

# 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
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.	<p>timestamp: a timestamp denoting when the event occurred.</p> <p>userSessionId: a unique id for the session.</p> <p>userId: the current user's ID.</p> <p>teamId: the current user's team.</p> <p>assignmentId: the team assignment id for the user to the team.</p> <p>sessionType: whether the event is the start or end of a session.</p> <p>teamLevel: the level of the team during this session.</p> <p>platformType: the type of platform of the user during this session.</p>
Ad-clicks.csv	A line is added to this file when a player clicks on an advertisement in the Flamingo app.	<p>timestamp: when the click occurred.</p> <p>txId: a unique id (within ad-clicks.log) for the click</p> <p>userSessionid: the id of the user session for the user who made the click</p> <p>teamid: the current team id of the user who made the click</p> <p>userid: the user id of the user who made the click</p> <p>adId: the id of the ad clicked on</p>

		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.	<p>timestamp: when the purchase was made.</p> <p>txId: a unique id (within buy-clicks.log) for the purchase</p> <p>userSessionId: the id of the user session for the user who made the purchase</p> <p>team: the current team id of the user who made the purchase</p> <p>userId: the user id of the user who made the purchase</p> <p>buyId: the id of the item purchased</p> <p>price: the price of the item purchased</p>
Users.csv	This file contains a line for each user playing the game.	<p>timestamp: when user first played the game.</p> <p>userId: the user id assigned to the user.</p> <p>nick: the nickname chosen by the user.</p> <p>twitter: the twitter handle of the user.</p> <p>dob: the date of birth of the user.</p> <p>country: the two-letter country code where the user lives.</p>
Team.csv	This file contains a line for each team terminated in the game.	<p>teamId: the id of the team</p> <p>name: the name of the team</p> <p>teamCreationTime: the timestamp when the team was created</p>

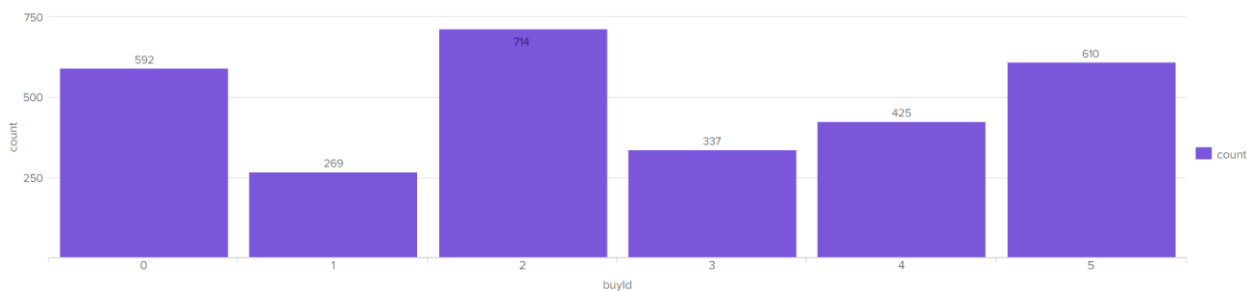
		<p>teamEndTime: the timestamp when the last member left the team</p> <p>strength: a measure of team strength, roughly corresponding to the success of a team</p> <p>currentLevel: the current level of the team</p>
Team-assignments.csv	A line is added to this file each time a user joins a team. A user can be in at most a single team at a time.	<p>timestamp: when the user joined the team.</p> <p>team: the id of the team</p> <p>userId: the id of the user</p> <p>assignmentId: a unique id for this assignment</p>
Level-events.csv	A line is added to this file each time a team starts or finishes a level in the game	<p>timestamp: when the event occurred.</p> <p>eventId: a unique id for the event</p> <p>teamId: the id of the team</p> <p>teamLevel: the level started or completed</p> <p>eventType: the type of event, either start or end</p>
Game-clicks.csv	A line is added to this file each time a user performs a click in the game.	<p>timestamp: when the click occurred.</p> <p>clickId: a unique id for the click.</p> <p>userId: the id of the user performing the click.</p> <p>userSessionId: the id of the session of the user when the click is performed.</p> <p>isHit: denotes if the click was on a</p>

