Technical Appendix
Catch the Pink Flamingo Analysis

Varun Gawande
June 2021, Big Data Specialization, Capstone Project

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	Records on every instance an ad was clicked	Timestamp: when exactly was the ad clicked txId: unique ID for every ad click userSessionid: The ID of the active user session when the user clicked the ad teamid: The ID of the team the user belonged to when he clicked the ad userid: The ID of the user who clicked the ad adId: Unique of the ad that was clicked adCategory: Category of the app
buy-clicks.csv	Records every transaction from the game's store	Timestamp: when exactly was item bought txId: unique ID for every transaction userSessionId: The ID of the active user session when the user bought the item team: The ID of the team the user belonged to userid: The ID of the user buyId: ID of the item bought price: Price of the item

users.csv	Records each User joining the game	timestamp: When did user register userId: Unique User ID nick: Chosen Nickname twitter: Twitter Handle dob: Date of Birth country: 2-Letter code of User’s country
team.csv	Record for each team terminated	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	Record added when User joins 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

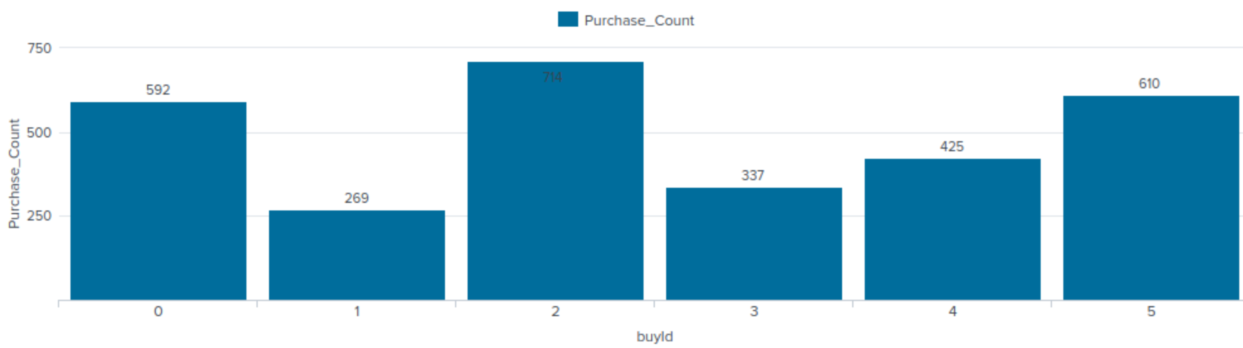
level-events.csv	Record added each time a team starts or finishes a level	timestamp: when the event occurred. eventId: a unique id for the event teamId: the id of the team teamLevel: the level started or completed eventType: the type of event, either start or end
user-session.csv	Record added for every new UserSession. i.e User starts and stops playing the game, or a team moves to the next level	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	Record added 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 clicking userSessionId: the id of the userSession at the time. isHit: denotes if the click was on a flamingo(value 1) or missed(value 0). teamId: the id of the team of the user teamLevel: the current level of the team of the user

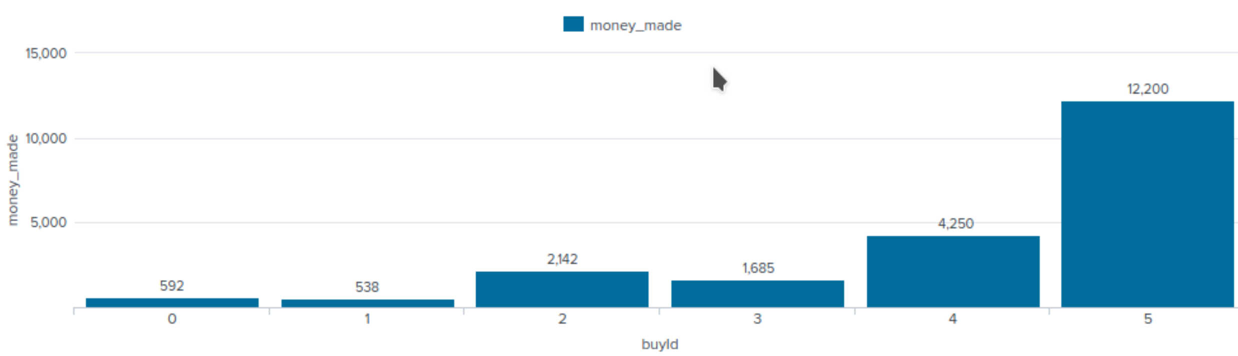
Aggregation

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

A bar chart showing how many times each item is purchased:

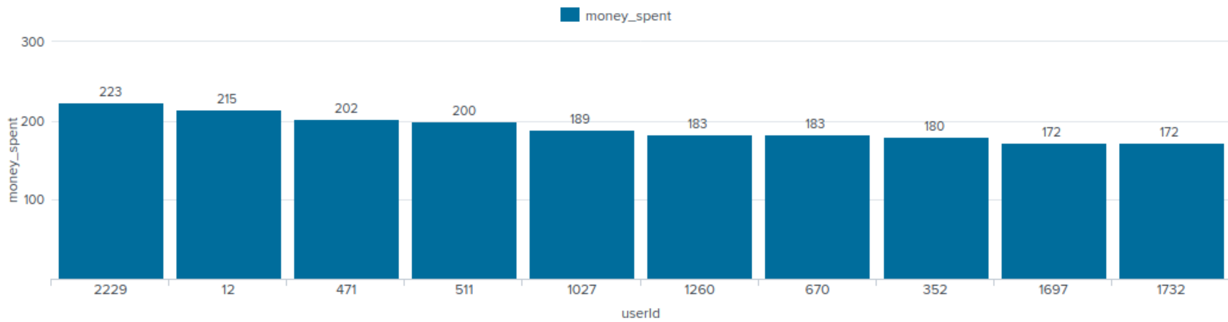


A bar chart showing how much money was made from each item:



Filtering

A bar chart showing the 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	11.596
2	12	iphone	13.068
3	471	iphone	14.503

Data Preparation

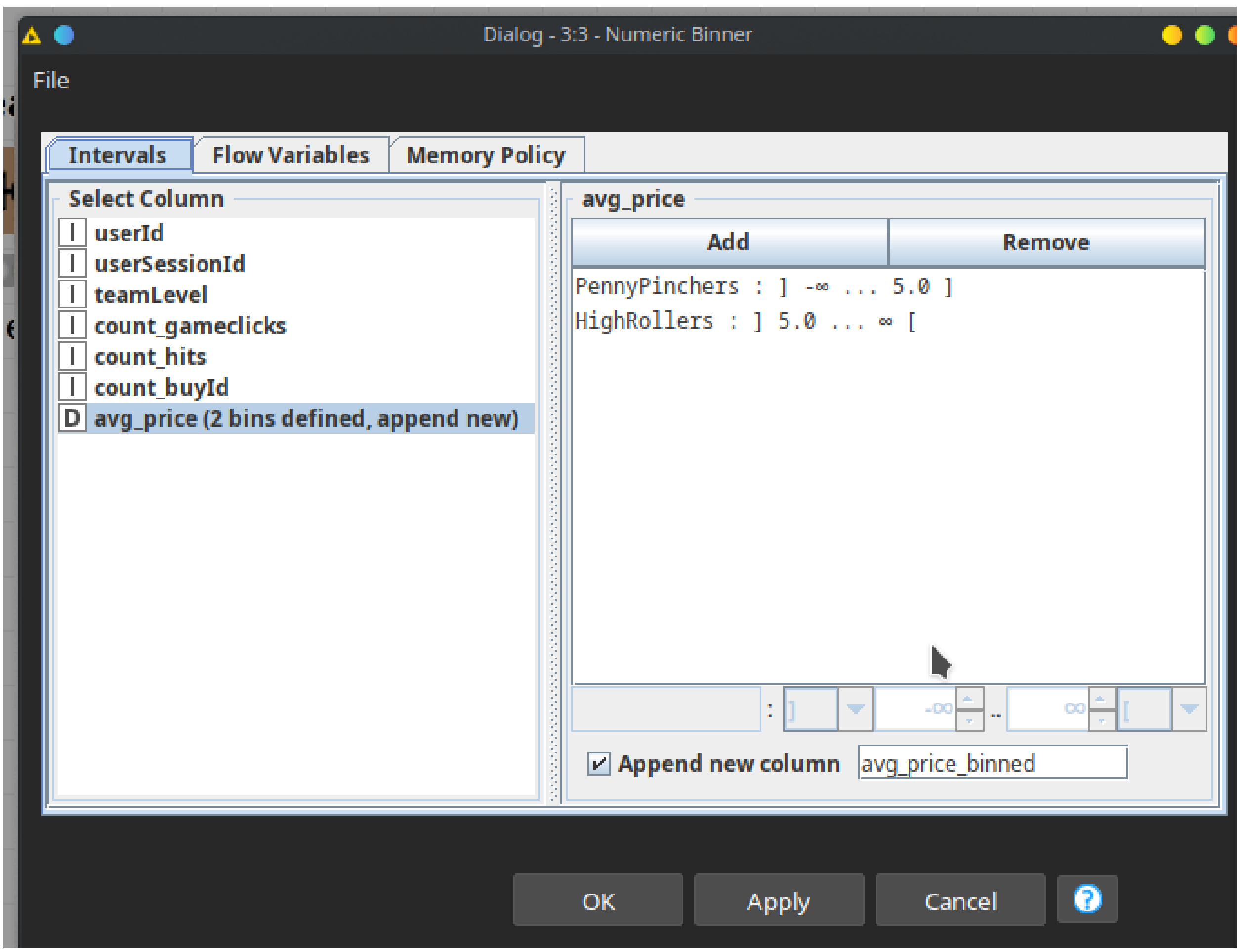
Analysis of combined_data.csv

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:



The table after the new attributes are created:

File Edit Hilite Navigation View

Table "default" - Rows: 1411 | Spec - Columns: 9 | Properties | Flow Variables

Row ID	userId	userSe...	teamLe...	platfor...	count_g...	count_h...	count_b...	avg_price	avg_price_...
Row4	937	5652	1	android	39	0	1	1	PennyPinchers
Row11	1623	5659	1	iphone	129	9	1	10	HighRollers
Row13	83	5661	1	android	102	14	1	5	PennyPinchers
Row17	121	5665	1	android	39	4	1	3	PennyPinchers
Row18	462	5666	1	android	90	10	1	3	PennyPinchers
Row31	819	5679	1	iphone	51	8	1	20	HighRollers
Row49	2199	5697	1	android	51	6	2	2.5	PennyPinchers
Row50	1143	5698	1	android	47	5	2	2	PennyPinchers
Row58	1652	5706	1	android	46	7	1	1	PennyPinchers
Row61	2222	5709	1	iphone	41	6	1	20	HighRollers
Row68	374	5716	1	android	47	7	1	3	PennyPinchers
Row72	1535	5720	1	iphone	76	7	1	20	HighRollers
Row73	21	5721	1	android	52	2	1	3	PennyPinchers
Row101	2379	5749	1	android	62	9	1	3	PennyPinchers
Row122	1807	5770	1	iphone	177	25	2	7.5	HighRollers
Row127	868	5775	1	iphone	54	5	1	10	HighRollers
Row129	1567	5777	1	android	27	4	2	4	PennyPinchers
Row131	221	5779	1	iphone	37	2	1	20	HighRollers
Row135	2306	5783	1	android	67	5	1	1	PennyPinchers
Row137	1065	5785	1	iphone	37	5	2	11.5	HighRollers
Row140	827	5788	1	iphone	75	5	1	20	HighRollers
Row150	1304	5798	1	mac	71	9	2	11.5	HighRollers
Row158	1264	5806	1	linux	81	12	1	5	PennyPinchers
Row159	1026	5807	1	iphone	52	10	1	20	HighRollers
Row163	649	5811	1	linux	51	9	1	1	PennyPinchers
Row169	1958	5817	1	android	40	3	1	20	HighRollers
Row172	1300	5820	1	android	58	1	2	3	PennyPinchers
Row186	178	5834	1	iphone	54	6	1	20	HighRollers
Row196	670	5844	1	iphone	38	3	2	20	HighRollers
Row207	208	5855	1	iphone	32	3	1	20	HighRollers
Row210	157	5858	1	iphone	32	2	1	10	HighRollers
Row212	2221	5860	1	iphone	191	18	2	11.5	HighRollers

The players with a Purchase History have been made into two categories, PennyPinchers and HighRollers. PennyPinchers are those players who have spent 5$ or less on average. HighRollers are those who spend more.

The creation of this new categorical attribute was necessary because It's more insightful to us to be able to tell if a user is a high-spender or not. Instead of trying to predict his average spend, we can bin that amount into two categories and make it a simpler, revealing insight.

Attribute Selection

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

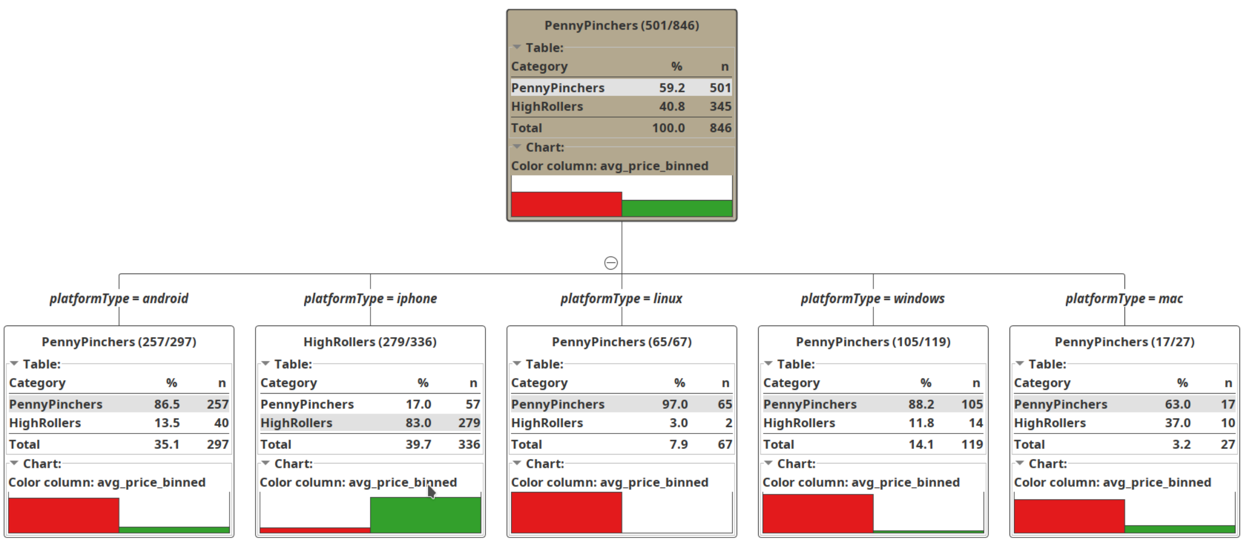
Attribute	**Rationale for Filtering**
userId	Unique Identifiers are not statistically insightful, the value of the ID of a user doesn’t tell us anything about his spending.
userSessionId	Unique Identifiers are not statistically insightful, the value of the ID of a userSession doesn’t tell us anything about his spending.
avg_price	The binned version of this attribute is to be predicted, hence, this is an attribute we can’t use for this job.

