

BIG DATA SPECIALIZATION - CAPSTONE PROJECT

TECHNICAL APPENDIX

Ken Wood

December 14, 2020

Table of Contents

Data Exploration	3
Aggregation	5
Filtering	7
Data Preparation	8
Analysis of combined_data.csv	8
Attribute Deletion	9
Data Partitioning and Modeling	9
Evaluation	10
Analysis Conclusions.....	11
Clustering Analysis	12
Training Data Set Creation	12
Cluster Centers.....	13
Recommended Actions.....	15
Graph Analytics.....	16
Modeling Chat Data using a Graph Data Model	16
Creation of the Graph Database for Chats	16
Finding the longest conversation chain and its participants	18
Top 10 chattiest users and top 10 chattiest teams... ..	19
How Active Are Groups of Users?.....	20
Most Active Users (based on Cluster Coefficients)	22

Data Exploration

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.

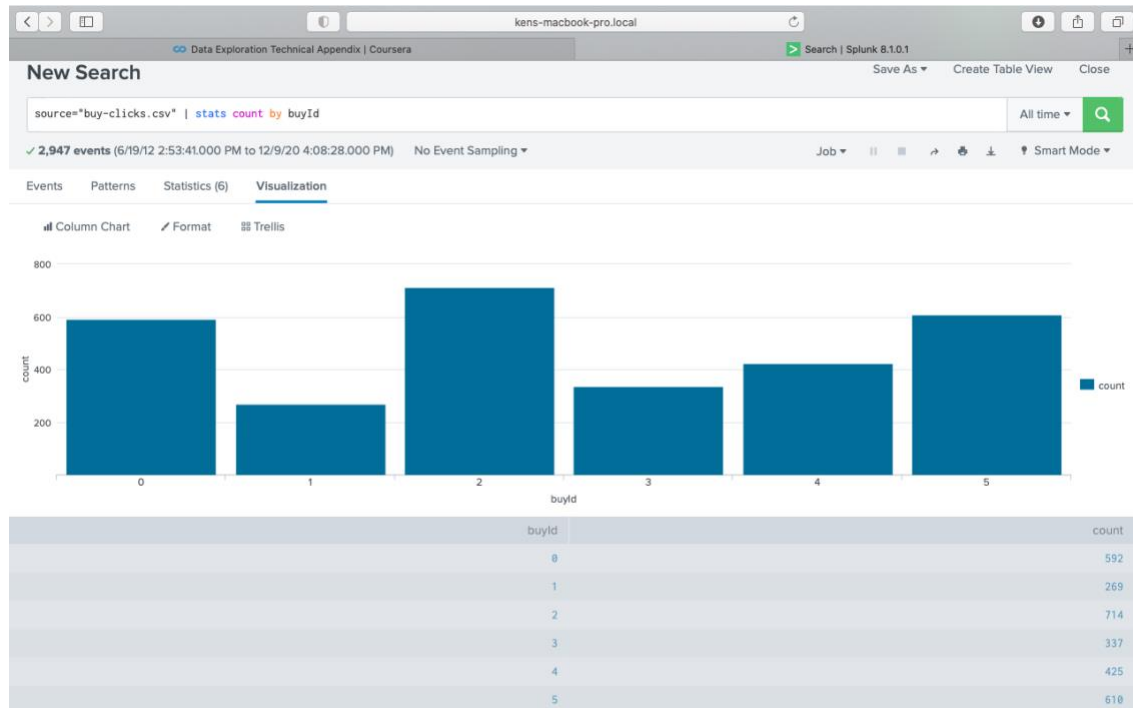
File Name	Description	Fields
team.csv	This file contains a line for each team terminated in the game.	teamId: the id of the team name: the name of the team teamCreationTime: the timestamp when the team was created teamEndTime: the timestamp when the last member left the team strength: a measure of team 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.

File Name	Description	Fields
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

