Final Project Report

Vibons - Journey App - Team 1

Group members: Jiaqi Zhao, Lingyan Jiang, Ruoqi Shi

CS506 Computational Tools of Data Science

Instructor: Lance Galletti

Client: Vibons Journey App

Project Manager: Shubhangi Jain

Table of Contents

Abstract

In this paper, we presented how the given dataset by our clients, Journey App, is reevaluated and converted into a dataframe with the index of the user id. We demonstrated the thinking process of developing an algorithm that can find the optimal sending time of the notifications for each day of the week for current Journey App users. Based on our client's need, we focused on users with a completion rate from 85 to 100 inclusive. By mapping and grouping, we finally generated a dataframe with the notification sending time suggestion for each existing user.

1. Motivation

Journey App, designed by Vibons, is a motivational app for users to easily jot down their daily ideas, write down their daily emotions, and receive educational articles about customized topics through notifications. Nevertheless, as every user can be inundated with so many app notifications, our client wants to find an optimal time to notify the users of the contents and inspire them to read.

In a nutshell, based on our client's requirement, our goal is to find the send time optimization for each user of the Journey App using the dataset given by our clients. Send time optimization analyzes when is the most likely time for each recipient to open the notifications. Such an analysis can boost the open rates for our client and enhance engagement with all the users.

2. Dataset and Representation

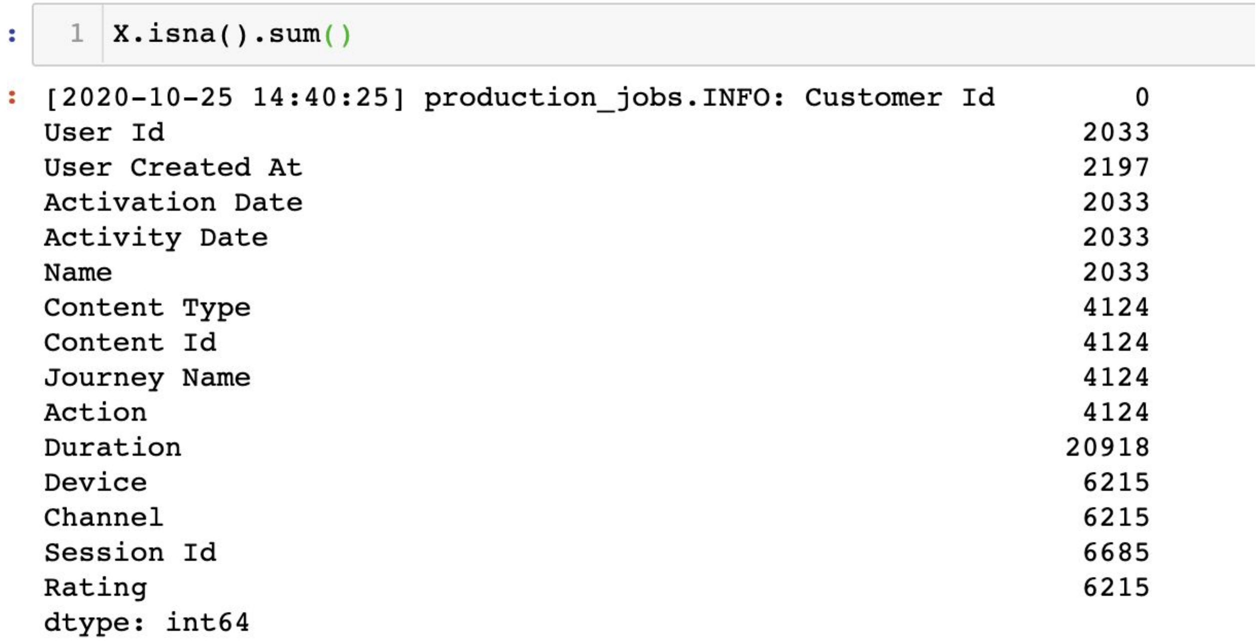
Our dataset is given by our client - Journey App. Thus, our group did not need to collect the data by ourselves.

For all the features, we knew their meanings before we preprocessed the data. For the definition of data, since each company has its own idea of formulating data titles, the meaning of data may be completely different. Therefore, we asked our customers in detail. We will also add the explanation of the data labels in the README file so that people can understand clearly our data.

- Info Customer ID - The ID number of the users' company, there are about 20 company IDs in total.
- User Id - Each user's Id on behalf of which notifications are sent.
- User-Created At - The first date of hire of the user to whom notification is sent.
- Activation Date - Date and time when a notification is sent.
- Activity Date - Date and time when a notification is opened by the receiving user.
- Name - The title of the notification content.
- Content ID - the ID of the notification content name.
- Content Type - The type of notification content.
- Journey Name - The journey that the content of the notification comes from.
- Action - Percentage of content read by the end-user.
- Device - The device used by the user to read the notification. Due to the current universality and convenience of mobile phones, the focus is on mobile phone users.
- Channel - The channel that users click on the notification.
- Session Id: When a notification is sent to the user, a session is locked in the system.

3. Data Preprocessing

After loading the CSV file as dataframe, we first checked about NaN values inside our dataframe:



```
X.isna().sum()
```

```
[2020-10-25 14:40:25] production_jobs.INFO: Customer Id       0
User Id                                                    2033
User Created At                                            2197
Activation Date                                            2033
Activity Date                                              2033
Name                                                       2033
Content Type                                               4124
Content Id                                                 4124
Journey Name                                               4124
Action                                                     4124
Duration                                                  20918
Device                                                     6215
Channel                                                    6215
Session Id                                                 6685
Rating                                                     6215
dtype: int64
```

For content type, device, and channel, they can be categorical variables. So we converted them into categorical variables with integers starting from one and if we detect a NaN value, we labeled it as zero.

Then, we used fillna() method to replace all the NaN values with the mode corresponding to each variable. The reason why we fit the mode value here is that we want the tendency of the overall users but we do not want outliers (if exist) to influence the data. We used dropna() method to drop the row if there is a NaN value for that user.

At last, we deleted “FLIPBOOK” entries inside the column “User Id”.

4. Technology Approach

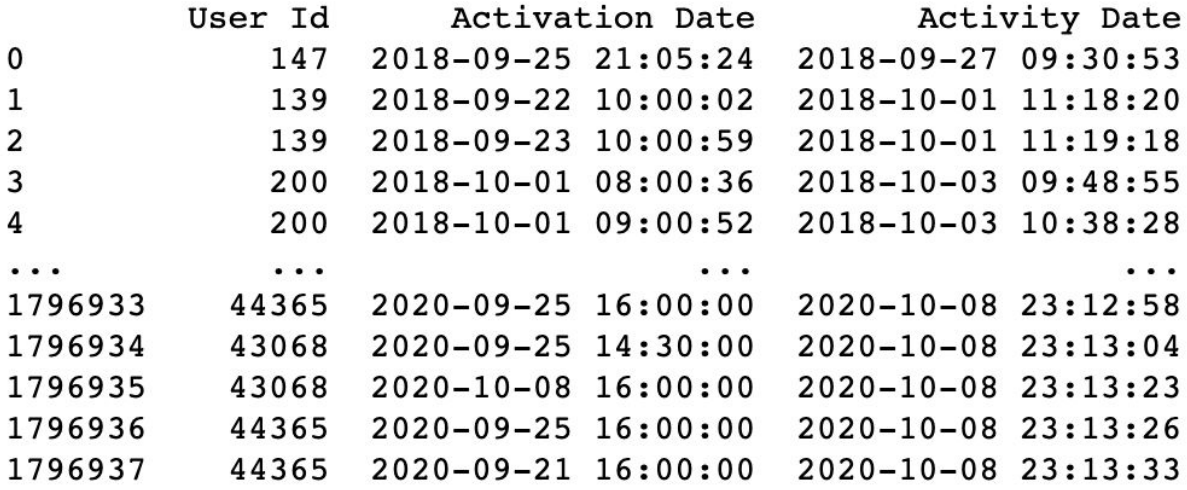
Iteration 1:

Aim:

To find the optimal notification sending time, we used the activity date column to parse out the hour and find out the hour time with the highest frequency number and stored in a dictionary.

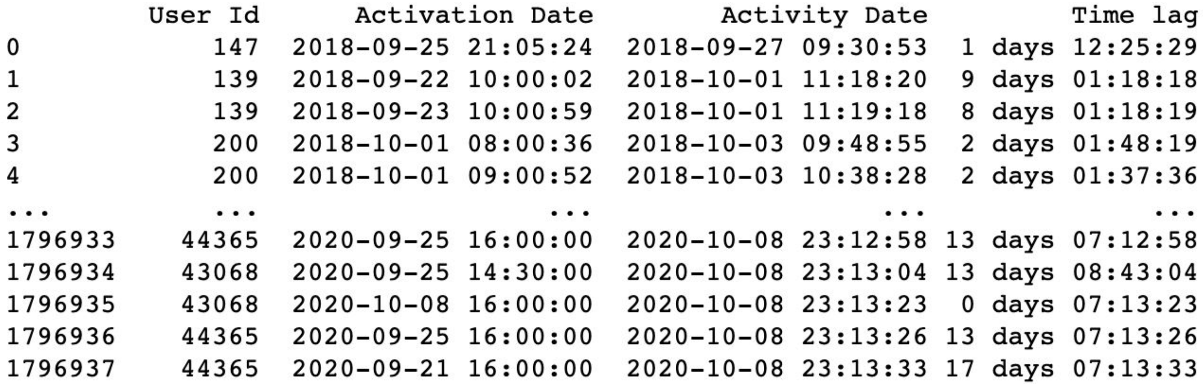
Strategy, Result and Visualization:

1. Initial Stage:
   We created a new dataframe called open_time with columns of User Id, Activation Date, and Activity Date from clean_data.