		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
--	--	---

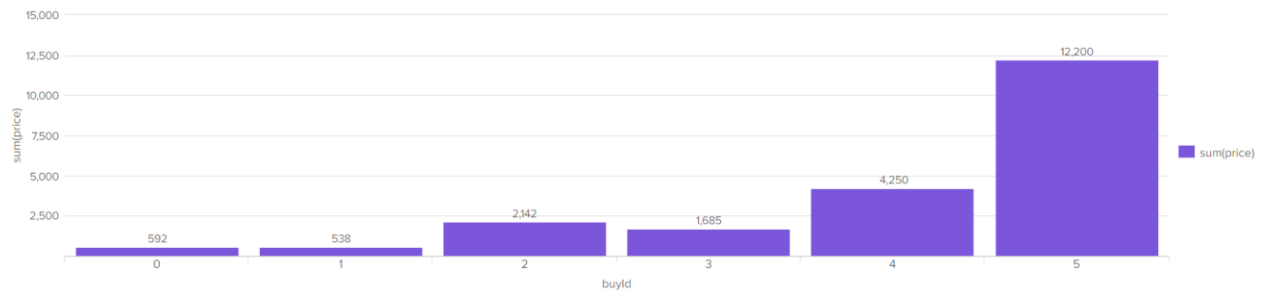
## Aggregation

Amount spent buying items	\$21407
Number of unique items available to be purchased	6
Company revenue made from the purchased in buy-clicks.csv	\$428
Item cost (max)	\$20
Item cost (min)	\$1

A histogram showing how many times each item is purchased:

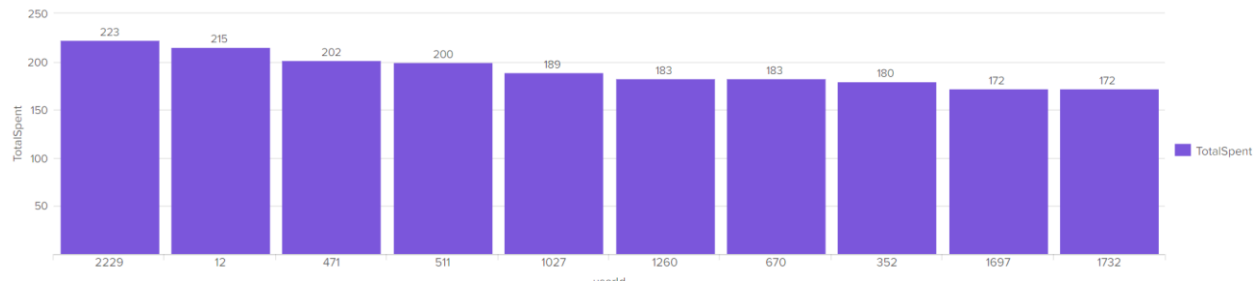


A histogram showing how much money was made from each item:



## 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	11.60
2	12	iphone	13.07
3	471	iphone	14.50

# Classification

## Data Preparation

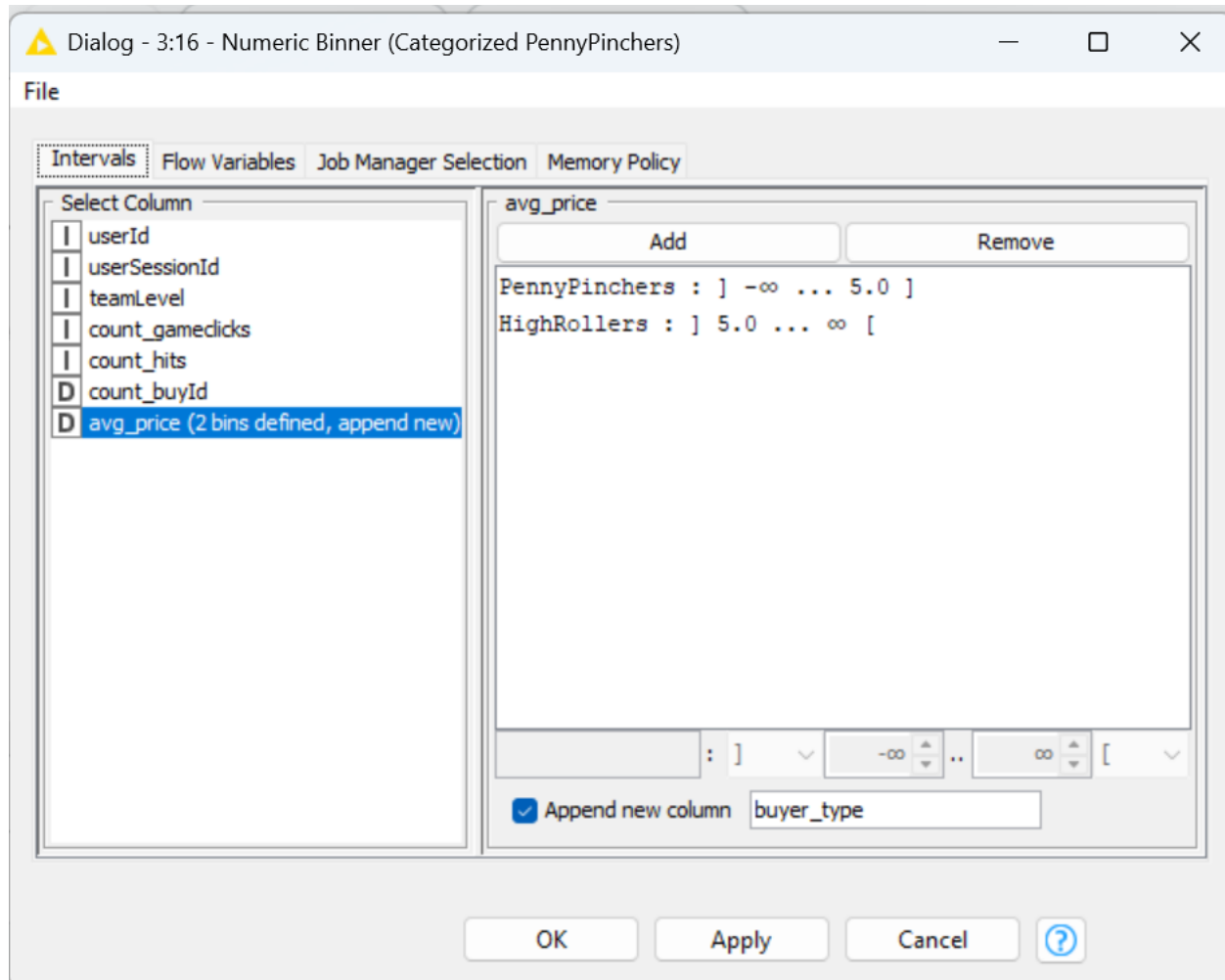
Analysis of combined\_data.csv

## 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:



PennyPinchers are defined as those whose purchase items \$5.00 or less.  
 Highrollers are defined as those who purchase items greater than \$5.00.  
 A new column attribute “buyer\_type” created to categorize each sample either with PennyPinchers or HighRollers.

The creation of this new categorical attribute was necessary because our goal is to predict whether a user is likely to buy expensive items (HighRollers) or cheaper ones (PennyPinchers). Classification models required a clear target variable (buyer\_type) to train on.

### Attribute Selection

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

Attribute	Rationale for Filtering
userId	These are unique identifiers but hold no predictive value for spending habits. They do not affect whether a user is HighRoller or PennyPincher.

userSessionId	It doesn't provide any meaningful features for classification. A session ID do not indicate whether a user will purchase expensive items.
Team_level	While this tracks game progression, it may not directly correlate with spending behavior. If there's no strong relationship between team level and purchases, it should be removed.
Avg_price	A new column attribute "buyer_type" was converted from this column. Therefore, the avg_price column is redundant.



## Data Partitioning and Modeling

The data was partitioned into train and test datasets.

