BAX 423

What happens in Vegas stays in Venmo

Submitted by The Analyst Amigos

Text Analytics

Q0 - our first task is to open your Venmo app, find 10 words that are not already in the dictionary and add them to it. Make sure you don't add to the dictionary a duplicate word by hitting Control+F before adding your word

1. profile-People;
2. pickup-Activity;
3. san mateo-Activity;
4. hihi-People;
5. yumyum-Food;
6. sofa-Utility;
7. 5guy-Food;
8. additional-Cash;
9. tofu soup-Food;
10. shipment-Utility

Q1 - Use the text dictionary and the emoji dictionary to classify Venmo's transactions in your sample dataset.

We choose 1% of the data as sample data. Below are schemas we used.

```
Sample dataset schema:
root
 |-- user1: integer (nullable = true)
 |-- user2: integer (nullable = true)
 |-- transaction_type: string (nullable = true)
 |-- datetime: timestamp (nullable = true)
 |-- description: string (nullable = true)
 |-- is_business: boolean (nullable = true)
 |-- story_id: string (nullable = true)

Text dictionary schema:
root
 |-- People: string (nullable = true)
 |-- Food: string (nullable = true)
 |-- Event: string (nullable = true)
 |-- Activity: string (nullable = true)
 |-- Travel: string (nullable = true)
 |-- Transportation: string (nullable = true)
 |-- Utility: string (nullable = true)
 |-- Cash: string (nullable = true)
 |-- Illegal/Sarcasm: string (nullable = true)

Emoji dictionary schema:
root
 |-- Event: string (nullable = true)
 |-- Travel: string (nullable = true)
 |-- Food: string (nullable = true)
 |-- Activity: string (nullable = true)
 |-- Transportation: string (nullable = true)
 |-- People: string (nullable = true)
 |-- Utility: string (nullable = true)
```

Here to show the top 20 rows in the final output.

```
final_df.show(truncate=False)
```

user1	user2	transaction_type	datetime	description	is_business	story_id	text_category	emoji	emoji_category
6471126	10046265	charge	2016-05-18 08:00:50	Butterfinger mklk	false	573bbec2cd03c9af22d27811	null	null	null
4892258	5697557	payment	2016-04-30 03:25:17	🍕 Mary had a little lamb	false	5723c32dcd03c9af22160da3	Travel	🍕	Food
4892258	5697557	payment	2016-04-30 03:25:17	🍕 Mary had a little lamb	false	5723c32dcd03c9af22160da3	People	🍕	Food
4892258	5697557	payment	2016-04-30 03:25:17	🍕 Mary had a little lamb	false	5723c32dcd03c9af22160da3	Food	🍕	Food
1718828	353851	payment	2015-10-02 08:02:23	Thank you	false	560dd79fcd03c9af223f674f	People	null	null
1718828	353851	payment	2015-10-02 08:02:23	Thank you	false	560dd79fcd03c9af223f674f	People	null	null
2778043	1933164	payment	2016-04-24 05:20:09	Goods	false	571bf519cd03c9af228bfbd7	null	null	null
2309177	1850775	charge	2016-06-08 06:04:20	Ancient pair	false	575752f4cd03c9af22b199f5	Transportation	null	null
2950949	6828833	payment	2016-07-30 23:34:34	Paddle boarding 🏄	false	579cd71b23e064eac0929710	Illegal/Sarcasm	🏄	Activity
2950949	6828833	payment	2016-07-30 23:34:34	Paddle boarding 🏄	false	579cd71b23e064eac0929710	Food	🏄	Activity
2950949	6828833	payment	2016-07-30 23:34:34	Paddle boarding 🏄	false	579cd71b23e064eac0929710	Activity	🏄	Activity
2540988	2762195	payment	2015-02-08 02:24:40	Rite aid	false	54d6586891bd05aa935a49b0	Event	null	null
693634	2471579	payment	2015-07-13 07:17:58	Merge come height now	false	55a303b607f81c2e966346bf	Utility	null	null
1369746	1366233	payment	2016-04-08 08:33:56	😘😘😘	false	57070a84cd03c9af221c88de	null	😘	People
1407592	1692018	charge	2015-10-25 05:56:24	Gym 10-23-15	false	562c0c98cd03c9af22af8c33	null	null	null
2778570	828348	payment	2016-09-19 20:44:03	☕	false	57dfeba323e064eac053240e	null	☕	Food
5183325	3463466	payment	2015-12-23 14:06:42	🍊	false	567a39f2cd03c9af228a9eef	null	🍊	Food
728795	728780	payment	2013-11-26 13:53:27	Pizza	false	52943759d56b6bac5cf36a84	Food	null	null
3108605	2532918	payment	2015-02-22 09:30:06	Tequila betch	false	54e9311fcd03c9af22c5f9da	Travel	null	null
3108605	2532918	payment	2015-02-22 09:30:06	Tequila betch	false	54e9311fcd03c9af22c5f9da	Illegal/Sarcasm	null	null

only showing top 20 rows

Q2 - What is the percentage of emoji only transactions? Which are the top 5 most popular emoji? Which are the top three most popular emoji categories?

Q2 [5 pts]: What is the percent of emoji only transactions? Which are the top 5 most popular emoji? Which are the top three most popular emoji categories?

Answer:

Percent of emoji only transactions: 23.71%.

Top 5 most popular emoji are: null, 🍻, 🍕, 🎉, 🍷.

Top 3 most popular emoji categories are: null, Food, People.

```
Percent of emoji only transactions: 23.71%
Top 5 most popular emojis:
+-----+-----+
|emoji|count|
+-----+-----+
| null|73727|
|   🍻| 1225|
|   🍕| 1183|
|   🎉|  927|
|   🍷|  837|
+-----+-----+

Top 3 most popular emoji categories:
+--------------+-----+
|emoji_category|count|
+--------------+-----+
|          null|73727|
|          Food|13781|
|        People| 9595|
+--------------+-----+
```

Q3 - For each user, create a variable to indicate their spending behavior profile. For example, if a user has made 10 transactions, where 5 of them are food and the other 5 are activity, then the user's spending profile will be 50% food and 50% activity.

```
+-----+--------------+-----------------+
|user1|text_category |percentage       |
+-----+--------------+-----------------+
|4    |Food          |50.0             |
|4    |Illegal/Sarcasm|50.0            |
|10   |Food          |100.0            |
|43   |null          |100.0            |
|52   |Activity      |25.0             |
|52   |Food          |25.0             |
|52   |People        |25.0             |
|52   |Utility       |25.0             |
|879  |Illegal/Sarcasm|33.33333333333333|
|879  |People        |33.33333333333333|
|879  |Transportation |33.33333333333333|
|1009 |null          |100.0            |
|1241 |Food          |100.0            |
|2504 |null          |100.0            |
|2794 |Food          |50.0             |
|2794 |Illegal/Sarcasm|50.0            |
|3310 |null          |100.0            |
|3664 |Food          |80.0             |
|3664 |Travel        |20.0             |
|4715 |Activity      |50.0             |
+-----+--------------+-----------------+
only showing top 20 rows
```

Q4 - What do you observe? Does the spending profile of the average customer stabilize after some point in time?

