# INC. GAME| CATCH THE PINK FLAMINGO- GROWTH STRATEGY TECHNICAL APPENDIX


Submitted by

**Muhammed Abdul-Fattah Salem**

# Table of Contents:

# CHAPTER1:DATA EXPLORATION VIA SPLUNK >

## CH1.1: Dataset 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** | ERD table: **AdClicks**<br><br>A line is added to this file when a player clicks on an advertisement in the Flamingo app.<br><br>Example: **[timestamp, txId, userSessionid, teamid, userid, adId, adCategory]**<br><br>[20160514 21:11:46, 6164, 6036, 91, 2116, 11, sports] | **timestamp**: when the click occurred.<br>**txId**: a unique id (within adclicks. log) for the click.<br>**userSessionid**: the id of the user session for the user who made the click.<br>**teamid**: the current team id of the user who made the click.<br>**userid**: the user id of the user who made the click.<br>**adId**: the id of the ad clicked on.<br>**adCategory**: the category/type of ad clicked on. |
| **buy-clicks.csv** | ERD table: **InAppPurchases**<br><br>A line is added to this file when a player makes an in-app purchase in the Flamingo app.<br><br>Example: **[timestamp, txId, userSessionId, team, userId, buyId, price]**<br><br>[20160514 21:41:36, 6193, 5759, 107, 418, 2, 2.99] | **timestamp**: when the purchase was made.<br>**txId**: a unique id (within buy-clicks.log) for the purchase.<br>**userSessionId**: the id of the user session for the user who made the purchase.<br>**team**: the current team id of the user who made the purchase.<br>**userId**: the user id of the user who made the purchase.<br>**buyId**: the id of the item purchased.<br>**price**: the price of the item purchased. |

| | | |
|---|---|---|
| **users.csv** | ERD table: **User**<br><br>This file contains a line for each user playing the game.<br><br>Example: **[timestamp, userId, nick, twitter, dob, country]**<br><br>[20120607 13:18:39, 1727, ygT3AjR9, @xF6DPo, 19920823, TO] | **timestamp**: when user first played the game.<br>**userId**: the user id assigned to the user.<br>**nick**: the nickname chosen by the user.<br>**twitter**: the twitter handle of the user.<br>**dob**: the date of birth of the user.<br>**country**: the two-letter country code where the user lives. |
| **team.csv** | ERD table: **Team**<br><br>This file contains a line for each team terminated in the game.<br><br>Example: **[teamId, name, teamCreationTime, teamEndTime, strength, currentLevel]**<br><br>[71, MVGOSu, 20160531 20:02:08, 99991231 23:59:59, 0.929398025213,1] | **teamId**: the id of the team.<br>**name**: the name of the team.<br>**teamCreationTime**: the timestamp when the team was created.<br>**teamEndTime**: the timestamp when the last member left the team.<br>**strength**: a measure of team strength, roughly corresponding to the success of a team.<br>**currentLevel**: the current level of the team. |
| **teamassignments.csv** | ERD table: **TeamAssignment**<br><br>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.<br><br>Example: **[timestamp, team, userId, assignmentId]**<br><br>[20160601 16:42:18, 37, 894, 5525] | **timestamp**: when the user joined the team.<br>**team**: the id of the team.<br>**userId**: the id of the user.<br>**assignmentId**: a unique id for this assignment. |
| **levelevents.csv** | ERD table: **LevelEvent**<br><br>A line is added to this file each time a team starts or finishes a level in the game.<br><br>Example: **[timestamp, eventId, teamId, teamLevel, eventType]**<br><br>[20160516 04:41:36, 0, 4, 1, end] | **timestamp**: when the event occurred.<br>**eventId**: a unique id for the event.<br>**teamId**: the id of the team.<br>**teamLevel**: the level started or completed.<br>**eventType**: the type of event, either start or end. |
| **usersession.csv** | ERD table: **User_Sessions**<br><br>Each line in this file describes a user session, which denotes when a user starts and stops playing the game. | **timestamp**: a timestamp denoting when the event occurred.<br>**userSessionId**: a unique id for the session.<br>**userId**: the current user's ID. |

| | | |
|---|---|---|
| | 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.<br><br>Example: **[timestamp, userSessionId, userId, teamId, assignmentId, sessionType, teamLevel, platformType]**<br><br>[20160526 14:47:30, 5675, 170, 90, 5423, start, 1, iphone] | **teamId**: the current user's team.<br>**assignmentId**: the team assignment id for the user to the team.<br>**sessionType**: whether the event is the start or end of a session.<br>**teamLevel**: the level of the team during this session.<br>**platformType**: the type of platform of the user during this session. |
| **gameclicks.csv** | ERD table: **GameClicks**<br><br>A line is added to this file each time a user performs a click in the game.<br><br>Example: **[timestamp, clickId, userId, userSessionId, isHit, teamId, teamLevel]**<br>[20160514 20:41:36, 298, 2236, 6104, 0, 80, 1] | **timestamp**: when the click occurred.<br>**clickId**: a unique id for the click.<br>**userId**: the id of the user performing the click.<br>**userSessionId**: the id of the session of the user when the click is performed.<br>**isHit**: denotes if the click was on a flamingo (value is 1) or missed the flamingo (value is 0).<br>**teamId**: the id of the team of the user.<br>**teamLevel**: the current level of the team of the user. |

## CH1.2: Aggregation

| | |
|---|---|
| Amount spent buying items | `source="buy-clicks.csv"\| stats sum(price)`<br>Results:<br>`21407.0` |
| Number of unique items available to be purchased | `source="buy-clicks.csv"\| stats count by buyId`<br>Results:<br>`6.0` |

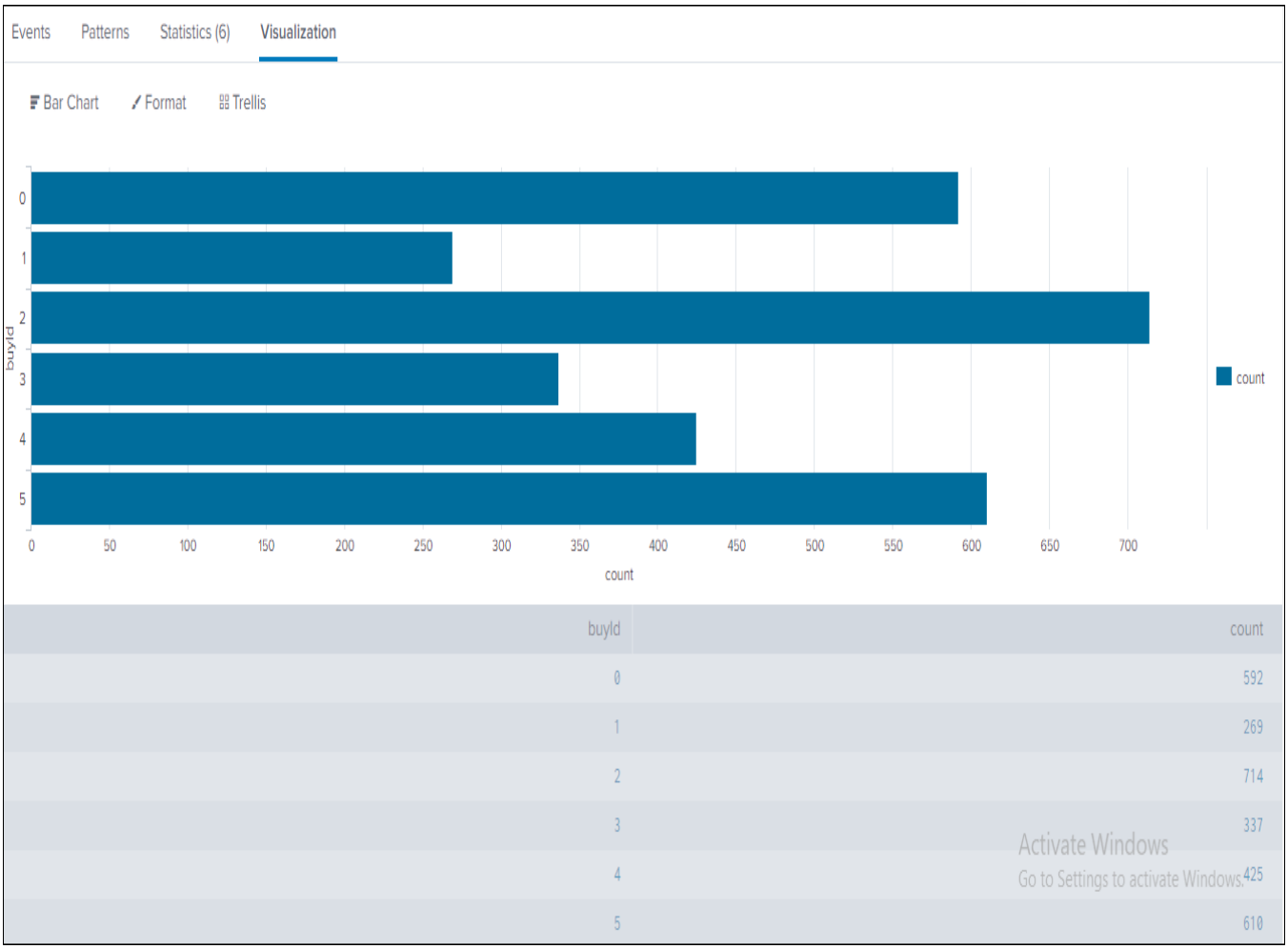



**Figure 1.2.1**: A HISTOGRAM SHOWING HOW MANY TIMES EACH UNIQUE ITEM IS PURCHASED (`source = "buy-clicks.csv"| stats count by buyId`).