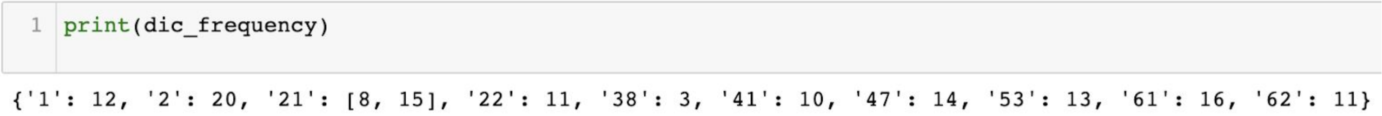
```
         User Id     Activation Date        Activity Date
0            147 2018-09-25 21:05:24  2018-09-27 09:30:53
1            139 2018-09-22 10:00:02  2018-10-01 11:18:20
2            139 2018-09-23 10:00:59  2018-10-01 11:19:18
3            200 2018-10-01 08:00:36  2018-10-03 09:48:55
4            200 2018-10-01 09:00:52  2018-10-03 10:38:28
...          ...                 ...                  ...
1796933    44365 2020-09-25 16:00:00  2020-10-08 23:12:58
1796934    43068 2020-09-25 14:30:00  2020-10-08 23:13:04
1796935    43068 2020-10-08 16:00:00  2020-10-08 23:13:23
1796936    44365 2020-09-25 16:00:00  2020-10-08 23:13:26
1796937    44365 2020-09-21 16:00:00  2020-10-08 23:13:33
```

2. Developing Stage 1:
   By this dataframe, we were able to find out the time lag between activity date and activation date by subtracting activation date from activity date, which is the time lag between the date and time when a notification is sent and when a notification is opened by the receiving user. The result is stored in a new column Time lag in open_time data frame shown below:



```
         User Id     Activation Date        Activity Date          Time lag
0            147 2018-09-25 21:05:24  2018-09-27 09:30:53  1 days 12:25:29
1            139 2018-09-22 10:00:02  2018-10-01 11:18:20  9 days 01:18:18
2            139 2018-09-23 10:00:59  2018-10-01 11:19:18  8 days 01:18:19
3            200 2018-10-01 08:00:36  2018-10-03 09:48:55  2 days 01:48:19
4            200 2018-10-01 09:00:52  2018-10-03 10:38:28  2 days 01:37:36
...          ...                 ...                  ...              ...
1796933    44365 2020-09-25 16:00:00  2020-10-08 23:12:58 13 days 07:12:58
1796934    43068 2020-09-25 14:30:00  2020-10-08 23:13:04 13 days 08:43:04
1796935    43068 2020-10-08 16:00:00  2020-10-08 23:13:23  0 days 07:13:23
1796936    44365 2020-09-25 16:00:00  2020-10-08 23:13:26 13 days 07:13:26
1796937    44365 2020-09-21 16:00:00  2020-10-08 23:13:33 17 days 07:13:33
```

3. Developing Stage 2:
    a. We want to create a dictionary for the “hr” item of each user so that we can give the optimal sending time by observing the frequency of the “hr”. For here, we used the first 10 users as an example. By importing *datetime.strptime* library, we can easily parse out the hour from the activity date.
    b. Then, we stored the user id as the key and all the corresponding hours as a list for the value of the key. Next, we want to find the frequency for each user. We used Counter() and most_common() function from collections library.
    For each user, we got the first two highest frequency hours for each user id by most_common(). This method gives us a result of all the hours of a certain user with the frequency of hour as the value in a dictionary ranking by the highest frequency to the lowest one.
    c. By setting the parameter in the most_common() function as “2”, only the first 2 most common values of the “hour” item will be shown. If there's only one result of the most_common() function, we will store it directly in our dictionary. If one of the “hour” is higher than the other one, we will only store the highest one. If the frequencies of the two “hour” items are the same, we will save both hour time towards the user id as a list in the dictionary. By this method, the result of the first ten users are shown below:



```
print(dic_frequency)
```

```
{'1': 12, '2': 20, '21': [8, 15], '22': 11, '38': 3, '41': 10, '47': 14, '53': 13, '61': 16, '62': 11}
```

Insights:

In this iteration, we used a dictionary to find out the corresponding hour time with the highest frequency for each user. The reason why we are using a dictionary is that it is easy for the client to find out the corresponding hour time for each user by searching the user id.

Limitations and Challenge:

First of all, though a dictionary is easy for our client to query the sending time for each user, we did not pay attention to the high running time of the dictionary. It takes us a really long time to generate the result. Thus, it is important for us to find out a new data structure to store and run the algorithm. We choose dataframe from the pandas library. The built-in functions inside this library can reduce and avoid the use of for loops which can save a lot of running time.

Secondly, we did not categorize the hour time with different weekdays. It is pivotal to find out the optimal sending time with the corresponding day of the week. We should not consider the seven days as a whole because our client might need to send several notifications in a week. We should give our client which day of the week and the sending time together.

Thirdly, in this iteration, we only consider the highest one. From our result, there’s a case that the *most_common(2)* gives as two hour times with the same frequency. In future iterations, we need to increase the number chosen for the most_common() function. Or we might need new criteria to choose the top n th frequency for each user.

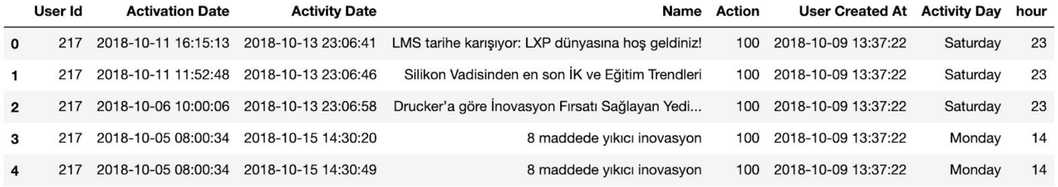
Iteration 2:

Aim:

To find the optimal notification sending time based on the existing the “Activity date” column (when the notification was opened) and “Action” column (the completion rate when a user opens the notification) in the dataset of the existing users

Strategy, results and visualizations

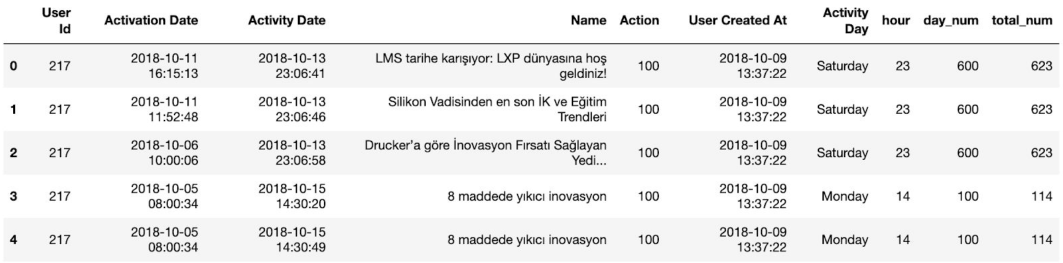
1. Initial stage:
   a. Based on our client’s requirement, we deleted the “time difference” and “time lag” which we created in Iteration 1.
   b. We focused on the “activity date” and “action rate”
2. Data preprocessing:
   a. First of all, we converted each date into the corresponding weekday of that week, as shown in the ‘Activity Day’ column
   b. We then extracted the hour time of the notification open time for each row, as shown in the ‘hour’ column.



	User Id	Activation Date	Activity Date	Name	Action	User Created At	Activity Day	hour
0	217	2018-10-11 16:15:13	2018-10-13 23:06:41	LMS tarihe karışıyor: LXP dünyasına hoş geldiniz!	100	2018-10-09 13:37:22	Saturday	23
1	217	2018-10-11 11:52:48	2018-10-13 23:06:46	Silikon Vadisinden en son İK ve Eğitim Trendleri	100	2018-10-09 13:37:22	Saturday	23
2	217	2018-10-06 10:00:06	2018-10-13 23:06:58	Drucker’a göre İnovasyon Fırsatı Sağlayan Yedi...	100	2018-10-09 13:37:22	Saturday	23
3	217	2018-10-05 08:00:34	2018-10-15 14:30:20	8 maddede yıkıcı inovasyon	100	2018-10-09 13:37:22	Monday	14
4	217	2018-10-05 08:00:34	2018-10-15 14:30:49	8 maddede yıkıcı inovasyon	100	2018-10-09 13:37:22	Monday	14

   Take the first row as an example. The ‘hour’ column for the first row is ‘23’. This means that user 217 opened this particular notification at 23:00, i.e. 11 pm, on October 13th, 2018 (as shown in the ‘Activity Date’ column of the first row)

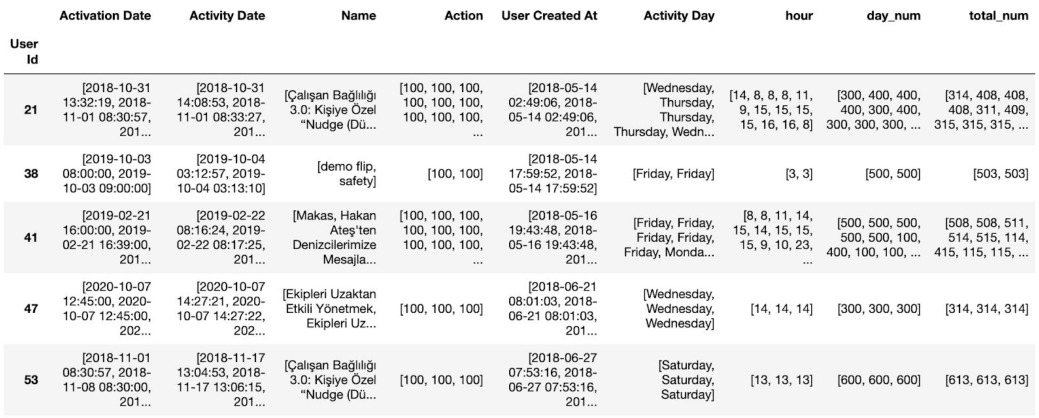
   c. According to the day of the week, we converted it into an integer stored in a column called hour_num. For instance, Monday was converted into 100; Tuesday was converted into 200. After that, each hour was stored as an integer in a column called day_num. So it is easy to do calculations and statistics compared with a string.
   d. Then, we added day_num and hour_num together as a new column called total_num.



	User Id	Activation Date	Activity Date	Name	Action	User Created At	Activity Day	hour	day_num	total_num
0	217	2018-10-11 16:15:13	2018-10-13 23:06:41	LMS tarihe karışıyor: LXP dünyasına hoş geldiniz!	100	2018-10-09 13:37:22	Saturday	23	600	623
1	217	2018-10-11 11:52:48	2018-10-13 23:06:46	Silikon Vadisinden en son İK ve Eğitim Trendleri	100	2018-10-09 13:37:22	Saturday	23	600	623
2	217	2018-10-06 10:00:06	2018-10-13 23:06:58	Drucker’a göre İnovasyon Fırsatı Sağlayan Yedi...	100	2018-10-09 13:37:22	Saturday	23	600	623
3	217	2018-10-05 08:00:34	2018-10-15 14:30:20	8 maddede yıkıcı inovasyon	100	2018-10-09 13:37:22	Monday	14	100	114
4	217	2018-10-05 08:00:34	2018-10-15 14:30:49	8 maddede yıkıcı inovasyon	100	2018-10-09 13:37:22	Monday	14	100	114

Take the first row of the screenshot above as an example, the ‘total_num’ column is ‘623’. This means that the user opens a notification on Saturday (‘6’) at 11 pm (‘23’).