<u>Observations:</u>

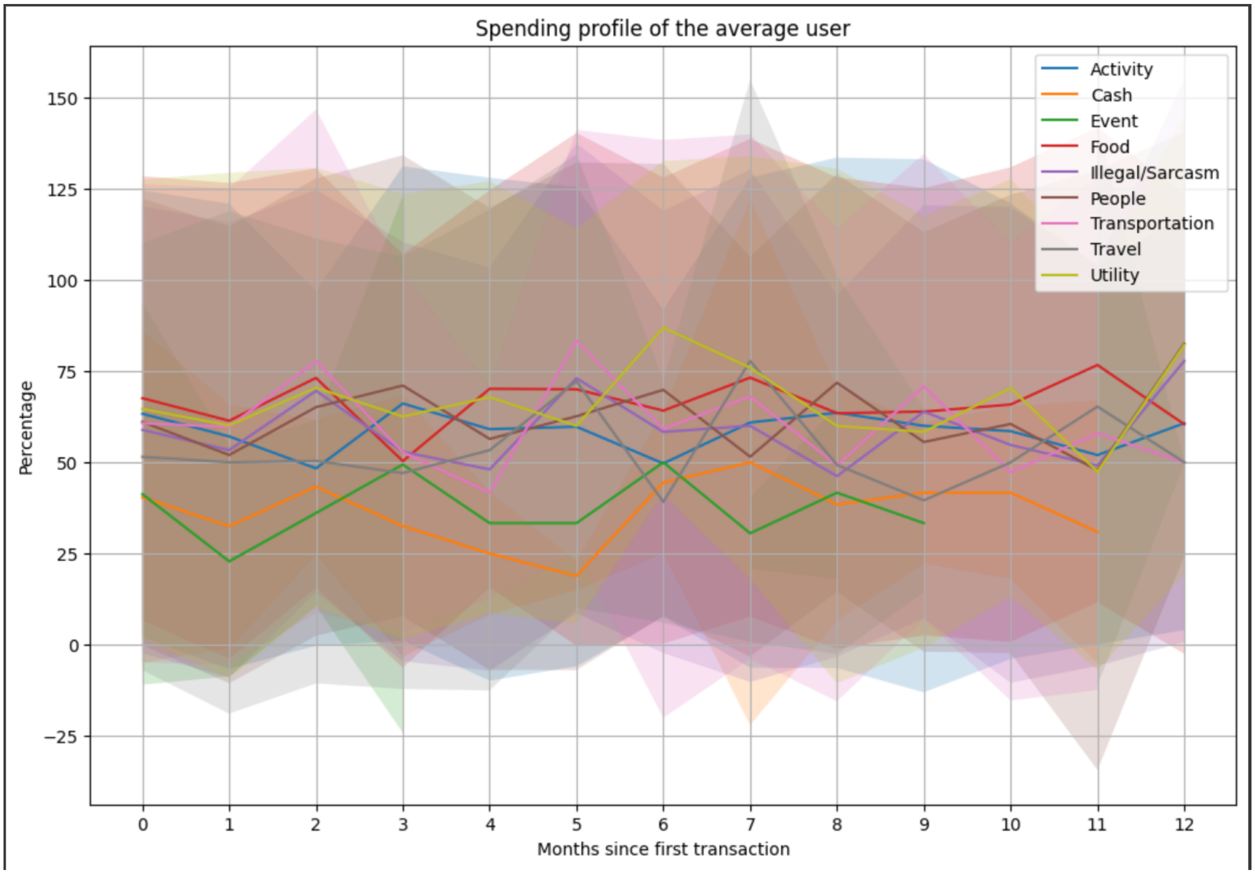
As we can see from the graph below, each spending category across all users fluctuates a lot over time. It seems that after month 6, the spending profile of the average customer stabilizes in time.

Categories of Food, Transportation and Utility usually take the largest proportion. (Has the highest percentage at a specific time point) Event and Cash categories tend to have the lowest percentage overall.

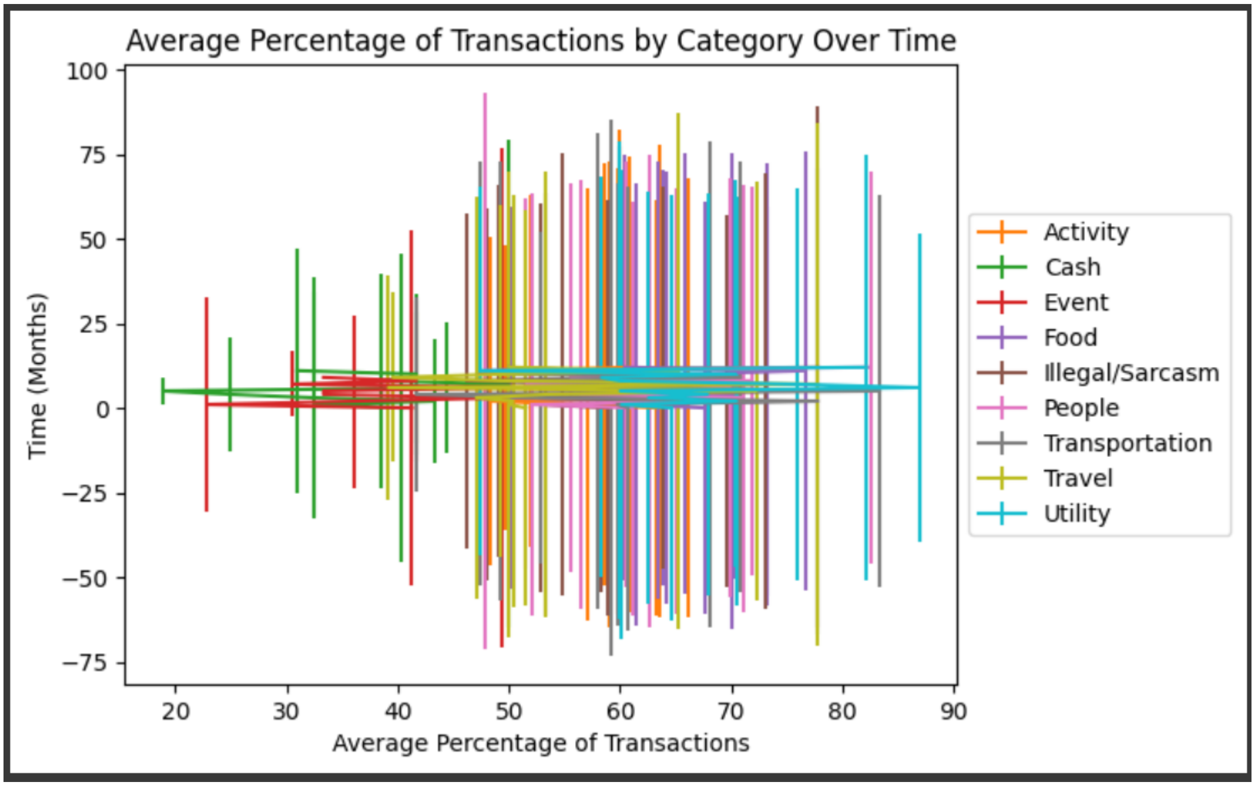
For the Travel category, other than the month 5 and month 7 reach the highest percentage point, this category does not fluctuate a lot on the other time.

(In order to have more details for each category, we generate plots for different time point, and the average percentage change table to interpret in the following steps.)