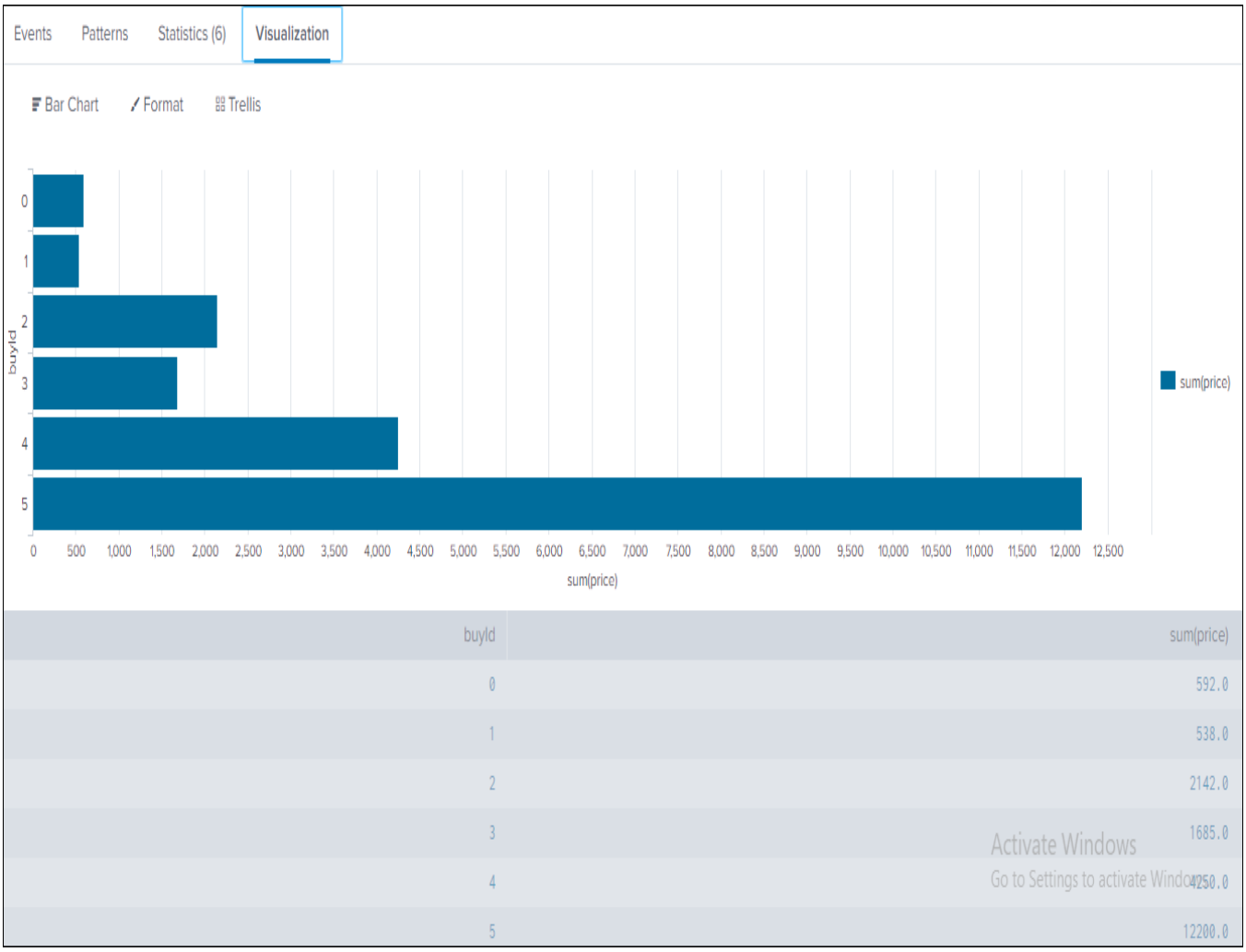



**Figure 1.2.2**: A HISTOGRAM SHOWING HOW MUCH MONEY EACH UNIQUE ITEM HAS MADE(`source = "buy-clicks.csv"| stats sum(price) by buyId`).

# CH 1.3: Filtering

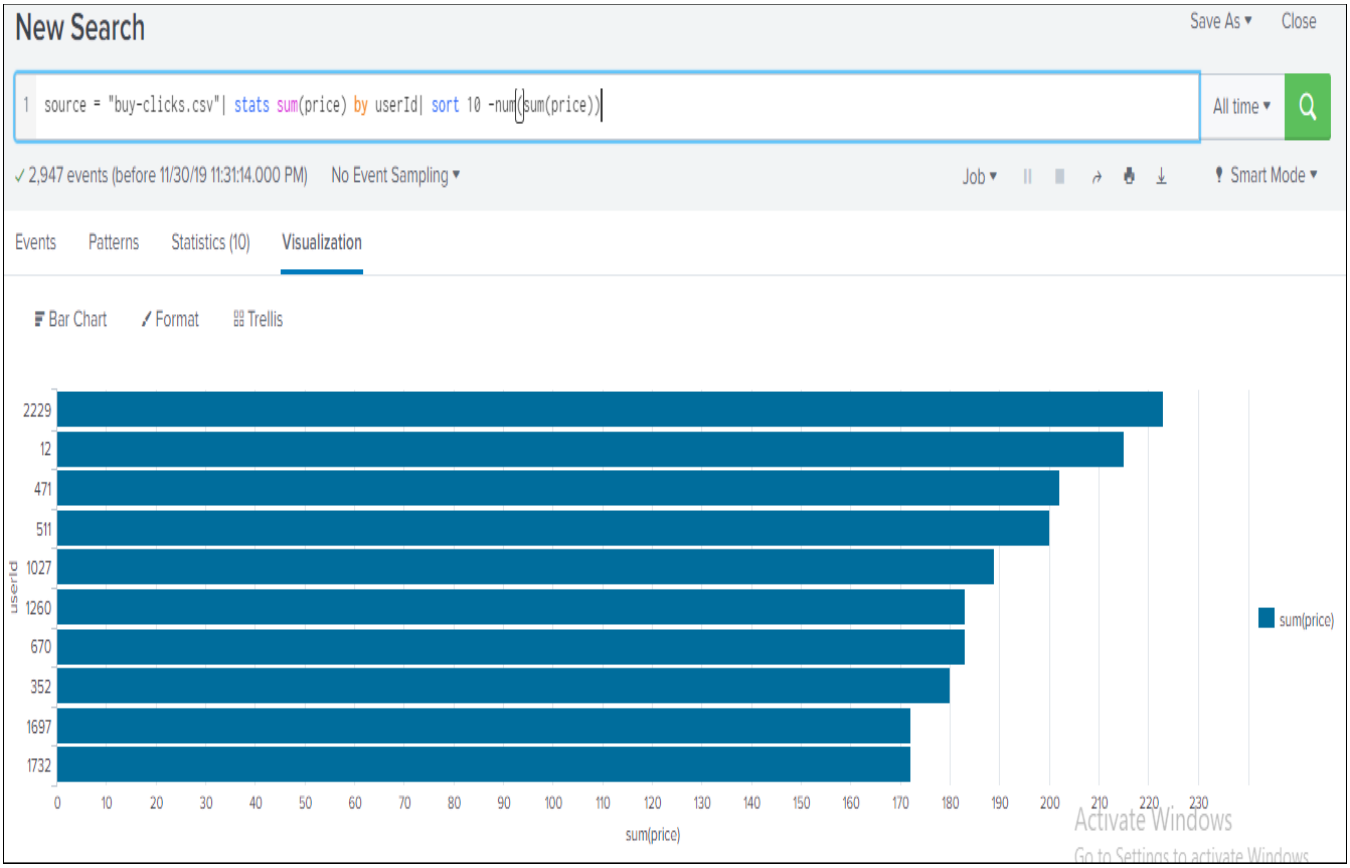



**Figure 1.3.1**: A HISTOGRAM SHOWING TOP 10 USERS RANKED BY AMOUNT OF MONEY THEY HAVE SPENT IN PURCHASE (`source = "buy-clicks.csv"| stats sum(price) by userId| sort 10 -num(sum(price))`)

The following table shows the user id, platform, and hit-ratio percentage for the top three buying users:

| **Rank** | **User Id** | **Platform**<br>`source="user-session.csv" userId="!"| stats count by platformType` | **Hit-Ratio (%)**<br>`source = "game-clicks.csv" userId ="!"| stats sum(isHit) as Sum, c(isHit) as Count by userId| eval Hit-Ratio% = (Sum / Count) * 100` |
|---|---|---|---|
| **1** | **2229** | `iphone` | `11.6` |
| **2** | **12** | `iphone` | `13.1` |
| **3** | **471** | `iphone` | `14.5` |

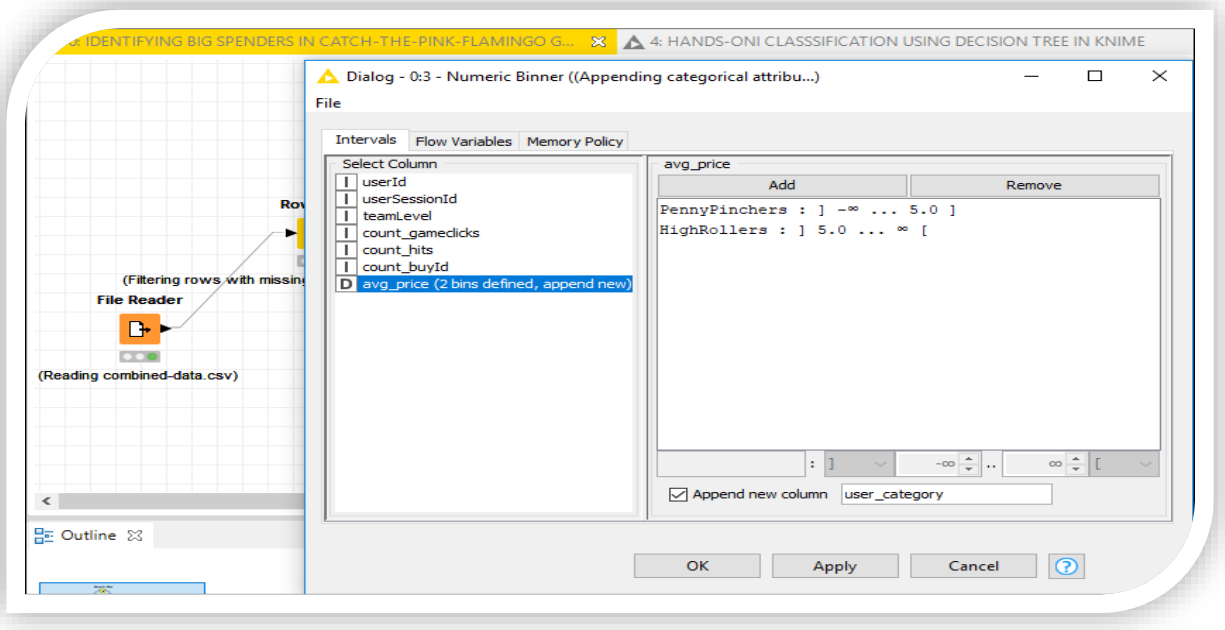
# CHAPTER 2: CLASSIFICATION ANALYSIS IN KNIME

