Technical Appendix Content

Data Exploration with Splunk

Data Classification with KNIME

Clustering with Spark

Graph Analytics with Neo4j

Data Exploration with Splunk

Data Exploration with Splunk

This appendix contains and exploration of the Catch the Pink Flamingo data. The data is explored using Splunk.

Data Set Overview

The table below lists each of the files available for analysis with a short description of what is found in each one.

File Name	Description	Fields
ad-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. **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 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.	**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	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 **teamId**: the id of the team

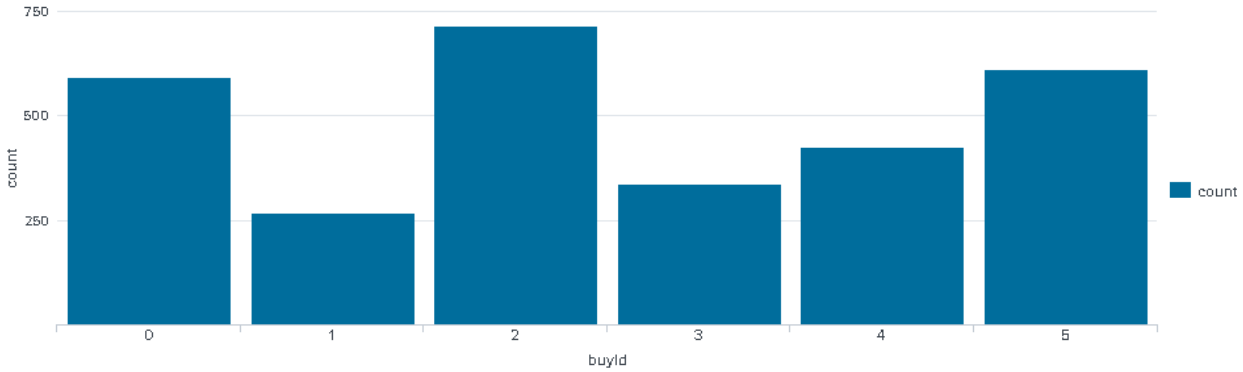
		teamLevel: the level started or completed **eventType**: the type of event, either start or end
user-session.csv	Each line in this file describes a user session, which denotes when a user starts and stops playing the game. Additionally, when a team goes to the next level in the game, the session is ended 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.	**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

Aggregation

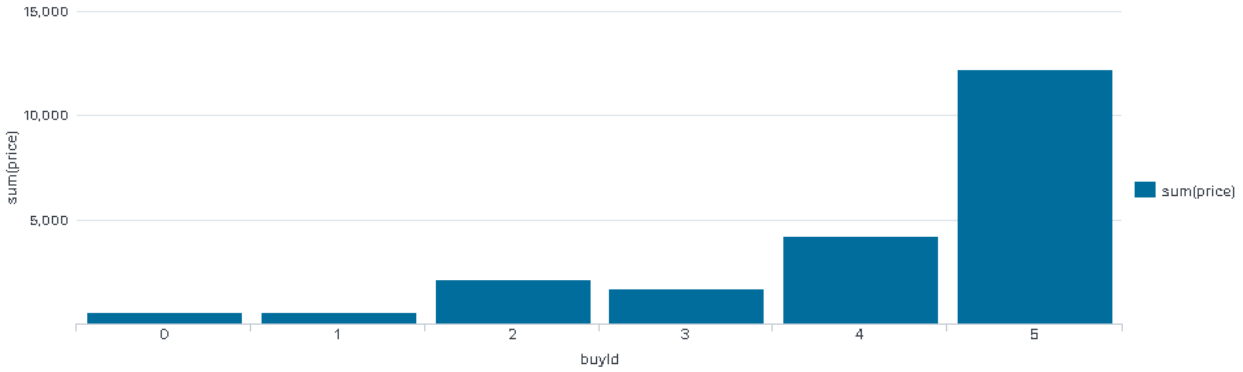
The table below shows the total amount of money spend with in-game purchases and the number of unique items which can be bought in-game.

Amount spent buying items	21,407
Number of unique items available to be purchased	6

A histogram showing how many times each item is purchased:

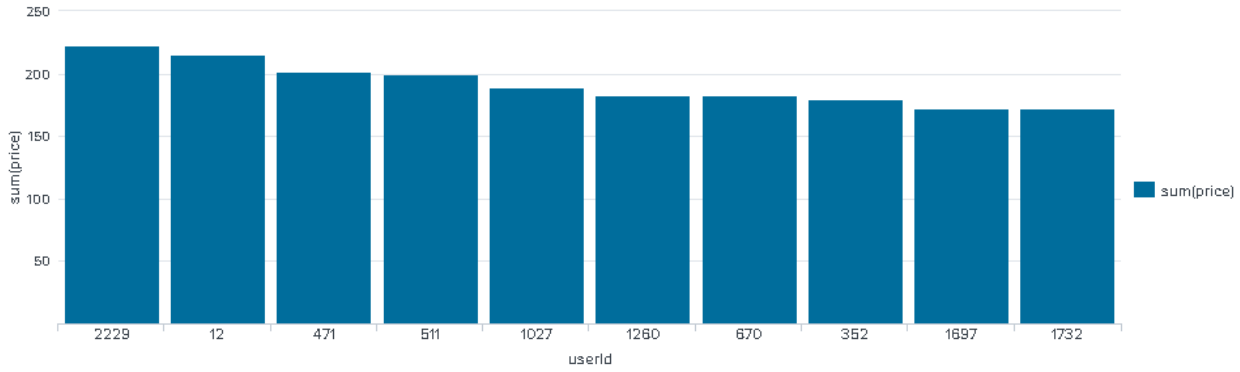


A histogram showing how much money was made from each item:



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	12
2	12	iphone	13
3	471	iphone	15

Data Classification with KNIME

Data Classification with KNMIE

This appendix contains a step by step approach for classification of the Cath the Pink Flamingo data. The Goal of this classification is to determine is a user is a HighRoller or a PennyPincher.

Data Preparation

The classification will be performed on the combined_data.csv dataset. This is a subset of the Catch the Pink Flamingo dataset.

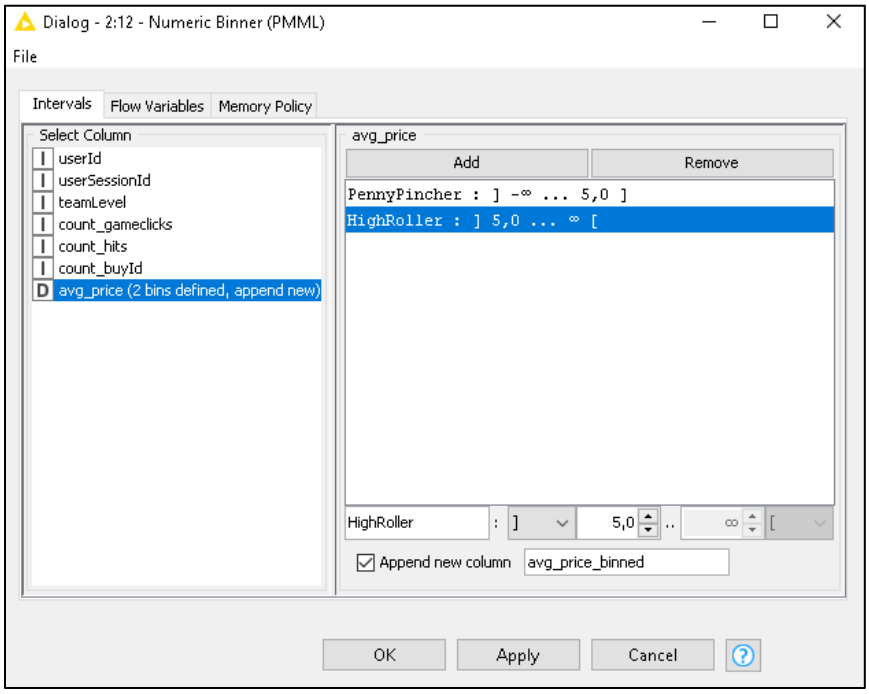
Sample Selection

The table below shows the total number of samples in the dataset and the number of samples with in-game purchases.

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

Attribute Creation

To enable to classify a user as a HighRoller or a PennyPincher a new categorical attribute is created from the numerical “avg_price” attribute. Below a screenshot is show of the creation of the new attribute



A user is classified as a PennyPincher when the avg_price is lower than or equal to 5 dollar. And a user is classified as a HighRoller when the “avg_price” is higher than 5 dollar.

The creation of this new categorical attribute is required for classification when using a decision tree. Also, with this new attribute the numerical attribute with a wide range (avg_price) is compressed into a binary attribute. This reduction is beneficial when constructing the decision tree.

Attribute Selection

To construct the decision tree highly correlated and irrelevant attributes must be filtered. In the table below the filtered attributes are show together with the reason for filtering.