1. Plot of Spending Category over Time (y-axis percentage, x-axis month)



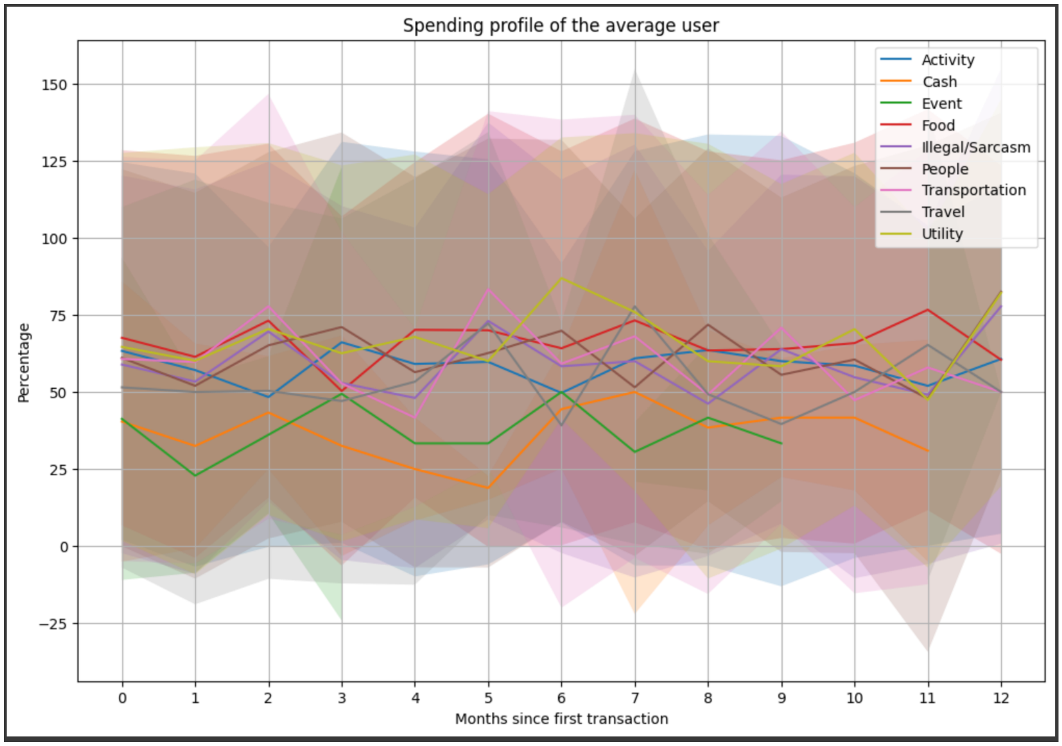
2. Plot of Spending Category over Time (x-axis percentage, y-axis month)



3. Average Percentage Change Table
Observations:

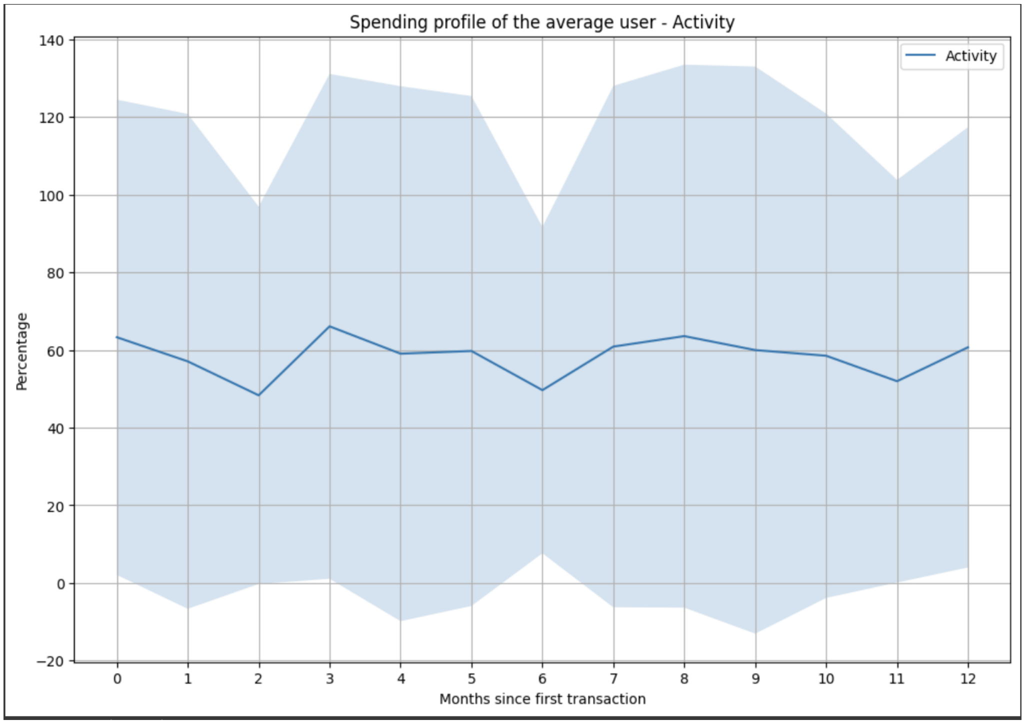
The People category has the highest average change through a year, which is 35.07%. Also, one notable thing is the Illegal/Sarcasm category increased a lot. This may need to do more analysis to get insights and have corresponding strategies to prevent that.

	text_category	percentage_change
0	NaN	NaN
1	Activity	-4.093673
2	Cash	NaN
3	Event	NaN
4	Food	-10.594748
5	Illegal/Sarcasm	32.032893
6	People	35.071671
7	Transportation	-17.646481
8	Travel	-2.921525
9	Utility	27.117235



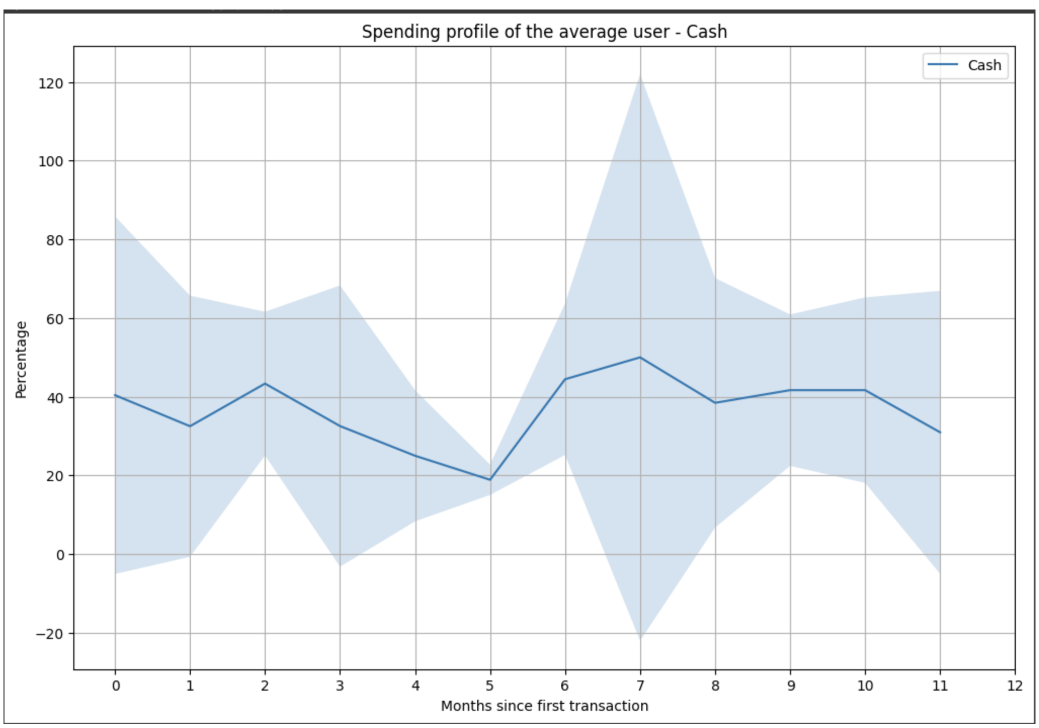
4. Average Percentage of Transactions for EACH Month