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… in order to test the generalisability of the decision tree model, it needs to be tested on unseen data, i.e data it was NOT trained on.

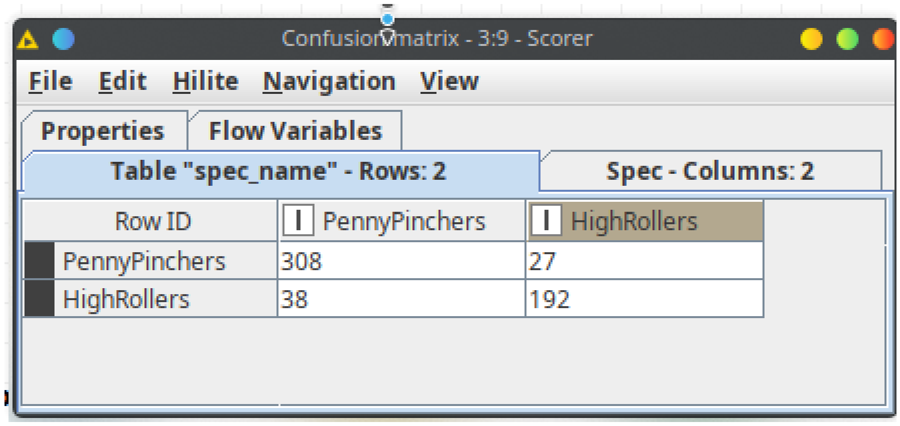
When partitioning the data using sampling, it is important to set the random seed because… the partitions made have to be reproducible for experimental purposes, such as comparisons with other methods or other people's approaches.

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



Evaluation

A screenshot of the confusion matrix can be seen below:



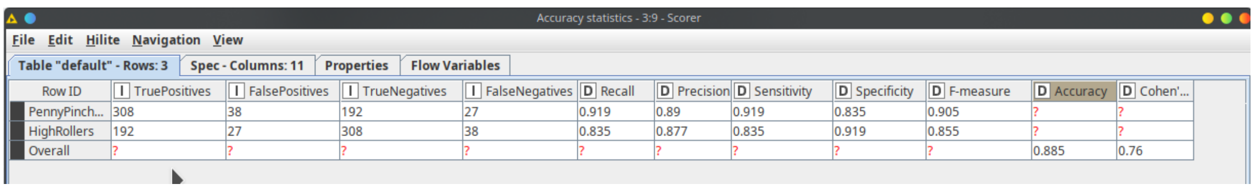
Confusion matrix - 3:9 - Scorer

File Edit Hilite Navigation View

Properties | Flow Variables

Table "spec_name" - Rows: 2 | Spec - Columns: 2

Row ID	PennyPinchers	HighRollers
PennyPinchers	308	27
HighRollers	38	192



Accuracy statistics - 3:9 - Scorer

File Edit Hilite Navigation View

Table "default" - Rows: 3 | Spec - Columns: 11 | Properties | Flow Variables

Row ID	TruePositives	FalsePositives	TrueNegatives	FalseNegatives	Recall	Precision	Sensitivity	Specificity	F-measure	Accuracy	Cohen'...
PennyPinch...	308	38	192	27	0.919	0.89	0.919	0.835	0.905	?	?
HighRollers	192	27	308	38	0.835	0.877	0.835	0.919	0.855	?	?
Overall	?	?	?	?	?	?	?	?	?	0.885	0.76