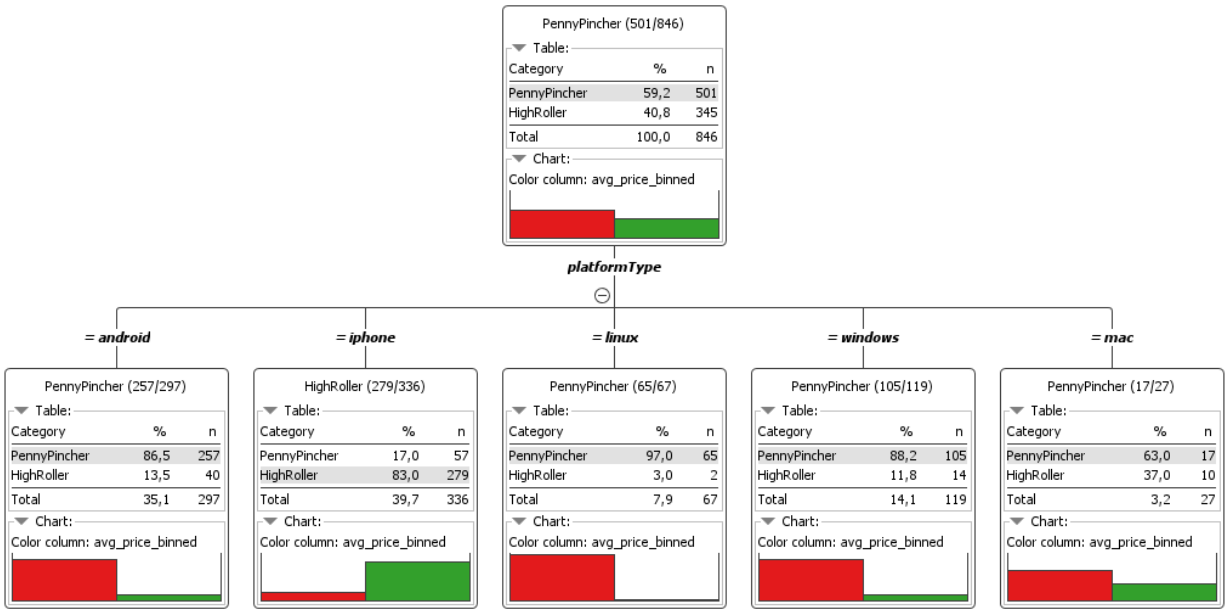
Attribute	**Rationale for Filtering**
userID	This attribute is filtered since this is user specific. We wat to create a model applicable for all users, and thus also new users.
userSessionID	The userSessionID is a unique value for each session. This does not contribute in any way when making predictions.
avg_price	This attribute is highly correlated to the binned version of this attribute. Using this when creating the model will result in a model with an extremely high accuracy based only on the avg_price.

Data Partitioning and Modelling

The remaining data is partitioned into a train and a test set with a size ratio of 60/40. The training data set is used to train the decision tree model. The trained model is then applied to the test set to predict whether a user is a HighRoller or a PennyPincher.
To avoid overtraining it is important that the model is trained on the train set and not on the test set.

The partitioning of the data set is done with sampling. With sampling it is important to set the random seed because this ensures that every time the partitioning node is executed the partitions are the same which makes the results reproducible.

A screenshot of the resulting decision tree is provided below.



Evaluation

The decision tree has been applied to the test set. Its performance can be visualized by means of the confusion matrix, as shown below.

Row ID	PennyP...	HighRoller
PennyPincher	308	27
HighRoller	38	192

Below the values in the confusion matrix are described somewhat more.

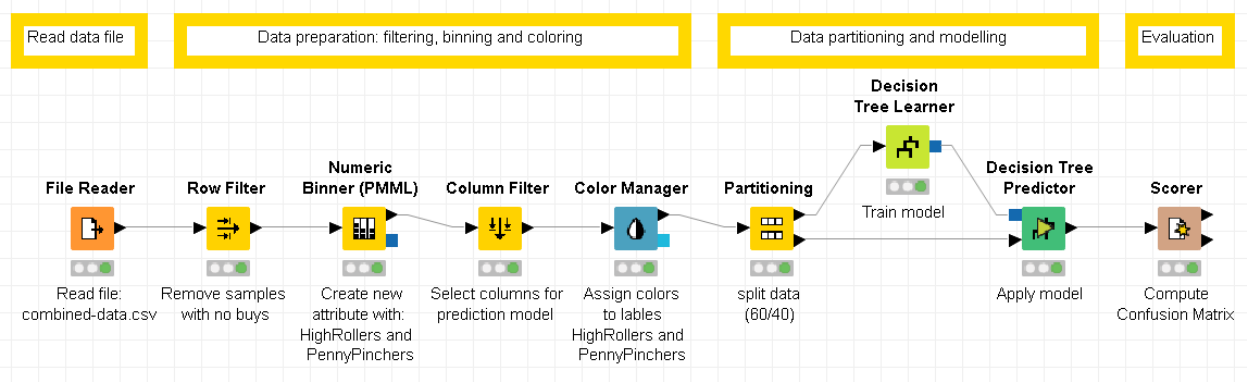
308 True Positives
192 True Negatives +
500 Correctly predicted values

38 False Positives
27 False Negatives +
65 Wrongly predicted values

The overall accuracy of the model is the total number of correct predictions divided by the total number of predictions, which is ([500 / 565] * 100 =) 88.496%. Furthermore, it shows that the model predicted 308 of the 346 PennyPinchers correctly, this is 89%. The model also predicted 192 of the 219 HighRollers correctly, which is of 88%.

Analysis Conclusions

The final KNIME workflow is shown below:



What makes a HighRoller vs. a PennyPincher?