- Activity



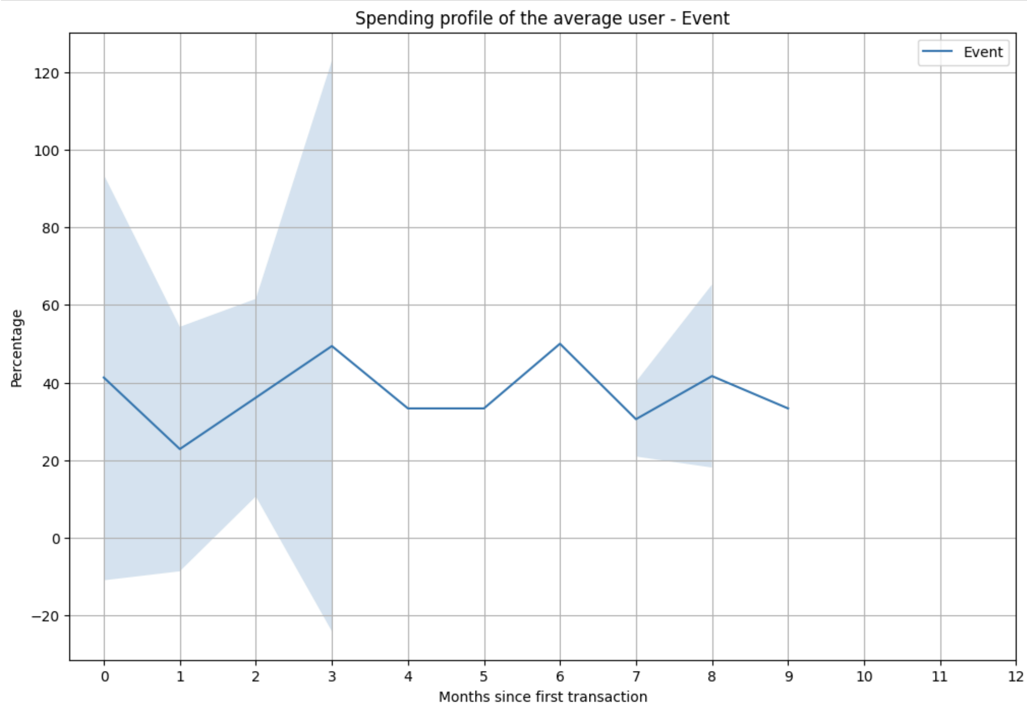
	months_since_first	average
13	0	63.305819
14	1	57.092593
15	2	48.350340
16	3	66.111111
17	4	59.074074
18	5	59.761905
19	6	49.679487
20	7	60.897436
21	8	63.591270
22	9	60.000000
23	10	58.541667
24	11	51.984127
25	12	60.714286

- Cash



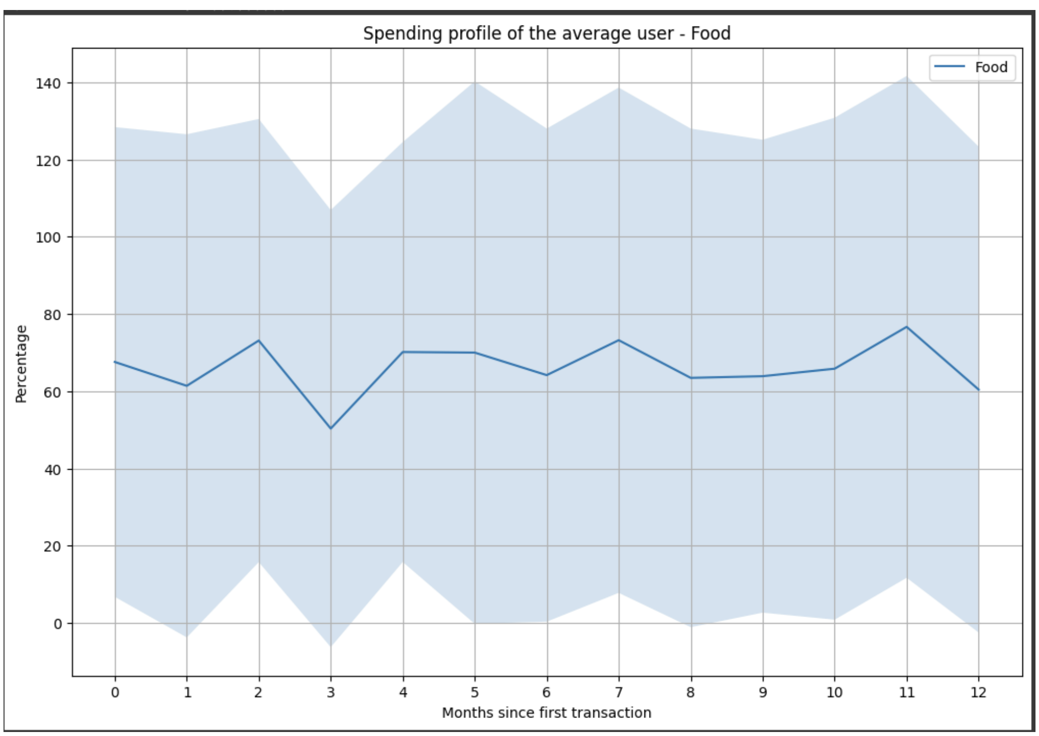
	months_since_first	average
26	0	40.390196
27	1	32.500000
28	2	43.333333
29	3	32.539683
30	4	25.000000
31	5	18.888889
32	6	44.444444
33	7	50.000000
34	8	38.444444
35	9	41.666667
36	10	41.666667
37	11	30.952381

- Event



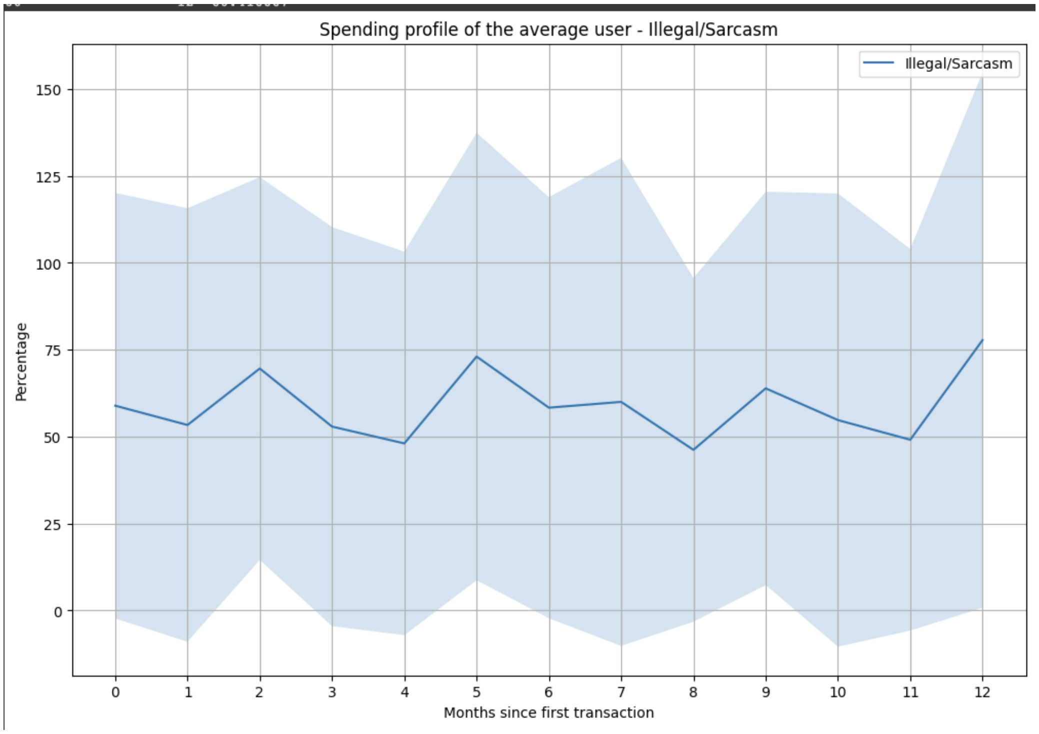
months_since_first	average
0	41.291558
1	22.857143
2	36.111111
3	49.404762
4	33.333333
5	33.333333
6	50.000000
7	30.555556
8	41.666667
9	33.333333