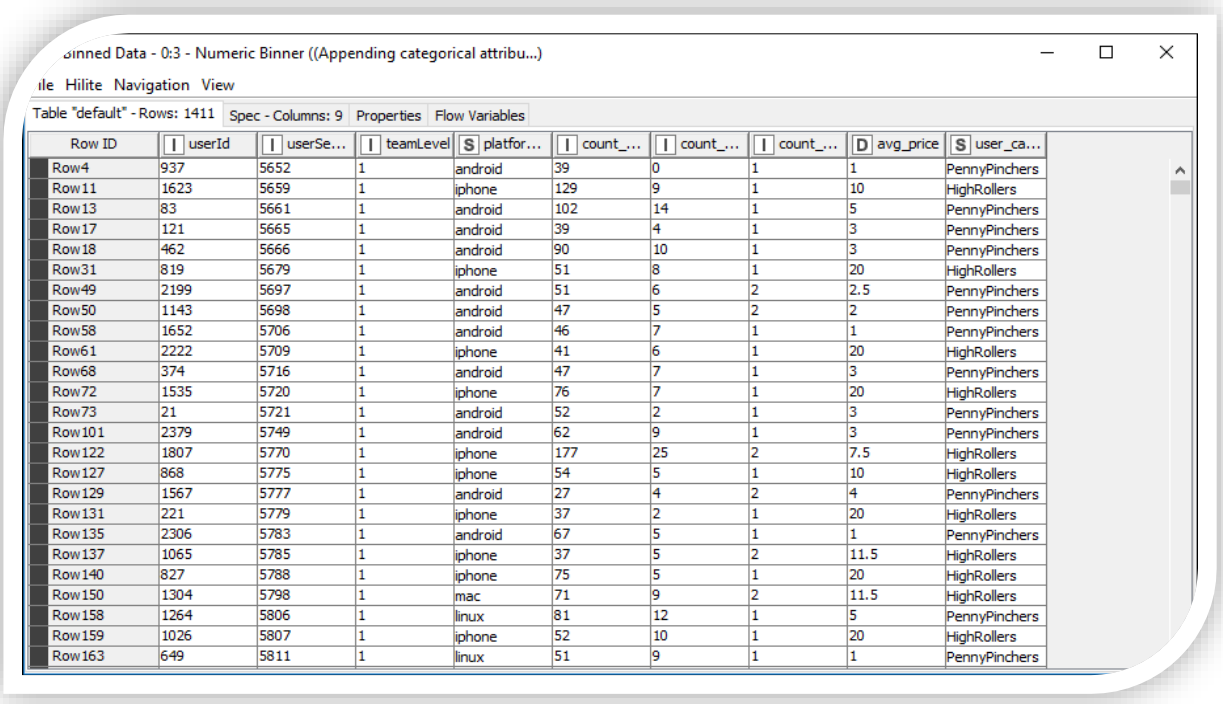
## CH 2.1: DATA PREPARATION| Analysis of combined_data.csv

### CH 2.1.1: Sample Selection

| Item | Amount |
|---|---|
| # of Samples | 4619 |
| # of Samples with Purchases | 1411 |

### CH 2.1.2: Attribute Creation






| Row ID | userId | userSe... | teamLevel | platfor... | count_... | count_... | count_... | avg_price | user_ca... |
|---|---|---|---|---|---|---|---|---|---|
| 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 |

**Figure 2.2.2:** SHOWS APPENDING NEW ATTRIBUTE CALLED USER CATEGORY DIFFERENTIATING USERS INTO "HIGHROLLERS" FOR USERS SPENDING MORE THAN 5$ IN PURCHASES AND "PENNYPINCHERS" FOR USERS SPENDING LESS THAN OR EQUAL 5$.

The creation of the new categorical attribute was necessary to insightfully differentiate users into two categories because it is not recommended for a classification problem building a decision tree model to use an attribute with continuous values such as avg_price.

## CH 2.2: Attribute Selection

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

| **Attribute** | **Rationale for Filtering** |
|---|---|
| **avg_price** | No need since we instead appended a new attribute called "user_category " categorizing users into HighRollers for those who spent more than 5$ in game purchases and PennyPinchers for less than or equal 5$. |
| **UserId** | No need since it is generated randomly and not related to our user_category |

| | |
|---|---|
| | classification problem. |
| **userSessionId** | No need since it is generated randomly and not related to our user_category classification problem. |

## CH 2.3: Data Partitioning and Modeling

- The data was partitioned into train and test datasets.
- The train dataset was used to create the decision tree model.
- The trained model was then applied to the test dataset.
- This is important because while building a decision tree model, we need to test its accuracy for a test dataset different from the train one used for building the trained model. Since we have our trained decision tree model, we apply this model for our classification problem making predictions against the test dataset.
- When partitioning the data using sampling, it is important to set the random seed to make sure that the partition is the same every time we are running the program; it is important to reproduce results again and again.

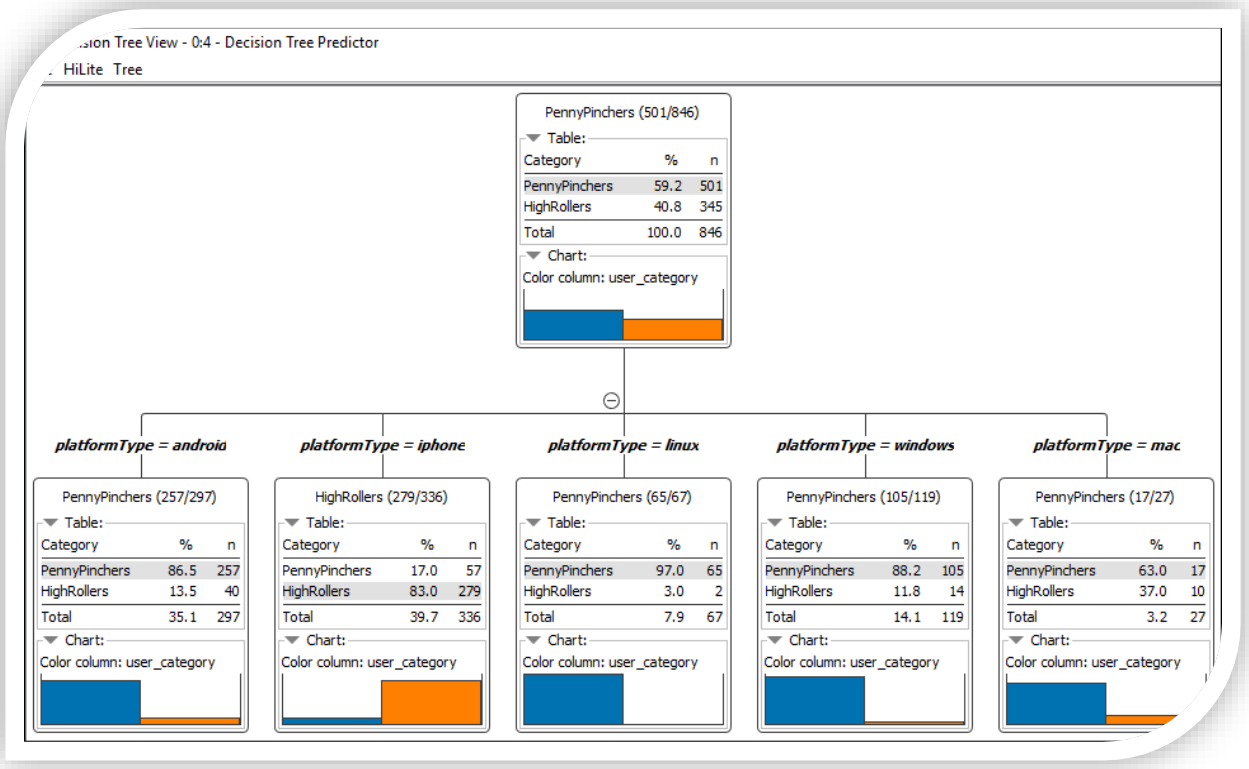



Figure 2.3.1: SHOWS RESULTING DECISION TREE PREDICTOR MODEL.

## CH 2.4: Evaluation

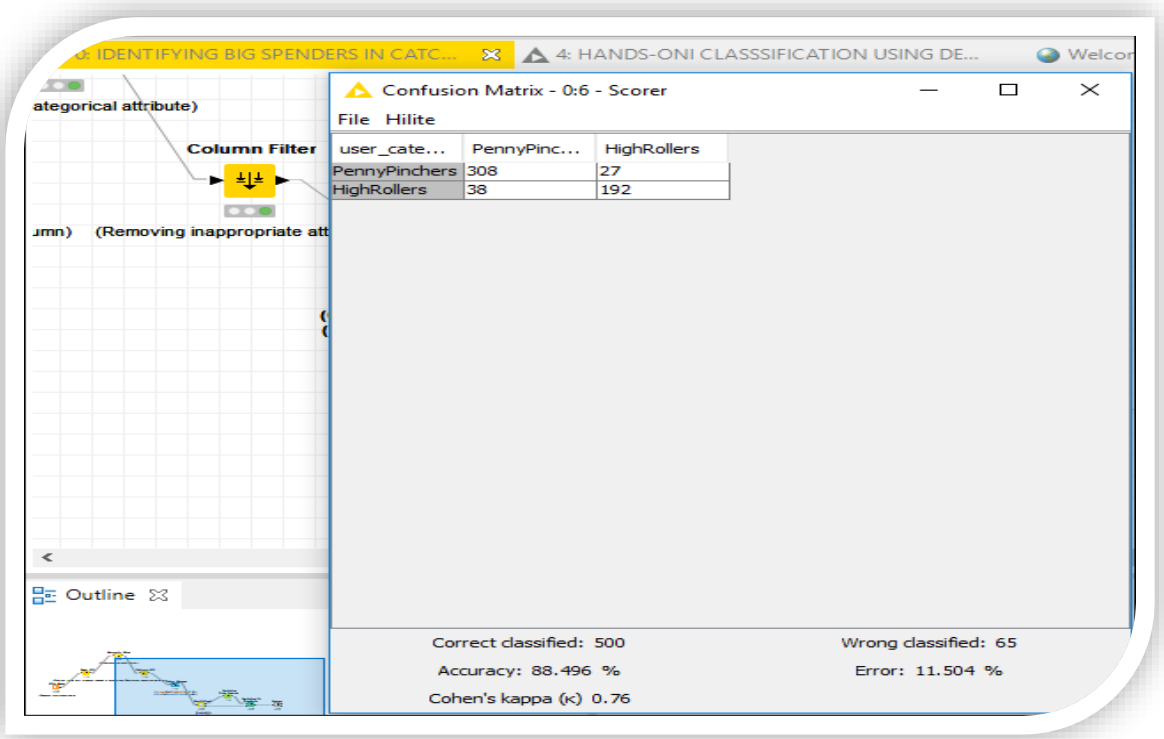



Figure 2.4.1: SHOWS RESULTING CONFUSION MATRIX WITH OVERALL **ACCURACY** = **88.5%**.

