

Venmo Project

- BAX 423 Big Data Group Project

By Team ConFiveDance: Guanying Deng, Miaoqi Yang, Stella Li, Yvette Peng, Wenbin Lu

Table of Contents

Text Analytics	2
Q0. Adding Words	2
Q1. Classification	2
Q2. Emoji Analytics	4
Q3. User Static Spending Profile	6
Q4. User Lifetime Spending Profile	6
Social Network Analytics	8
Q5. Friends and Friends of Friends	8
Q6. Social Network Metrics	9
i) Number of Friends of Friends and Number of Friends of Friends	9
ii) Clustering coefficient	12
iii) PageRank	15
Predictive Analytics with MLlib	17
Q7. Create the dependent variable Y	17
Q8. Create the recency and frequency variables	17
Q9. Regression on Recency and Frequency	19
Q10. Regression on Recency, Frequency and Spending Profile	22
Q11. Regression on Social Network Metrics	23
Q12. Regression on Social Network Metrics and Spending Profile of Social Network	25
Appendix. MSE output for each regression	27

Text Analytics

When a transaction happens in Venmo, a "note" is required to describe the reason for the payment or charge. Some users simply enter emojis, some users prefer to write text explanations, and some enter emojis along with the text. That is, Venmo messages can be 1) emoji, 2) text, and 3) a combination of emoji and text. The notes ('description' column in the transaction table provided) in Venmo transactions offer information about what people are spending their money on.

In this part of the project, we are trying to figure out the meaning behind transactions on Venmo. We will classify each transaction based on the notes and create both static and dynamic spending profiles for each user to see and later analyze their spending behavior.

Q0. Adding Words

Ten words we added to the dictionary: printing, hot pot, sandwich, KTV, dimsum, mask, Jollibee, cocktail, milktea, and graduation.

Q1. Classification

In this question, we will add a column indicating the category of the description of each transaction to the original transaction table.

After importing three datasets¹, lists were created to include words or emojis that belong to each category. For example, the list emoji_lst_Food contains all emojis in the emoji dictionary under the Food category, and the list txt_lst_Utility contains all words in the word dictionary under the Utility category.. We created txt_lst and emoji_lst to include all words and emojis in text and emoji dictionaries.

The function 'to_category' takes a single word or an emoji as input and returns the category of the input. This function will be called in the function 'classify' to assign categories to each word/emoji.

The main function 'classify' takes strings as input and returns the category of the string. If a single string involves more than one category, the returned category should be the one that has the highest occurrence.

After applying the function to add a new column to the transaction table, we now have a table with a new column 'category' which classifies the description of each row of Venmo transactions.

user1	user2	transaction_type	datetime	description	is_business	story_id	category
1218774	1528945	payment	2015-11-27 10:48:19	Uber	false	5657c473cd03c9af2...	Transportation
5109483	4782303	payment	2015-06-17 11:37:04	Costco	false	5580f9702b64f70ab...	Food
4322148	3392963	payment	2015-06-19 07:05:31	Sweaty balls	false	55835ccb1a624b14a...	Illegal_Sarcasm
469894	1333620	charge	2016-06-03 23:34:13	👊	false	5751b185cd03c9af2...	Event
2960727	3442373	payment	2016-05-29 23:23:42	⚡	false	574b178ecd03c9af2...	Utility

¹Dataset imported: VenmoSample.snappy.parquet, Venmo_Word_Classification_Dictionary and Venmo_Emoji_Classification_Dictionary

Q2. Emoji Analytics






Emoji Only Percentage

Now we want to know what percentage of the transactions' notes contain emoji only. We approach the problem by first creating a function that takes a string as input and returns '1' if it is an emoji-only string, and '0' otherwise. We apply this function to add a new column 'Emoji_only' to the transaction table, labeling if the description of each transaction is emoji_only or not. After that, we wrote a SQL to sum up the number of transactions which are emoji-only, and divide that number by the total count of transactions. The result shows that 23.42% of transactions are emoji-only.

Top 5 Most Popular Emojis

We started by writing a function to extract all emojis from the input string. And then apply it to add a new column 'emoji' to the transaction table with extracted emojis from the description. If the description does not contain any emoji, the emoji column will be 'null'. We then remove those rows with null values in the 'emoji' column. In this way, the rows are only those which contain at least one emoji in the description. Next, we create a list to put in all the emojis from the 'emoji' column and count the number of each emoji.