- Food



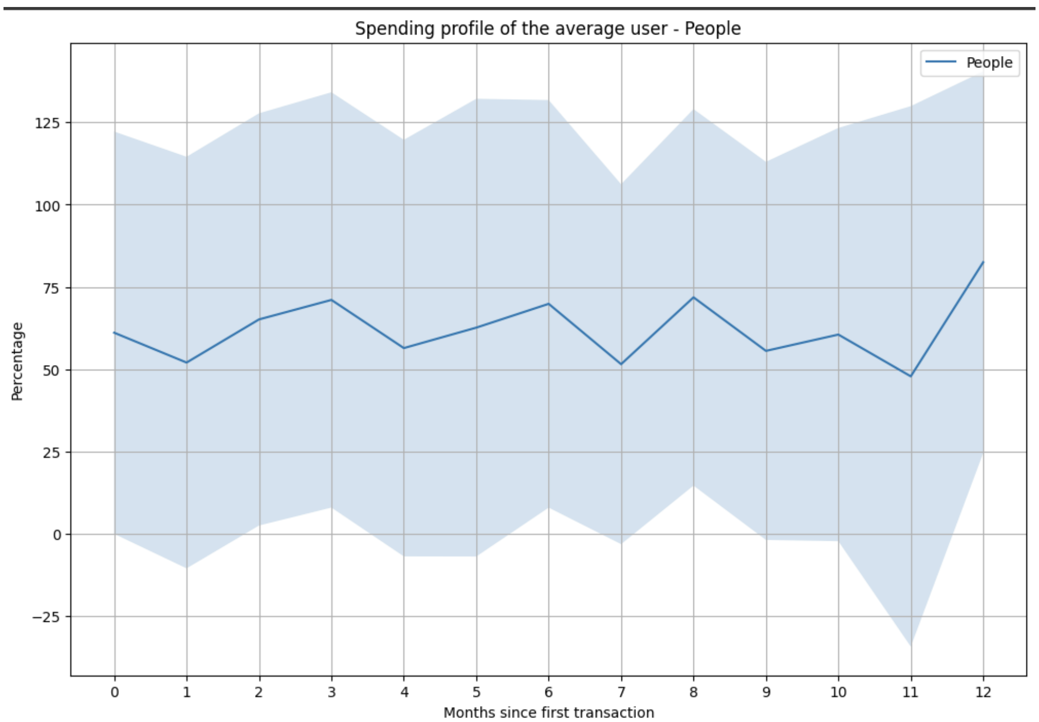
	months_since_first	average
48	0	67.576195
49	1	61.391723
50	2	73.125000
51	3	50.327381
52	4	70.151515
53	5	70.000000
54	6	64.166667
55	7	73.214286
56	8	63.456790
57	9	63.888889
58	10	65.833333
59	11	76.666667
60	12	60.416667

- Illegal/Sarcasm



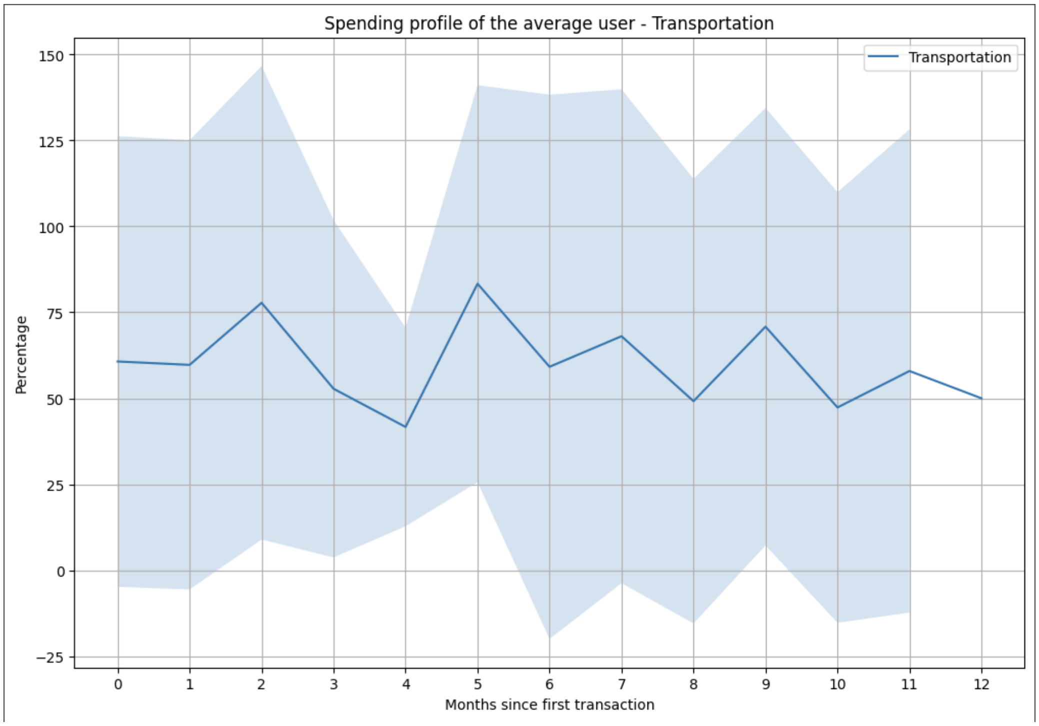
	months_since_first	average
61	0	58.907880
62	1	53.354701
63	2	69.593254
64	3	52.893773
65	4	48.055556
66	5	73.030303
67	6	58.333333
68	7	60.000000
69	8	46.203704
70	9	63.888889
71	10	54.761905
72	11	49.107143
73	12	77.777778