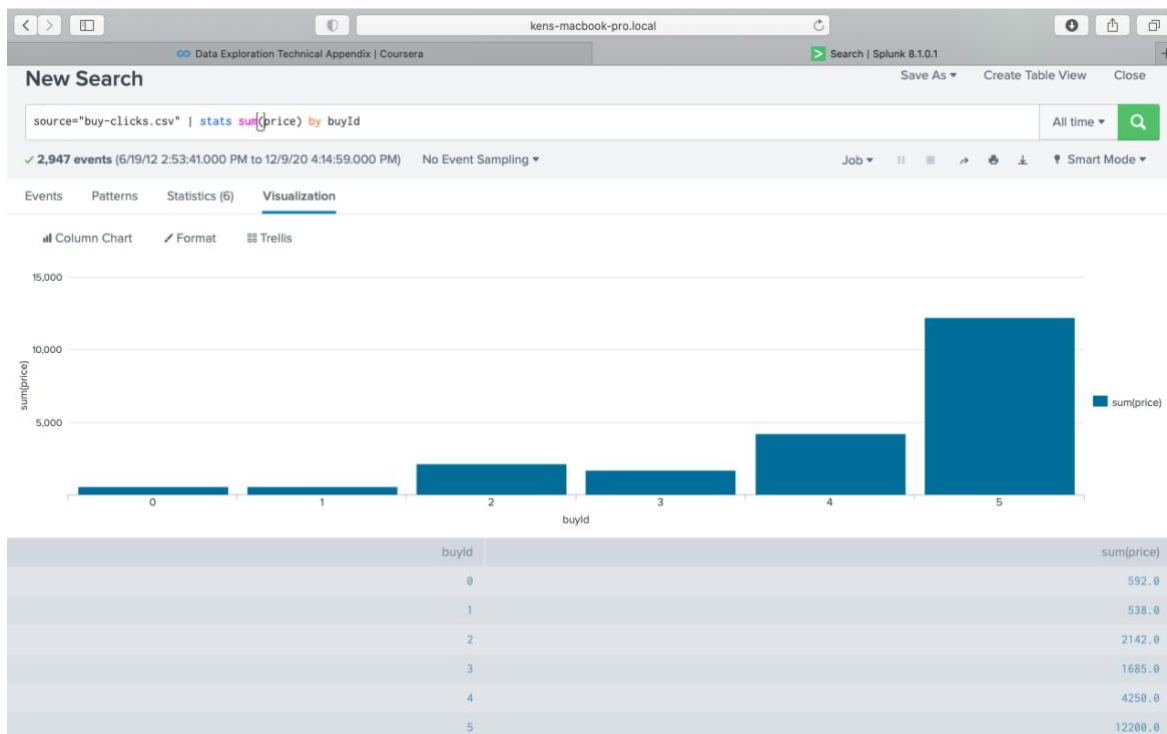
Aggregation

Amount spent buying items	source="buy-clicks.csv" stats sum(price) \$21,407														
Number of unique items available to be purchased	<div># buyId</div> <hr/> <div>6 Values, 100% of events</div> <table> <tr> <th>Values</th><th>%</th></tr> <tr> <td>2</td><td>24.7%</td></tr> <tr> <td>5</td><td>19.9%</td></tr> <tr> <td>0</td><td>19.8%</td></tr> <tr> <td>4</td><td>16.6%</td></tr> <tr> <td>3</td><td>10.4%</td></tr> <tr> <td>1</td><td>8.6%</td></tr> </table>	Values	%	2	24.7%	5	19.9%	0	19.8%	4	16.6%	3	10.4%	1	8.6%
Values	%														
2	24.7%														
5	19.9%														
0	19.8%														
4	16.6%														
3	10.4%														
1	8.6%														

A histogram showing how many times each item is purchased:

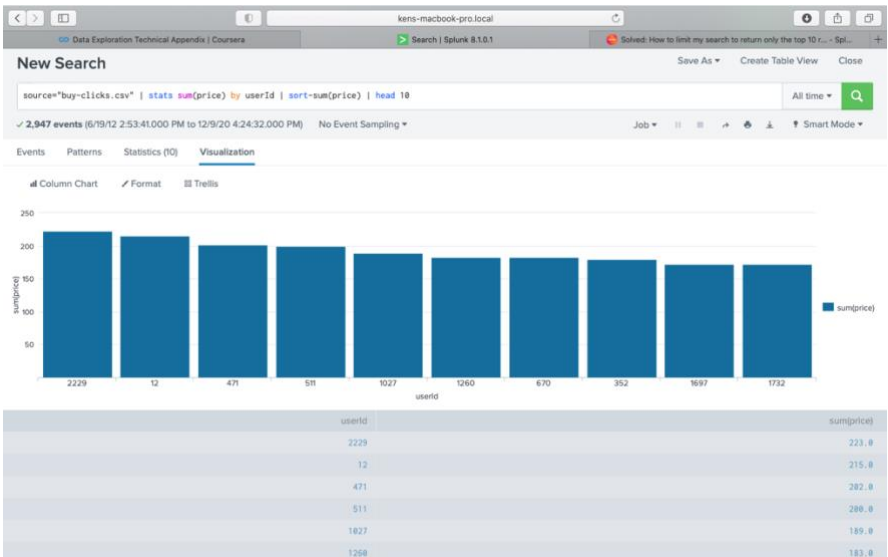


A histogram showing how much money was made from each item:



Filtering

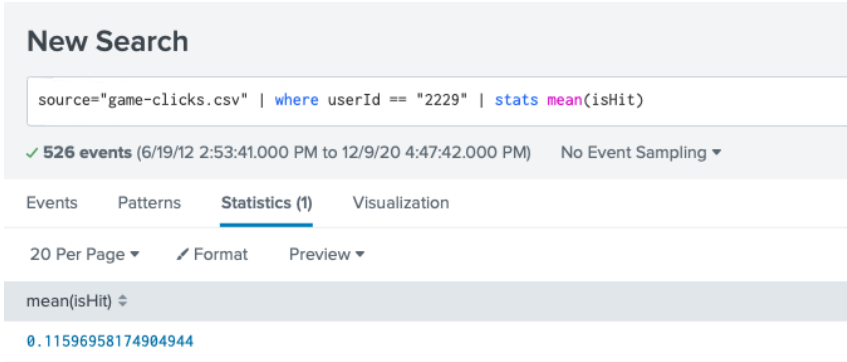
A histogram showing total amount of money spent by the top ten users (ranked by how much money they spent).