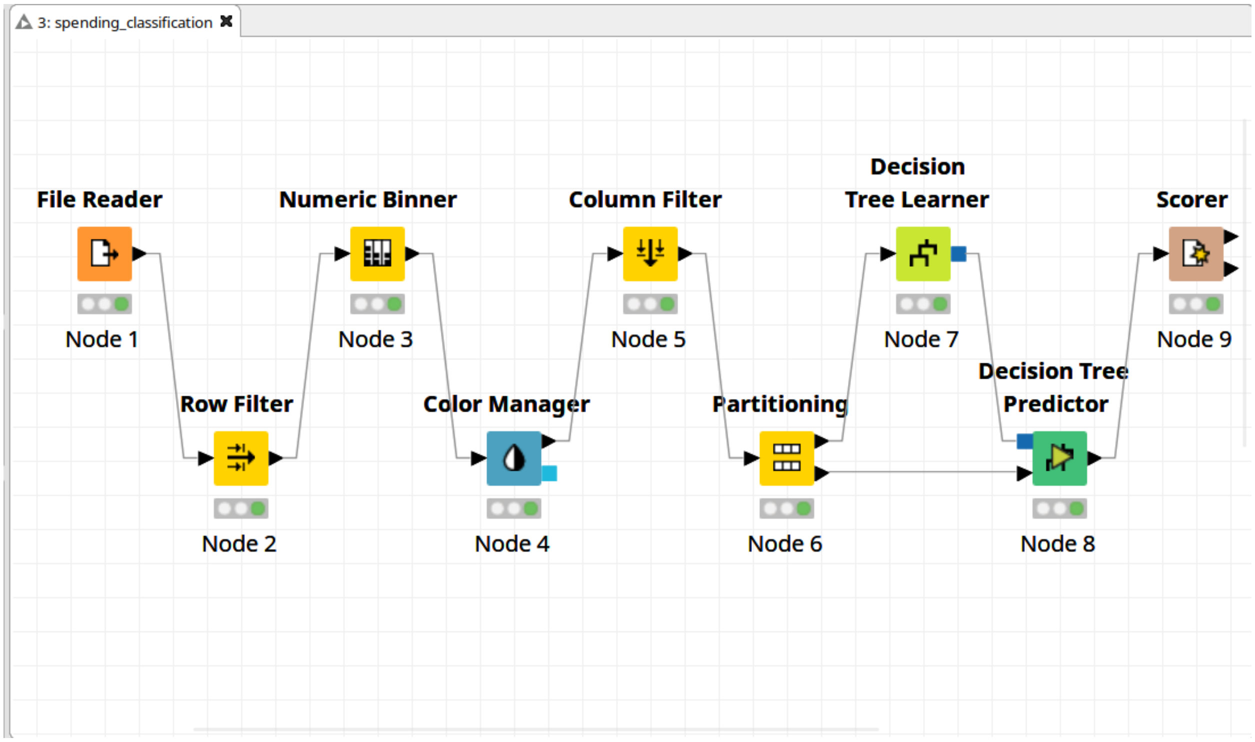
As seen in the screenshot above, the overall accuracy of the model is **88.5%**

For us the Positive Class is HighRoller, Negative is PennyPincher.

True Positive: 192. The number of samples that were correctly classified as HighRollers
False Positive: 27. The number of samples that were incorrectly classified as HighRollers(were actually PennyPinchers)
True Negatives: 308. The number of samples that were correctly classified as PennyPinchers
False Negatives: 38. The number of samples that were incorrectly classified as PennyPinchers(were actually HighRollers)

Analysis Conclusions

The final KNIME workflow is shown below:



What makes a HighRoller vs. a PennyPincher?
The best predictor for a HighRoller seems to be the platform of the players. Players on the iPhone platform are very likely to be HighRollers, the rest are very likely to be PennyPinchers.

Specific Recommendations to Increase Revenue
1. Provide extra perks for the iPhone platform so that new iPhone players feel incentivised to start playing the game.
2. On special occasions, gift in-game purchase bonuses to players. If some players like the bonuses, we'll be able to convert some non-purchasers into purchasers.

Attribute Selection

features_used = ["adsClicks",”totalPurchase”,”teamStrength”,”avg_clicks/hr”]

Attribute	Rationale for Selection
# of ads clicked by user	This attribute captures the ad clicking behaviour of a user
Total purchase amount by user	This attribute captures how much revenue the user generates from the game’s store.
teamStrength of the user	This attribute captures how strong a user’s team is.
Average clicks per hour	This attribute captures how much a user tends to click in one hour, note: Has nothing to do with click accuracy.

Training Data Set Creation

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

```
[86]: df show 5

+--------+--------------+--------------+------------------+
|adClicks|Total Purchase|  teamStrength|      avg_clicks/hr|
+--------+--------------+--------------+------------------+
|      51|         202.0|0.141376627543|1.7583892617449663|
|      41|          16.0|0.320057042827|2.5310880829015545|
|      17|           8.0|0.132214897776|2.0707964601769913|
|      56|          14.0|0.350676528613|1.9941176470588236|
|      39|          20.0|0.340788463107|1.8021582733812949|
+--------+--------------+--------------+------------------+
only showing top 5 rows
```

Dimensions of the final data set: (357, 4)
# of clusters created: 3

Cluster Centers

The code used in creating cluster centers is given below:

Cluster centers formed are given in the table below

Cluster #	**Center** **adClicks** **Total Purchase** **teamStrength** **avg_clicks/hr**
1	0.5657139554913375, 0.5638219939345769, -0.7355293672389471, -0.13910127144715376
2	-1.2507587950221988, -0.5508930923104604, -0.105071834780552, -0.1993074766586675
3	0.5306311917352996, -0.14220019574626339, 0.9800822528171507, 0.35843527965628746

These clusters can be differentiated from each other as follows:

Cluster 1 is a segment of users who have **moderately high ad clicks** and **purchase amount**(high or low comes relatively to most of the population since we are using Standard Deviations). Their **teams are very weak** and they **click slower than normal**.