- People



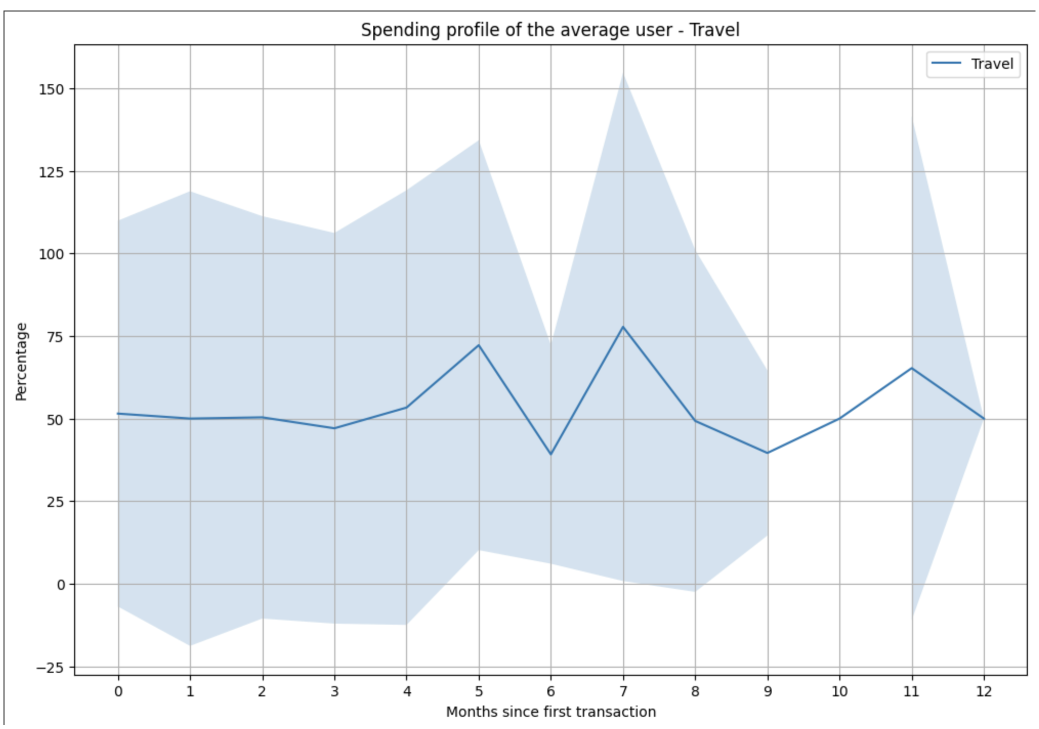
	months_since_first	average
74	0	61.078685
75	1	52.006803
76	2	65.119048
77	3	71.041667
78	4	56.403509
79	5	62.619048
80	6	69.852941
81	7	51.515152
82	8	71.825397
83	9	55.555556
84	10	60.512821
85	11	47.817460
86	12	82.500000

- Transportation



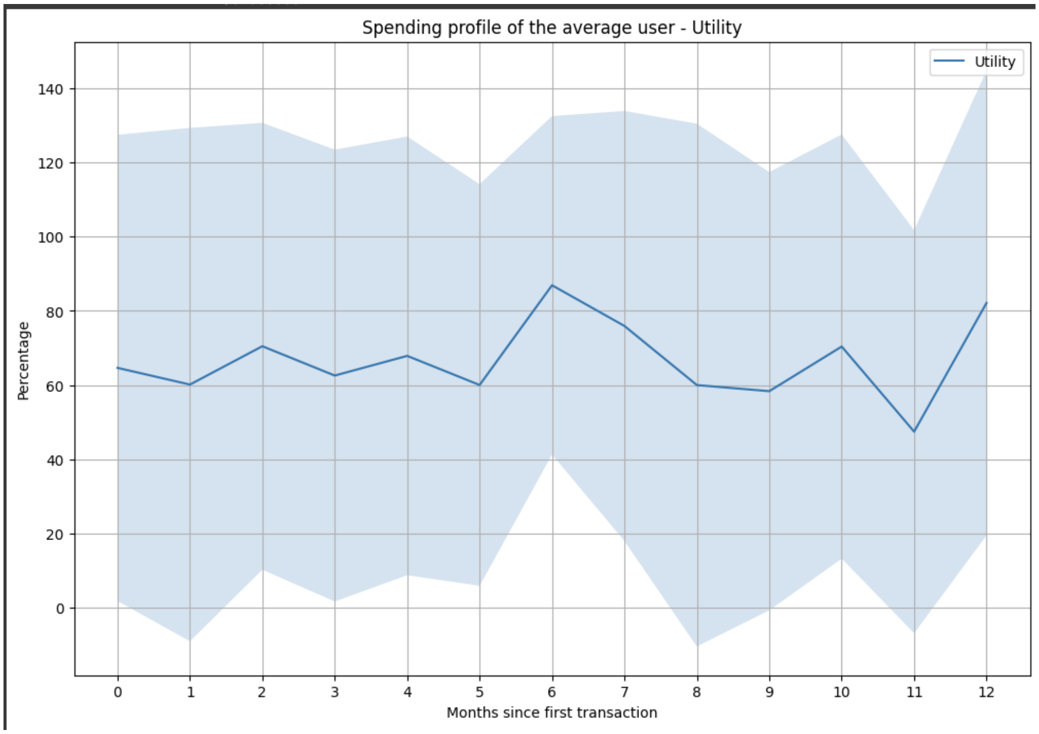
	months_since_first	average
87	0	60.713860
88	1	59.722222
89	2	77.777778
90	3	52.777778
91	4	41.666667
92	5	83.333333
93	6	59.166667
94	7	68.055556
95	8	49.166667
96	9	70.833333
97	10	47.333333
98	11	57.936508
99	12	50.000000

- Travel



	months_since_first	average
100	0	51.504723
101	1	50.000000
102	2	50.370370
103	3	47.056277
104	4	53.333333
105	5	72.222222
106	6	39.166667
107	7	77.777778
108	8	49.285714
109	9	39.583333
110	10	50.000000
111	11	65.277778
112	12	50.000000

- Utility



	months_since_first	average
113	0	64.619764
114	1	60.116959
115	2	70.454545
116	3	62.547619
117	4	67.857143
118	5	60.000000
119	6	86.904762
120	7	75.925926
121	8	59.978632
122	9	58.333333
123	10	70.370370
124	11	47.420635
125	12	82.142857

Social Network Analytics

Q5 - Write a script to find a user's friends and friends of friends (Friend definition: A user's friend is someone who has transacted with the user, either sending money to the user or receiving money from the user). Describe your algorithm and calculate its computational complexity. Can you do it better?

<u>To find friends of a user:</u>

```
friends = []
For record in dataframe:
  if user1 == user OR user2 == user:
    friends.append(other user)
```

=> get unique values from friends list

<u>To find friends of friends:</u>

```
friends_of_friend = []
For user in friends:
  For record in dataframe:
    if user1 == user OR user2 == user:
      friends.append(other user)
```

=> get unique values from friends_of_friend list O(n^2)

The algorithms mentioned above have a Time Complexity of O(n^2) based on below results, therefore we choose a different method.

```
[ ] %%time
    find_friends(df_first_12_months, 4892258)

    CPU times: user 304 ms, sys: 25.2 ms, total: 329 ms
    Wall time: 27.1 s
    []

[ ] %%time
    find_friends_of_friends(df_first_12_months,find_friends(df_first_12_months, 4892258))

    CPU times: user 290 ms, sys: 989 µs, total: 291 ms
    Wall time: 1.69 s
    []

[ ] %%time
    find_friends(df_first_12_months, 5697557)

    CPU times: user 169 ms, sys: 3.96 ms, total: 173 ms
    Wall time: 1.57 s
    []
```