Apparently the platform type is the best estimator for predicting if someone is a HighRoller or a PennyPincher. Most people with an i-phone are HighRollers while almost all Linux users are PennyPinchers. From the Windows and Android users also most are PennyPinchers, while Mac users have a relative high amount of HighRollers.

The table below contains two recommendations based on the conclusions to increase the revenue.

Specific Recommendations to Increase Revenue
1. Target i-phone and Mac users mainly with ads for expensive items (> $5).
2. Target Android, Linux and Windows users mainly with ads for cheaper items (≤ $5).

Clustering with Spark

Clustering with Spark

In this appendix K-means clustering is performed on three attributes of the Catch the Pink Flamingo dataset. Loading the data set, selecting the features and training the K-means clustering model is done in Apache Spark.

Attribute Selection

The clustering analysis will be performed on three attributes. In the table below the three attributes are described and an argumentation for their selection is given.

Attribute	**Rationale for Selection**
“teamLifeSpan”	The lifespan of a team, which is the teamEndTime – teamCreationTime. In other words, how old is the team or how long did the team last?
“strengthLevelRatio”	The ratio between the strength of a team and the teamLevel. The higher this ratio the higher the “success” of this team.
“revenuePerTeam”	The total amount of money spent per team.

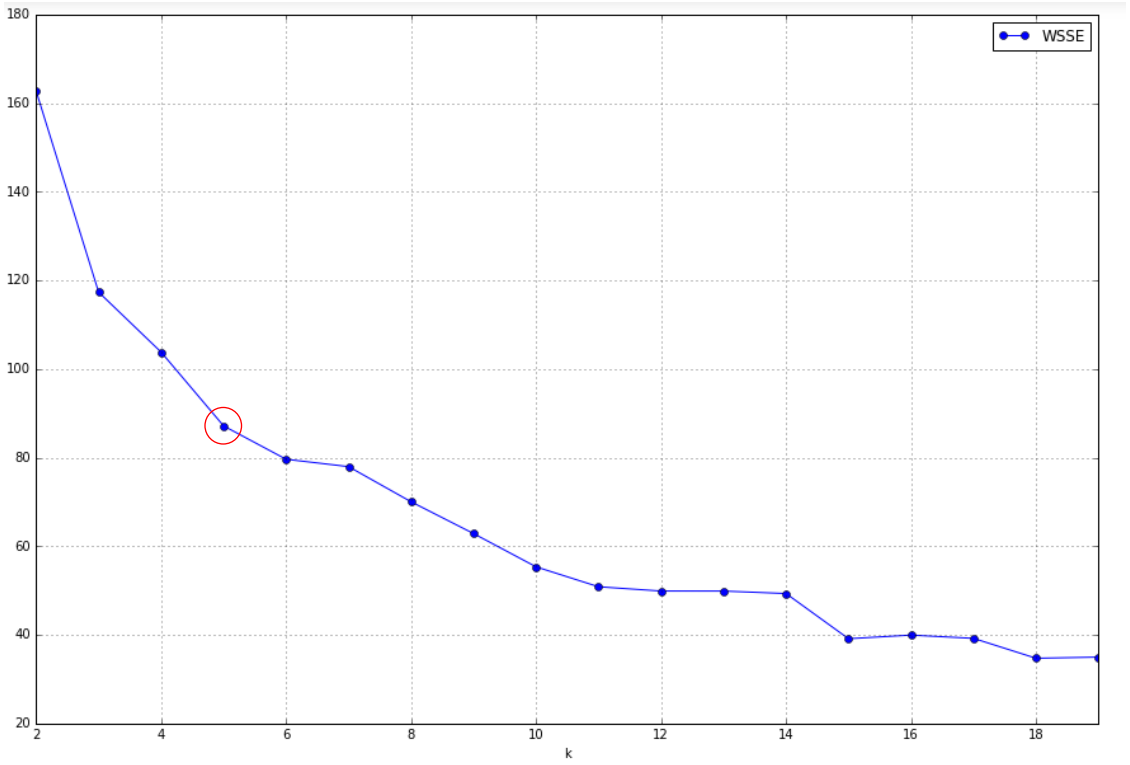
Training Data Set Creation

The remaining dataset has dimensions 169 rows x 3 columns. The first five rows from this dataset are presented below.

```
+------------+------------------+--------------+
|teamLifeSpan|strengthLevelRatio|revenuePerTeam|
+------------+------------------+--------------+
|251936258640|    0.913882906472|           0.0|
|251936474923|    0.276723269022|         141.0|
|251936361720|    0.867932831197|           0.0|
|251936240417|    0.636515436129|           0.0|
|251936434312|    0.836061494696|         710.0|
+------------+------------------+--------------+
only showing top 5 rows
```

Next, the number of clusters (k) is determined. This is done based on the elbow-method and the weighted sum squared error (WSSE). In the graph below the WSSSE is provided for each number of clusters from 2 up to 20.

As one can see, there is no clear elbow visible. Therefore, a trade-off is sought between the reduction in WSSE and increase in clusters. I selected k = 5, because, from this point on the number of cluster must increase rather much in relation to the decrease in the WSSE.