The result shows that the top 5 most popular emojis are:

Emoji	Occurrence
	215039
	145233
	124727
	111157
	94327

Top 3 Most Popular Emoji Categories

We created a for loop to iterate the list that contains all emojis from the 'emoji' column in the last step and count the occurrences of emojis in each category.

According to the result, the top 3 most popular emoji categories are:

Category	Count
Food	1744390
People	1113540
Utility	430868

Q3. User Static Spending Profile

In this question, we are going to create a table that shows each users' spending profile. We wrote a SQL query to keep only those transactions, which are 'payment' type and count the ratio of each category of each user, and then pivot the table so that each category will be the columns, and each row represents a user's spending profile.

The result is a new table with a new column 'category'.

user1	Activity	Cash	Event	Food	Illegal_Sarcasm	Not_classified	People	Transportation	Travel	Utility
2	null	null	null	null	null	null	null	null	null	1.0
3	0.17	null	0.17	null	null	0.17	null	null	0.5	null
4	0.2	null	null	0.4	null	0.2	null	null	0.2	null
10	0.1	null	null	0.3	null	0.4	0.2	null	null	null
11	0.08	null	0.12	0.12	null	0.52	0.12	null	0.04	null

Q4. User Lifetime Spending Profile

In Q3, we got the static spending profile of each user. In this question, we will create a dynamic spending profile for each user during their first year's lifetime. To evaluate the change of spending profile over time, we treat time as a sequence of discrete monthly intervals. A lifetime of 0 refers to the time where a user made his first transaction after joining Venmo. A lifetime of 1 means the user made the transactions between day 1 and day 30 since the first transaction, and a lifetime of 2 means the user made the transactions between day 31 and day 60 since the first transaction, and so on. We are only looking at lifetimes 0-12 in this project. To achieve that, we used a window function to get the date of the first transaction of each user and then wrote a SQL query to assign numbers 1-12 to

the transaction dates by taking the date difference between transaction dates and the date of the first transaction for that user, and dividing that by 30.

user1	lifetime	Activity	Cash	Event	Food	Illegal_Sarcasm	Not_classified	People	Transportation	Travel	Utility
2	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
3	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
3	1	0.2	0.0	0.2	0.0	0.0	0.2	0.0	0.0	0.4	0.0
4	0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
4	1	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0

Social Network Analytics

Q5. Friends and Friends of Friends

Finding a user's friends

First, we selected the 'user1' and 'user2' columns in the transactions and unioned the two columns with them reversed. In this way, all the users who appeared in the transactions are in the 'user1' column. Then, to speed up our algorithm, we kept only distinct pairs of users to reduce the amount of data. After this, we grouped the 'user2' column with the 'user1' column as keys to generate key-value pairs in the format of {user1: user1's friends}, using the 'group by key' function.

Computational complexity: $O(N^2)$, or $O(|U_1| * |U_2|)$, where $|U_1|$ = distinct number of user1, $|U_2|$ = distinct number of user2.

Finding a user's friends of friends

After collecting the key-value pairs of users and their friends as a dictionary, we wrote a user-defined function to iterate through a user's friends, find friends of the user's friends, and generate a set for those friends to keep only distinct users in the set. We applied this function to all users to find their friends of friends, and then calculated the set difference

between the 'friends of friends' set and the set of the users' direct friends. This method removed all the users' direct friends in the 'friends of friends' set and kept only second-degree friends.

Computational complexity: $O(N^3)+O(N^3)$, or $O(|U_1|*|U_f|*|U_{f2}|) + O(|U_{f2}|*|U_f|*|U_1|)$, where $|U_1|$ = distinct number of users (the keys), $|U_f|$ = length of friend lists for all users combined, $|U_{f2}|$ = length of friend lists for each user's friends combined. The first step describes the computational complexity of finding the friends of friends of all users, and the second step describes the computational complexity of calculating the set differences.