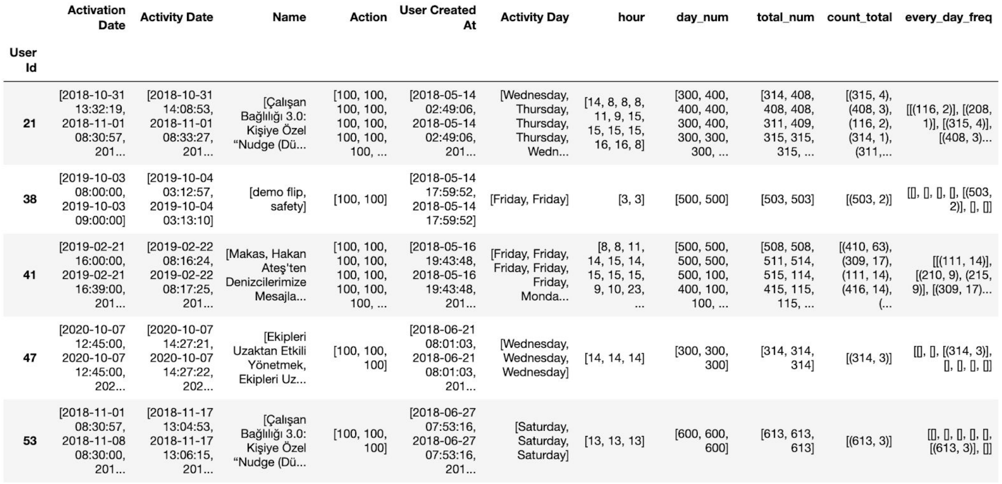
e. Next, we aggregated each row by user id. The user id became the index of each row.



	Activation Date	Activity Date	Name	Action	User Created At	Activity Day	hour	day_num	total_num
User Id									
21	[2018-10-31 13:32:19, 2018-11-01 08:30:57, 201...	[2018-10-31 14:08:53, 2018-11-01 08:33:27, 201...	[Çalışan Bağlılığı 3.0: Kişiye Özel “Nudge (Dü...	[100, 100, 100, 100, 100, 100, 100, 100, 100, ...	[2018-05-14 02:49:06, 2018-05-14 02:49:06, 201...	[Wednesday, Thursday, Thursday, Thursday, Wedn...	[14, 8, 8, 8, 11, 9, 15, 15, 15, 15, 16, 16, 8]	[300, 400, 400, 400, 300, 400, 300, 300, 300, ...	[314, 408, 408, 408, 311, 409, 315, 315, 315, ...
38	[2019-10-03 08:00:00, 2019-10-03 09:00:00]	[2019-10-04 03:12:57, 2019-10-04 03:13:10]	[demo flip, safety]	[100, 100]	[2018-05-14 17:59:52, 2018-05-14 17:59:52]	[Friday, Friday]	[3, 3]	[500, 500]	[503, 503]
41	[2019-02-21 16:00:00, 2019-02-21 16:39:00, 201...	[2019-02-22 08:16:24, 2019-02-22 08:17:25, 201...	[Makas, Hakan Ateş'ten Denizcilerimize Mesajla...	[100, 100, 100, 100, 100, 100, 100, 100, 100, ...	[2018-05-16 19:43:48, 2018-05-16 19:43:48, 201...	[Friday, Friday, Friday, Friday, Friday, Monda...	[8, 8, 11, 14, 15, 14, 15, 15, 15, 9, 10, 23, ...	[500, 500, 500, 500, 500, 100, 400, 100, 100, ...	[508, 508, 511, 514, 515, 114, 415, 115, 115, ...
47	[2020-10-07 12:45:00, 2020-10-07 12:45:00, 202...	[2020-10-07 14:27:21, 2020-10-07 14:27:22, 202...	[Ekipleri Uzaktan Etkili Yönetmek, Ekipleri Uz...	[100, 100, 100]	[2018-06-21 08:01:03, 2018-06-21 08:01:03, 201...	[Wednesday, Wednesday, Wednesday]	[14, 14, 14]	[300, 300, 300]	[314, 314, 314]
53	[2018-11-01 08:30:57, 2018-11-08 08:30:00, 201...	[2018-11-17 13:04:53, 2018-11-17 13:06:15, 201...	[Çalışan Bağlılığı 3.0: Kişiye Özel “Nudge (Dü...	[100, 100, 100]	[2018-06-27 07:53:16, 2018-06-27 07:53:16, 201...	[Saturday, Saturday, Saturday]	[13, 13, 13]	[600, 600, 600]	[613, 613, 613]

The screenshot above shows the result after aggregating rows with the same user id. Take the first row of the screenshot above as an example. The ‘total_num’ column becomes a list of [315, 408, 408, 408, 311, 409, 315, 315, 315, ...]. This list shows when and what time did this user 21 opened the notifications in the whole dataset.

f. Then, we went through the total number and found out the frequency/impression (numbers of times) of opening the notification at a specific hour time for each user on each day of the week. Our algorithm gave a list with seven elements corresponding from Monday to Sunday with the hour time and frequency as a list in it.



	Activation Date	Activity Date	Name	Action	User Created At	Activity Day	hour	day_num	total_num	count_total	every_day_freq
User Id											
21	[2018-10-31 13:32:19, 2018-11-01 08:30:57, 201...	[2018-10-31 14:08:53, 2018-11-01 08:33:27, 201...	[Çalışan Bağlılığı 3.0: Kişiye Özel “Nudge (Dü...	[100, 100, 100, 100, 100, 100, 100, 100, 100, ...	[2018-05-14 02:49:06, 2018-05-14 02:49:06, 201...	[Wednesday, Thursday, Thursday, Thursday, Wedn...	[14, 8, 8, 8, 11, 9, 15, 15, 15, 15, 16, 16, 8]	[300, 400, 400, 400, 300, 400, 300, 300, 300, ...	[314, 408, 408, 408, 311, 409, 315, 315, 315, ...	[(315, 4), (408, 3), (116, 2), (314, 1), (311,...	[[(116, 2)], [(208, 1)], [(315, 4)], [(408, 3)...
38	[2019-10-03 08:00:00, 2019-10-03 09:00:00]	[2019-10-04 03:12:57, 2019-10-04 03:13:10]	[demo flip, safety]	[100, 100]	[2018-05-14 17:59:52, 2018-05-14 17:59:52]	[Friday, Friday]	[3, 3]	[500, 500]	[503, 503]	[(503, 2)]	[[], [], [], [], [(503, 2)], [], []]
41	[2019-02-21 16:00:00, 2019-02-21 16:39:00, 201...	[2019-02-22 08:16:24, 2019-02-22 08:17:25, 201...	[Makas, Hakan Ateş'ten Denizcilerimize Mesajla...	[100, 100, 100, 100, 100, 100, 100, 100, 100, ...	[2018-05-16 19:43:48, 2018-05-16 19:43:48, 201...	[Friday, Friday, Friday, Friday, Friday, Monda...	[8, 8, 11, 14, 15, 14, 15, 15, 15, 9, 10, 23, ...	[500, 500, 500, 500, 500, 100, 400, 100, 100, ...	[508, 508, 511, 514, 515, 114, 415, 115, 115, ...	[(410, 63), (309, 17), (111, 14), (416, 14), (...	[[(111, 14)], [(210, 9), (215, 9)], [(309, 17)...
47	[2020-10-07 12:45:00, 2020-10-07 12:45:00, 202...	[2020-10-07 14:27:21, 2020-10-07 14:27:22, 202...	[Ekipleri Uzaktan Etkili Yönetmek, Ekipleri Uz...	[100, 100, 100]	[2018-06-21 08:01:03, 2018-06-21 08:01:03, 201...	[Wednesday, Wednesday, Wednesday]	[14, 14, 14]	[300, 300, 300]	[314, 314, 314]	[(314, 3)]	[[], [], [(314, 3)], [], [], [], []]
53	[2018-11-01 08:30:57, 2018-11-08 08:30:00, 201...	[2018-11-17 13:04:53, 2018-11-17 13:06:15, 201...	[Çalışan Bağlılığı 3.0: Kişiye Özel “Nudge (Dü...	[100, 100, 100]	[2018-06-27 07:53:16, 2018-06-27 07:53:16, 201...	[Saturday, Saturday, Saturday]	[13, 13, 13]	[600, 600, 600]	[613, 613, 613]	[(613, 3)]	[[], [], [], [], [], [(613, 3)], []]

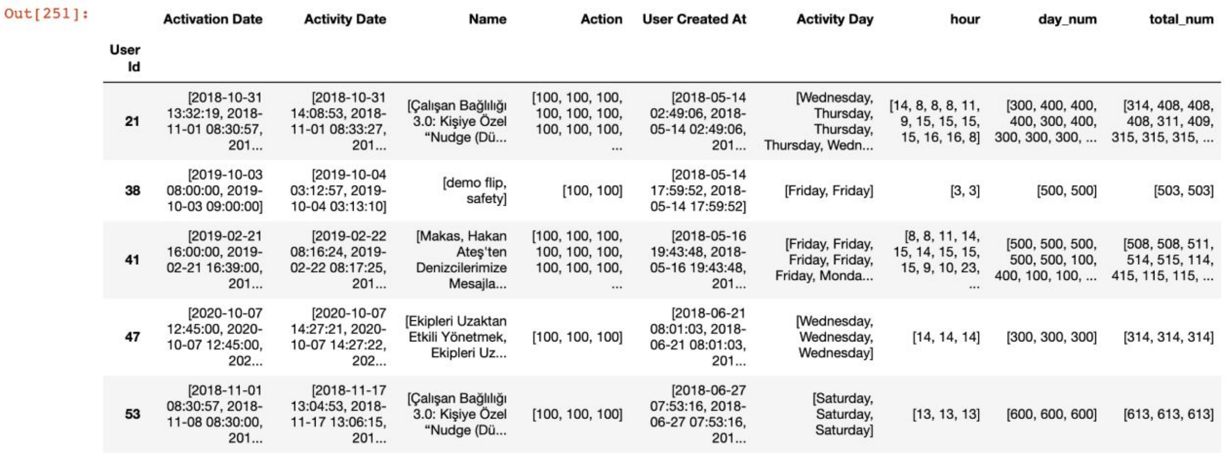
3. Developing Stage 1 :

a. Goal: We focused on the users who have a completion rate of 100 and found the hour time on each day of the week for each user when they open the notification and complete 100% of the contents.