Cluster Centres

Before training the K-means model, the data set has been split in a training and test set. Below a summary of the train data set is provided.

	0	1	2	3	4
summary	count	mean	stddev	min	max
teamLifeSpan	169	1.535470142469586E11	1.2327739042364684E11	0	251936558792
strengthLevelRatio	169	0.31185658308037445	0.32222852216250214	0.0	0.994851162257
revenuePerTeam	169	126.66863905325444	190.1401338453616	0.0	880.0

The code to train the K-means model with k = 5 is provided below together with the code to retrieve the cluster centres.

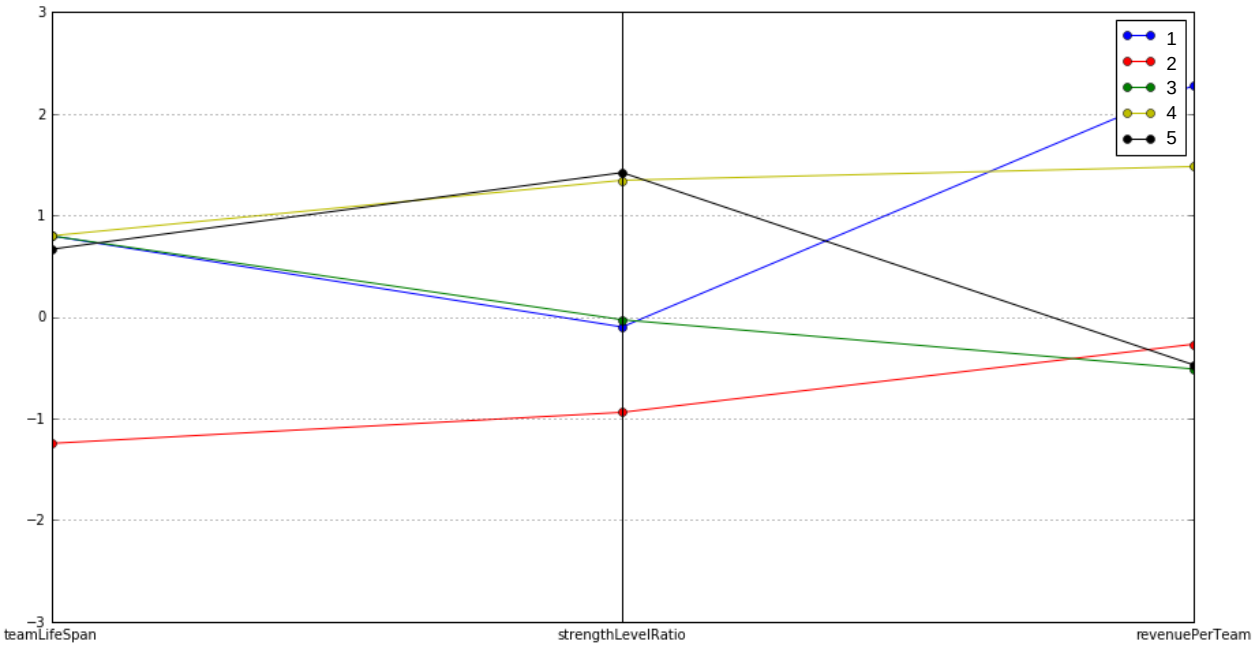
```
kmeans = KMeans(k=5, seed=1)
model = kmeans.fit(scaledData)
transformed = model.transform(scaledData)
```

```
centers = model.clusterCenters()
centers
```

In the table below the cluster centres are provided.

Cluster #	Centre
1	0.79811356, -0.09910393, 2.27760802
2	-1.24554073, -0.93950776, -0.26919232
3	0.79811345, -0.02803835, -0.51249789
4	0.79811367, 1.34517521, 1.47997576
5	0.66626475, 1.42078479, -0.47074411

For a better understanding of the clusters, they are visualized in the graph below.



These clusters can be differentiated from each other as follows:

Cluster 1 contains older teams (an above average teamLifeSpan) with a normal success rate (strengthLevelRatio) and very high revenuePerTeam. These teams are willing to do in-game purchases.

Cluster 2 is the most unique cluster. It contains young teams (a low teamLifeSpan) with a below average success rate (strengthLevelRatio) and a just below average revenuePerTeam. These teams are new and not jet having much success, and maybe therefore do not spend that much on in-game purchases.

Cluster 3 is similar to cluster 1, except that the revenuePerTeam is below average. So, it consists of old teams (an above average teamLifeSpan) with a normal success rate (strengthLevelRatio)

and a low revenuePerTeam. These teams are doing average and are not jet willing to spend much on in-game purchases.

Cluster 4 consists of older teams (a high teamLifeSpan) with a high success rate (strengthLevelRatio) and a very high revenuePerTeam. These teams are willing to do in-game purchases.