- **500** samples have been **correctly classified**, but **65** have been **wrong classified** with **11.5%** **error**.
- **308** **of PennyPinchers** are correctly classified.
- **27** **of PennyPinchers** are wrong classified.
- **192** **of HighRollers** are correctly classified.
- **38** **of HighRollers** are wrong classified

## CH 2.5: Analysis Conclusions

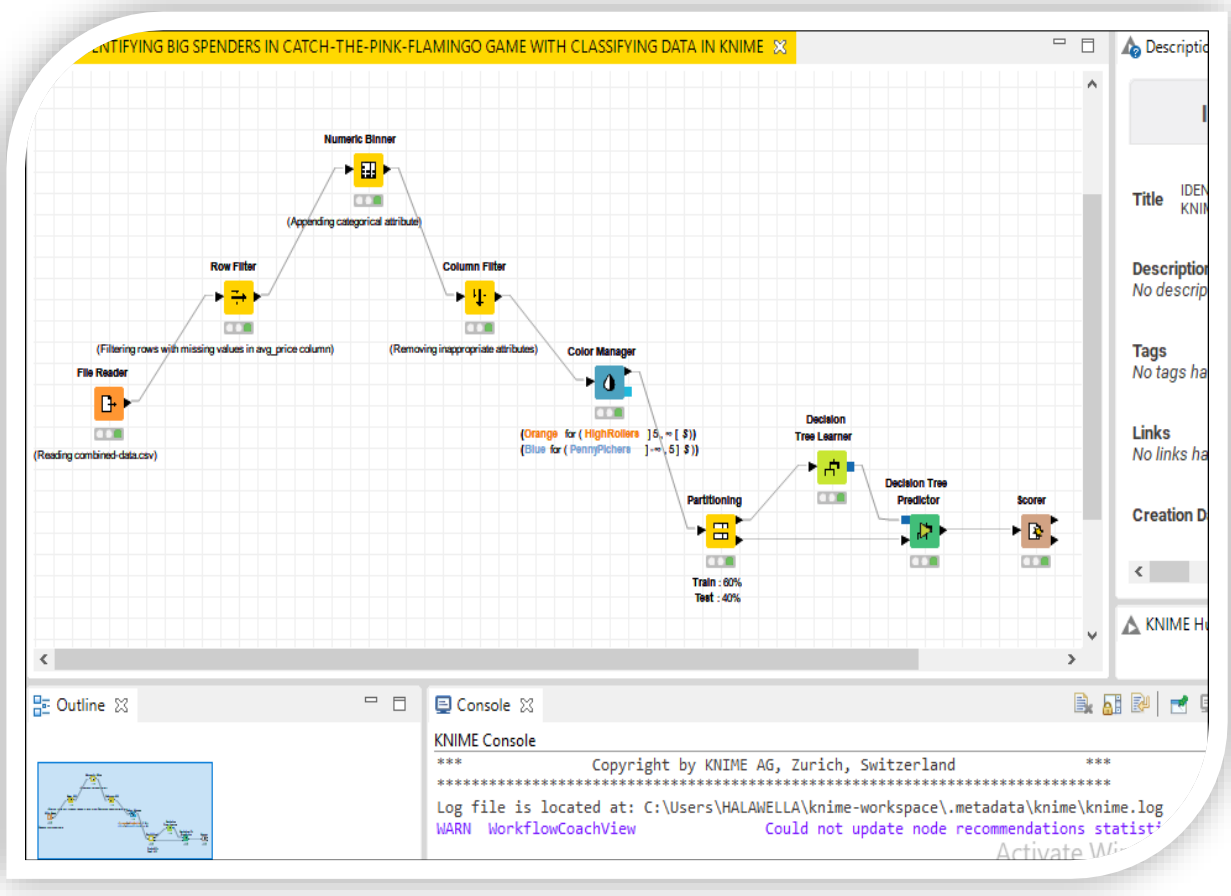



Figure 2.5.1: SHOWS FINAL KNIME WORKFLOW BUILDING A DECISION TREE MODEL.

### QUESTION: What makes a HighRoller vs. a PennyPincher?

Insightfully, it is observed that iphone users are more likely to be HighRollers, but other platforms' users are PennyPinchers.

| **Specific Recommendations to Increase Revenue** |
|---|
| **1**. Promote limited offers to encourage PennyPinchers for buying. |
| **2**. Show more expensive ads targeting iphone platform users and high_level_spending ones. |

# CHAPTER 3: CLUSTERING ANALYSIS VIA SPARK MLlib


## CH 3.1: Attribute Selection

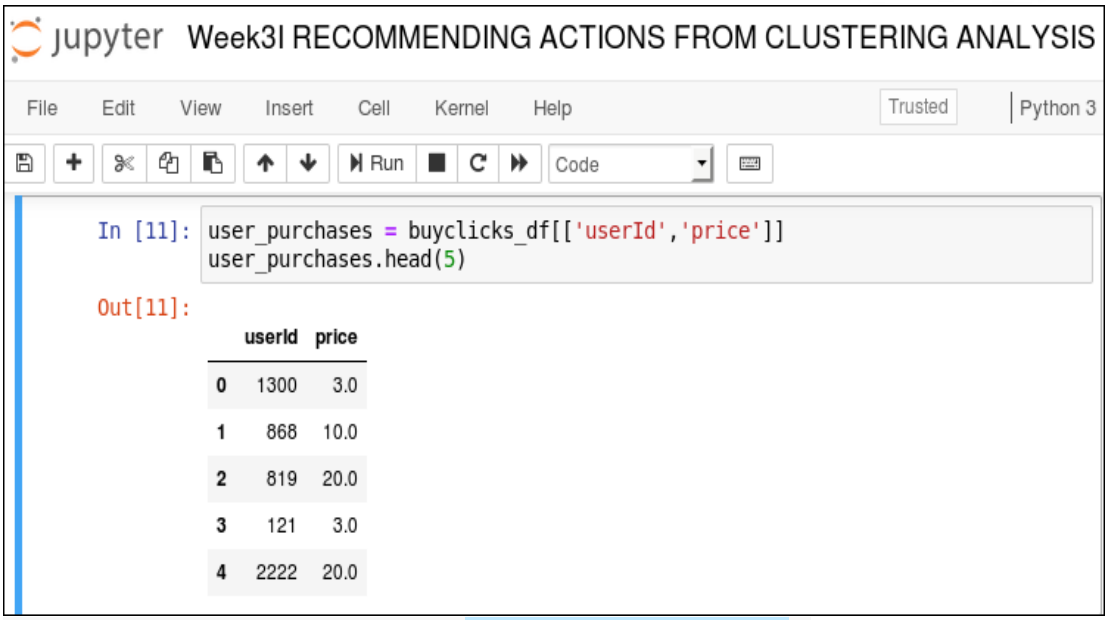
| Attribute | Rationale for Selection |
|---|---|
| Total_game_clicks/ User | This attribute describes the total number of times a user has clicked in the game. This is the result of adding **clickCount** column counting game clicks/user in the **game_clicks** dataset.<br>It will help understanding how involved the user in the game is. |
| Total_ad_clicks/User | This attribute describes the total number of times a user has clicked ads during all the user's history in the game. This is the result of adding **adCount** column counting ad clicks/user in the **ad_clicks** dataset.<br>It will help understanding what kind of ads each user is interested in. |
| Revenue/User | This attribute describes the sum of the average price of ads each user has clicked. This is the result of adding **price** column counting purchases' price/user in the **buy_clicks** dataset.<br>This attribute will help to understand the value of users. Users who click on high priced ads are more valuable and generate more revenue. |