b. Steps:

   i. Subtract all the users who have a completion rate of "100"

   ii. Under the condition of completion rate equal to "100", create a list of the days and the hour time about when they opened the notification for each user, as shown in the 'total_num'column in the screenshot below:

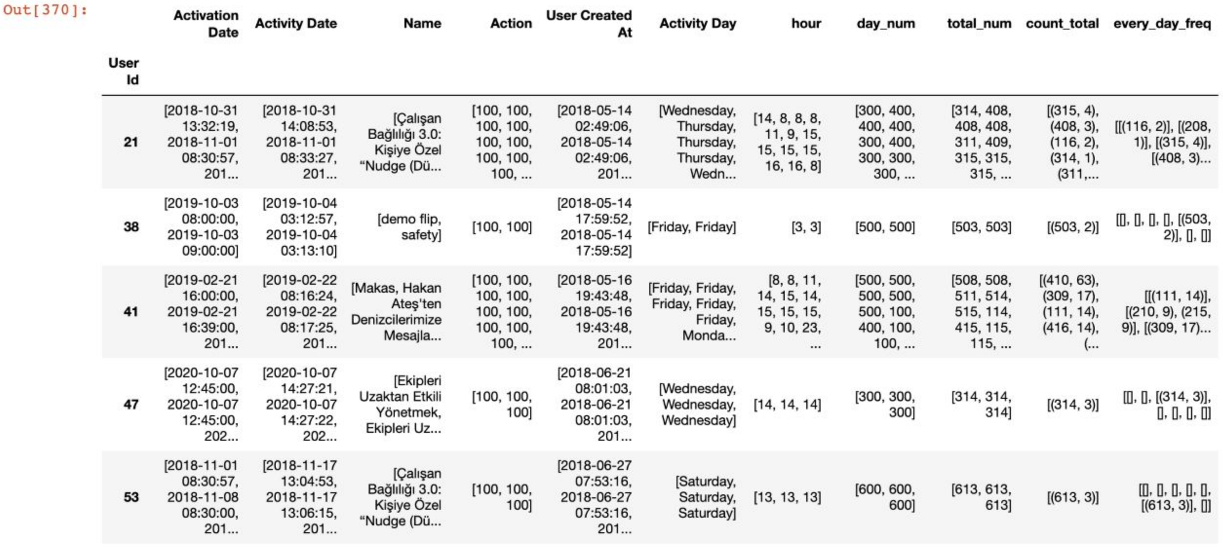


Out[251]:

User Id	Activation Date	Activity Date	Name	Action	User Created At	Activity Day	hour	day_num	total_num
21	[2018-10-31 13:32:19, 2018-11-01 08:30:57, 201...	[2018-10-31 14:08:53, 2018-11-01 08:33:27, 201...	[Çalışan Bağlılığı 3.0: Kişiye Özel "Nudge (Dü...	[100, 100, 100, 100, 100, 100, 100, 100, 100, ...	[2018-05-14 02:49:06, 2018-05-14 02:49:06, 201...	[Wednesday, Thursday, Thursday, Thursday, Wedn...	[14, 8, 8, 8, 11, 9, 15, 15, 15, 15, 16, 16, 8]	[300, 400, 400, 400, 300, 400, 300, 300, 300, ...	[314, 408, 408, 408, 311, 409, 315, 315, 315, ...
38	[2019-10-03 08:00:00, 2019-10-03 09:00:00]	[2019-10-04 03:12:57, 2019-10-04 03:13:10]	[demo flip, safety]	[100, 100]	[2018-05-14 17:59:52, 2018-05-14 17:59:52]	[Friday, Friday]	[3, 3]	[500, 500]	[503, 503]
41	[2019-02-21 16:00:00, 2019-02-21 16:39:00, 201...	[2019-02-22 08:16:24, 2019-02-22 08:17:25, 201...	[Makas, Hakan Ateş'ten Denizcilerimize Mesajla...	[100, 100, 100, 100, 100, 100, 100, 100, 100, ...	[2018-05-16 19:43:48, 2018-05-16 19:43:48, 201...	[Friday, Friday, Friday, Friday, Friday, Monda...	[8, 8, 11, 14, 15, 14, 15, 15, 15, 9, 10, 23, ...	[500, 500, 500, 500, 500, 100, 400, 100, 100, ...	[508, 508, 511, 514, 515, 114, 415, 115, 115, ...
47	[2020-10-07 12:45:00, 2020-10-07 12:45:00, 202...	[2020-10-07 14:27:21, 2020-10-07 14:27:22, 202...	[Ekipleri Uzaktan Etkili Yönetmek, Ekipleri Uz...	[100, 100, 100]	[2018-06-21 08:01:03, 2018-06-21 08:01:03, 201...	[Wednesday, Wednesday, Wednesday]	[14, 14, 14]	[300, 300, 300]	[314, 314, 314]
53	[2018-11-01 08:30:57, 2018-11-08 08:30:00, 201...	[2018-11-17 13:04:53, 2018-11-17 13:06:15, 201...	[Çalışan Bağlılığı 3.0: Kişiye Özel "Nudge (Dü...	[100, 100, 100]	[2018-06-27 07:53:16, 2018-06-27 07:53:16, 201...	[Saturday, Saturday, Saturday]	[13, 13, 13]	[600, 600, 600]	[613, 613, 613]

(In the 'total_num'column, the number "314", for example, means that User 21 opened the notification on Wednesday-represented as "3", at 14:00, or 2 pm.)

   iii. List out the frequency/impression (numbers of times) of opening the notification at a specific hour time for each user on each day of the week, as shown in the 'every_day_freq'column in the screenshot below:

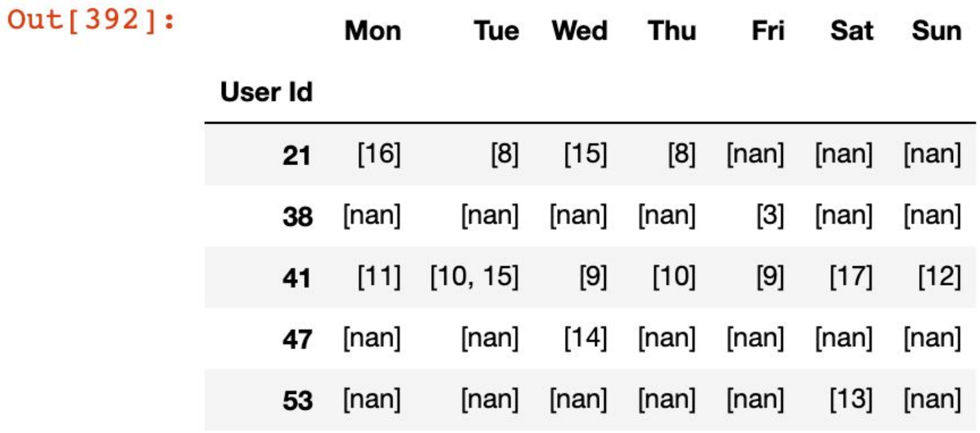


Out[370]:

User Id	Activation Date	Activity Date	Name	Action	User Created At	Activity Day	hour	day_num	total_num	count_total	every_day_freq
21	[2018-10-31 13:32:19, 2018-11-01 08:30:57, 201...	[2018-10-31 14:08:53, 2018-11-01 08:33:27, 201...	[Çalışan Bağlılığı 3.0: Kişiye Özel "Nudge (Dü...	[100, 100, 100, 100, 100, 100, 100, 100, 100, ...	[2018-05-14 02:49:06, 2018-05-14 02:49:06, 201...	[Wednesday, Thursday, Thursday, Thursday, Wedn...	[14, 8, 8, 8, 11, 9, 15, 15, 15, 16, 16, 8]	[300, 400, 400, 400, 300, 400, 300, 300, 300, ...	[314, 408, 408, 408, 311, 409, 315, 315, 315, ...	[(315, 4), (408, 3), (116, 2), (314, 1), (311,...	[[(116, 2)], [(208, 1)], [(315, 4)], [(408, 3)...
38	[2019-10-03 08:00:00, 2019-10-03 09:00:00]	[2019-10-04 03:12:57, 2019-10-04 03:13:10]	[demo flip, safety]	[100, 100]	[2018-05-14 17:59:52, 2018-05-14 17:59:52]	[Friday, Friday]	[3, 3]	[500, 500]	[503, 503]	[(503, 2)]	[[], [], [], [], [(503, 2)], [], []]
41	[2019-02-21 16:00:00, 2019-02-21 16:39:00, 201...	[2019-02-22 08:16:24, 2019-02-22 08:17:25, 201...	[Makas, Hakan Ateş'ten Denizcilerimize Mesajla...	[100, 100, 100, 100, 100, 100, 100, 100, 100, ...	[2018-05-16 19:43:48, 2018-05-16 19:43:48, 201...	[Friday, Friday, Friday, Friday, Friday, Monda...	[8, 8, 11, 14, 15, 14, 15, 15, 15, 9, 10, 23, ...	[500, 500, 500, 500, 500, 100, 400, 100, 100, ...	[508, 508, 511, 514, 515, 114, 415, 115, 115, ...	[(410, 63), (309, 17), (111, 14), (416, 14), (...	[[(111, 14)], [(210, 9), (215, 9)], [(309, 17)...
47	[2020-10-07 12:45:00, 2020-10-07 12:45:00, 202...	[2020-10-07 14:27:21, 2020-10-07 14:27:22, 202...	[Ekipleri Uzaktan Etkili Yönetmek, Ekipleri Uz...	[100, 100, 100]	[2018-06-21 08:01:03, 2018-06-21 08:01:03, 201...	[Wednesday, Wednesday, Wednesday]	[14, 14, 14]	[300, 300, 300]	[314, 314, 314]	[(314, 3)]	[[], [], [(314, 3)], [], [], [], []]
53	[2018-11-01 08:30:57, 2018-11-08 08:30:00, 201...	[2018-11-17 13:04:53, 2018-11-17 13:06:15, 201...	[Çalışan Bağlılığı 3.0: Kişiye Özel "Nudge (Dü...	[100, 100, 100]	[2018-06-27 07:53:16, 2018-06-27 07:53:16, 201...	[Saturday, Saturday, Saturday]	[13, 13, 13]	[600, 600, 600]	[613, 613, 613]	[(613, 3)]	[[], [], [], [], [], [(613, 3)], []]

(In the ‘every_day_freq’column, (116,2) means that User 21 opened the notification at 16:00, or 4 pm, on Monday twice. For User 38, you can see some empty ‘[ ]’ in it. This means that for other days of the week except for Friday, this user did not 100% complete any article after opening the notifications.)

