nodes. We then use the formula on line 11 to calculate the coefficient as was indicated in the assignment and return the coefficients from highest to lowest.

```
match (u1:User)-[r1:InteractsWith]->(u2:User)
where u1.id <> u2.id
AND u1.id in [394,2067,1087,209,554,999,516,1627,461,668]
with u1, collect(u2.id) as neighbors, count(distinct(u2)) as neighborCount
match (u3:User)-[r2:InteractsWith]->(u4:User)
where (u3.id in neighbors) AND (u4.id in neighbors) AND (u3.id <> u4.id)
with u1, u3, u4, neighborCount,
case when count(r2) > 0 then 1
else 0
end as answer
return u1.id, sum(answer)*1.0/(neighborCount*(neighborCount-1)) as coeff order by coeff
desc limit 10;
```

Most Active Users (based on Cluster Coefficients rounded to 4 decimals)

User ID	Coefficient
209	0.9523
554	0.9047
1087	0.8

Part 5: Recommended Actions

- Target iPhone users with high price items

  The decision tree classification created in KNIME indicated that iPhone users are more likely to be HighRollers - users who spend more than $5.0 on items - therefore, one way to increase revenue could be by targeting these users with more high price items or items with higher prices.
- Increase the number of low price items for other platform users

  The decision tree implemented in KNIME also indicated that other platform users such as Android, Linux, Windows and Mac users are more likely to be PennyPinchers - users who spend $5.0 or less on items - therefore, while the price of items can remain low, we can increase the number of items shown to these users.
- Target users in cluster 1 with promotional offers

  The implementation of the k-means clustering algorithm in PySpark indicated that users in the first cluster with cluster center [67.448, 34.144,   8.128] should be targeted with promotional offers to bring the users to buy more items and click more ads since they are users on the margin. Figuring out what incentivizes these users to become as “active” (high ad clickers and item buyers) users as cluster 2 users could change their behavior and eventually shift them to cluster 2 users.
- Target users in cluster 2 with more diverse products

  The implementation of the k-means clustering algorithm in PySpark also indicated that users in the second cluster with cluster center [145.51111111, 41.06666667,   8.48888889] should be targeted with more products since they are high spenders and have higher ad-click ratio and their ad category preference is higher. Maybe diverting some of the resources from cluster 3 users to cluster 2 users could benefit more since we will be allocating resources to higher spending users.