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	0.11596958174904944
2	12	iPhone	0.13068181818181818
3	471	iPhone	0.13496280552603612

Example Splunk screenshot:



Data Preparation

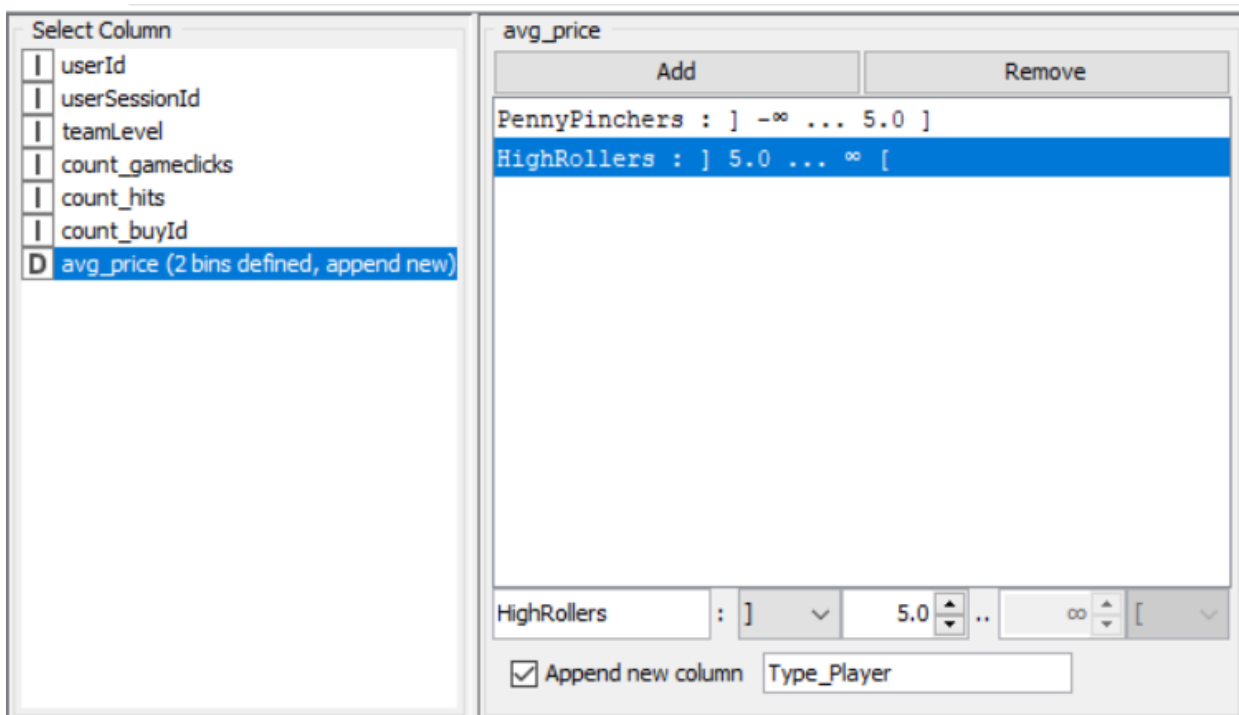
Analysis of combined_data.csv

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:



New column named “avg_price_binned” is the new attribute where avg_price > 5 belongs to “HighRollers” because the prices of them are over \$5, while avg_price ≤ 5 belongs to “PennyPinchers” because the prices of those are not over \$5.

Attribute Deletion

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

Attribute	Rationale for Filtering
avg_price	We don't need the average price anymore since we are coding it into a categorical variable
userId	Don't need this since it is just a computer-generated number
userSessionId	Don't need this since it is just a computer-generated number

Data Partitioning and Modeling

The data was partitioned into train and test datasets.