Q6. Social Network Metrics

i) Number of Friends of Friends and Number of Friends of Friends

Number of friends in each lifetime month

We first swapped the "user1" and "user2" columns and union with the original dataset, considering the undirected relationship between users. Then, we used a SQL window function to calculate a "lifetime" column, which indicated the lifetime month when the transaction occurred. SQL aggregation function was employed to calculate the number of "new" friends with whom users transacted within each lifetime month. We only considered

the first transactions between the same pair of users, because we did not want to double count the same friend.

```
↗
```

user1	lifetime	num_friends
2	0	1
2	1	0
2	2	0
2	3	0
2	4	0
2	5	0
2	6	0
2	7	0
2	8	0
2	9	0
2	10	0
2	11	0
2	12	0
3	0	1
3	1	1
3	2	0
3	3	1
3	4	4
3	5	1
3	6	0

only showing top 20 rows

Then we used a window function to calculate the cumulative number of “new” friends over the lifetime.

user1	lifetime	num_friends	cum_num_friends
2	0	1	1
2	1	0	1
2	2	0	1
2	3	0	1
2	4	0	1
2	5	0	1
2	6	0	1
2	7	0	1
2	8	0	1
2	9	0	1
2	10	0	1
2	11	0	1
2	12	0	1
3	0	1	1
3	1	1	2

only showing top 15 rows

Number of friends of friends in each lifetime month

To calculate the number of second-degree friends, we left-joined two transaction tables using SQL by making the transaction receiver (user2) of table1 equal to the initiator (user1) of table2. Also, we made sure that table1's 'datetime' was greater than table2's datetime so that for each user, we could find his/her friends of friends acquainted only before his/her transaction at the date.

Then we used SQL to subtract the 1st-degree friends from the user's friends of friends list because 1st-degree friends may have common friends with the user. We further applied the window function to calculate the cumulative number of "new" friends of friends of each lifetime of the user.

user1	lifetime	num_fnd_of_fnd
2	0	0
2	1	0
2	2	0
2	3	0
2	4	0
2	5	0
2	6	0
2	7	0
2	8	0
2	9	0
2	10	0
2	11	0
2	12	0
3	0	19
3	1	15
3	2	0
3	3	2
3	4	17
3	5	16
3	6	0

only showing top 20 rows

ii) Clustering coefficient

The Clustering coefficient measures the degree of a user's social network being bonded together. To approach this problem, we first found the maths equation for computing the clustering coefficient:

Clustering coefficient = the number of triangles through the users / number of possible vertices in the user's network. The denominator was easily calculated by $\text{degree} * (\text{degree} - 1) / 2$. The degree is the number of friends a user has in each lifetime month.

```
[27] num_friends_one_year.createOrReplaceTempView("num_friends")
# calculate the number of possible vertices for each lifetime month
num_vertices = spark.sql("""SELECT user1, lifetime, cum_num_friends*(cum_num_friends-1)/2 AS num_vertices
FROM num_friends
ORDER BY user1, lifetime""")

num_vertices.show(15)
```

```
+---+-----+-----+
|user1|lifetime|num_vertices|
+---+-----+-----+
| 2| 0| 0.0|
| 2| 1| 0.0|
| 2| 2| 0.0|
| 2| 3| 0.0|
| 2| 4| 0.0|
| 2| 5| 0.0|
| 2| 6| 0.0|
| 2| 7| 0.0|
| 2| 8| 0.0|
| 2| 9| 0.0|
| 2| 10| 0.0|
| 2| 11| 0.0|
| 2| 12| 0.0|
| 3| 0| 0.0|
| 3| 1| 1.0|
+---+-----+-----+
only showing top 15 rows
```