Cluster 5 is similar to cluster 4, except that the revenuePerTeam is below average. It consists of old teams (a high teamLifeSpan) with an high success rate and a below average revenuePerTeam. These teams perform well but are not so willing to do in-game purchases.

Recommended Actions

Based on the results two recommendations are made in order to increase the revenue of the Catch the Pink Flamingo game.

Action Recommended	**Rationale for the action**
Increase ads to users of teams who exist longer than average and have a below average strengthLevelRatio	It was seen that teams that exist longer than average but have a lower than average strengthLevelRatio are willing to do in-game purchases. By targeting the users in these teams with high priced items revenue will increase.
Give discount on high priced items to users of new teams with a low strengthLevelRatio.	These teams have the lowest strengthLevelRatio, which indicate that these teams have low success in relation to their level. Mostly when people are not that good in a game they tend to like it less and therefore don't spend much on in-game purchases. By helping them with discounts on expensive in-game purchases they will get better in the game and like it more and eventually spend more.

Graph Analytics with Neo4j

Graph Analytics with Neo4j

Modeling Chat Data using a Graph Data Model

This graph model contains the chat data of the Catch the Pink Flamingo game. It basically includes the relations between the users of the game, the different teams formed by the users, and the chats between all users and between users in a team. Users can join and leave teams, create chats, respond to chats and, mention other users in a chat.

Creation of the Graph Database for Chats

The graph database is constructed from six .csv files. Below the characteristics of these files are described:

File 1	**chat_create_team_chat.csv**	
Description	A line is added to this file when a player creates a new chat with their team.	
Columns	userid	id of the user
	teamid	id of the team
	TeamChatSessionID	id of the chat session
	timestamp	datetime of creation team chat

File 2	**chat_item_team_chat.csv**	
Description	A line is added to this file when a player, who is part of a team, creates a chat item.	
Columns	userid	id of the user
	TeamChatSessionID	id of the chat session
	chatitemid	id of the chat item
	timestamp	datetime of creation chat item

File 3	**chat_join_team_chat.csv**	
Description	A line is added when a player joins a team.	
Columns	userid	id of the user
	TeamChatSessionID	id of the chat session
	timestamp	datetime of moment user joins team

File 4	**chat_leave_team_chat.csv**	
Description	A line is added when a player leaves a team.	
Columns	userid	id of the user
	TeamChatSessionID	id of the chat session
	timestamp	datetime of moment user leaves team

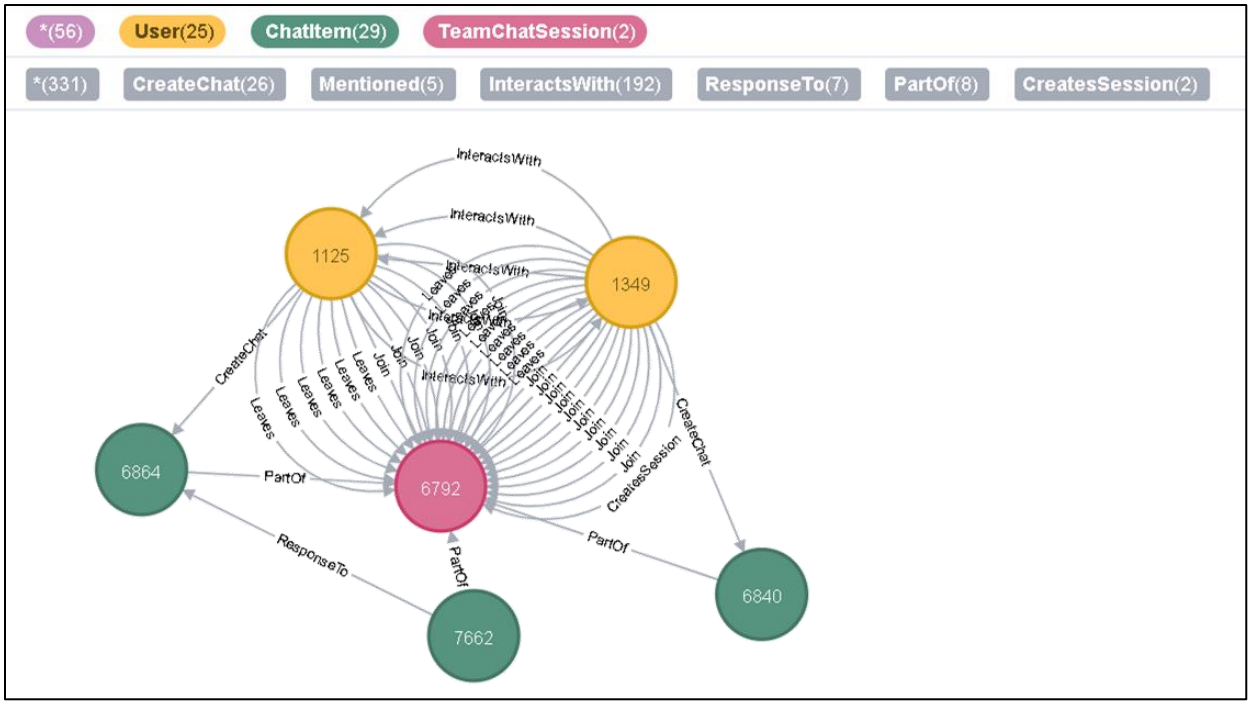
File 5	**chat_mention_team_chat.csv**	
Description	A line is added when a player is mentioned in a chat item	
Columns	ChatItem	id of the chat item
	userid	id of the user
	timeStamp	datetime of moment user is mentioned