- iv. We then chose the hour time with the highest impression for each day as the optimal notification sending time for a particular user on a particular day of a week.
    - After the two steps above, we got the following result, shown in the screenshot.



Out[392]:

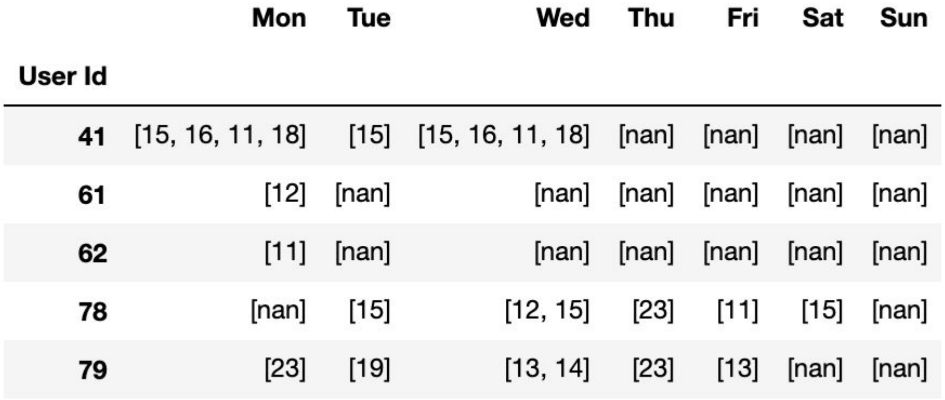
User Id	Mon	Tue	Wed	Thu	Fri	Sat	Sun
21	[16]	[8]	[15]	[8]	[nan]	[nan]	[nan]
38	[nan]	[nan]	[nan]	[nan]	[3]	[nan]	[nan]
41	[11]	[10, 15]	[9]	[10]	[9]	[17]	[12]
47	[nan]	[nan]	[14]	[nan]	[nan]	[nan]	[nan]
53	[nan]	[nan]	[nan]	[nan]	[nan]	[13]	[nan]

(For example, for User 21, we saved ‘16’ for Monday. This means that we suggest sending the notification to User 21 at 16:00, i.e. 4 pm. We chose ‘16’ because, in the last screenshot, we found that ‘116’ has the highest frequency for this user, where ‘1’indicates Monday and ‘16’ indicates 16:00.)

4. Development Stage 2:
    a. Goal: To fill the 'nan' values in the results we got from the last stage for every user, which means to find the optimal, we will go through the time when each user opened the notification but with a different completion rate.
    b. Steps:
        - i. We first continued to distinguish between different daily best notification sending times when users are at different completion rates.
            - Here, in order to find out the situation of the completion rate and write code conveniently, we divided the completion rate into six buckets: 100 (analyzed in developing stage 1), 90-100, 80-90, 50-80, 0-50, 0.

    ii. We then repeated the steps in developing stage 1 for each bucket and found the best notification sending time each day under different completion rates
    iii. When the best delivery time of the day was not found at completion rate=100, we chose to use completion rate = 90-100 instead, and so on until completion close to 0.

c. Results:
Take the bucket when the completion rate equals 90-100 as an example:



	Mon	Tue	Wed	Thu	Fri	Sat	Sun
User Id							
41	[15, 16, 11, 18]	[15]	[15, 16, 11, 18]	[nan]	[nan]	[nan]	[nan]
61	[12]	[nan]	[nan]	[nan]	[nan]	[nan]	[nan]
62	[11]	[nan]	[nan]	[nan]	[nan]	[nan]	[nan]
78	[nan]	[15]	[12, 15]	[23]	[11]	[15]	[nan]
79	[23]	[19]	[13, 14]	[23]	[13]	[nan]	[nan]

These are the first five users when their completion rate is less than 100. Take User 41 as an example. We saved a list of ‘15,16,11,18’ for Monday. This means that we suggest sending the notification to User 41 on Monday at 15:00, 16:00, 11:00 and 18:00, i.e. 3 pm, 4 pm, 11 am and 6 pm. The ‘nan’ value here indicates that the user does not have a completion rate of 90-100 on the particular day of the week.

Lessons Learned and Insights:

First of all, we learned the importance of simplifying the elements when the dataset is pretty large. Our dataset is a quite large one with almost 2 million data entries, specifically 1926480 rows. Thus, for instance, it will take a long time for the algorithm to process if we are dealing with the weekday element in the format of string. So we creatively turned the weekday element from string to numbers, such as ‘100’, as shown in the ‘day_num’ column. So it is much easier to do calculations and statistics compared with a string.

Second, we learned the importance of result demonstration. This means that sometimes for our client, they care more about the results regarding the primary goal of the project. They care less about what approaches or how did we do to get the results, especially when the client is not a technical person. Thus, it is important to output results that are readable and easy-to-understand, as we did in which we eliminated unnecessary columns and only showed the optimal notification sending time for each user on each day of the week.

Limitations and Challenge:

As for this iteration of the project, when picking the optimal notification sending time for each user on each day of the week, we only provided one choice based on which hour time has the largest impression/frequency. This approach may delete some valuable information when the largest and second to the largest impression/ frequency of the hour time are quite close to each other.

For instance, for user 21, on Monday, they opened the notification at 2 pm 10 times, while at 6 pm for 9 times. The approach in this iteration only keeps 2 pm as the optimal notification sending time on Monday, while does not keep the 6 pm. However, 6 pm may also be valuable for the client to try out. Thus, we tried different ways to improve this issue in the next iteration.

Iteration 3:

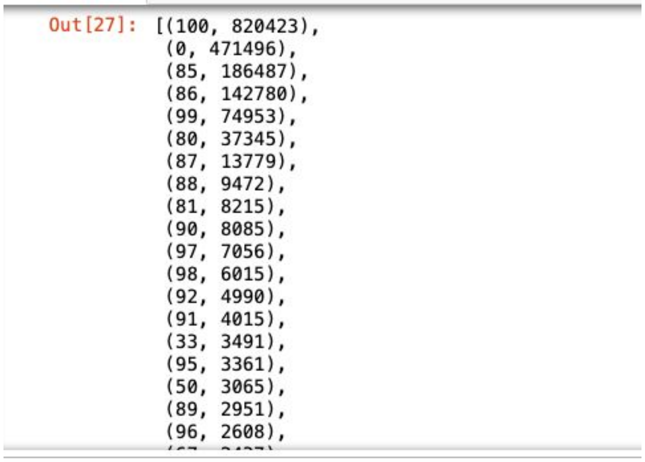
Aim:

According to the client's request to add the customer impression into the final result, the algorithm of optimal notification sending time was changed again, and the format required by the client was changed to get a better result performance.

Strategy, Result and Visualizations:

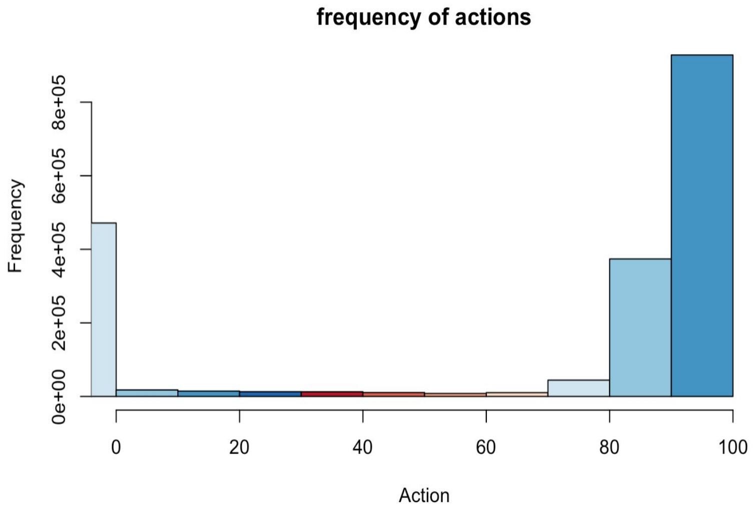
After we met with our project manager and client, our client gave us some feedback and a new request for this project. Instead of finding the optimal notification sending time for all application users, the client wanted our team to only focus on the users whose notification completion rate is between 85 and 100 and added a new request of considering the customer impression in our final result. In this case, our team needed to find different optimal times under different completion rates and then compare the completion rate to find the optimal time for each day. In addition, we needed to consider the different content types and list the contents that users read at that optimal notification sending time.

1. Initial stage:
    a. Our team added the action distribution to see the distribution for different completion rate and users range. Then, we divided the users into three groups by the completion rate: 85-100, 0-85, and 0.

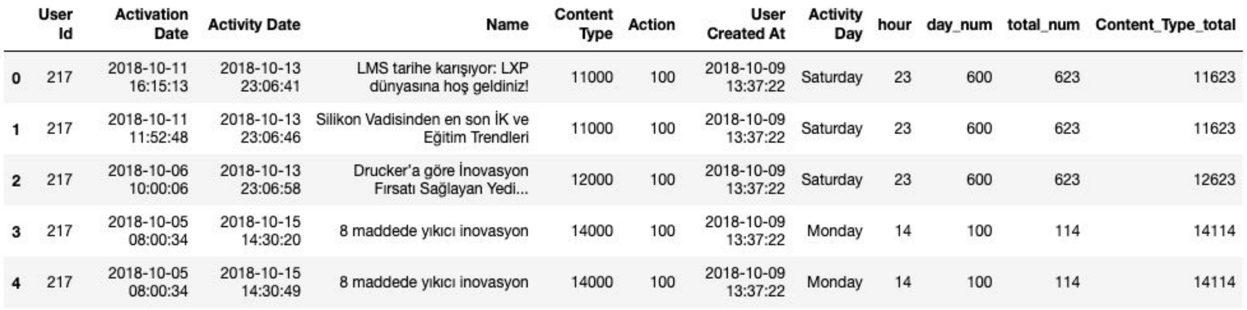


```
Out[27]: [(100, 820423),
          (0, 471496),
          (85, 186487),
          (86, 142780),
          (99, 74953),
          (80, 37345),
          (87, 13779),
          (88, 9472),
          (81, 8215),
          (90, 8085),
          (97, 7056),
          (98, 6015),
          (92, 4990),
          (91, 4015),
          (33, 3491),
          (95, 3361),
          (50, 3065),
          (89, 2951),
          (96, 2608),
```

In order to facilitate data analysis and processing, we first classified different action rates. From the screenshot below, we can see that most of the user action rates are above 80%. The ratio of users whose action rate from 0 to 80% is very small. However, users whose action rate is directly equal to 0 are no longer a minority. Therefore, we directly consider users whose action rate is above 85% and find out their optimal notification sending time.