To get the numerator, we left-joined three transaction tables using SQL. The join conditions were $t1_user2 = t2_user1$, $t2_user2 = t3_user1$ and $t1_user1 = t3_user2$. In this way, we can see a closed triangle going through the node of $t1_user1$.

```
+---+---+-----+-----+-----+-----+-----+-----+-----+
|t1_user1|t1_user2|t1_start_date|t1_datetime|t2_user1|t2_user2|t2_datetime|t3_user1|t3_user2|
+---+---+-----+-----+-----+-----+-----+-----+-----+
| 9| 243|2012-06-28 04:28:32|2012-06-28 04:28:32| 243| 367955|2013-10-06 09:59:31| 367955| 9|2016-0
| 9| 755956|2012-06-28 04:28:32|2015-05-19 04:58:47| 755956| 185494|2014-11-07 03:54:46| 185494| 9|2016-0
| 9| 185494|2012-06-28 04:28:32|2016-02-08 02:20:19| 185494| 755956|2014-11-07 03:54:46| 755956| 9|2015-0
| 9| 367955|2012-06-28 04:28:32|2016-08-24 01:57:40| 367955| 243|2013-10-06 09:59:31| 243| 9|2012-0
| 11| 50225|2012-05-15 22:08:58|2012-08-10 02:01:05| 50225| 119906|2013-11-27 11:28:08| 119906| 11|2014-0
| 11| 49778|2012-05-15 22:08:58|2012-10-07 04:57:45| 49778| 60183|2015-02-18 02:56:26| 60183| 11|2015-0
| 11| 146274|2012-05-15 22:08:58|2012-09-19 01:52:50| 146274| 146271|2013-12-16 00:39:32| 146271| 11|2013-1
| 11| 49778|2012-05-15 22:08:58|2012-10-07 04:57:45| 49778| 119906|2014-08-01 20:12:52| 119906| 11|2014-0
| 11| 160300|2012-05-15 22:08:58|2012-11-18 03:58:43| 160300| 41798|2014-04-08 22:36:49| 41798| 11|2013-0
| 11| 160300|2012-05-15 22:08:58|2012-11-18 03:58:43| 160300| 51792|2014-02-13 07:16:37| 51792| 11|2014-0
| 11| 160300|2012-05-15 22:08:58|2012-11-18 03:58:43| 160300| 119906|2013-05-16 07:41:13| 119906| 11|2014-0
| 11| 41798|2012-05-15 22:08:58|2013-08-01 23:04:08| 41798| 183628|2014-04-23 23:59:04| 183628| 11|2014-0
| 11| 41798|2012-05-15 22:08:58|2013-08-01 23:04:08| 41798| 51792|2014-01-11 03:15:37| 51792| 11|2014-0
| 11| 41798|2012-05-15 22:08:58|2013-08-01 23:04:08| 41798| 160300|2014-04-08 22:36:49| 160300| 11|2012-1
| 11| 171286|2012-05-15 22:08:58|2013-10-11 21:51:24| 171286| 183628|2013-08-04 06:47:26| 183628| 11|2014-0
| 11| 146271|2012-05-15 22:08:58|2013-10-13 07:12:24| 146271| 146274|2013-12-16 00:39:32| 146274| 11|2012-0
| 11| 183628|2012-05-15 22:08:58|2014-02-08 04:59:35| 183628| 171286|2013-08-04 06:47:26| 171286| 11|2013-1
| 11| 183628|2012-05-15 22:08:58|2014-02-08 04:59:35| 183628| 119906|2014-02-01 12:17:51| 119906| 11|2014-0
| 11| 183628|2012-05-15 22:08:58|2014-02-08 04:59:35| 183628| 41798|2014-04-23 23:59:04| 41798| 11|2013-0
| 11| 119906|2012-05-15 22:08:58|2014-03-15 00:22:34| 119906| 160300|2013-05-16 07:41:13| 160300| 11|2012-1
+---+---+-----+-----+-----+-----+-----+-----+-----+
only showing top 20 rows
```

Based on the above joint table, we applied the max function to judge the latest date of the three datetime values from three tables in each row. The latest date is the date when the triangle of the three users form. The dates when the triangles were created provided valuable information for us to indicate in which lifetime month of the t1_user he/she had formed such a triangle relationship.

Then, we calculated the cumulative number of triangles for each user1 over its lifetime months 0-12 using the SQL window function. The result is as follows:

```
# calculate the cumulative # triangles in each lifetime month for each user
num_tri_window = Window.partitionBy("user1").orderBy("lifetime")
from pyspark.sql.functions import sum, col
num_triangles_one_year = num_triangles_one_year.withColumn("cum_num_triangles",
                                                             sum("num_triangles").over(num_tri_window)).sort(col("user1"),
                                                                 col("lifetime"))

num_triangles_one_year.show(15)
```

user1	lifetime	num_triangles	cum_num_triangles
2	0	0.0	0.0
2	1	0.0	0.0
2	2	0.0	0.0
2	3	0.0	0.0
2	4	0.0	0.0
2	5	0.0	0.0
2	6	0.0	0.0
2	7	0.0	0.0
2	8	0.0	0.0
2	9	0.0	0.0
2	10	0.0	0.0
2	11	0.0	0.0
2	12	0.0	0.0
3	0	0.0	0.0
3	1	0.0	0.0

only showing top 15 rows

After this stage, we can divide the cumulative number of triangles by the number of possible vertices in the network. The result is shown below:

```
num_triangles_one_year.createOrReplaceTempView("t1")
num_vertices.createOrReplaceTempView("t2")

cluster_coef = spark.sql("""
    SELECT t1.user1, t1.lifetime,
           cum_num_triangles/NULLIF(num_vertices,0) AS cluster_coefficient
    FROM t1
    JOIN t2 ON t1.user1 = t2.user1
    AND t1.lifetime = t2.lifetime
    ORDER BY t1.user1, t1.lifetime
    """)

cluster_coef.show(15)
```