## CH 3.2: Training Dataset Overview

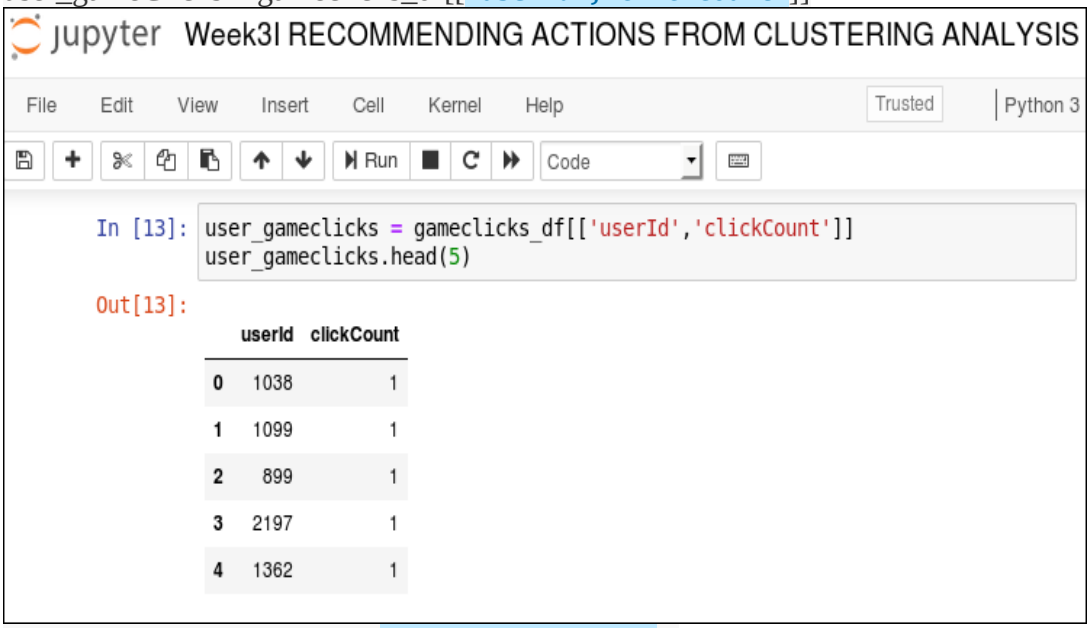
### Ch 3.2.1: Training Dataset creation

**The training data set used for this analysis is shown below:**

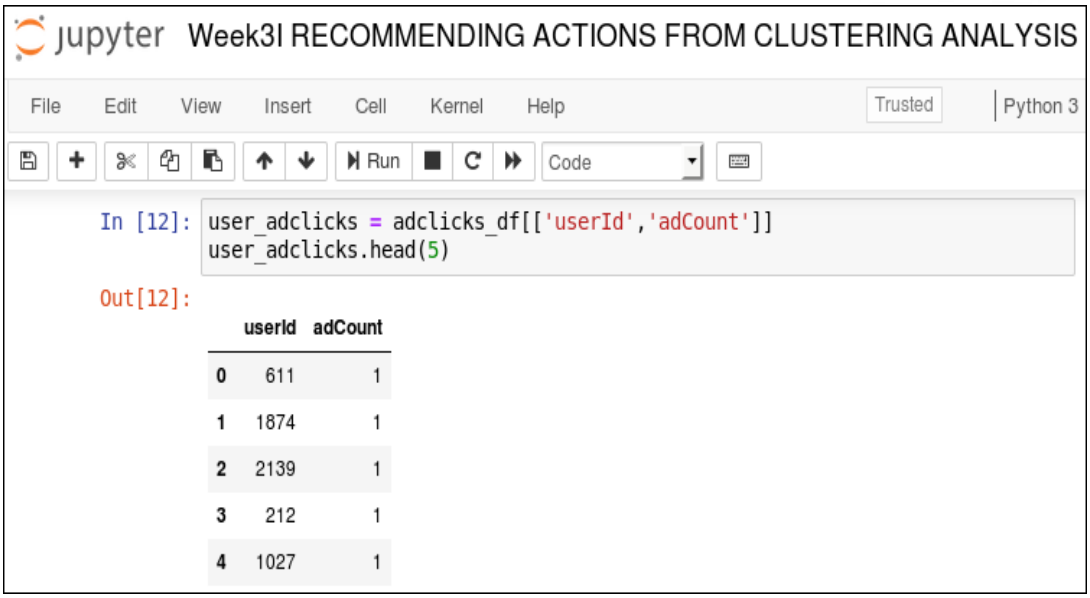
- Read of the following files: `game-clicks.csv, buy-clicks.csv, and ad-clicks.csv`
- Feature Selection of the selected columns
  - user_purchases = buyclicks_df[[`'userId','price'`]]




- user_gameClicks = gameclicks_df[['userId','clickCount']]

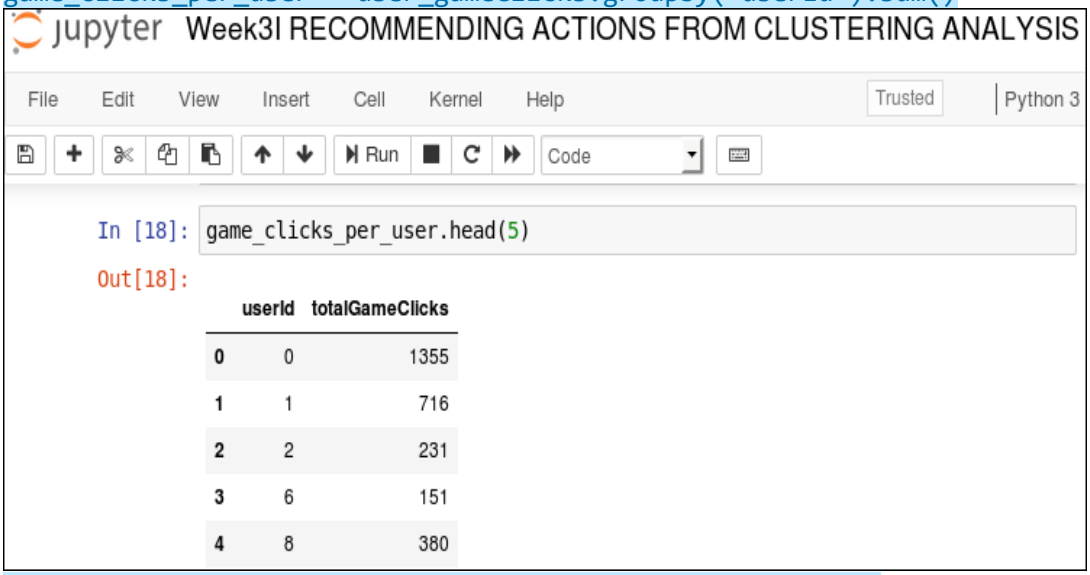



- user_adclicks = adclicks_df[['userId','adCount']]



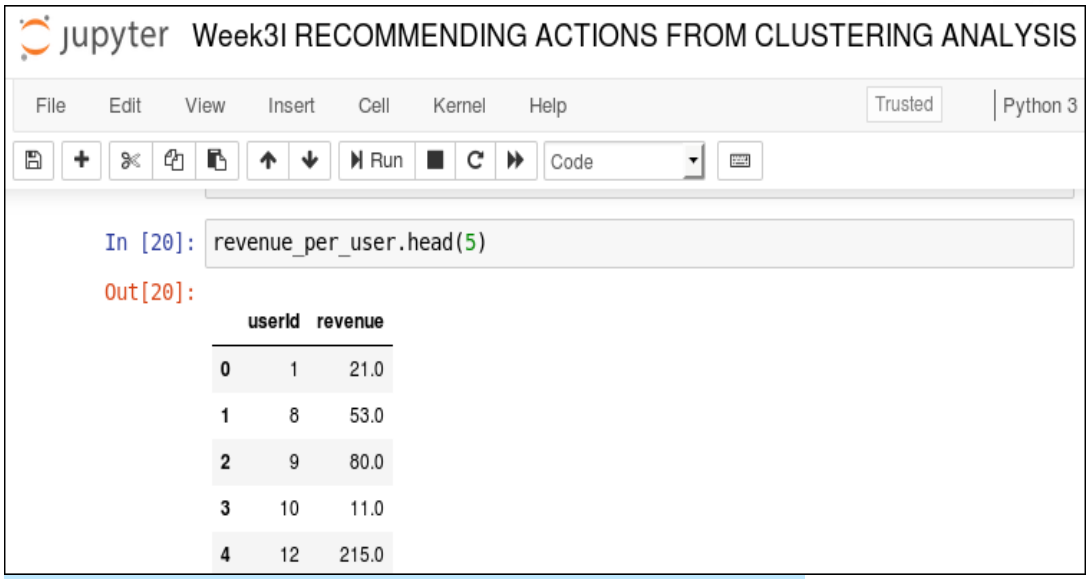

| | userId | adCount |
|---|---|---|
| 0 | 611 | 1 |
| 1 | 1874 | 1 |
| 2 | 2139 | 1 |
| 3 | 212 | 1 |
| 4 | 1027 | 1 |

- Sum the target columns per User Id
  - `game_clicks_per_user = user_gameClicks.groupby('userId').sum()`

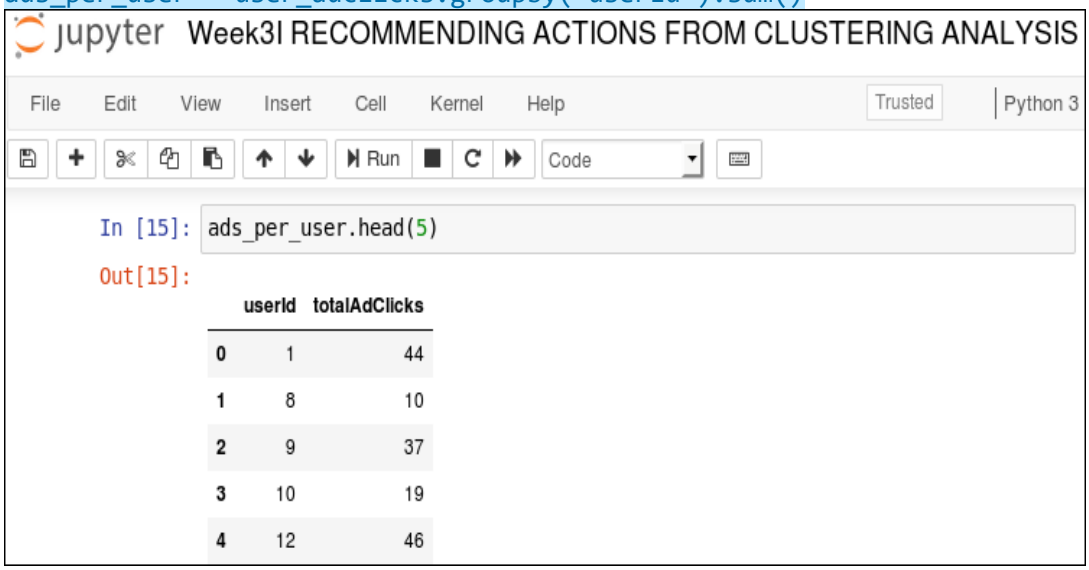



| | userId | totalGameClicks |
|---|---|---|
| 0 | 0 | 1355 |
| 1 | 1 | 716 |
| 2 | 2 | 231 |
| 3 | 6 | 151 |
| 4 | 8 | 380 |

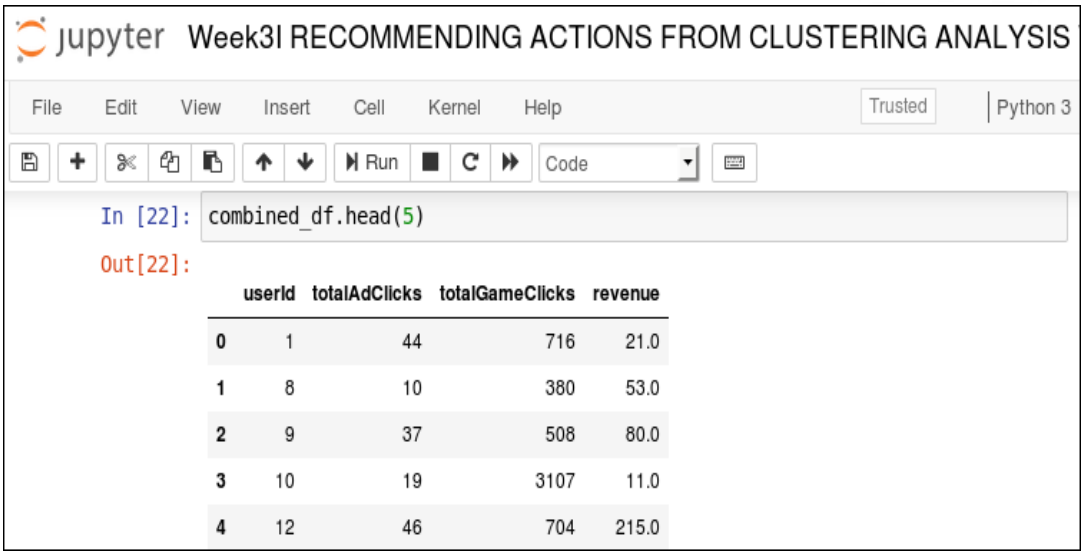
  - `revenue_per_user = user_purchases.groupby('userId').sum()`