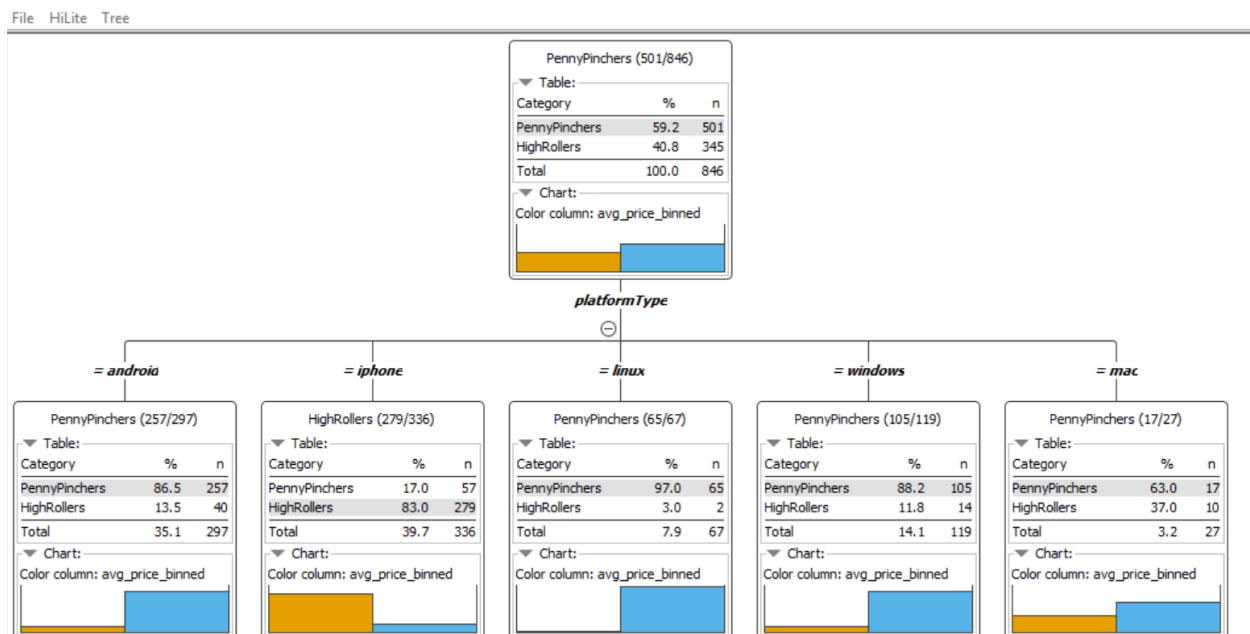
The training data set was used to create the decision tree model.

The trained model was then applied to the test dataset.

This is important because splitting a dataset into training and testing sets is a critical step in machine learning as to ensure the model is evaluated fairly and performs well in real world scenarios.

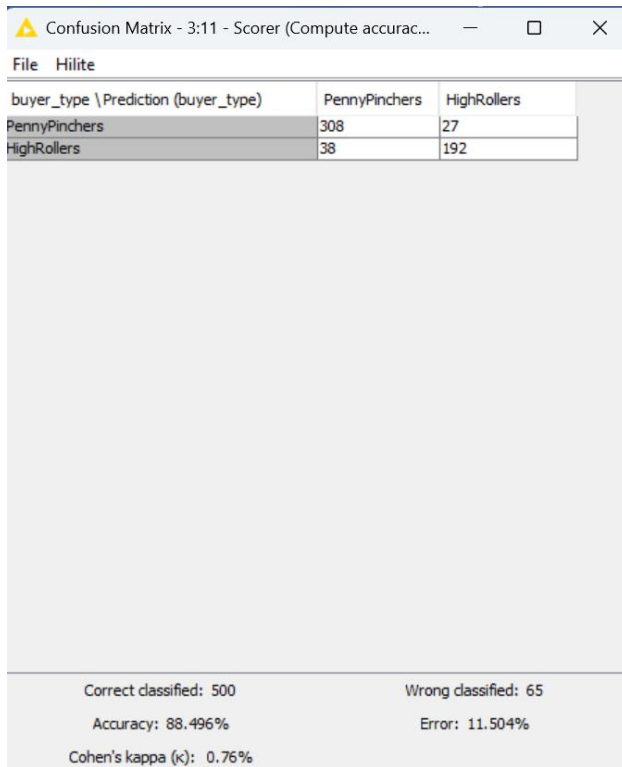
When partitioning the data using sampling, it is important to set the random seed because to ensure the split is reproducible every time we run the workflow in KNIME and also consistency.

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 - 3:11 - Scorer (Compute accurac...)". Inside, there is a table with the following data:

buyer_type \ Prediction (buyer_type)	PennyPinchers	HighRollers
PennyPinchers	308	27
HighRollers	38	192

Below the table, the following summary statistics are displayed:

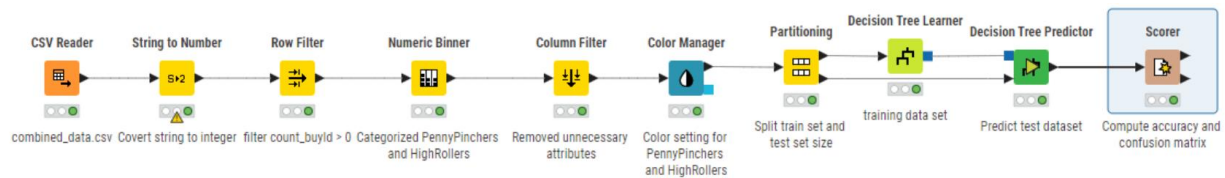
- Correct classified: 500
- Wrong classified: 65
- Accuracy: 88.496%
- Error: 11.504%
- Cohen's kappa ( $\kappa$ ): 0.76%

As seen in the screenshot above, the overall accuracy of the model is 88.50%.