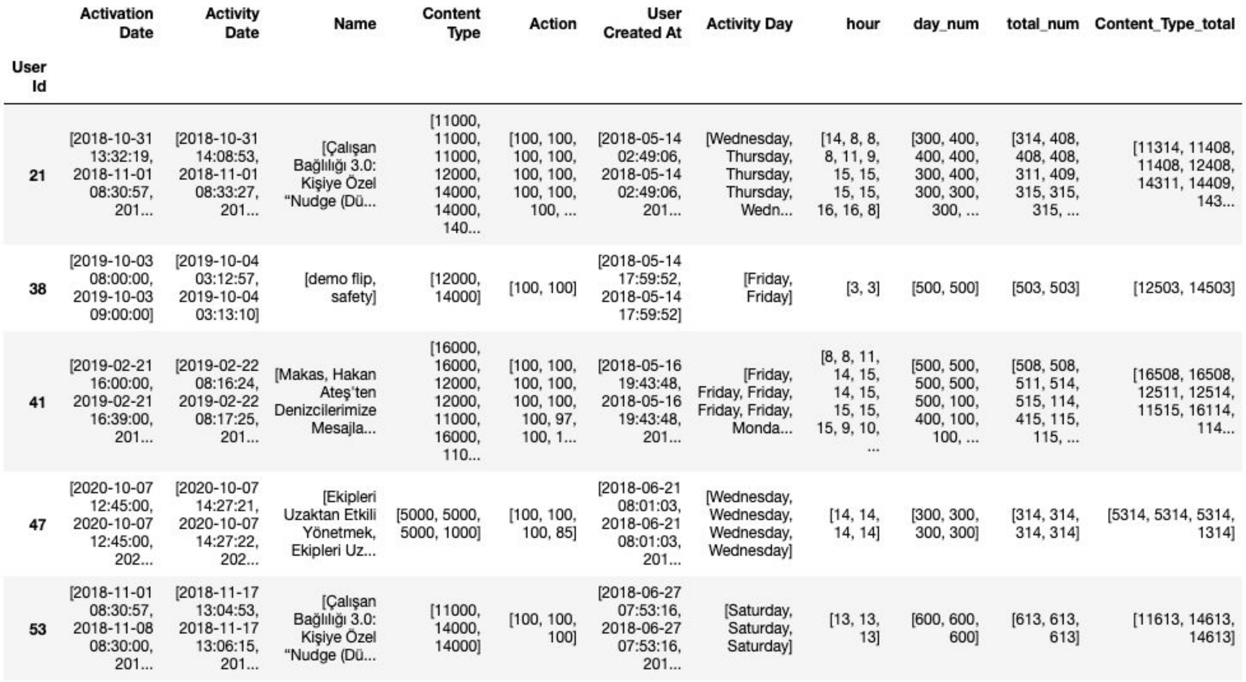


As you can see in the screenshots above, our team first found out the competition rate distribution. It clearly shows that the users who have the completion rate =100 are in the majority. The following screenshot is the dataframe of all users whose completion rate between 85 and 100.



	User Id	Activation Date	Activity Date	Name	Content Type	Action	User Created At	Activity Day	hour	day_num	total_num	Content_Type_total
0	217	2018-10-11 16:15:13	2018-10-13 23:06:41	LMS tarihe karışıyor: LXP dünyasına hoş geldiniz!	11000	100	2018-10-09 13:37:22	Saturday	23	600	623	11623
1	217	2018-10-11 11:52:48	2018-10-13 23:06:46	Silikon Vadisinden en son İK ve Eğitim Trendleri	11000	100	2018-10-09 13:37:22	Saturday	23	600	623	11623
2	217	2018-10-06 10:00:06	2018-10-13 23:06:58	Drucker'a göre İnovasyon Fırsatı Sağlayan Yedi...	12000	100	2018-10-09 13:37:22	Saturday	23	600	623	12623
3	217	2018-10-05 08:00:34	2018-10-15 14:30:20	8 maddede yıkıcı inovasyon	14000	100	2018-10-09 13:37:22	Monday	14	100	114	14114
4	217	2018-10-05 08:00:34	2018-10-15 14:30:49	8 maddede yıkıcı inovasyon	14000	100	2018-10-09 13:37:22	Monday	14	100	114	14114

b. We created five new dataframe columns for this dataframe and they are “Activity Day”, “hour”, “day_num”, “total_num”, and “Content_Type_total”. “Activity Day” is the date of each “Activity Date”. “Hour” is the hour time of each “Activity Date”. “Day_num” is the transformation of “Activity Day” in order to facilitate future statistics and calculations. “Total_num” is the result of adding “hour” and “day_num” in order to put the activity time of the same user in a space in the dataframe. “Content_Type_total” is the result of adding “total_num” and “Content Type” together so that we can easily report all the content that users read at each optimal notification sending time.

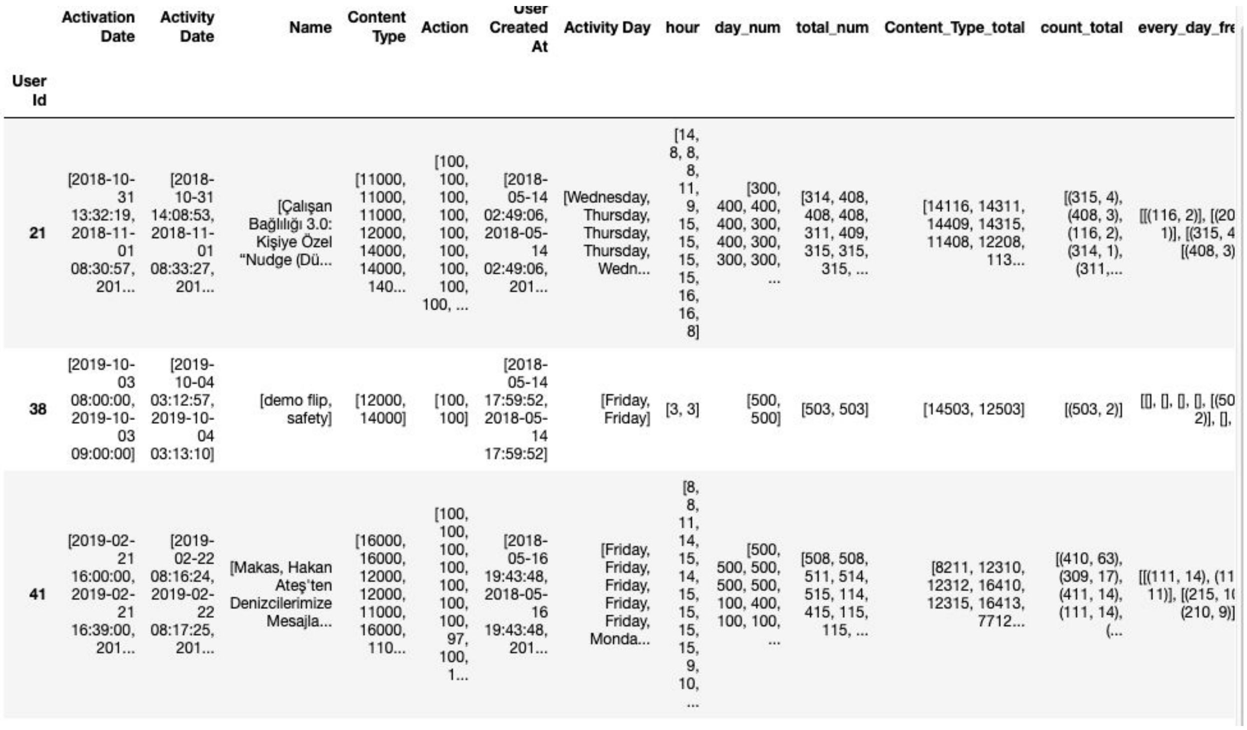


User Id	Activation Date	Activity Date	Name	Content Type	Action	User Created At	Activity Day	hour	day_num	total_num	Content_Type_total
21	[2018-10-31 13:32:19, 2018-11-01 08:30:57, 201...	[2018-10-31 14:08:53, 2018-11-01 08:33:27, 201...	[Çalışan Bağlılığı 3.0: Kişiye Özel "Nudge (Dü...	[11000, 11000, 11000, 12000, 14000, 14000, 140...	[100, 100, 100, 100, 100, 100, 100, 100, 100, ...	[2018-05-14 02:49:06, 2018-05-14 02:49:06, 201...	[Wednesday, Thursday, Thursday, Thursday, Wedn...	[14, 8, 8, 8, 11, 9, 15, 15, 15, 15, 16, 16, 8]	[300, 400, 400, 400, 300, 400, 300, 300, 300, ...	[314, 408, 408, 408, 311, 409, 315, 315, 315, ...	[11314, 11408, 11408, 12408, 14311, 14409, 143...
38	[2019-10-03 08:00:00, 2019-10-03 09:00:00]	[2019-10-04 03:12:57, 2019-10-04 03:13:10]	[demo flip, safety]	[12000, 14000]	[100, 100]	[2018-05-14 17:59:52, 2018-05-14 17:59:52]	[Friday, Friday]	[3, 3]	[500, 500]	[503, 503]	[12503, 14503]
41	[2019-02-21 16:00:00, 2019-02-21 16:39:00, 201...	[2019-02-22 08:16:24, 2019-02-22 08:17:25, 201...	[Makas, Hakan Ateş'ten Denizcilerimize Mesajla...	[16000, 16000, 12000, 12000, 11000, 16000, 110...	[100, 100, 100, 100, 100, 100, 100, 97, 100, 1...	[2018-05-16 19:43:48, 2018-05-16 19:43:48, 201...	[Friday, Friday, Friday, Friday, Friday, Monda...	[8, 8, 11, 14, 15, 14, 15, 15, 15, 15, 9, 10, ...	[500, 500, 500, 500, 500, 100, 400, 100, 100, ...	[508, 508, 511, 514, 515, 114, 415, 115, 115, ...	[16508, 16508, 12511, 12514, 11515, 16114, 114...
47	[2020-10-07 12:45:00, 2020-10-07 12:45:00, 202...	[2020-10-07 14:27:21, 2020-10-07 14:27:22, 202...	[Ekipleri Uzaktan Etkili Yönetmek, Ekipleri Uz...	[5000, 5000, 5000, 1000]	[100, 100, 100, 85]	[2018-06-21 08:01:03, 2018-06-21 08:01:03, 201...	[Wednesday, Wednesday, Wednesday, Wednesday]	[14, 14, 14, 14]	[300, 300, 300, 300]	[314, 314, 314, 314]	[5314, 5314, 5314, 1314]
53	[2018-11-01 08:30:57, 2018-11-08 08:30:00, 201...	[2018-11-17 13:04:53, 2018-11-17 13:06:15, 201...	[Çalışan Bağlılığı 3.0: Kişiye Özel "Nudge (Dü...	[11000, 14000, 14000]	[100, 100, 100]	[2018-06-27 07:53:16, 2018-06-27 07:53:16, 201...	[Saturday, Saturday, Saturday]	[13, 13, 13]	[600, 600, 600]	[613, 613, 613]	[11613, 14613, 14613]

c. Then, we added the corresponding content type of each hour time on each day of the week. We first made changes to the categorical variable. The numbers of content type are stored as integers from 1 to 17. To make it easy to parse the day, hour, and content type, we multiply each integer by 100. We added hr_num, day_num, content type together as integers in a new column content_type_total. For instance, 17513 typically means that there's a user who finishes reading a notification of type 17 at 1 PM on Friday. Same procedures were done as the previous one.