user1	lifetime	cluster_coefficient
2	0	null
2	1	null
2	2	null
2	3	null
2	4	null
2	5	null
2	6	null
2	7	null
2	8	null
2	9	null
2	10	null
2	11	null
2	12	null
3	0	null
3	1	0.0

only showing top 15 rows

iii) PageRank

In the context of social network analytics, PageRank is used to calculate the importance of an individual in their network. Since our friend network is an undirected graph here, we assumed that a user's PageRank would be higher if she/he just connects to more users.

Also, PageRank is a global metric. Since social networks are evolving continuously and different users are at different points in their lifetimes at a given point of time in real life, it is almost impossible to calculate the PageRank for each user at each lifetime point for them. Hence, we calculated PageRank for all users only once for the whole time frame of the data.

Using the package `networkx`, we generated a graph of all users, where individual users are nodes, and edges between the nodes indicate that the two connected users have transacted. After running the `PageRank` function, we obtained key-value pairs of users with their PageRanks to be used later in the regressions.

Predictive Analytics with MLlib

Q7. Create the dependent variable Y

For each user, we calculated the total number of transactions the user had completed in Venmo by lifetime¹². A sample of results is shown below.

```
↗
```

user1	num_transaction
2	1
3	6
4	2
10	7
11	6
12	4
13	4
16	3
19	1
28	1

only showing top 10 rows

Q8. Create the recency and frequency variables

In the analysis, we will focus on not only the spending profile and social network, but also the traditional user-based metrics in CRM analysis. In Question8, we introduced the RFM framework. Because the analysis does not involve monetary data, we mainly focus on Recency and Frequency.

Recency refers to how recent a user has made a transaction. It is the number of days since the last time a user was active. Frequency refers to how often a user makes a transaction. It is generally the number of transactions divided by the number of days in a specified time interval.

The SQL query below shows how we define recency and frequency. Recency is the number of days since the last transaction for a user at the time for this transaction. We cumulatively calculated frequency: the frequency of transactions for a user at a point in their lifetime is the total number of transactions until this point divided by the number of days elapsed since the first day of the user's lifetime started. To illustrate, the cumulative frequency of transactions for a user in month2 would be the total number of transactions in months 0 to 2 / 60 days.

```
# calculate frequency and recency
# frequency: how often a user uses Venmo in a month. It is standardized and equals to (number of transactions/30)
# recency: the last time a user was active
# if a user has used Venmo twice during her first month in Venmo with the second time being on day x,
# then her recency in month 1 is "30-x"
df_rfmodel_full.createOrReplaceTempView("df_rfmodel_full")
rf_model = spark.sql("SELECT user1, lifetime, lifetime_days, \
    IFNULL(((sum(num_txn) OVER (PARTITION BY user1 ORDER BY lifetime))/lifetime_days, 0) AS frequency, \
    (lifetime_days - MAX(latest_txn) \
    OVER(PARTITION BY user1 \
    ORDER BY lifetime ASC)) AS recency \
FROM df_rfmodel_full \
ORDER BY user1, lifetime")

rf_model.createOrReplaceTempView("rf_model")
rf_model.show(24)
```

The following table displays recency and frequency for users 2 and 3 in each lifetime point.

user1	lifetime	lifetime_days	latest_txn	num_txn	frequency	recency
2	0	0	0	1	null	0
2	1	30	null	null	0.0333333333333333	30
2	2	60	null	null	0.0166666666666666	60
2	3	90	null	null	0.0111111111111111	90
2	4	120	null	null	0.0083333333333333	120
2	5	150	null	null	0.0066666666666666	150
2	6	180	null	null	0.0055555555555555	180
2	7	210	null	null	0.0047619047619047	210
2	8	240	null	null	0.0041666666666666	240
2	9	270	null	null	0.0037037037037037	270
2	10	300	null	null	0.0033333333333333	300
2	11	330	null	null	0.0030303030303030	330
2	12	360	null	null	0.0027777777777777	360
3	0	0	0	1	null	0
3	1	30	17	5	0.2	13
3	2	60	null	null	0.1	43
3	3	90	null	null	0.0666666666666666	73
3	4	120	null	null	0.05	103
3	5	150	null	null	0.04	133
3	6	180	null	null	0.0333333333333333	163
3	7	210	null	null	0.0285714285714285	193
3	8	240	null	null	0.025	223
3	9	270	null	null	0.0222222222222222	253
3	10	300	null	null	0.02	283


Q9. Regression on Recency and Frequency

For this question, we tried two ways of calculating frequency: frequency for within each month and cumulative frequency as calculated above. For the first approach, the frequency of transactions for a user in month 2 would be *# of transactions in month 2 / 30 days*.