- 308 PennyPinchers were correctly predicted as PennyPinchers, meaning the model accurately identified them. (True Negative)
- 27 PennyPinchers were incorrectly predicted as HighRollers, suggesting the model misclassified them as big spenders. (False Positive)
- 38 HighRollers were incorrectly predicted as PennyPinchers, meaning the model underestimated their spending behaviour. (False Negative)
- 192 HighRollers were correctly predicted as HighRollers, showing the model effectively recognized big spenders. (True Positive)

## Analysis Conclusions

The final KNIME workflow is shown below:



What makes a HighRoller vs. a PennyPincher?

iPhone Users Are More Likely to Be HighRollers

- 83% of iPhone users purchase big-ticket items (HighRollers).
- This suggests iPhone users tend to spend more in the game.

Android, Windows, and Linux Users Are Predominantly PennyPinchers

- Android: 86.5% PennyPinchers
- Windows: 88.2% PennyPinchers
- Linux: 97% PennyPinchers
- Users on these platforms mostly buy inexpensive items.

Specific Recommendations to Increase Revenue
1. Targeted Promotions for HighRollers (iPhone Users) <ul style="list-style-type: none"><li>- Since iPhone users are far more likely to spend on big-ticket items, Eglence should focus marketing and in-app incentives toward them.</li><li>- Implement exclusive limited-time offers, premium bundles, and VIP perks tailored for HighRollers.</li></ul>
2. Convert PennyPinchers into Higher Spenders (Android, Windows, Linux Users) <ul style="list-style-type: none"><li>- Discounted bundle deals (e.g., "Buy 2 small items, get 1 big-ticket item at a discount").</li><li>- Loyalty-based rewards (e.g., spend \$5+ over time to unlock exclusive items).</li></ul>

# Clustering

## Attribute Selection

features\_used = ["ad\_click\_count", "inapp\_purchase\_revenue", "revenue\_per\_adclick"]

Attribute	Rationale for Selection
ad_click_count	Represents the total number of ad clicks per user. Directly conveys the overall engagement with ads.
InApp_Purchase_Revenue	Captures the total revenue generated by the user through in-app purchases. Indicates this measure tracks monetary contributions.
Revenue_Per_AdClick	Denotes the average revenue obtained per ad click (i.e., total revenue divided by the total ad clicks). Reflects the conversion efficiency of ad interactions into revenue.

## Training Data Set Creation

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

Create a Derived Feature

```
✓ [32] training_df = training_df.withColumn("Revenue_Per_AdClick",  
0s      when(col("Ad_Click_Count") != 0, col("InApp_Purchase_Revenue") / col("Ad_Click_Count"))  
      .otherwise(0.0)  
      )
```

```
✓ 1s ▶ training_df.show(5)
```

```
↗ +-----+-----+-----+-----+  
|userId|Ad_Click_Count|InApp_Purchase_Revenue|Revenue_Per_AdClick|  
+-----+-----+-----+-----+  
|    1|          44|             21.0|    0.47727272727273|  
|    8|          10|             53.0|              5.3|  
|    9|          37|             80.0|    2.1621621621621623|  
|   10|          19|             11.0|    0.5789473684210527|  
|   12|          46|            215.0|    4.673913043478261|  
+-----+-----+-----+-----+  
only showing top 5 rows
```