- `ads_per_user = user_adclicks.groupby('userId').sum()`

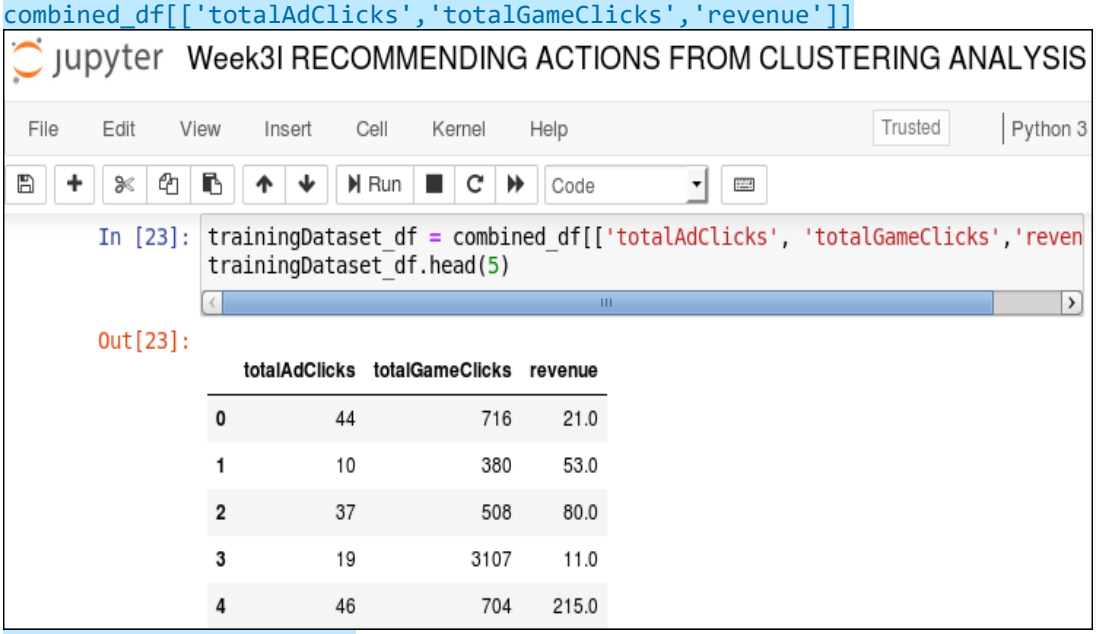



- Merge the datasets into one for the analysis
  - `combined_df = hits_per_user.merge(revenue_per_user, on='userId')`



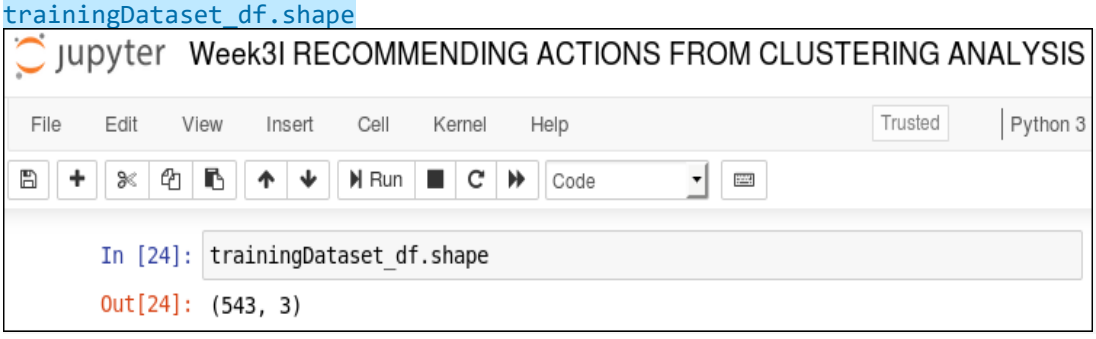

| | userId | totalAdClicks | totalGameClicks | revenue |
|---|---|---|---|---|
| 0 | 1 | 44 | 716 | 21.0 |
| 1 | 8 | 10 | 380 | 53.0 |
| 2 | 9 | 37 | 508 | 80.0 |
| 3 | 10 | 19 | 3107 | 11.0 |
| 4 | 12 | 46 | 704 | 215.0 |

- Dimensions of the final training dataset
  - `trainingDataset_df = combined_df[['totalAdClicks','totalGameClicks','revenue']]`




| | totalAdClicks | totalGameClicks | revenue |
|---|---|---|---|
| 0 | 44 | 716 | 21.0 |
| 1 | 10 | 380 | 53.0 |
| 2 | 37 | 508 | 80.0 |
| 3 | 19 | 3107 | 11.0 |
| 4 | 46 | 704 | 215.0 |

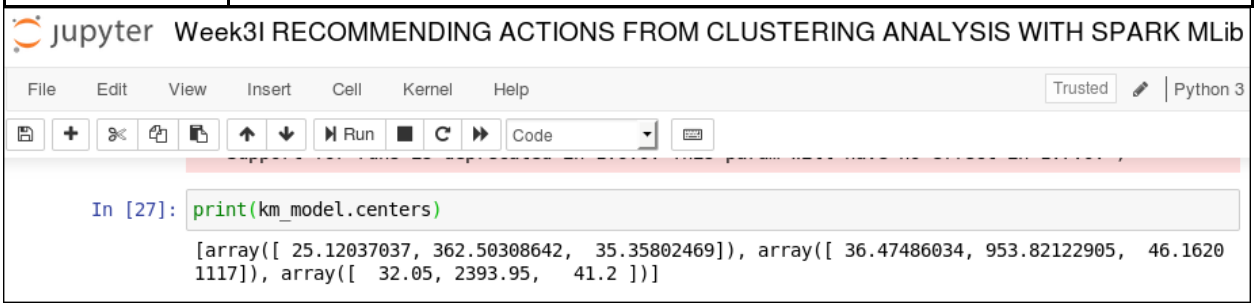
  - `trainingDataset_df.shape`



- **No. of clusters**: 3 [High_level_spenders/ Neutral_users/ Low_level_spenders]

## CH 3.2.2: CLuster Centers

| Cluster | Center |
|---|---|
| 1 | [ 25.12037037, 362.50308642, 35.35802469] |
| 2 | [ 32.05, 2393.95, 41.2 ] |
| 3 | [ 36.47486034, 953.82122905, 46.16201117] |




**These clusters can be differentiated from each other as follows:**

For each array, (field1) refers to ad-clicks/user, (field2) refers to game-clicks/user, and (field3) refers to revenue/user in the game.

- **1ST Cluster** is centered at array([ 25.12037037, 362.50308642, 35.35802469 ]).

  It is different from other clusters in that the users with the less game-clicks also produces the less revenue and ad-clicks count. This cluster of users can be called "**Low_level_spenders**".

- **2ND Cluster** is centered at array([ 32.05, 2393.95, 41.2 ]).

  It is different from other clusters in that the ad-clicks is not the least, game-clicks is the most, but their revenue is in the mid-range. This cluster of users can be called "**Neutral_users**".

- **3RD Cluster** is centered at array([ 36.47486034, 953.82122905, 46.16201117 ]).

  It is different from the other clusters in that the users' ad-clicks, game-clicks and revenue/user are all more than others, this kind of users can be called "**High_level spenders**".

## CH 3.3: Recommended Actions

| Actions Recommended | Rationale for the action |
|---|---|
| **1.** Offering more higher-price ads and limited products to users described with **High_level_spenders** | For high_level_spenders, there should be higher-price ads for limited products and subscriptions so we could significantly increase revenue/user while they are not playing that much. |
| **2.** Offering limited promotions and low-price packages to users described with **Low_level_spenders** | While low_level_spenders are playing the most, there should be more ads for limited promotions and lower-price packages so theY could be involved more to the game, following promotions, and paying more for worth experience. |

# CHAPTER 4: CHAT DATA GRAPH ANALYTICS| NEO4J 4.0.0

---

## CH 4.1: Modeling Chat Data using a Graph Data Model

We are using Neo4j's graph query language: Cypher for graph analytics of catch the pink flamingo chat data, building a graph data model illustrating how users are involved and interact with chat data of the game. For a user assigned to a team, this user could:

- join/leave an existed team-chat session.
- initially create a chat session, then create a chat item in the session.
- be mentioned by a chat item, and a chat item can be a response to another chat item created by another user in the same team.

## CH 4.2: Creation of the Graph Database for Chats

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

- Write the schema of the 6 CSV files

| File Name | Fields' Description |
|---|---|
| **1. chat_create_team_chat.csv** | ▪ userid: the id assigned to a user.<br>▪ teamid: the id assigned to a team.<br>▪ teamChatSessionID: a unique id assigned to a chat session. |

| | |
|---|---|
| | ▪ timestamp: a timestamp denoting when a chat session created. |
| **2. chat_item_team_chat.csv** | ▪ userid: the id assigned to a user.<br>▪ teamChatSessionID: a unique id assigned to a chat session.<br>▪ chatItemid: a unique id for the chat item.<br>▪ timestamp: a timestamp denoting when a chat item created. |
| **3. chat_join_team_chat.csv** | ▪ userid: the id assigned to a user.<br>▪ teamChatSessionid: a unique id assigned to a chat session.<br>▪ timestamp: a timestamp denoting when a user joins a chat session. |
| **4. chat_leave_team_chat.csv** | ▪ userid: the id assigned to a user.<br>▪ teamChatSessionid: a unique id assigned to a chat session.<br>▪ timestamp: a timestamp denoting when a user leaves a chat session. |
| **5. chat_mention_team_chat.csv** | ▪ chatItemid: the id assigned to a chat item.<br>▪ userid: the id assigned to a user.<br>▪ timestamp: a timestamp denoting when a user mentioned by a chat item. |
| **6. chat_respond_team_chat.csv** | ▪ chatid1: the id assigned to a chat post1.<br>▪ chatid2: the id assigned to a chat post2.<br>▪ timestamp: a timestamp denoting when the chat post1 responds to the chat post2. |

- Explain the loading process and include a sample LOAD command

Using Cypher Query Language to load the CSV data into neo4j, each row of script is parsed for refine the nodes, the edges and its timestamp. Let's consult the following script as an example:

```
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)
;
```