2. Developing stage 1
According to the client's request, we used the hour as the smallest unit and provided the customer service with a time block spanning one hour.
    a. We took the "content type" into our consideration and changed the amount of impressions shown in our result. If a user has similar frequencies in two different hours, we will report both.
    b. For instance, on Monday, a user reads at 2 PM for 14 impressions and 4 PM for 12 impressions. The number of impressions is close and both 2 PM and 4 PM will be reported with impressions shown in the result table. If a user reads at 2 PM for 14 impressions and 4 PM for 1 impression on Monday, the difference between the frequency is huge and we can ignore the 4 PM. Only 2 PM and the corresponding impressions will be shown in the result table. To test the closeness of frequency, we use average. For each day, we find the average frequency of each hour. If the

frequency is larger than the average, we will remain the hour and frequency; if not, we will ignore the hour time.

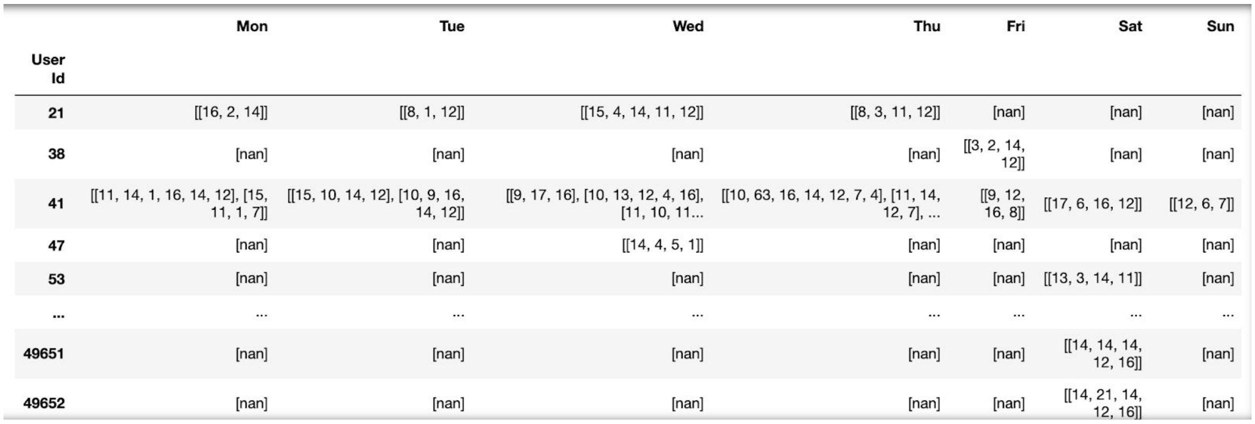


User Id	Activation Date	Activity Date	Name	Content Type	Action	User Created At	Activity Day	hour	day_num	total_num	Content_Type_total	count_total	every_day_fre
21	[2018-10-31 13:32:19, 2018-11-01 08:30:57, 201...	[2018-10-31 14:08:53, 2018-11-01 08:33:27, 201...	[Çalışan Bağlılığı 3.0: Kişiye Özel “Nudge (Dü...	[11000, 11000, 11000, 12000, 14000, 14000, 140...	[100, 100, 100, 100, 100, 100, 100, 100, 100, ...	[2018-05-14 02:49:06, 2018-05-14 02:49:06, 201...	[Wednesday, Thursday, Thursday, Thursday, Wedn...	[14, 8, 8, 8, 11, 9, 15, 15, 15, 15, 16, 16, 8]	[300, 400, 400, 400, 300, 400, 300, 300, 300, ...	[314, 408, 408, 408, 311, 409, 315, 315, 315, ...	[14116, 14311, 14409, 14315, 11408, 12208, 113...	[(315, 4), (408, 3), (116, 2), (314, 1), (311,...	[[(116, 2)], [(20 1)], [(315, 4 [(408, 3)
38	[2019-10-03 08:00:00, 2019-10-03 09:00:00]	[2019-10-04 03:12:57, 2019-10-04 03:13:10]	[demo flip, safety]	[12000, 14000]	[100, 100]	[2018-05-14 17:59:52, 2018-05-14 17:59:52]	[Friday, Friday]	[3, 3]	[500, 500]	[503, 503]	[14503, 12503]	[(503, 2)]	[[], [], [], [], [(50 2)], [],
41	[2019-02-21 16:00:00, 2019-02-21 16:39:00, 201...	[2019-02-22 08:16:24, 2019-02-22 08:17:25, 201...	[Makas, Hakan Ateş'ten Denizcilerimize Mesajla...	[16000, 16000, 12000, 12000, 11000, 16000, 110...	[100, 100, 100, 100, 100, 100, 100, 97, 100, 1...	[2018-05-16 19:43:48, 2018-05-16 19:43:48, 201...	[Friday, Friday, Friday, Friday, Friday, Monda...	[8, 8, 11, 14, 15, 14, 15, 15, 15, 15, 9, 10, ...	[500, 500, 500, 500, 500, 100, 400, 100, 100, ...	[508, 508, 511, 514, 515, 114, 415, 115, 115, ...	[8211, 12310, 12312, 16410, 12315, 16413, 7712...	[(410, 63), (309, 17), (411, 14), (111, 14), (...	[[(111, 14), (11 11)], [(215, 1((210, 9)]

In the above screenshot, in order to meet the demands of the client for the customer impression, we used frequency and average to calculate the completion rate at the same time. We first gathered all the frequencies for each user at their different optimal notification sending time. To test the closeness of frequency, we use the average function. For each day, we find the average frequency of each hour. If the frequency is larger than the average, we will remain the hour and frequency; if not, we will ignore the hour time. As you can see above, the column “every_day_freq” is the result that is over the average frequency. This helps us determine which content type is the most popular and acceptable by the users overall, which can be used to send notifications for our new users (the secondary goal of our project). At the same time, we can also check the frequency of each user and see which content type they like the best which can be used to increase the completion rate of each notification.

3. Developing stage 2
    a. Next, our team wanted to combine content type with day and hour as a list. We used list(set()) to get a list of content_type_total with unique elements for each user. We also defined a new function nth_digit to return the nth digit of an integer. By using this function, we could easily find out the day by pointing to the

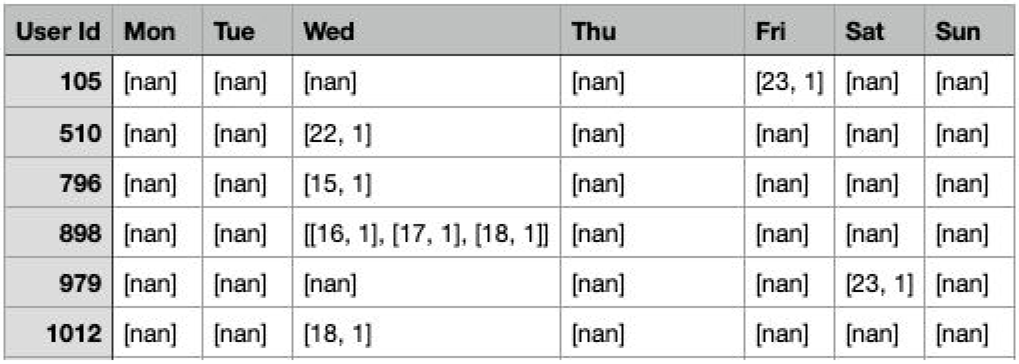
3rd digit. By for-loops and nth_digit, we could combine day and hour with the corresponding content type. The result is shown below:



	Mon	Tue	Wed	Thu	Fri	Sat	Sun
User Id							
21	[[16, 2, 14]]	[[8, 1, 12]]	[[15, 4, 14, 11, 12]]	[[8, 3, 11, 12]]	[nan]	[nan]	[nan]
38	[nan]	[nan]	[nan]	[nan]	[[3, 2, 14, 12]]	[nan]	[nan]
41	[[11, 14, 1, 16, 14, 12], [15, 11, 1, 7]]	[[15, 10, 14, 12], [10, 9, 16, 14, 12]]	[[9, 17, 16], [10, 13, 12, 4, 16], [11, 10, 11...	[[10, 63, 16, 14, 12, 7, 4], [11, 14, 12, 7], ...	[[9, 12, 16, 8]]	[[17, 6, 16, 12]]	[[12, 6, 7]]
47	[nan]	[nan]	[[14, 4, 5, 1]]	[nan]	[nan]	[nan]	[nan]
53	[nan]	[nan]	[nan]	[nan]	[nan]	[[13, 3, 14, 11]]	[nan]
...	...	...	...	...	...	...	...
49651	[nan]	[nan]	[nan]	[nan]	[nan]	[[14, 14, 14, 12, 16]]	[nan]
49652	[nan]	[nan]	[nan]	[nan]	[nan]	[[14, 21, 14, 12, 16]]	[nan]

b. As you can see above, in the column representing Monday, "Mon", the result follows the format as follows: user_id: [[time1, frequency1, corresponding content_type], [[time2, frequency2, corresponding content_type] ...]. Take user id 21 as an example. On Monday, that user finished 2 times at 16:00 (4 pm) with the content type of 14. On Wednesday, that user finished 4 times at 15 with content types of 14, 11, and 12. So that our client could easily recognize the data that we produced and find the information that they need. Then, we stored all the optimal notification sending time in the CSV documents. In the CSV file, the columns include user_id and each day of the week, so that the client can use the CSV file and can be imputed in SQL or other software and easily used.

4. Developing stage 3
   a. For the users whose completion rate is below 85, we also gathered a dataframe just in case our client may need this file for future improvement. So we completely separate the completion rate from 85 to 100 and users below 85 and list them in a new table.