Cluster 2 is a segment of users that **barely click ads**, and have **fewer purchase amounts** too. Their **teams are slightly on the weaker side**, and so is their **click speed, slow**.

Cluster 3 is a segment of users that have **moderately high ad clicks** but they **purchase lesser** than most, their teams are **extremely strong**, and their **click speeds are fairly high**.

Below you can see the summary of the train data set:

```
[123]: model.clusterCenters foreach println

[0.5657139554913375,0.5638219939345769,-0.7355293672389471,-0.13910127144715376]
[-1.2507587950221988,-0.5508930923104604,-0.105071834780552,-0.1993074766586675]
[0.5306311917352996,-0.14220019574626339,0.9800822528171507,0.35843527965628746]
```

[0.5657139554913375,0.5638219939345769,-0.7355293672389471,-0.13910127144715376]
[-1.2507587950221988,-0.5508930923104604,-0.105071834780552,-0.1993074766586675]
[0.5306311917352996,-0.14220019574626339,0.9800822528171507,0.35843527965628746]

Recommended Actions

Action Recommended	Rationale for the action
Increased Ads and Store Promotions to extremely weak Team Players	It was seen that players from the **weakest teams** tend to click on ads and buy from the store a lot. Hence increasing the ads shown, and promotions such as discounts given to these users, the higher the revenue generated.
Show higher price and higher frequency of ads to players of extremely strong teams.	It was seen that players from the **strongest teams** tend to click on ads a lot. Hence increasing the ads shown or showing more revenue generating ads to these users, the revenue generated could be increased.

Graph Analytics

Modeling Chat Data using a Graph Data Model

The behaviour of users and their chats is modelled via graphs:
Users create TeamChatSessions, TeamChatSessions are owned by Teams, other Users Join or Leave TeamChatSessions, Users create ChatItems which are part of TeamChatSessions, ChatItems can be responses to other ChatItems, ChatItems may mention Users.

Creation of the Graph Database for Chats

Describe the steps you took for creating the graph database. As part of these steps

i) The schema of the 6 CSV files

Chat_create_team_chat:
userid, teamid, TeamChatSessionID, timestamp

Chat_item_team_chat:
userid, teamchatsessionid, chatitemid, timestamp

Chat_join_team_chat:
userid, TeamChatSessionID, teamstamp

Chat_leave_team_chat:
userid, teamchatsessionid, timestamp

Chat_mention_team_chat:
ChatItem, userid, timeStamp

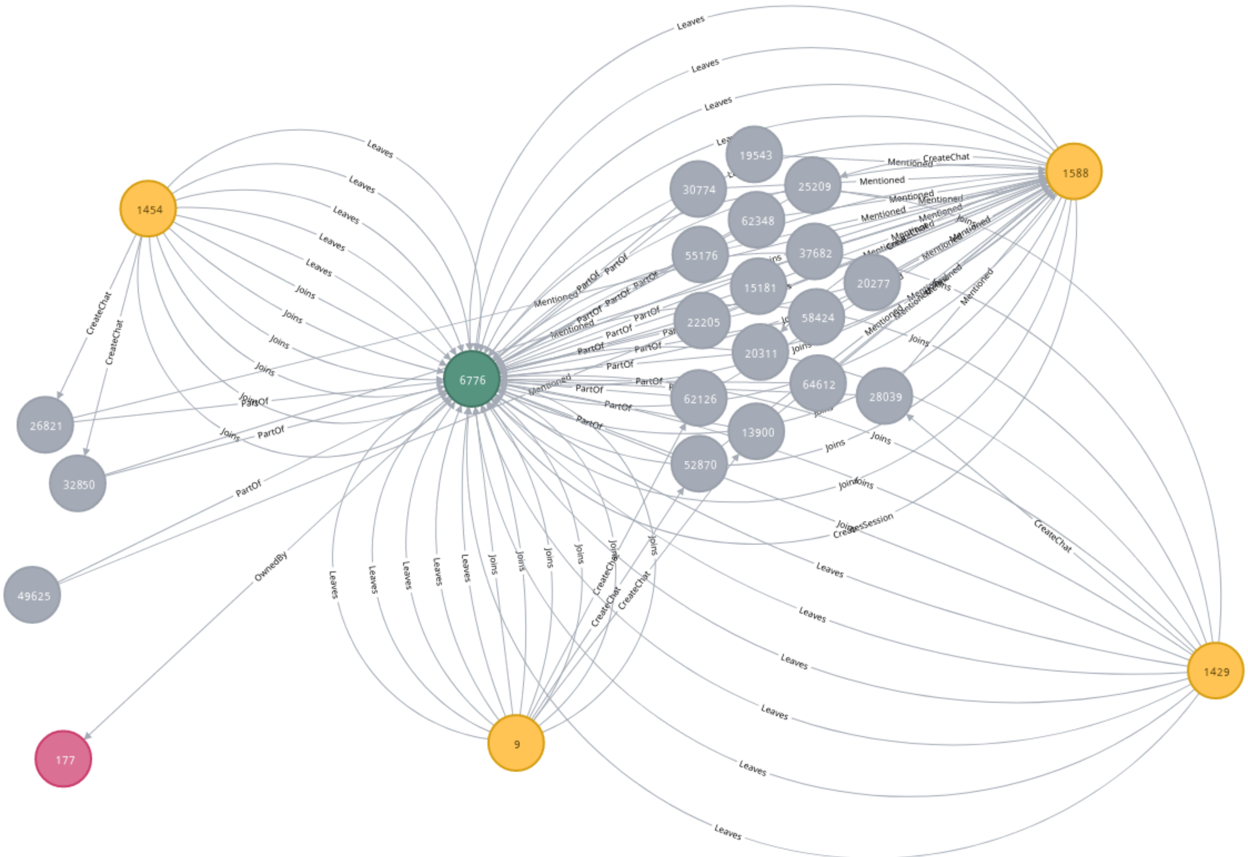
Chat_respond_team_chat:
chatid1, chatid2,timestamp