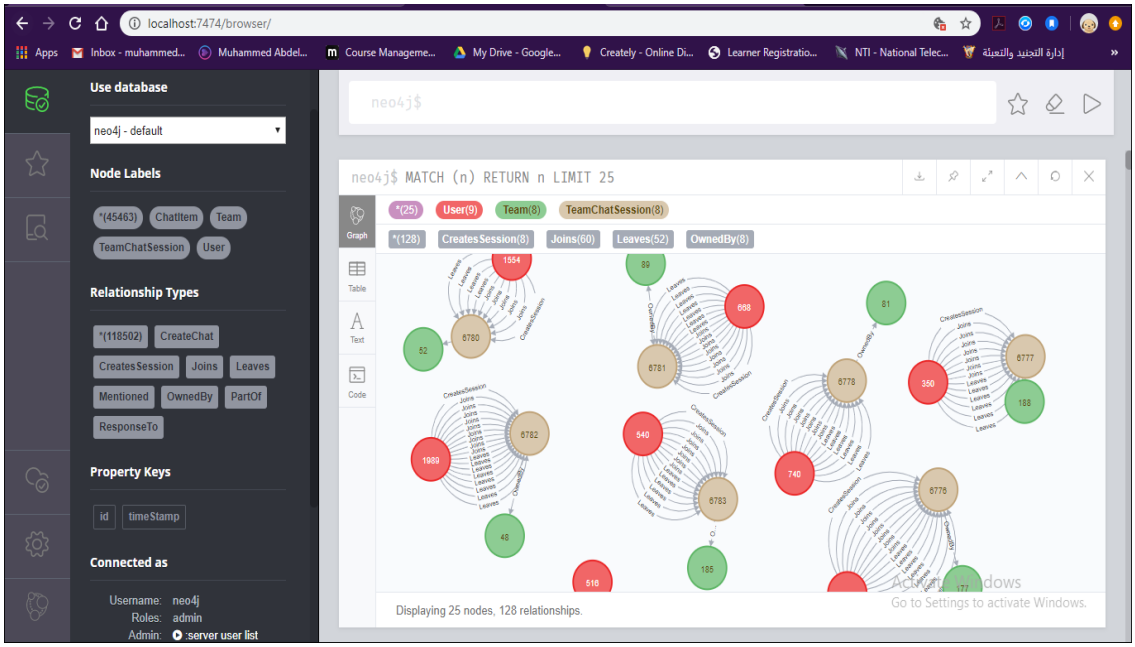
✓ The first line gives the path of the file, this command reads the chat_item_team_chat.csv at a time and create user nodes. The 0th column value is

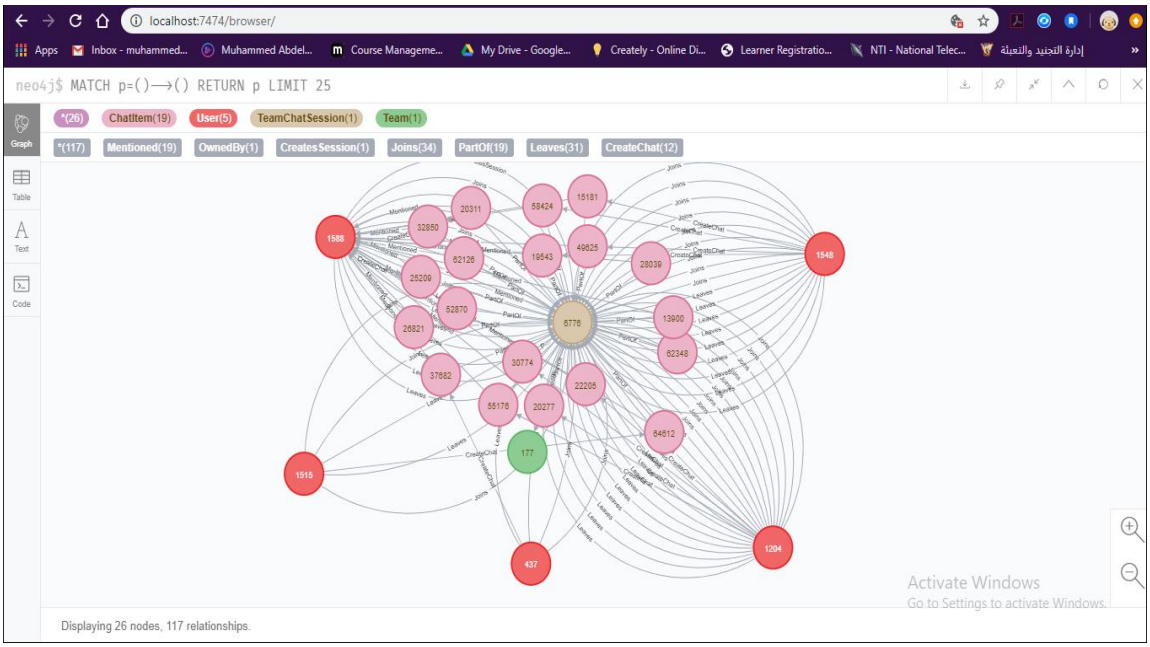
converted to an integer and is used to populate the id attribute. Similarly, the other nodes are created.

- ✓ Line 5, `MERGE (u)-[:CreateChat{timeStamp: row[3]}]->(i)` creates an edge labeled "CreateChat" between the User node u and the ChatItem node i. This edge has a property called timeStamp. This property is filled by the content of column 3 of the same row.

- ✓ Line 6, `MERGE (i)-[:PartOf{timeStamp: row[3]}]->(c)` creates an edge labeled "PartOf" between the ChatItem node i and the TeamChatSession node c. This edge has a property called timeStamp. This property is filled by the content of column 3 of the same row.

- 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. Below are two acceptable examples:



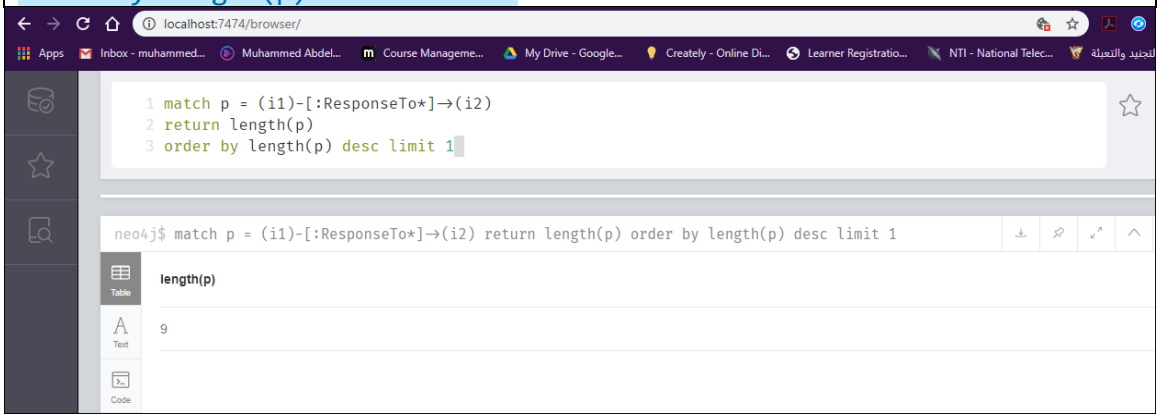



The first example is a rendered in Neo4j 4.0.0 distribution, the second has had some nodes moved to expose the edges more clearly. Both include examples of most node and edge types.

## CH 4.3: The longest conversation chain and its participants

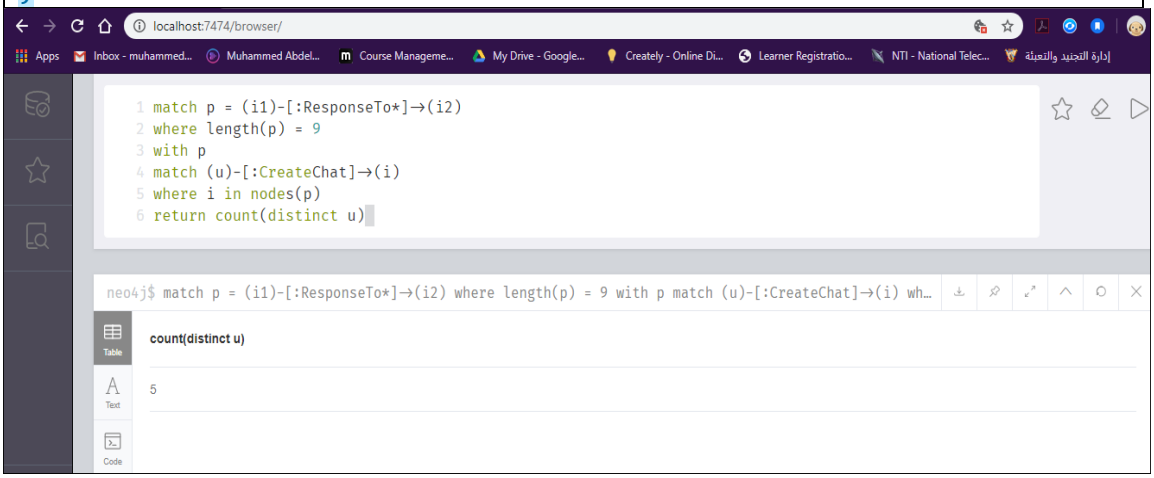
The longest conversation chain is traced via ChatItem nodes which are connected by ResponseTo edges, order the length and find the longest one. Follow the query shown below to get the longest conversation chain with **length of 9**, and the longest conversation with **10 chats**.

```
match p = (i1)-[:ResponseTo*]->(i2)
return length(p)
order by length(p) desc limit 1
```



Since we've already known the longest conversation chain has path **length of 9**, next we will find all the distinct users which are part of this path. Follow the query shown below **5 unique users**.

```
match p = (i1)-[:ResponseTo*]->(i2)
where length(p) = 9
with p
match (u)-[:CreateChat]->(i)
where i in nodes(p)
return count(distinct u)
;
```