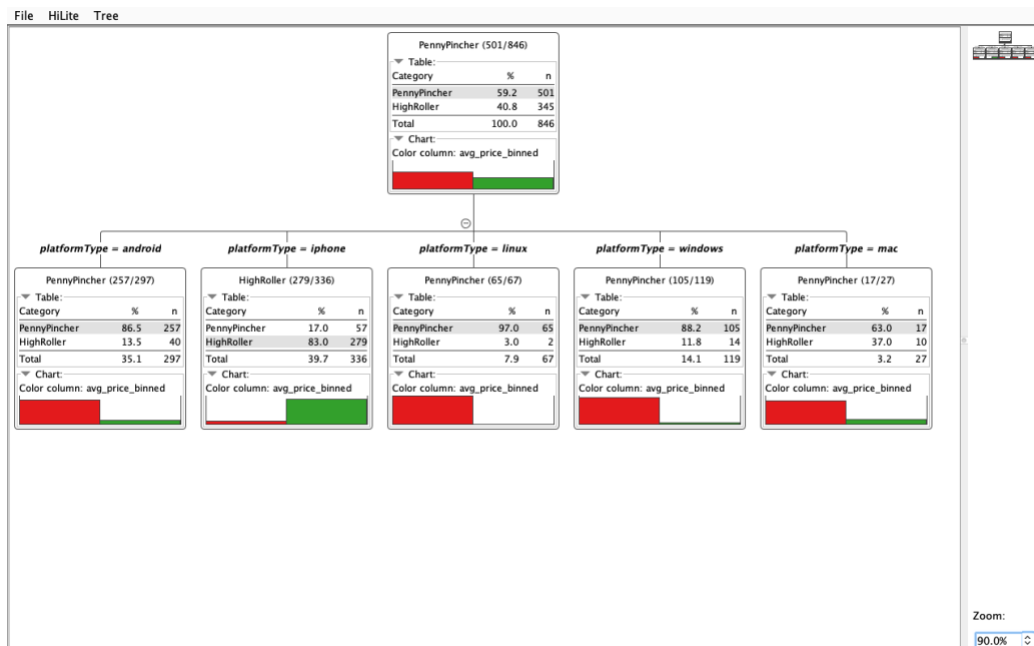
The **train** data set was used to create the decision tree model.

The trained model was then applied to the **test** dataset.

This is important because we want to ensure that we are not overfitting our Decision Tree Model on the train dataset. Running the model on the test data set allows us to determine whether our model generalizes well when introduced with new data.

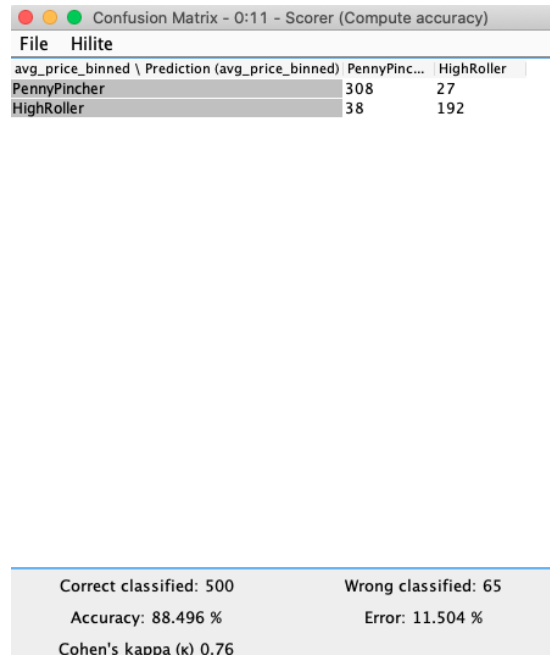
When partitioning the data using sampling, it is important to set the random seed because we want to be able to obtain repeatable results.

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



Evaluation

A screenshot of the confusion matrix can be seen below:



The screenshot shows a window titled "Confusion Matrix - 0:11 - Scorer (Compute accuracy)". It contains a table with the following data:

avg_price_binned \ Prediction (avg_price_binned)	PennyPinc...	HighRoller
PennyPincher	308	27
HighRoller	38	192

Below the table, the following statistics are displayed:

- Correct classified: 500
- Wrong classified: 65
- Accuracy: 88.496 %
- Error: 11.504 %
- Cohen's kappa (κ) 0.76