The first approach

Using the first approach, we regressed recency and frequency on the total number of transactions at lifetime in month 12 (y) for each user's lifetime points and plotted the MSE for each lifetime point.

The sample dataset of this question is shown in the following table.

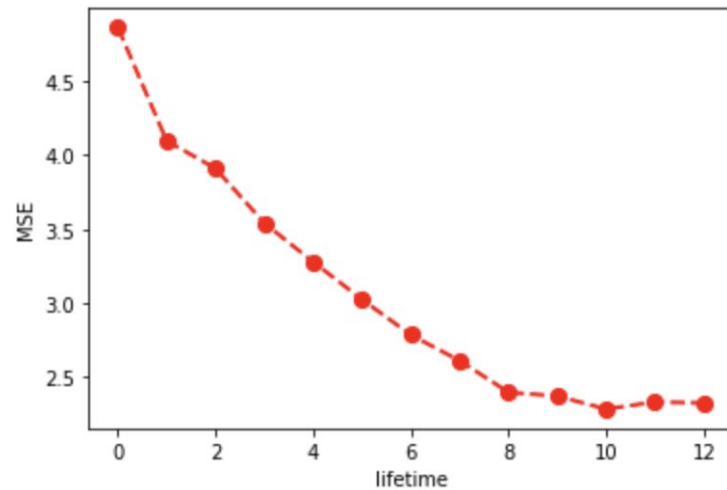


user1	lifetime	frequency	recency	num_transaction
2	0	0.03333333333333333	0	1
2	1	0.0	30	1
2	2	0.0	60	1
2	3	0.0	90	1
2	4	0.0	120	1
2	5	0.0	150	1
2	6	0.0	180	1
2	7	0.0	210	1
2	8	0.0	240	1
2	9	0.0	270	1
2	10	0.0	300	1
2	11	0.0	330	1
2	12	0.0	360	1
3	0	0.03333333333333333	0	6
3	1	0.16666666666666666	13	6
3	2	0.0	43	6
3	3	0.0	73	6
3	4	0.0	103	6
3	5	0.0	133	6
3	6	0.0	163	6

only showing top 20 rows

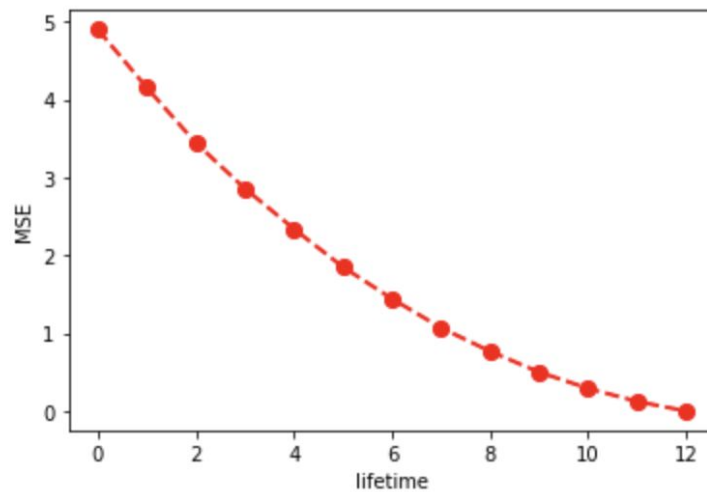
Firstly, we split the dataset to training and testing sets in a ratio of 0.7 to 0.3. Then we use the training set to build and train the linear regression model and test it on the testing set to see how well it performs.

We also reported the Mean Squared Error (MSE) for the linear regression model. The following graph illustrates the MSE for each lifetime point. The MSEs here, together with the MSE outputs in later questions in numerical values, can be found in the appendix in this report.



The second approach

Using the cumulative way of calculating frequency, we redid the regression with all else equal and obtained a new set of MSE plot for each lifetime point.