File 6	**chat_respond_team_chat.csv**	
Description	A line is added to this file when player responds to a chat post by another player.	
Columns	chatid1	id of the chat of the responding user
	chatid2	id of the chat which is responded to
	timestamp	datetime of moment response is posted

Each file is loaded separately into the graph database. By loading the files Nodes and/or relationships are created. A sample LOAD command in Cypher language is show below:

```
LOAD CSV FROM "file:/chat-data/chat_item_team_chat.csv" AS row
MERGE (u:User {id: toInteger(row[0])})
MERGE (c:TeamChatSession {id: toInteger(row[1])})
MERGE (i:ChatItem {id: toInteger(row[2])})
MERGE (u)-[:CreateChat{timeStamp: row[3]}]->(i)
MERGE (i)-[:PartOf{timeStamp: row[3]}]->(c)
```

The first line loads the .csv file and works through it row by row. Line two to four creates the “User”, “TeamChatSession” and “ChatItem” nodes and line five and six create the “CreateChat” and “PartOf” relations between these nodes. A part of the created graph database with most of the nodes and relations is shown in the screenshot below.

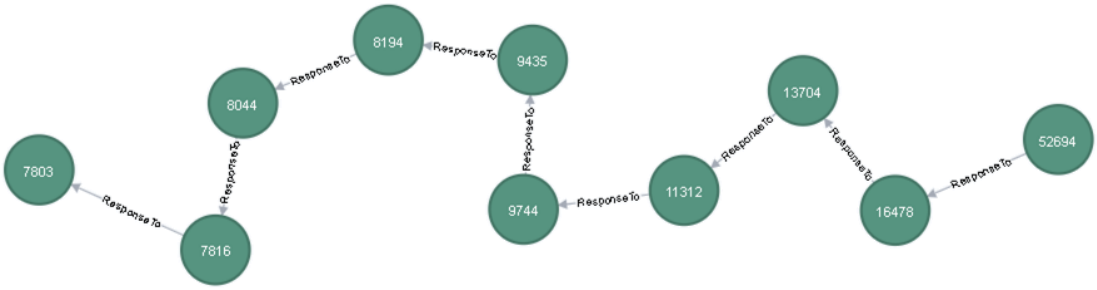


Finding the longest conversation chain and its participants

The longest conversation chain and its participants can be found using two queries. First, the longest conversation chain is determined using the “ResponseTo” relation, see code below.

```
MATCH p=(a)-[:ResponseTo*]->(c)
RETURN length(p) ORDER BY length(p) DESC LIMIT 1
```

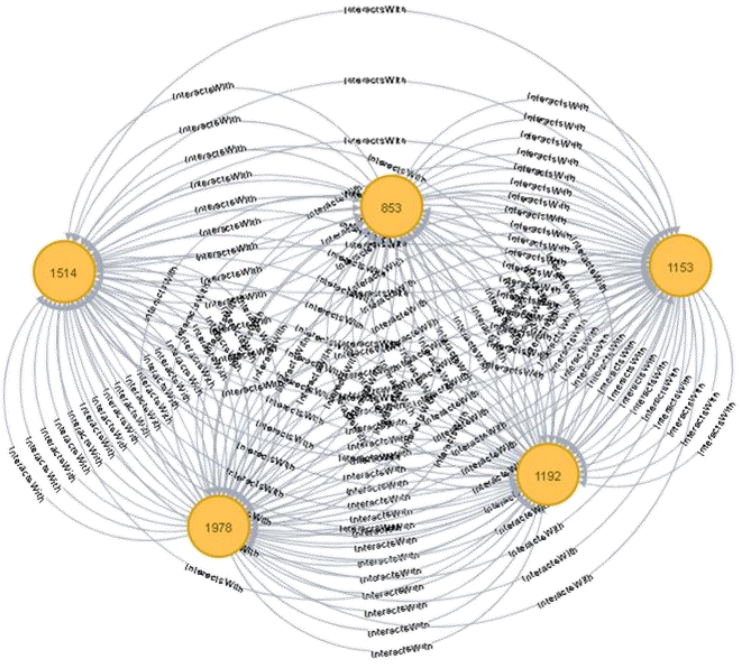
This query result in a path length of 9, which means 9 “ResponseTo” relations and thus 1 original chat an 9 reply chats. The path is show in the figure below.