		Predicted 0	Predicted 1
Actual 0	TN	FP	
Actual 1	FN	TP	

Therefore: TN = 308, FP = 27, FN = 38, TP = 192, total = 565

Accuracy = (TP + TN)/total = (192+308)/565 = **0.885**

Error = (FP + FN)/total = (27+38)/565 = **0.115**

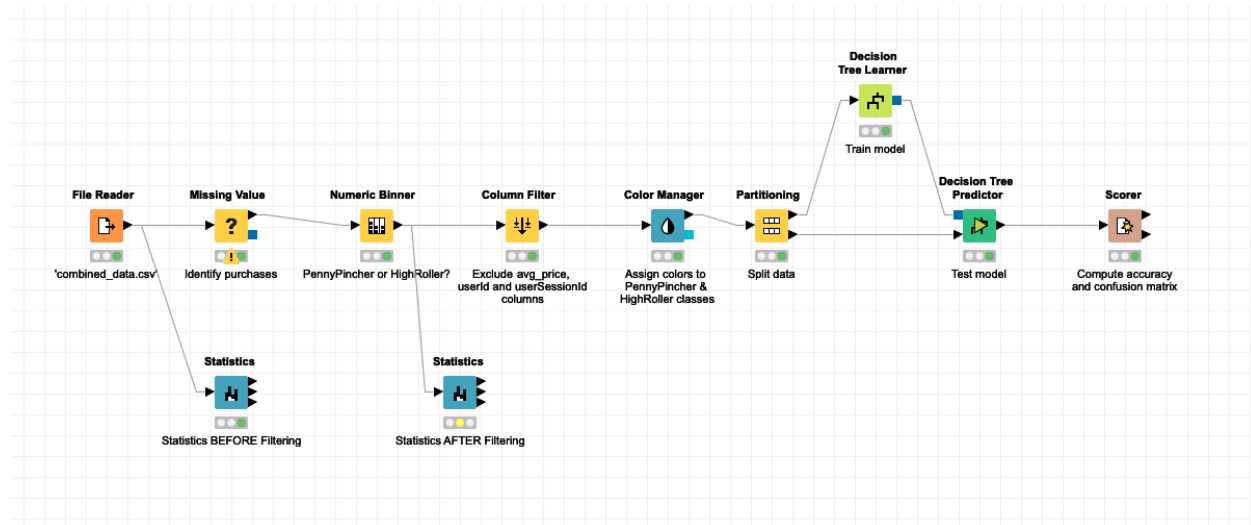
Precision = TP/(TP+FP) = 192/(192+27) = **0.877**

Recall = TP/(TP+FN) = 192/(192+38) = **0.835**

F1 Score = 2(Precision*Recall)/(Precision+Recall) = 2*0.887*0.835/(0.877+0.835) = **0.865**

Analysis Conclusions

The final KNIME workflow is shown below:



What makes a HighRoller vs. a PennyPincher?

Looking at the decision tree created by the model, HighRollers mostly use the iPhone as their platform to play the game, followed (distantly) by Android. PennyPinchers are more prevalent on desktops or laptops.

Overall, we can see that more valuable purchases are made on mobile devices rather than desktops or laptops.

Specific Recommendations to Increase Revenue

1. Create a marketing initiative that entices PennyPinchers to play the game on a mobile device.
2. Build out more game features on mobile platforms to maintain a continued and growing interest in the game.

Clustering Analysis

```
features_used = ["sum(count_gameclicks)", "sum(count_hits)", "sum(price)"]
```

Attribute	Rationale for Selection
"sum(count_gameclicks)" by userId	"count_gameclicks" is provided in the 'combined_data.csv' file. Does the number of game_clicks generated by a userId correlate with money spent?
"sum(count_hits)" by userId	"count_hits" is provided in the 'combined_data.csv' file. Does the number of hits generated by a userId correlate with money spent?
"sum(price)" by userId	"price" paid is provided by userId in the 'buy-clicks.csv' file. Total amount of money spent by user on the game.

Training Data Set Creation

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

userId	sum(count_gameclicks)	sum(count_hits)	sum(price)
231	262	28	63.0
2032	638	59	20.0
233	250	29	28.0
34	665	79	95.0
1234	590	73	53.0
1434	772	89	9.0
1634	2546	266	27.0
1235	367	39	40.0
1835	734	94	27.0
236	606	70	43.0
436	4392	494	43.0
1436	622	65	16.0
1636	317	29	25.0
2236	299	36	15.0
1837	714	90	67.0
38	1425	155	30.0
239	526	50	20.0
439	379	33	25.0
1639	1806	225	155.0
1640	559	73	72.0

only showing top 20 rows

Dimensions of the final data set:

userId	sum(count_gameclicks)	sum(count_hits)	sum(price)	features_unscaled	features
231	262	28	63.0	[262.0,28.0,63.0]	[-0.6344202852895...]
2032	638	59	20.0	[638.0,59.0,20.0]	[-0.0018534459321...]
233	250	29	28.0	[250.0,29.0,28.0]	[-0.6546085886733...]
34	665	79	95.0	[665.0,79.0,95.0]	[0.04357023668131...]
1234	590	73	53.0	[590.0,73.0,53.0]	[-0.0826066594671...]
1434	772	89	9.0	[772.0,89.0,9.0]	[0.22358260851973...]
1634	2546	266	27.0	[2546.0,266.0,27.0]	[3.20808679208386...]
1235	367	39	40.0	[367.0,39.0,40.0]	[-0.4577726306817...]
1835	734	94	27.0	[734.0,94.0,27.0]	[0.15965298113786...]
236	606	70	43.0	[606.0,70.0,43.0]	[-0.0556889216221...]
436	4392	494	43.0	[4392.0,494.0,43.0]	[6.31372079595049...]
1436	622	65	16.0	[622.0,65.0,16.0]	[-0.0287711837771...]
1636	317	29	25.0	[317.0,29.0,25.0]	[-0.5418905614473...]
2236	299	36	15.0	[299.0,36.0,15.0]	[-0.5721730165230...]
1837	714	90	67.0	[714.0,90.0,67.0]	[0.12600580883161...]
38	1425	155	30.0	[1425.0,155.0,30.0]	[1.32216278431870...]
239	526	50	20.0	[526.0,50.0,20.0]	[-0.1902776108471...]
439	379	33	25.0	[379.0,33.0,25.0]	[-0.4375843272980...]
1639	1806	225	155.0	[1806.0,225.0,155.0]	[1.96314141675271...]
1640	559	73	72.0	[559.0,73.0,72.0]	[-0.1347597765418...]