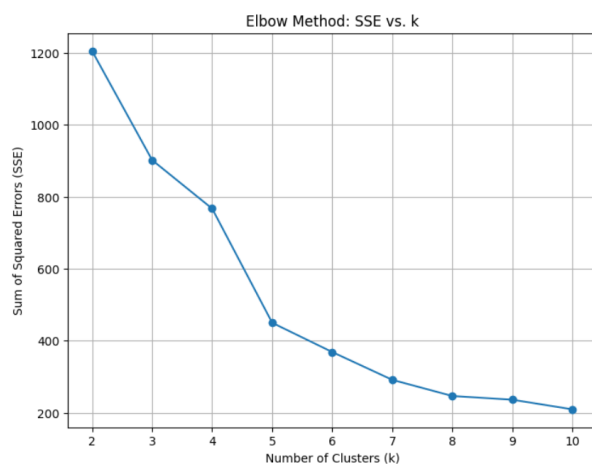
Dimensions of the final data set:

```
▶ num_rows = training_df.count()  
  num_cols = len(training_df.columns)  
  print("Dimensions (rows x columns):", num_rows, "x", num_cols)
```

```
↗ Dimensions (rows x columns): 600 x 4
```

# of clusters created: 4

Using Elbow method, K = 4 is chosen as an optimal value for this case.



## Cluster Centers

The code used in creating cluster centers is given below:

Based on the Elbow method plot above, K = 4 looks good. We can now perform KMeans clustering to generate 4 clusters:

```
✓ [29] kmeans = KMeans(k=4, seed=1)
56s model = kmeans.fit(scaledData)
transformed = model.transform(scaledData)
```

Print the center of these four clusters:

```
✓ [30] centers = model.clusterCenters()
0s print("Cluster Centers:")
for idx, center in enumerate(centers):
    print(f"Cluster {idx+1}: {center}")
```

```
➔ Cluster Centers:
Cluster 1: [-0.19744081 -0.27800498 -0.17299374]
Cluster 2: [0.22539263 1.79276725 1.55888795]
Cluster 3: [-1.17500884 -0.7201748 -0.31932685]
Cluster 4: [ 1.09493924  0.00255386 -0.32795829]
```

Cluster centers formed are given in the table below

Cluster #	Center
1	[-0.19744081 -0.27800498 -0.17299374]
2	[0.22539263 1.79276725 1.55888795]
3	[-1.17500884 -0.7201748 -0.31932685]
4	[ 1.09493924  0.00255386 -0.32795829]

These clusters can be differentiated from each other as follows:

### **Cluster 1: [-0.1974, -0.2780, -0.1729]**

Users in this cluster are slightly below the overall average on every metric—ad clicks, in-app purchase revenue, and revenue per ad click. Although their behaviour is broadly “average,” they tend to be a bit less active and generate slightly less revenue than a typical user.

### **Cluster 2: [0.2254, 1.7928, 1.5589]**

This group has a modestly above-average number of ad clicks (first dimension) but stands out with very high values in both revenue and revenue per ad click. In other words, the users in Cluster 2 spend substantially more than average and are highly efficient in converting ad interactions into revenue. They can be considered your “high rollers” or premium users.

### **Cluster 3: [-1.1751, -0.7202, -0.3193]**

With strongly negative values—especially for the first dimension (ad clicks)—users in Cluster 3 are significantly less active and generate much lower revenue and conversion efficiency than average. These users seem to be low-engagement, low-spending customers.

#### Cluster 4: [1.0949, 0.0026, -0.3280]

Although the users in this cluster have a high number of ad clicks (a value well above the overall mean), their revenue is around the global average and their revenue per ad click is somewhat below average. This pattern suggests that they are very active in clicking ads, but their spending behaviour does not follow suit. They may represent a segment that is highly engaged in terms of interactions but not converting those interactions into higher spending.

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

▶ training\_df.describe().show()

summary	userId	Ad_Click_Count	InApp_Purchase_Revenue	Revenue_Per_AdClick
count	600	600	600	600
mean	1207.975	27.205	35.678333333333335	1.4559759635271425
stddev	686.534288965143	16.048537865260123	40.830422516718	1.9522302773363274
min	1	0	0.0	0.0
max	2387	67	223.0	25.0

## Recommended Actions

Action Recommended	Rationale for the action
For the users in Cluster 2, design and promote premium in-app offers (such as exclusive bundles, or limited-time packages) tailored to their demonstrated willingness to spend.	As Cluster 2 shows <b>much higher-than-average spending and conversion</b> , this group represents your “high rollers.” Enhancing their experience with premium offers could capture even more value, thereby directly increasing revenue.
For users in Cluster 4, implement targeted conversion strategies. These may include personalized discount offers, loyalty rewards triggered by reaching a threshold of ad clicks, or nudges that suggest bundled purchase options.	Although this cluster is very active in engaging with ads, they do not convert that engagement into high spending. By introducing incentives or by optimizing the user journey to reduce barriers to purchase, the company can indirectly drive up the average revenue per user within this segment, thereby increasing overall revenue.