ii) Explain the loading process and include a sample LOAD command

Load every row of the input CSVs as a 'row'. Create nodes or edges with required property values by indexing the values inside the row. For example

```
LOAD CSV FROM "file:/capstone_chat_data/chat_leave_team_chat.csv" AS row
MERGE (u:User {id: toInteger(row[0])})
MERGE (c:TeamChatSession {id: toInteger(row[1])})
MERGE (u)-[:Leaves{timeStamp: row[2]}]→(c)
```

iii) Present a screenshot of some part of the graph you have generated. The graphs must include clearly visible examples of most node and edge types.



Finding the longest conversation chain and its participants

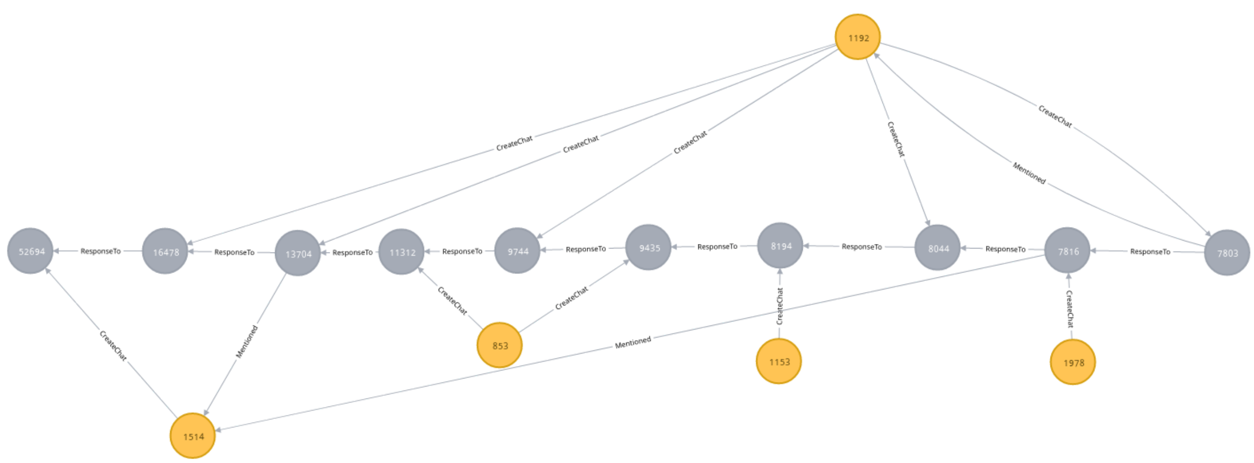
The longest Chain: 9 Responses after the 1 Original ChatItem

Find all paths between all ChatItems, let the maximum length be max_length. Find the path which is as long as max_length. Store its' ChatItem nodes in list named items. Return all distinct paths where User creates ChatItem that is in the items list.

Query:

```
MATCH p=(i1:ChatItem)-[:ResponseTo*]→(i2:ChatItem)
WITH max(length(p)) as max_length
MATCH p=(i1:ChatItem)-[:ResponseTo*]→(i2:ChatItem) WHERE length(p) = max_length
WITH [i in nodes(p)] as items
MATCH path=(u:User)-[:CreateChat]→(i:ChatItem)
WHERE i IN items
RETURN DISTINCT path
```

Longest Path:



The ChatItems of the Longest Path are marked in Grey, while the involved Users are Marked in Yellow. The no. of Users involved is 5.