only showing top 20 rows

Cluster Centers

The code used in creating cluster centers is given below:

We can now perform K-Means clustering to generate 3 clusters:

```
In [18]: kmeans = KMeans(k=3, seed=1)
model = kmeans.fit(scaledData)
transformed = model.transform(scaledData)
```

Print the center of these three clusters...

```
In [19]: centers = model.clusterCenters()
centers
```

```
Out[19]: [array([-0.31333318, -0.33179717, -0.37859269]),
array([ 2.35752989,  2.3388565 , -0.03049294]),
array([-0.04642581,  0.05337151,  1.8170137 ])]
```

Convert the cluster centers back to their original scale...

```
In [20]: centers = [cluster * scalerModel.std + scalerModel.mean
for cluster in model.clusterCenters()]
centers

Out[20]: [array([ 452.85532995,  50.17766497,  24.24111675]),
array([ 2040.42592593,  229.05555556,  38.66666667]),
array([ 611.5060241 ,  75.97590361, 115.22891566])]
```

```
In [ ]:
```

Cluster centers formed are given in the table below

Cluster #	Center [sum(count_gameclicks), sum(count_hits), sum(price)]
1	452.85532995, 50.17766497, 24.24111675
2	2040.42592593, 229.05555556, 38.66666667
3	611.5060241, 75.97590361, 115.22891566

These clusters can be differentiated from each other as follows:

Cluster 1 is different from the others in that users with a smaller number of gameclicks and count_hits tend to have the lowest number of in-game purchases.

Cluster 2 is different from the others in that the users who generate large gameclicks and count_hits numbers spend slightly more in in-game purchases.

Cluster 3 is different from the others in that the users who generate an intermediate number of gameclicks and count_hits generate the most purchases

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

```
Select the features column make the data persist:

In [17]: scaledData = scaledData.select("features")
scaledData.persist()

Out[17]: DataFrame[features: vector]

In [18]: scaledData.show()

+-----+
|          features|
+-----+
| [-0.6344202852895...|
| [-0.0018534459321...|
| [-0.6546085886733...|
| [0.04357023668131...|
| [-0.0826066594671...|
| [0.22358260851973...|
| [3.20808679208386...|
| [-0.4577726306817...|
| [0.15965298113786...|
| [-0.0556889216221...|
| [6.31372079595049...|
| [-0.0287711837771...|
| [-0.5418905614473...|
| [-0.5721730165230...|
| [0.12600580883161...|
| [1.32216278431870...|
| [-0.1902776108471...|
| [-0.4375843272980...|
| [1.96314141675271...|
| [-0.1347597765418...|
+-----+
only showing top 20 rows
```

Recommended Actions

Action Recommended	Rationale for the action
Increase ads to users who play a lot	It was seen that users who play a lot are also the users who spend less. If we increase ads to users who play a lot, it will promote these users to spend more and therefore increase the revenue
Show higher price ads to users who spend more	If we show higher price ads to users who spend more, we can increase the revenue faster. The users who spend more also do not play too much, thus by showing them the more valuable ads first, we can increase the revenue faster.

Graph Analytics

Modeling Chat Data using a Graph Data Model

All chat-related data is contained in six .csv files that were downloaded in Week 1 of the Capstone Project. 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.

Creation of the Graph Database for Chats

Describe the steps taken to create the graph database.

- i) Write the schema of the 6 CSV files

chat_create_team_chat.csv
0. UserId
1. TeamId
2. TeamChatSessionId
3. timestamp

chat_join_team_chat.csv
0. UserId
1. TeamChatSessionId
2. timestamp

chat_leave_team_chat.csv
0. UserId
1. TeamChatSessionId
2. timestamp

chat_item_team_chat.csv
0. UserId
1. TeamChatSessionId
2. ChatItemId
3. timestamp

chat_mention_team_chat.csv
0. ChatItemId
1. UserId
2. timestamp

chat_respond_team_chat.csv
0. ChatItemId
1. ChatItemId
2. timestamp

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

In the Neo4J system we first execute the following instructions:



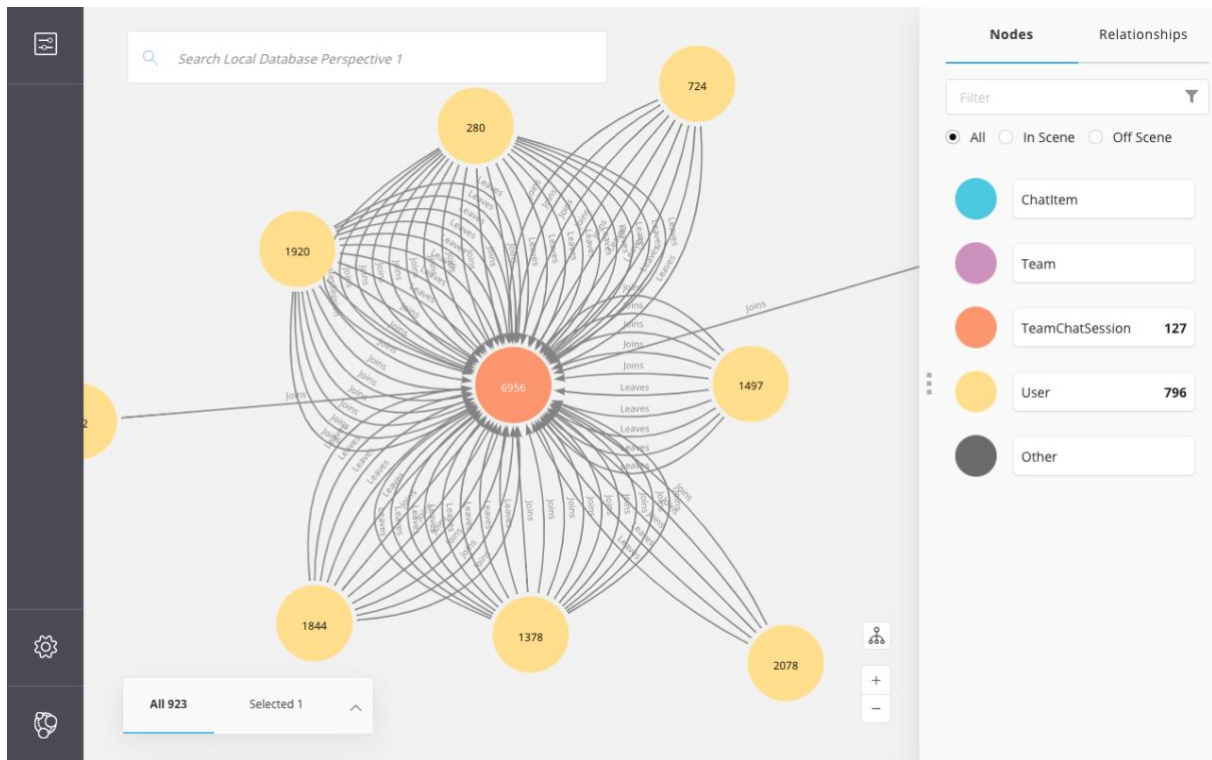
```
neo4j$  
$ CREATE CONSTRAINT ON (u:User) ASSERT u.id IS UNIQUE; CREATE CONSTRAINT ON (t:Team) ASSERT t.id IS U...  
$ CREATE CONSTRAINT ON (u:User) ASSERT u.id IS UNIQUE  
$ CREATE CONSTRAINT ON (t:Team) ASSERT t.id IS UNIQUE  
$ CREATE CONSTRAINT ON (c:TeamChatSession) ASSERT c.id IS UNIQUE  
$ CREATE CONSTRAINT ON (i:ChatItem) ASSERT i.id IS UNIQUE
```

We then load the chat .csv files into Neo4J. Here is an example LOAD sequence:



```
1 LOAD CSV FROM "http://localhost:11001/project-dfdb3533-4631-4a04-9aca-b168b64da413/chat_join_team_chat.csv" AS row  
2 MERGE (u:User {id: toInteger(row[0])})  
3 MERGE (c:TeamChatSession {id: toInteger(row[1])})  
4 MERGE (u)-[:Joins{timeStamp: row[2]}]-(c)  
neo4j$ LOAD CSV FROM "http://localhost:11001/project-dfdb3533-4631-4a04-9aca-b168b64da413/cha...  
Added 381 labels, created 381 nodes, set 635 properties, created 254 relationships, completed after 834 ms.  
Table  
Code  
Added 381 labels, created 381 nodes, set 635 properties, created 254 relationships, completed after 834 ms.
```

- 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

Report the results including the length of the conversation (path length) and how many unique users were part of the conversation chain.