The cumulative approach of calculating frequency added much more predictive power, as this way of calculation incorporates all transactions up until each month, not just each individual month separately. However, this introduces a problem at month 12, when the

cumulative frequency is almost exactly collinear with the total number of transactions for 12 months (since total number of transactions = frequency*12 months*30days), hence the near-0 (3.0441×10^{-25}) MSE for month 12.

The MSEs for both regressions showed a steadily decreasing trend, since as we get closer to month 12 from 0, more data from the previous months are getting incorporated in the independent variables, and they would better predict the total number of transactions as an aggregated metric of spending behavior for that user in the first year of their lifetimes.

Q10. Regression on Recency, Frequency and Spending Profile

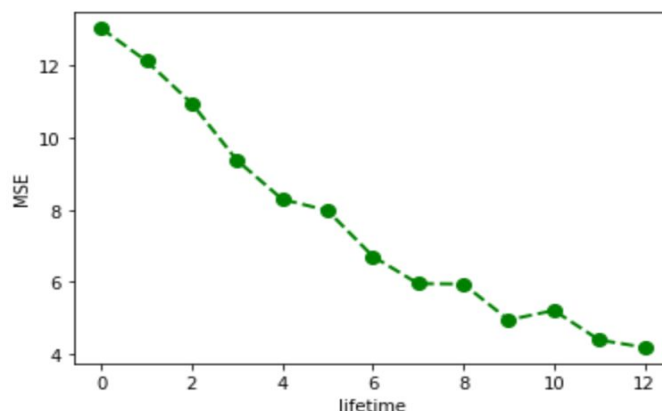
The independent variables in this question are users' recency, frequency, and their dynamic spending behaviors. The dependent variable is the total number of transactions at lifetime month 12 of the user. The sample dataset is shown in the following table:

user1	lifetime	Activity	Cash	Event	Food	Illegal_Sarcasm	Not_classified	People	Transportation	Travel	Utility
2	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
3	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
3	1	0.2	0.0	0.2	0.0	0.0	0.2	0.0	0.0	0.4	0.0
4	0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
4	1	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
10	1	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
10	2	0.5	0.0	0.0	0.0	0.0	0.5	0.0	0.0	0.0	0.0
10	4	0.0	0.0	0.0	0.5	0.0	0.0	0.5	0.0	0.0	0.0
10	5	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0

only showing top 10 rows

The ratio of training and test sets is 7/3. Compared with Question 9, we added the users' dynamic spending behaviors into the model and ran a regression for each month in lifetimes. The results of MSE in each month are plotted in the graph below. From the

results, we can conclude that adding the dynamic spending behaviors did not improve the predictive power of the model, since the MSE values overall incremented.

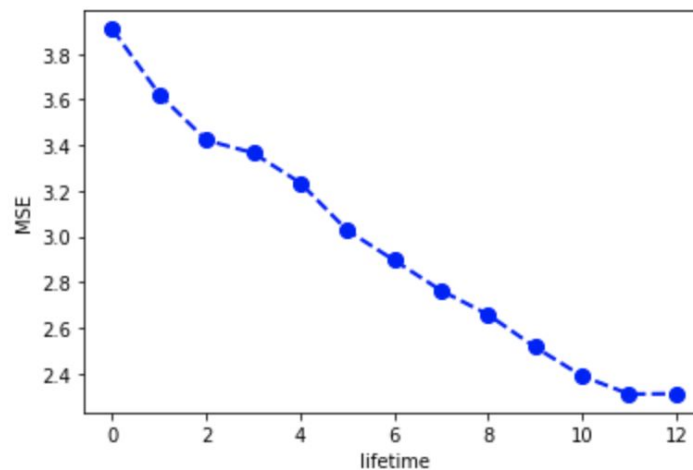


Q11. Regression on Social Network Metrics

The independent variables in this question are user's number of friends, number of friends of friends, number of triangles going through the user and page rank. The reason for us not using the clustering coefficient but the number of triangles is that some users have only 1 friend, making their clustering coefficient indefinite. We wanted to minimize the number of null values in the dataset, so we took the numerator of the clustering coefficient, number of triangles as an independent variable of the regression. In addition, we observed that the values of page rank were very minimal. In order not to affect the regression result, we adopted the z-score standardization method to scale the page rank, i.e. $(\text{page rank} - \text{mean}) / \text{standard deviation}$, when we wanted to see the regression output for month 12.