# Graph Analytics

## Modeling Chat Data using a Graph Data Model

In this chat data graph model, conversations are represented using nodes and relationships.

- Users are nodes (**User**).
- Chat messages are nodes (**ChatItem**).
- Users create chat messages, forming edges (**CreateChat**).
- Replies between messages form edges (**ResponseTo**).
- Mentions of other users in chat create edges (**Mentioned**).
- Messages belong to teams or sessions, forming edges (**PartOf** and **OwnedBy**).

This structure allows efficient analysis of conversations, including tracking response chains, identifying active users, and detecting community patterns.

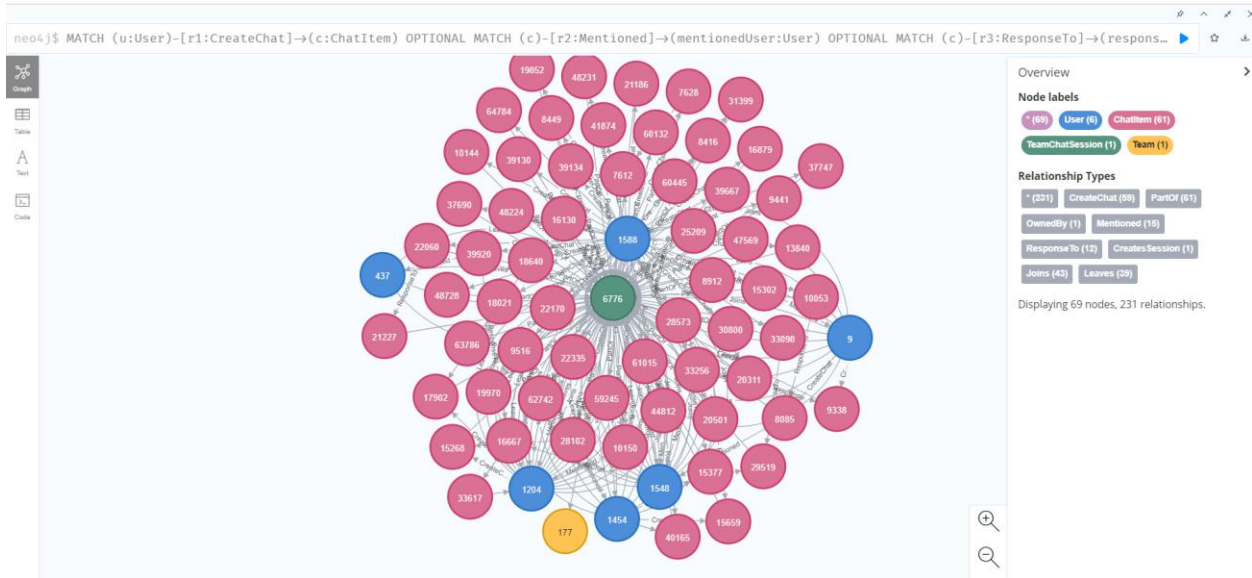
## Creation of the Graph Database for Chats

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

- i) Write the schema of the 6 CSV files  
The database is structured using nodes and relationships, and CSV files hold the data to populate them.
  1. User.csv
    - a. UserID (Unique identifier)
    - b. Name (Optional username)
  2. ChatItems.csv
    - a. ChatID (Unique identifier for each message)
    - b. Text (Message content)
    - c. Timestamp (Time sent)
  3. Teams.csv
    - a. TeamID (Unique identifier)
    - b. TeamName (Name of the chat group)
  4. CreateChat.csv (Relationships)
    - a. UserID - > ChatID (CreateChat relationship)
  5. Mentioned.csv (Relationships)
    - a. ChatID - > UserID (Mentioned relationship)
  6. ResponseTo.csv (Relationships)
    - a. ChatID\_A - > ChatID\_B (ResponseTo relationship)
- ii) Explain the loading process and include a sample LOAD command  
To load data, we use Neo4j "LOAD CSV" function.  
Below is a sample LOAD command used.  
LOAD CSV WITH HEADERS FROM 'file:///CreateChat.csv' AS row  
MATCH (u:User {id: row.UserID})  
MATCH (c:ChatItem {id: row.ChatID})  
MERGE (u)-[:CreateChat]->(c);

MATCH ensures nodes exist before creating relationships.  
MERGE prevents duplicate connections.