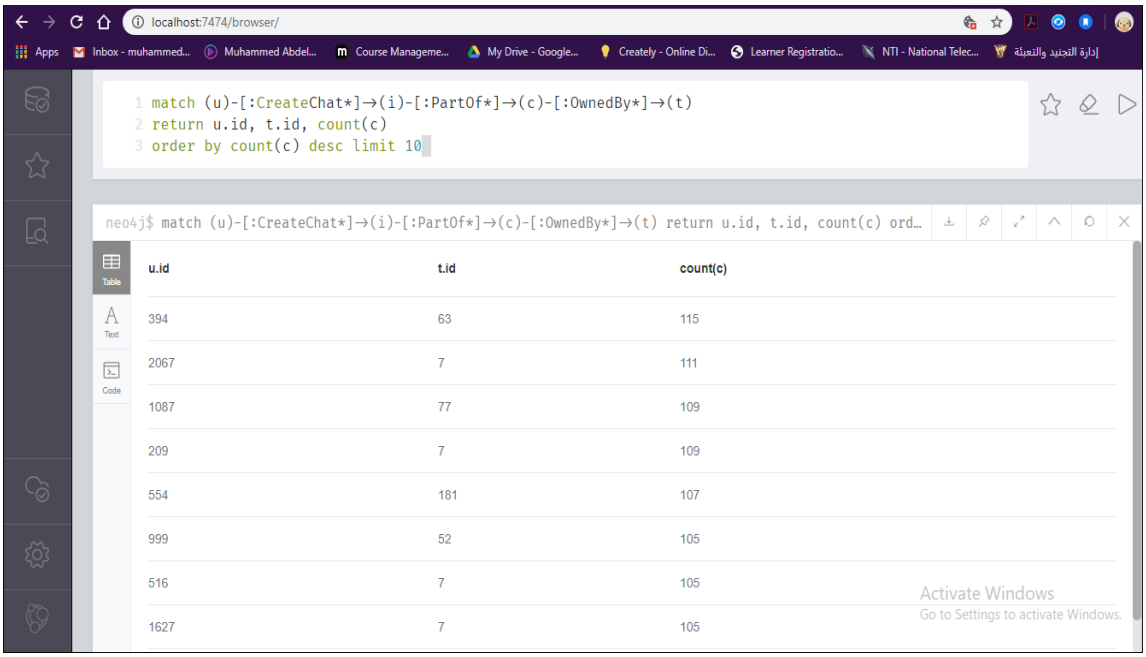

## CH 4.4: Relationship between top 10 chattiest users and top 10 chattiest teams:

Describe your steps from Question 2. In the process, create the following two tables. You only need to include the top 3 for each table. Identify and report whether any of the chattiest users were part of any of the chattiest teams.

- **Chattiest Users:**

  Firstly, we match the CreateChat edge from User node to ChatItem node, then return the ChatItem amount per user, and order by the amount in descending order. Follow the query shown below to fulfill this table next:

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

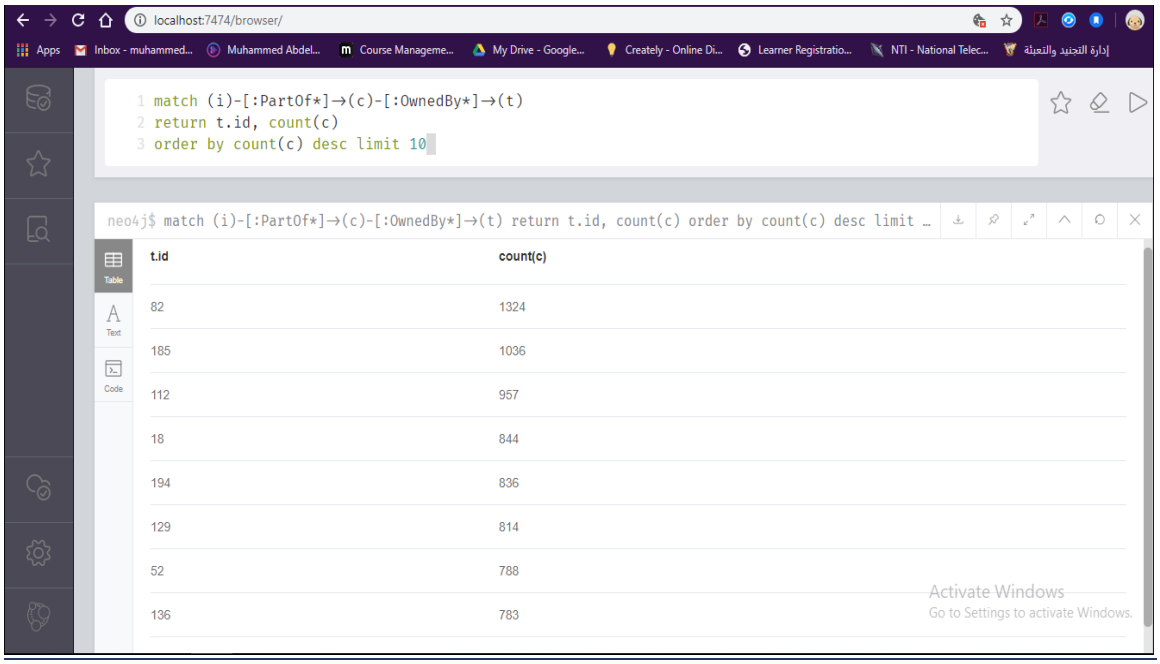



| Users | Number of Chats |
|---|---|
| 394 | 115 |
| 2067 | 111 |
| 1087 | 109 |
| 209 | 109 |

- **Chattiest Teams:**

  Firstly, we match the CreateChat edge from User node to ChatItem node, then return the ChatItem amount per user, and order by the amount in descending order following the query shown below:

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

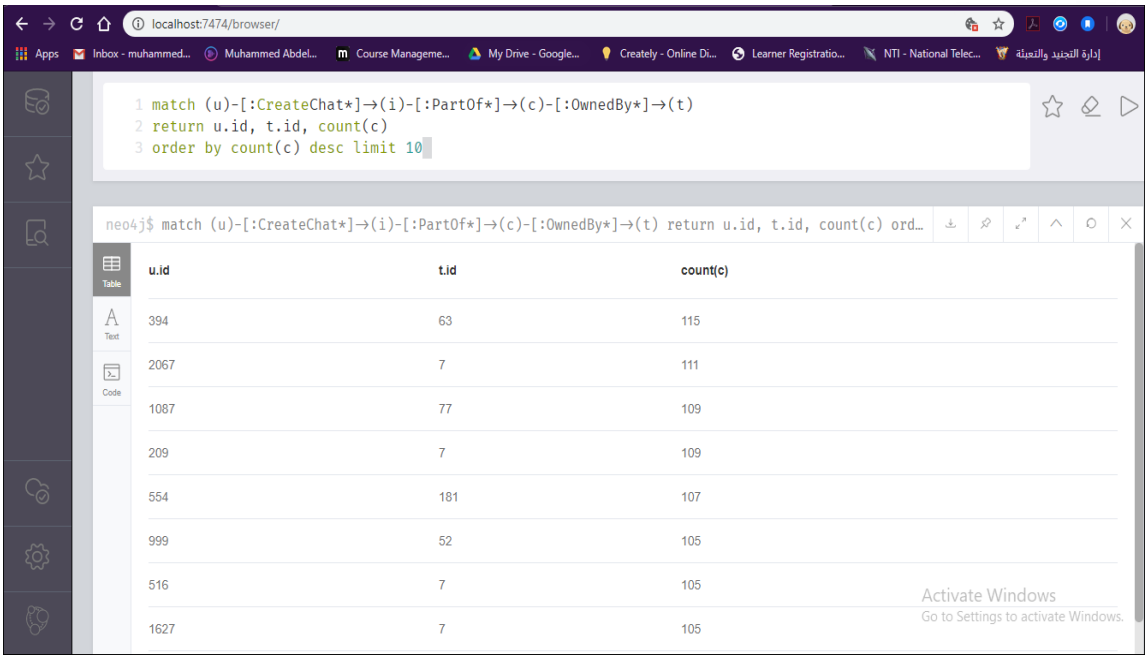



| Teams | Number of Chats |
|---|---|
| 82 | 1324 |
| 185 | 1036 |
| 112 | 957 |

## Final Result:

All we need is to combine the two queries above together as follows:

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




As result shows, the user with **U.id = 999**, which in the team with **T.id = 52** is part of the top 10 chattiest teams, but other 9 users are not part of the top 10 chattiest teams. This states that most of the chattiest users are not in the chattiest teams.

## CH 4.5: How Active Are Groups of Users?

In order to answer this question, 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. Follow these steps explained below:

1. 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 CreateChat in response to another user's ChatItem.

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

   - One user created a ChatItem in response to another user's ChatItem.

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

2. The above scheme will create an undesirable side effect if a user has responded to her own chatItem, because it will create a self-loop between two users. So, after the first two steps we need to eliminate all self-loops involving the edge "InteractsWith".

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

For each of these neighbors, we need to find:

- the number of edges it has with the other members on the same list:

```
match (u1:User)-[r1:InteractsWith]->(u2:User)
where u1.id <> u2.id
with u1, collect(u2.id) as neighbors, count(distinct(u2)) as
neighborAmount
match (u3:User)-[r2:InteractsWith]->(u4:User)
where (u3.id in neighbors)
AND (u4.id in neighbors)
AND (u3.id <> u4.id)
return u3.id, u4.id, count(r2)
```

- If one member has multiple edges with another member, we need to count it as 1 because we care only if the edge exists or not:

```
match (u1:User)-[r1:InteractsWith]->(u2:User)
where u1.id <> u2.id
with u1, collect(u2.id) as neighbors, count(distinct(u2)) as
neighborAmount
match (u3:User)-[r2:InteractsWith]->(u4:User)
where (u3.id in neighbors)
AND (u4.id in neighbors)
AND (u3.id <> u4.id)
return u3.id, u4.id, count(r2),
case
when count(r2) > 0 then 1
else 0
end as value
```

- The last step to program this computation in cypher, we need to combine all queries above together:

```
match (u1:User)-[r1:InteractsWith]->(u2:User)
where u1.id <> u2.id
with u1, collect(u2.id) as neighbors, count(distinct(u2)) as
neighborAmount
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, neighborAmount,
case
when (u3)-->(u4) then 1
```

```
else 0
end as value
return  u1,  sum(value)*1.0/(neighborAmount*(neighborAmount-1))  as
coeff
order by coeff desc limit 10
```

- Most active users (based on Cluster Coefficients).

| User ID | Coefficient |
|---|---|
| 209 | 0.9523809523809523 |
| 554 | 0.9047619047619048 |
| 1087 | 0.8 |

# CHAPTER 5: RECOMMENDED ACTIONS

I. Offer iPhone users more packages and limited promotions$.
**Explain:**
**Splunk〉:** the data exploration analysis reflects that most users with iphone platform are high-level spenders; offering them more products with higher price ads will increase revenue/user.

II. Add more products with higher-price ads$ to "High_level_spenders" & "HighRollers".
**Explain:**
**Knime SparkMLib :** according to classification and clustering analysis, high-level spenders clicked less and buy most; adding more ads for higher-price products will increase revenue/user.

III. Encourage low-level spenders and PennyPinchers with fixed pay packages and limited promotion to users.