Next, the path length of the longest conversation chain is used to find all participating users, see code below.

```
MATCH path = (a)-[:ResponseTo*9]->(c)
WITH NODES(path) AS ChatItems
MATCH (u:User) - [:CreateChat] -> (i:ChatItem)
WHERE i IN ChatItems
RETURN count(distinct(u))
```

This query results in a number of distinct users of 5. The users are shown in the figure below.



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

The relation between the top 10 chattiest users and the top 10 chattiest teams can be investigated using three queries. First, the top 10 chattiest users are determined using the code below.

```
MATCH (u:User) - [:CreateChat] -> (i:ChatItem)
RETURN u.id AS UserId, count(i) AS nrOfChats ORDER BY count(i) DESC LIMIT 10
```

The top 3 chattiest users including their number of chats are presented in the table below.

Chattiest Users

Users	Number of Chats
394	115
2067	111
1087	109

In the second step the top 10 chattiest teams are determiner using the code below.

```
MATCH (i:ChatItem) - [e:PartOf] - (c:TeamChatSession) - [:OwnedBy] -> (t)
RETURN t.id AS TeamID, count(i) AS nrOfChats ORDER BY count(i) DESC LIMIT 10
```

The top 3 chattiest teams including their number of chats are presented in the table below.

Chattiest Teams

Teams	Number of Chats
82	1324
185	1036
112	957

Finally, the relation between the top 10 chattiest users and top 10 chattiest teams is investigated using the following code:

```
MATCH (u:User) - [:CreateChat] - (i:ChatItem) - [:PartOf] - (c:TeamChatSession)
- [:OwnedBy] -> (t)
WITH count(i) AS nr, u.id AS top10Users, t.id AS TeamId
ORDER BY count(i) DESC
LIMIT 10
RETURN top10Users, TeamId
```

As show in the tables below, only one of the chattiest users (user 999) is in one of the chattiest teams (group 52).

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

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

"top10Users"	"TeamId"
394	63
2067	7
1087	77
209	7
554	181
999	52
516	7
1627	7
668	89
461	104

How Active Are Groups of Users?

The activity of a group of users can be defined by how mutually interactive this group is. This can be determined by estimating the “density” of the neighborhood of a node.
First of all, a new relation type “InteractsWith” is added. This “InteractsWith” relation will form edges between users who mention other users and users who respond to a chat of another user. In the code below the relations are created between users that mention other users.

```
MATCH (u1:User) - [:CreateChat] - (i:ChatItem) - [:Mentioned] -> (u2:User)
CREATE (u1) - [:InteractsWith] -> (u2)
```

In the code below the relations are created between users who respond to a chat of another user.

```
MATCH (u1:User) - [:CreateChat] - (i:ChatItem) - [:ResponseTo] - (i2:ChatItem)
- [:CreateChat] - (u2:User)
CREATE (u1) - [:InteractsWith] -> (u2)
```

The code above has an undesirable side effect of creating a self-loop if a user has responded to their own chat. These self-loops are deleted using the code below.

```
MATCH (u1) - [r: InteractsWith] -> (u1) DELETE r
```

Now, with these new relations in place, the density of a neighborhood can be determined and expressed as a “clustering coefficient”. This coefficient is determined as follows:

clustering coefficient = e / (k * (k – 1))
With:
e = number of unique interconnections between the nodes in the neighborhood
k = number of neighbors of a node

The density of the neighborhoods will be determined for the top 10 most chattiest users. Their ID's are [394, 2067, 1087, 209, 554,1627, 516, 999, 668, 461].

The code and a description is provided below.

```
MATCH (u1:User) - [r1:InteractsWith] -> (u2:User)
WHERE u1.id <> u2.id
AND u1.id IN [ 394, 2067, 1087, 209, 554, 1627, 516, 999, 668, 461 ]
WITH u1, collect(u2.id) AS neighbors, count(distinct(u2)) AS k
MATCH (u3:User) - [r2:InteractsWith] -> (u4:User)
WHERE (u3.id <> u4.id) AND (u3.id IN neighbors) AND (u4.id IN neighbors)
WITH u1, u3, u4, k,
CASE
    WHEN count(r2) > 0 THEN 1
    ELSE 0
END AS value
RETURN u1.id as UserId, sum(value) * 1.0 /(k * (k - 1)) AS coefficient
ORDER BY coefficient DESC
```

Line 1 get all users that are connected to other user through a "InteractsWith" edge.
Line 2 – 3 filter users that are not connected to themselves and part of the top 10 chattiest users.
Line 4 store the neighbors as a list and calculate the number of neighbors.
Line 5 – 6 get all users that are in the neighbors list and are not connected to themselves.
Line 7 – 11 for each neighbor node calculate the unique connections to other neighbor nodes.
Line 12 – 13 calculate the "clustering coefficient" and return it for each user in the top 10 chattiest users.

Below the three most active users in the top 10 chattiest users based on their cluster coefficient are provided.

Most Active Users (based on Cluster Coefficients)

User ID	Coefficient
209	0.952
554	0.905
1087	0.800