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

Chattiest Users

Match User nodes with CreateChat edge, for every such node, find the count of CreateChat edges, Order them by Count but Descending, Limit our results to the top 3.

```
MATCH (u:User)-[c:CreateChat]→()
RETURN u.id as UserID, COUNT(c) as Chats
ORDER BY Chats DESC
LIMIT 3
```

Users	Number of Chats
394	115
2067	111
1087	109

Chattiest Teams

Match ChatItems that should be PartOf TeamChatSessions which should be OwnedBy Teams. For every such Team, find the count of ChatItems, Order them by Count but Descending, Limit our results to the top 3.

```
MATCH (i:ChatItem)-[:PartOf]→(:TeamChatSession)-[:OwnedBy]→(t:Team)
RETURN t.id as TeamID, COUNT(i) as Chats
ORDER BY Chats DESC
LIMIT 3
```

Teams	Number of Chats
82	1324
185	1036
112	957

We found that **One** of the top 10 Chattiest Users was in the top 10 Chattiest Teams.

```
MATCH (u:User)-[c:CreateChat]->()
WITH u, COUNT(c) as UserChats
ORDER BY UserChats DESC LIMIT 10
WITH [u.id] as ChattiestUsers

MATCH (u:User)-[:CreateChat]->(:ChatItem)-[:PartOf]->(:TeamChatSession)-[:OwnedBy]->(t:Team)
WHERE u.id IN ChattiestUsers
  AND t.id IN [82,185,112,18,194,129,52,136,146,81]
return DISTINCT u.id AS User, t.id as Team
```

"User"	"Team"
999	52

Only User 999 is a chatty User belonging to a Chatty Team(i.e Team 52).

How Active Are Groups of Users?

We will construct the neighborhood of users. In this neighborhood, we will connect two users if

- One user mentioned another user in a chat
- One user created a chatItem in response to another user's chatItem

Creating New Edges:

For the first condition, this query would have the following structure:

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

Use the same logic to create the query statement for the second condition. This query will also have the form

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

So after the first two steps we need to eliminate all self loops involving the edge "Interacts with". This will take the form:

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

Part 1:

For Every TOP 10 Chatty User, we find the Neighbours and No. of Neighbours:

```
MATCH (u:User)-[c:CreateChat]→()
WITH u, COUNT(c) as Chats
ORDER BY Chats DESC LIMIT 10
WITH [u] as ChattiestUsers

//Getting the neighbours of all Users and the count
MATCH (u1:User)-[:InteractsWith]→(u2:User)
WHERE u1 in ChattiestUsers
WITH u1.id AS UserID, COLLECT(DISTINCT u2.id) AS Neighbours
RETURN UserID, Neighbours, SIZE(Neighbours) AS k
```

Output:

"UserID"	"Neighbours"	"k"
394	[1997,1012,2011,1782]	4
2067	[697,1672,63,209,1627,516,1265,2096]	8
1087	[426,772,929,1879,1311,1098]	6
209	[516,63,2067,1672,2096,1265,1627]	7
554	[2018,1687,1010,1959,1096,1412,610]	7
1627	[2096,1672,2067,63,516,209,697,1265]	8
516	[63,209,1672,2067,2096,1627,1265]	7
999	[1554,1587,778,1056,1606,1601,1398,1506,1839]	9
668	[2034,648,698,458]	4
461	[1675,1482,482]	3

Part 2:

Find Interecting Users such that both belong in Neighbours list, Finding Coefficient.

```
// Getting TOP 10 Chattiest Users
MATCH (u:User)-[c:CreateChat]->()
WITH u, COUNT(c) as Chats
ORDER BY Chats DESC LIMIT 10
WITH [u] as ChattiestUsers

// Getting the neighbours of TOP 10 Users and the count
MATCH (u1:User)-[:InteractsWith]->(u2:User)
WHERE u1 in ChattiestUsers
WITH u1.id AS UserID, COLLECT(DISTINCT u2.id) AS Neighbours
WITH UserID, Neighbours, SIZE(Neighbours) AS k

// Find Interecting Users
MATCH (u1:User)-[:InteractsWith]->(u2:User)
// Such that both belong in Neighbours list
WHERE u1.id IN Neighbours AND u2.id IN Neighbours
// For every valid combination of a User and its two neighbours,
// Value is 1 if neighbours have interacted atleast once, k is no. of Neighbours
WITH DISTINCT UserID, u1.id AS N1, u2.id as N2, CASE WHEN (u1)-[:InteractsWith]->(u2) THEN 1 ELSE 0 END AS VALUE, k

RETURN UserID,SUM(VALUE) as NUM,k*(k-1) AS DENUM, SUM(VALUE)/(k*(k-1)*1.0) AS ClusteringCoefficient
ORDER BY ClusteringCoefficient DESC
```

Output:

"UserID"	"NUM"	"DENUM"	"ClusteringCoefficient"
668	12	12	1.0
461	6	6	1.0
209	38	42	0.9047619047619048
516	38	42	0.9047619047619048
394	10	12	0.8333333333333334

Most Active Users (based on Cluster Coefficients)

User ID	Coefficient
668	1.0
461	1.0
209	0.9047619047619048