In the updated version below we leverage spark to find the list of friends and friends of friends faster.

```
[ ] from pyspark.sql.functions import collect_list, size

    # Create a new DataFrame with two columns: 'user' and 'friend'
    df_friends = df_first_12_months.selectExpr("user1 as user", "user2 as friend").union(df_first_12_months.selectExpr("user2 as user", "user1 as friend"))

    # Group by 'user' and aggregate the 'friend' column into a list and count the friends
    df_friends_list = df_friends.groupBy('user').agg(collect_list('friend').alias('friends'), size(collect_list('friend')).alias('count_friends'))

[ ] from pyspark.sql.functions import explode, flatten, array_distinct

    # Create a new DataFrame that maps each user to their friends of friends
    df_friends_of_friends = df_friends_list \
        .withColumn('friend', explode('friends')) \
        .join(df_friends_list.withColumnRenamed('user', 'friend').withColumnRenamed('friends', 'friends_of_friend'), on='friend') \
        .groupBy('user') \
        .agg(collect_list('friends_of_friend').alias('friends_of_friends'))

    # Flatten the list of friends of friends and remove duplicates
    df_friends_of_friends = df_friends_of_friends \
        .withColumn('friends_of_friends', array_distinct(flatten('friends_of_friends')))

    # Calculate the count of friends of friends
    df_friends_of_friends = df_friends_of_friends \
        .withColumn('count_friends_of_friends', size('friends_of_friends'))

    # Join back with df_friends_list to add the friends of friends list and the count
    df_friends_list = df_friends_list \
        .join(df_friends_of_friends, on='user', how='left_outer')
```

Q6 - Now that you have the list of each user's friends and friends of friends, you are in position to calculate many social network variables. Use the dynamic analysis from before, and calculate the following social network metrics across a user's lifetime in Venmo (from 0 up to 12 months).

1. **Number of friends and number of friends of friends.**

```
df_friends_list.show(5)
```

```
+------+---------+-------------+-----------------+-----------------------+
|  user|  friends|count_friends|friends_of_friends|count_friends_of_friends|
+------+---------+-------------+-----------------+-----------------------+
|219523|[1288863]|            1|         [219523]|                      1|
|351750|[2372811]|            1|         [351750]|                      1|
|360710| [167260]|            1|         [360710]|                      1|
|377599|[2331442]|            1|         [377599]|                      1|
|402253| [993909]|            1|         [402253]|                      1|
+------+---------+-------------+-----------------+-----------------------+
only showing top 5 rows
```

```
Row(user=219523, friends=[1288863], count_friends=1, friends_of_friends=[219523], count_friends_of_friends=1)
Row(user=351750, friends=[2372811], count_friends=1, friends_of_friends=[351750], count_friends_of_friends=1)
Row(user=360710, friends=[167260], count_friends=1, friends_of_friends=[360710], count_friends_of_friends=1)
Row(user=377599, friends=[2331442], count_friends=1, friends_of_friends=[377599], count_friends_of_friends=1)
Row(user=402253, friends=[993909], count_friends=1, friends_of_friends=[402253], count_friends_of_friends=1)
```

2. **Clustering coefficient of a user's network.**

   The graph would be considered undirected for this purpose.
   1. Identify the friends and store in a friend list - length of N.

   2. Calculate the potential number of connections that are possible between these friends - N*(N-1)/2.

   3. For each friend in the friend list - see if the friend has any of the og guys friends in his friends.

   The algorithm above would have a time complexity of O(n^2) again, therefore we once again leverage Spark to calculate the clustering coefficients of each user quicker.

   Algorithm:

```python
from pyspark.sql.functions import col, count

# Explode the 'friends' column into separate rows
df_exploded = df_friends_list.select('user', explode('friends').alias('friend'))

# Self-join the DataFrame on the 'user' column to find all pairs of friends for each user
df_pairs = df_exploded.alias('a').join(df_exploded.alias('b'), col('a.user') == col('b.user'))

# Count the number of connections between friends for each user
df_connections = df_pairs.where(col('a.friend') < col('b.friend')).groupBy('a.user').agg(count('*').alias('connections'))

# Calculate the total possible number of connections between friends for each user
df_possible_connections = df_friends_list.select('user', (col('count_friends')*(col('count_friends')-1)/2).alias('possible_connections'))

# Join the DataFrames and calculate the clustering coefficient
df_clustering_coefficient = df_connections.join(df_possible_connections, on='user')
df_clustering_coefficient = df_clustering_coefficient.withColumn('clustering_coefficient', col('connections')/col('possible_connections'))

# Join back with df_friends_list to add the clustering coefficient
df_friends_list = df_friends_list.join(df_clustering_coefficient.select('user', 'clustering_coefficient'), on='user', how='left_outer')
```

Results:

```python
df_friends_list.show(5)
```

```
+------+---------+-------------+-----------------+-----------------------+----------------------+
|  user|  friends|count_friends|friends_of_friends|count_friends_of_friends|clustering_coefficient|
+------+---------+-------------+-----------------+-----------------------+----------------------+
|219523|[1288863]|            1|         [219523]|                      1|                  null|
|351750|[2372811]|            1|         [351750]|                      1|                  null|
|360710| [167260]|            1|         [360710]|                      1|                  null|
|377599|[2331442]|            1|         [377599]|                      1|                  null|
|402253| [993909]|            1|         [402253]|                      1|                  null|
+------+---------+-------------+-----------------+-----------------------+----------------------+
only showing top 5 rows
```

```python
rows = df_friends_list.take(5)
for row in rows:
    print(row)
```

```
Row(user=219523, friends=[1288863], count_friends=1, friends_of_friends=[219523], count_friends_of_friends=1, clustering_coefficient=None)
Row(user=351750, friends=[2372811], count_friends=1, friends_of_friends=[351750], count_friends_of_friends=1, clustering_coefficient=None)
Row(user=360710, friends=[167260], count_friends=1, friends_of_friends=[360710], count_friends_of_friends=1, clustering_coefficient=None)
Row(user=377599, friends=[2331442], count_friends=1, friends_of_friends=[377599], count_friends_of_friends=1, clustering_coefficient=None)
Row(user=402253, friends=[993909], count_friends=1, friends_of_friends=[402253], count_friends_of_friends=1, clustering_coefficient=None)
```

3. Calculate the page rank of each user.

```python
df_friends_list.show()
```

```
+-------+------------------+-------------+------------------+------------------------+----------------------+------------------+
|   user|           friends|count_friends|friends_of_friends|count_friends_of_friends|clustering_coefficient|          pagerank|
+-------+------------------+-------------+------------------+------------------------+----------------------+------------------+
| 219523|         [1288863]|            1|          [219523]|                       1|                  null|0.6986984661425121|
| 351750|         [2372811]|            1|          [351750]|                       1|                  null|1.2925921623636472|
| 360710|          [167260]|            1|          [360710]|                       1|                  null|0.6986984661425121|
| 377599|         [2331442]|            1|          [377599]|                       1|                  null|0.6986984661425121|
| 402253|          [993909]|            1|          [402253]|                       1|                  null|0.6986984661425121|
| 478897|         [1229665]|            1|          [478897]|                       1|                  null|0.6986984661425121|
| 557836|         [2073653]|            1|          [557836]|                       1|                  null|1.2925921623636472|
| 566063|         [1017679]|            1|          [566063]|                       1|                  null|0.6986984661425121|
| 613396|         [1299335]|            1|          [613396]|                       1|                  null|1.2925921623636472|
| 636477|          [495709]|            1|          [636477]|                       1|                  null|0.6986984661425121|
| 640156|          [874985]|            1|          [640156]|                       1|                  null|1.2925921623636472|
| 687716|         [4742723]|            1|          [687716]|                       1|                  null|0.6986984661425121|
| 801532|          [778723]|            1|          [801532]|                       1|                  null|1.2925921623636472|
| 904487|         [3221529]|            1|          [904487]|                       1|                  null|1.2925921623636472|
| 969344|[5713634, 5464752]|            2|          [969344]|                       1|                   1.0|0.6986984661425121|
|1162591|         [3744940]|            1|         [1162591]|                       1|                  null|0.6986984661425121|
|1200753|         [1067298]|            1|         [1200753]|                       1|                  null|0.6986984661425121|
|1218886|          [694580]|            1|         [1218886]|                       1|                  null|0.6986984661425121|
|1271994|         [1389041]|            1|         [1271994]|                       1|                  null|1.2925921623636472|
|1306781|         [1732376]|            1|[1306781, 5204446]|                       2|                  null|0.9956453142530797|
+-------+------------------+-------------+------------------+------------------------+----------------------+------------------+
only showing top 20 rows
```

With each update of the social network there is a probability that the page rank of the user changes. In such a case we would have to constantly update the page rank of each user at each change to the network. This would be chaotic and computationally intensive.