- i) Longest conversation path length...

```
1 / Question 1
2 // Find the longest conversation chain in the chat data using the "ResponseTo"
3 // edge label. This question has two parts
4
5 // 1) How many chats are involved in it?
6 match p = (i1)-[:ResponseTo*]->(i2)
7 return length(p)
8 order by length(p) desc limit 1
```

```
neo4j$ match path = (i1)-[:ResponseTo*]->(i2) return length(path) order by length(path) d...
```

length(path)
9

Started streaming 1 records after 3 ms and completed after 791 ms.

ii) Number of unique users in conversation...

```
1 // 2) How many users participated in this chain?
2 match path = (i1)-[:ResponseTo*]→(i2)
3 where length(path) = 9
4 with path
5 match (u)-[:CreateChat]→(i)
6 where i in nodes(path)
7 return count(distinct u)
```

neo4j\$ match path = (i1)-[:ResponseTo*]→(i2) where length(path) = 9 with path match (u)-...

	count(distinct u)
1	5

Started streaming 1 records after 3 ms and completed after 556 ms.

Top 10 chattiest users and top 10 chattiest teams...

Chattiest Users:

```
1 // chattiest users
2 match (u)-[:CreateChat*]→(i)
3 return u.id, count(i)
4 order by count(i) desc limit 10
```

neo4j\$ match (u)-[:CreateChat*]→(i) return u.id, count(i) order by count(i) desc limit 3

	u.id	count(i)
1	394	115
2	2067	111
3	209	109

Started streaming 3 records after 3 ms and completed after 814 ms.

Chattiest Teams:

```
1 // chattiest teams
2 match (i)-[:PartOf*]-(c)-[:OwnedBy*]-(t)
3 return t.id, count(c)
4 order by count(c) desc limit 3
```

```
neo4j$ match (i)-[:PartOf*]-(c)-[:OwnedBy*]-(t) return t.id, count(c) order by count(c)...
```

	t.id	count(c)
1	82	1324
2	185	1036
3	112	957

Started streaming 3 records after 3 ms and completed after 222 ms.

How Active Are Groups of Users?

First, we will compute an estimate of how “dense” the neighborhood of a node is. In the context of chat, that translates to how mutually interactive a certain group of users are. If we can identify these highly interactive neighborhoods, we can potentially target some members of the neighborhood for direct advertising. We will do this in a series of steps.

(a) We will construct the neighborhood of users. In this neighborhood, we will connect two users if:

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

The way to make this connection will be to create a new edge called, for example, “InteractsWith” between users to satisfy either of the two conditions. So, we will write a query to create these edges for each condition.

```
neo4j$ Match (u1:User)-[:CreateChat]-(i)-[:Mentioned]-(u2:User)
create (u1)-[:InteractsWith]-(u2)
```

```
neo4j$ Match (u1:User)-[:CreateChat]-(i)-[:Mentioned]-(u2:User)
create (u1)-[:InteractsWith]-(u2)
```

Created 11084 relationships, completed after 790 ms.

Created 11084 relationships, completed after 790 ms.

We will use the same logic to create the query statement for the second condition.

```
1 match (u1:User) - [:CreateChat] -> (i1:ChatItem) - [:ResponseTo] -> (u2:User)
2 (i2:ChatItem)
3 with u1, i1, i2
4 match (u2) - [:CreateChat] -> (i2)
5 create (u1) - [:InteractsWith] -> (u2)
```

neo4j\$ match (u1:User) - [:CreateChat] -> (i1:ChatItem) - [:R... Created 22146 relationships, completed after 1103 ms.

Table

Code

Created 22146 relationships, completed after 1103 ms.

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 executing the query above, we will need to eliminate all self-loops involving the edge “InteractsWith”. This can be done through this query:

```
neo4j$ match (u1) - [r:InteractsWith] -> (u1) delete r
```

neo4j\$ match (u1) - [r:InteractsWith] -> (u1) delete r Deleted 6565 relationships, completed after 794 ms.

Table

Code

Deleted 6565 relationships, completed after 794 ms.

To create the clustering coefficient, we need to find all the users that are connected through the edge “InteractsWith” (line 1) and 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.

```

1 match (u1:User)-[r1:InteractsWith]→(u2:User)
2 where u1.id <> u2.id
3 AND u1.id in [394,2067,1087,209,554,999,516,1627,999,516,461,668]
4 with u1, collect(u2.id) as neighbors, count( distinct (u2)) as neighborCount
5 match (u3:User)-[r2:InteractsWith]→(u4:User)
6 where (u3.id in neighbors) AND (u4.id in neighbors) AND (u3.id <> u4.id)
7 with u1, u3, u4, neighborCount,
8 case when count(r2) > 0 then 1
9 else 0
10 end as answer
11 return u1.id, sum(answer)* 1.0 /(neighborCount*(neighborCount -1 )) as coeff

```

Most Active Users (based on Cluster Coefficients)

neo4j\$ match (u1:User)-[r1:InteractsWith]→(u2:User) wher...

	u1.id	coeff
1	461	1.0
2	668	1.0
3	209	0.9523809523809523
4	516	0.9523809523809523
5	394	0.9166666666666666
6	999	0.8194444444444444
7	554	0.8000000000000000

Started streaming 10 records after 32 ms and completed after 119 ms.