The dependent variable is the total number of transactions at lifetime month 12 of the user. The regression is dynamic and we ran one regression for each lifetime month. For

each month of lifetimes, we obtained one MSE and plotted the graph as below. Like the RF framework, the MSEs of social network models overall declined as the month in lifetime of the regression increased. However, the social network predictors produced smaller values of MSEs than RF models. We may conclude that social network metrics perform better than traditional RF models in predicting users' transaction counts in Venmo.



Since the data size was very large, the p-values of all the predictors were close to 0. Alternatively, we looked at the standard errors of the coefficients of each predictor, and it turned out that the β estimate of the **number of friends of friends** had the lowest standard error. Because a lower standard error indicates a higher precision of the slope coefficient, we suspect that the **number of friends of friends** has the strongest predictive power.

```
[46] print("Coefficients: %s" % str(lr_model.coef))
↳ Coefficients: [0.47149521753573875, 0.0010758762607959876, 0.25392387687192824, 0.28330846106808866]

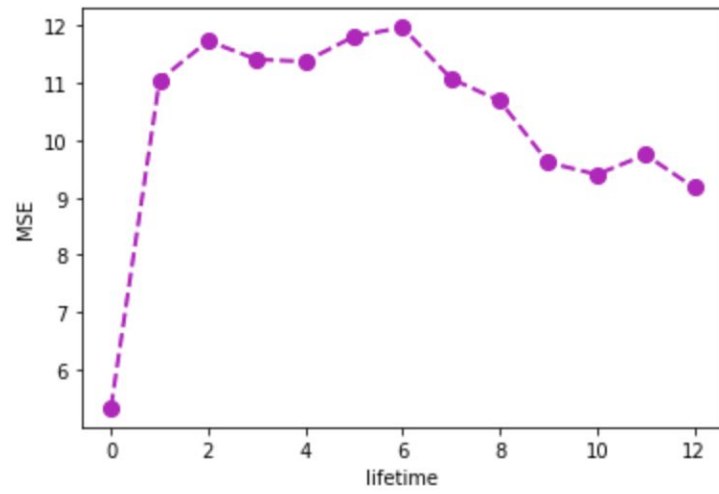
[47] print("P Values: " + str(trainingSummary.pValues))
↳ P Values: [0.0, 0.0, 0.0, 0.0, 0.0]

[48] print("Coefficient Standard Errors: " + str(trainingSummary.coefStandardErrors))
↳ Errors: [0.0005871103619620669, 3.4256845069149835e-05, 0.00066819957459062, 0.0011365252407445224, 0.0016983769658974383]
```

Q12. Regression on Social Network Metrics and Spending Profile of Social Network

In this regression analysis, the independent variables are 4 social network metrics including number of friends, number of friends of friends, number of triangles and page rank, and 10 transaction categories: activity, cash, event, food, illegal sarcasm, people, transportation, travel, utility, and not classified. The dynamic spending profile was calculated cumulatively by month, in the way that we computed the running average of the category proportions in each month in the lifetime of users.

The dynamic regression was conducted at each month in lifetimes, and the MSEs obtained in each period were visualized in the following line chart. Compared to the MSEs in the model with social network metrics only in Q11, the MSE figures dramatically increased after adding the spending profile predictors. Although the MSE gradually declines from lifetime month 6 to the end of one year, the fluctuation exhibited by the graph demonstrates that the predictive power of the model did not increase after adding the spending profile of users' social network.



Appendix. MSE output for each regression

Lifetime in month	Q9	Q9-cumulative freq	Q10	Q11	Q12
0	4.8662	4.9010	13.0266	3.9097	5.3376
1	4.0984	4.1537	12.1243	3.6235	11.0452
2	3.9113	3.4440	10.9507	3.4222	11.7284
3	3.5413	2.8586	9.3719	3.3661	11.4055
4	3.2782	2.3400	8.2937	3.2353	11.3693
5	3.0242	1.8505	7.9815	3.0272	11.8058
6	2.7841	1.4422	6.7134	2.8957	11.9645
7	2.6095	1.0701	5.9576	2.7629	11.0723
8	2.3953	0.7670	5.9463	2.6575	10.6863
9	2.3694	0.5006	4.9523	2.5151	9.6101
10	2.2816	0.2912	5.2149	2.3883	9.4018
11	2.3295	0.1237	4.3970	2.3090	9.7477
12	2.3236	0.0000	4.1928	2.3106	9.1743