- 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. Below are two acceptable examples. The first example is rendered in the default Neo4j distribution, the second has had some nodes moved to expose the edges more clearly. Both include 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. Describe your steps. Write the query that produces the correct answer.

The longest conversation chain is 9. There are 5 unique users. We focus on the “ResponseTo” relationship, which connects chat messages.

### Steps Taken

#### 1. Find the longest conversation chain

- Using MATCH p = (start:ChatItem)-[:ResponseTo\*]->(end:ChatItem), we search for paths of any length (\* operator).
- The longest path is identified using ORDER BY length(p) DESC LIMIT 1.

#### 2. Extract unique users from the longest chain

- We unwind nodes (ChatItem) along the longest path.
- Then, we match users who created each ChatItem.
- count(DISTINCT u) ensures we count unique users only.

```

MATCH p = (start:ChatItem)-[:ResponseTo*]->(end:ChatItem)
RETURN length(p) AS LongestConversationLength
ORDER BY LongestConversationLength DESC
LIMIT 1;
MATCH p = (start:ChatItem)-[:ResponseTo*]->(end:ChatItem)
WITH p
ORDER BY length(p) DESC
LIMIT 1
UNWIND nodes(p) AS chatItem
MATCH (u:User)-[:CreateChat]->(chatItem)
RETURN count(DISTINCT u) AS UniqueUsersInLongestChain;

```

### Analyzing the 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.*

#### Steps taken:

1. **Retrieve the top 10 chattiest users**
  - Count the number of messages each user has sent.
  - Sort by descending chat count.
  - Select the top 10.
2. **Retrieve the top 10 chattiest teams**
  - Count the number of messages within each team.
  - Sort by descending chat count.
  - Select the top 10.
3. **Check for overlap**
  - Identify users who belong to any of the top 10 teams.
  - Report whether any top chatters are part of top chatty teams.

#### **Chattiest Users**

Users	Number of Chats
394	115
2067	111
1087	109

#### **Chattiest Teams**

Teams	Number of Chats
82	1324

185	1036
112	957

Finally, present your answer, i.e. whether or not any of the chattiest users are part of any of the chattiest teams.

UserID 999 is part of TeamID 52, which confirms some overlap between top chattiest users and top chattiest teams.

```
MATCH (u:User)-[:CreateChat]-(c:ChatItem)-[:PartOf]->(session:TeamChatSession)-[:OwnedBy]->(t:Team)
WHERE u.id IN [394, 2067, 1087, 209, 554, 1627, 999, 516, 668, 461] AND t.id IN [82, 185, 112, 18, 194, 129, 52, 136, 146, 81]
RETURN DISTINCT u.id AS UserID, t.id AS TeamID;
```

	UserID	TeamID
1	999	52

## How Active Are Groups of Users?

Describe your steps for performing this analysis. Be as clear, concise, and as brief as possible. Finally, report the top 3 most active users in the table below.

We analyze interactions within chat session using clustering coefficients and message counts.

1. Retrieve user activity levels
  - o Count messages sent by each user in group chats.
  - o Identify users with high engagement.
2. Compute clustering coefficients
  - o Measure how connected a user's neighborhood is.
  - o Users in tightly linked networks score higher.
  - o Select the top 10.
3. Rank the most active users
  - o Sort by message count & clustering coefficient.
  - o Identify top 3 active users.

### Most Active Users (based on Cluster Coefficients)

User ID	Coefficient
209	1
668	1
999	0.958

```
MATCH (u:User)-[:InteractsWith]->(neighbor:User)
WHERE u.id IN [394, 2067, 1087, 209, 554, 1627, 999, 516, 668, 461]
WITH u, collect(neighbor) AS neighbors, count(neighbor) AS k
UNWIND neighbors AS n1
```

```
UNWIND neighbors AS n2
WITH u, k, n1, n2
WHERE n1 <> n2
MATCH (n1)-[:InteractsWith]->(n2)
WITH u, k, count(*) AS actualEdges
RETURN u.id AS UserID,
       CASE WHEN k > 1 THEN actualEdges * 1.0 / (k * (k - 1)) ELSE 0 END AS ClusteringCoefficient
ORDER BY ClusteringCoefficient DESC;
```