Instead, we just calculate the page rank for each user once in this case - at the end of a given time period to estimate the influence of the user in proportion to the entire network. However, this leads to a loss of understanding of how important of a role a particular user played at a particular moment of time.

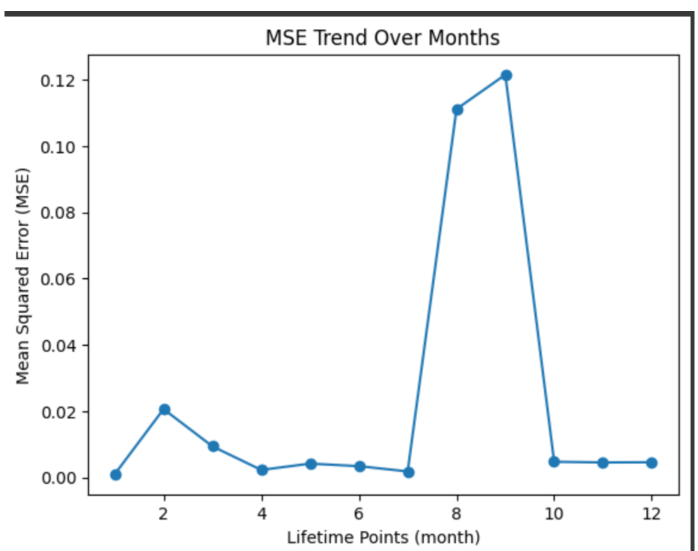
Predictive Analytics

Assumption- We have taken a sample of the dataset assuming it is an apt representation of the population

Q7 - grouped dataset by user1 and month to find the count of transactions. **Assumption** - one transaction is done when user1 sends user2 money on venmo

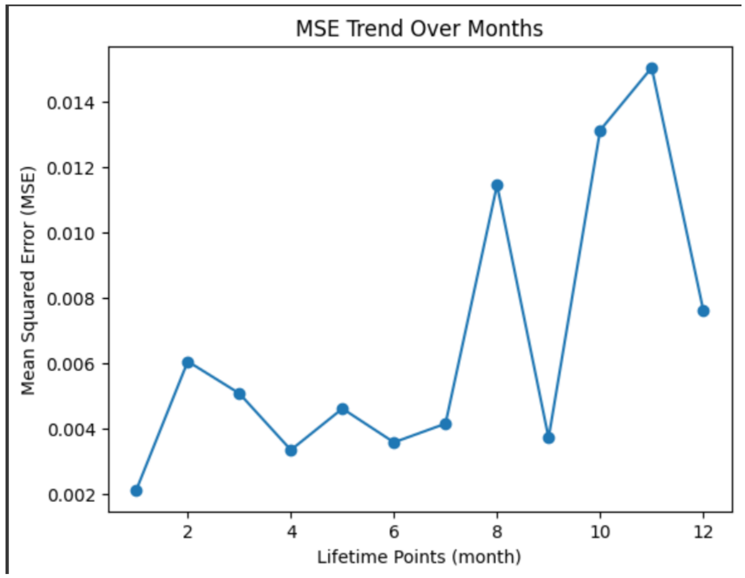
Q8 - For each month and user1, we have taken the latest transaction datetime, extracted the day of month, and subtracted this from 30 to get the recency. As mentioned in the question, frequency is 30/no of transactions

Q9 -



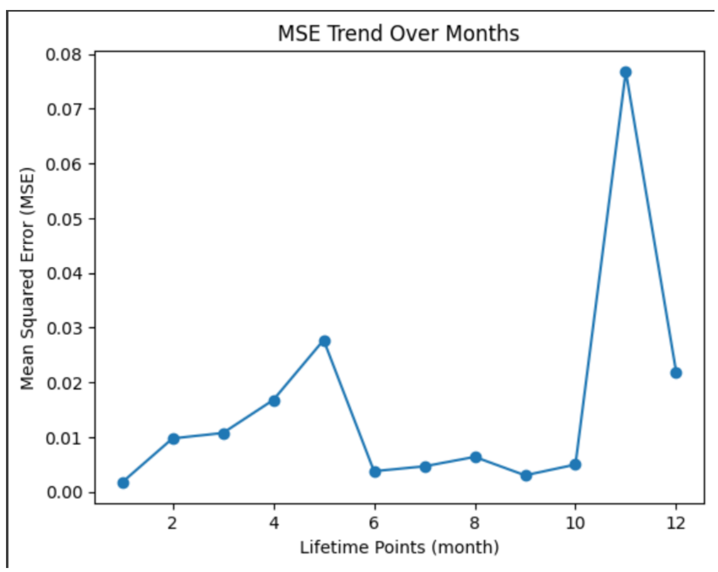
MSE range is 0-0.12

Q10



MSE range is 0.002-0.014

Q10

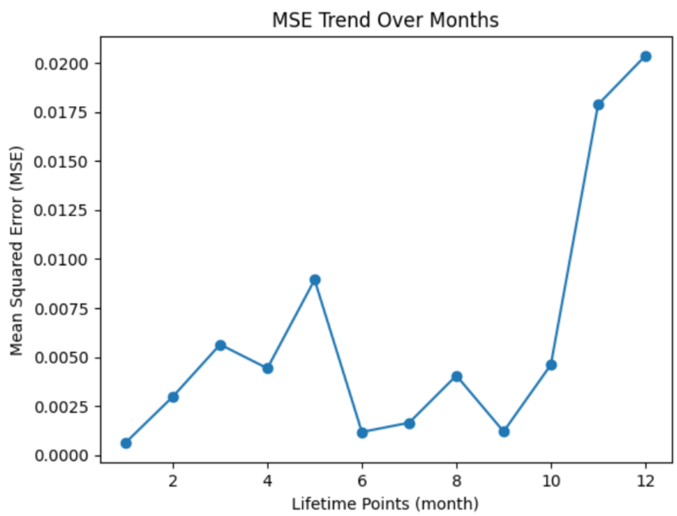


MSE range is 0-0.08

Q. What do you observe? How do social network metrics compare with the RF framework? What are the most informative predictors?

The MSE error range for regression with social network metrics is lower as compared to the MSE range for RF Framework So the network analytics columns are more informative that just knowing the recency and frequency

Q11



MSE range is 0-0.02

Q - Does the spending behavior of her social network add any predictive benefit compared to Q10?
Yes, including network and spending behavior reduces MSE, hence improving the predictive power of the model.