User Id	Mon	Tue	Wed	Thu	Fri	Sat	Sun
105	[nan]	[nan]	[nan]	[nan]	[23, 1]	[nan]	[nan]
510	[nan]	[nan]	[22, 1]	[nan]	[nan]	[nan]	[nan]
796	[nan]	[nan]	[15, 1]	[nan]	[nan]	[nan]	[nan]
898	[nan]	[nan]	[[16, 1], [17, 1], [18, 1]]	[nan]	[nan]	[nan]	[nan]
979	[nan]	[nan]	[nan]	[nan]	[nan]	[23, 1]	[nan]
1012	[nan]	[nan]	[18, 1]	[nan]	[nan]	[nan]	[nan]

In the screenshot, the format is the same as before except the completion rate is below 85. We are still not so sure what format that the client may want. So we still need to change our code in the future to meet the client's demands.

Lessons Learned and Insights:

During this iteration, we have a few improvements. Followed by iteration 2, we projected the different content type to number 1 to 17, instead of the one-hot coder so that the client can easily use and understand. And we multiplied 1000 to "content type" column, 100 to "activity date" column and added with "hour" column together to save the algorithm time and easy to calculate. Obviously, compared with string, numbers can be easily calculated and imported into different software.

Secondly, we used an average frequency function to classify the customer impression. Instead of storing all the time that is suitable for the optimal notification sending time, we need to see the customer impression and then to decide. So we added the average frequency on the base of iteration 2. When the frequency is below the frequency average we will just ignore that time and focus on the time that has a better impression.

Limitations and Future steps:

Also, this iteration is not as smooth as we write in the project report. Our team also met the pitfalls and solved them as best as we can. When the client had the request of considering the customer impression and adding content type. We had several plans to do that and not sure what format the client really wanted since we also need to add content type in the final result. As we mentioned above, we use a transformation from string to number type to solve the content type and customer impression problem. As to the format of the final result. We communicated several times and we decided to use the format as user_id : [[Day1, time1, frequency1, corresponding content_type], [Day2, time2, frequency2, corresponding content_type] ...].

For the future step, we may want to use the data results we get so far to train the model of the algorithm. So that we can predict the optimal notification sending time for the new users. In addition, the client can use our data analysis and training model in the future, so that the client doesn't need to spend more money on the data analysis processing. For the different algorithms, we may try linear regression, logistic regression, and other algorithms and adjust hyper-parameters to find the best-performed algorithm.

5. Reference

Marketing. (2019, July 15). The Science Behind Send Time Optimization - The Robly Blog. Retrieved November 27, 2020, from https://blog.robly.com/2019/07/16/the-science-behind-send-time-optimization/

6. Appendix

7.1 Python

```
# Vibons Team1 - Journey app
# Team member: Lingyan Jiang, Ruoqi Shi, Jiaqi Zhao

import pandas as pd
import numpy as np

X = pd.read_csv('new_data.csv', parse_dates = True, low_memory = False)
#print(X.shape)
X = X.values

# delete duplicates in device and content_type and prepare for projecting string to number
res_list = X.T.tolist()
b = list(set(res_list[10]))

device = X[:,11]
for i in range(len(device)):
    if device[i] == 'Device':
        print(i)

content_type = X[:,6]
for i in range(len(content_type)):
    if content_type[i] == 'Content Type':
        print(i)
#print(X[:,6])

#Mapping device, channel and content_type to number
device = X[:,11]
#print(device)
channel = X[:,12]
#print(channel)
content_type = X[:,6]
#print(content_type)
for i in range(len(device)):
    if device[i] == 'Android':
        device[i] = 4
    elif device[i] == 'iOS':
        device[i] = 3
    elif device[i] == 'Web':
        device[i] = 2
    elif device[i] == 'Mobile Web (Tablet)':
        device[i] = 1
    elif device[i] == 'Mobile Web':
        device[i] = 5
    else:
        device[i] = 0
#print(device)

for i in range(len(channel)):
    if channel[i] == 'Mobile Notification':
        channel[i] = 2
    elif channel[i] == 'From Email':
        channel[i] = 1
    elif channel[i] == 'Direct Connection':
        channel[i] = 3
    else:
        channel[i] = 0
#print(channel)

content_type_list = ['KEY_CONTACT_NEWHIRE', 'EXTERNAL_ARTICLE', 'CHECKLIST',
'KNOWLEDGE_REQUIREMENT_NEWHIRE', 'BOOK_SUGGESTION', 'QUIZ',
'KEY_CONTACT_MANAGER', 'EXTERNAL_VIDEO', 'SURVEY', 'MEETING', 'LIVE_EVENT',
'INTERNAL_ARTICLE', 'FLIPBOOK', 'KNOWLEDGE_REQUIREMENT_MANAGER',
'INFOGRAPHIC', 'Content Type', 'INTERNAL_VIDEO', 'QUOTES']

for i in range(len(content_type)):
    if content_type[i] == content_type_list[0]:
        content_type[i] = 17000
    elif content_type[i] == content_type_list[1]:
        content_type[i] = 1000
    elif content_type[i] == content_type_list[2]:
        content_type[i] = 2000
    elif content_type[i] == content_type_list[3]:
        content_type[i] = 3000
    elif content_type[i] == content_type_list[4]:
        content_type[i] = 4000
    elif content_type[i] == content_type_list[5]:
        content_type[i] = 5000
    elif content_type[i] == content_type_list[6]:
        content_type[i] = 6000
    elif content_type[i] == content_type_list[7]:
        content_type[i] = 7000
    elif content_type[i] == content_type_list[8]:
        content_type[i] = 8000
    elif content_type[i] == content_type_list[9]:
        content_type[i] = 9000
    elif content_type[i] == content_type_list[10]:
        content_type[i] = 10000
    elif content_type[i] == content_type_list[11]:
        content_type[i] = 11000
    elif content_type[i] == content_type_list[12]:
        content_type[i] = 12000
    elif content_type[i] == content_type_list[13]:
        content_type[i] = 13000
    elif content_type[i] == content_type_list[14]:
        content_type[i] = 14000
    elif content_type[i] == content_type_list[15]:
        content_type[i] = 15000
    elif content_type[i] == content_type_list[16]:
        content_type[i] = 16000
    else:
        content_type[i] = 0
#print(content_type)

#select the data from cleaned data
clean_data = pd.DataFrame(X)
#print(clean_data.head())
c = clean_data.values
#print(c)

# use mode() function to replace the nan value
max_value = clean_data[11].mode()
clean_data[11] = clean_data[11].replace([0],max_value)
#print(clean_data[11].value_counts())

# use mode() function to replace the nan value
content_max = clean_data[6].mode()
#print(content_max)
#print(clean_data[6].value_counts())
clean_data[6] = clean_data[6].replace([0],content_max)
#print('after: ',clean_data[6].value_counts())

# use mode() function to replace the nan value
max_value = clean_data[12].mode()
clean_data[12] = clean_data[12].replace([0],max_value)
#print(clean_data[12].value_counts())

#drop Flipbook from the user ID
```

```
clean_data.drop(clean_data[clean_data[1] == 'FLIPBOOK'].index, inplace = True)

#drop nan value
clean_data.dropna(how = 'any', axis = 0, inplace = True)

#reset index of cleaned data
clean_data.reset_index(inplace=True, drop=True)

#transfer userID index to integer
clean_data[1] = clean_data[1].fillna(0.0).astype('int')

#Replace the Action nan value to 0 and transfer Action type to integer
clean_data[9] = clean_data[9].replace(['nan'],'0')
clean_data[9] = clean_data[9].fillna(0.0).astype('int')

#Replace the Content_Type nan value to [] and transfer Action Content_Type type to string
#clean_data.drop(index = 1789049, axis = 0)
clean_data[5] = clean_data[5].replace(['nan'],"")
clean_data[5] = clean_data[5].astype('str')

#Give the new cleaned data the column title
clean_data.columns = ['Customer Id','User Id', 'User Created At', 'Activation Date', 'Activity Date','Name',
'Content Type', 'Content ID','Journey Name', 'Action', 'Duration','Device', 'Channel', 'Session Id', 'Rating']

#Group by with userID and order by descending

#clean_data = clean_data[:300].groupby(['User Id'],axis=1)
#print(Grouped.size())

clean_data.groupby(['User Id'])

#print(clean_data.groupby(['User Id']).count())

group_label = clean_data.groupby(['User Id']).groups
#print(group_label)

#choose the columns from cleaned data that we need to process later
open_time = pd.DataFrame()
open_time = clean_data.loc[:, ('User Id','Activation Date','Activity Date','Name','Content
Type','Action','User Created At')]
#diff = pd.Timestamp(open_time['Activity Date'])- pd.Timestamp(open_time['Activation Date'])
#diff = pd.to_datetime(open_time['Activity Date'])- pd.to_datetime(open_time['Activation Date'])
#print(diff)
#open_time.insert(5, "Time lag", diff)
```

```
#print(open_time)
#open_time.dropna(how = "any")

#transfer the Activity Date to date and hour seperately
date = open_time['Activity Date'].map(lambda x: str(x)[0:10])
date = pd.to_datetime(date)
open_time['Activity Day'] = date.dt.day_name()
hr = open_time['Activity Date'].map(lambda x: str(x)[11:13])
open_time['hour'] = hr

# # Action Distribution

#find the action distribution
import collections
actiondis = open_time["Action"].values.tolist()
collections.Counter(actiondis).most_common()


# # Action >= 85

#Find the data when action rate is >= 85
Action85 = open_time[open_time.loc[:,("Action")] >= 85]
Action = clean_data[["Content Type","Device","Channel","Action"]]

#Mapping the data to number for convenience
def tonum(x):
    """
    take x, activity day as input
    return int Mon -> 100, Tue -> , etc
    """
    if x == "Monday":
        #print("1")
        return 100
    elif x == "Tuesday":
        return 200
    elif x == "Wednesday":
        return 300
    elif x == "Thursday":
        return 400
    elif x == "Friday":
        return 500
    elif x == "Saturday":
        return 600
    else:
```

```
    return 700

#transfer the data to number using the function above
day_num = open_time.loc[:,('Activity Day')].map(lambda x: tonum(x))

#store the date into day_num column
Action85["day_num"] = day_num

#transfer the hour to number
hr_num = Action85['hour'].map(lambda x: int(x))

#store the hour into hr_num column
Action85["hour"] = hr_num

#add the hour and day_num together in order for calculating convenience
total_num = Action85['hour'].add(Action85['day_num'])

#store the total_num into total_num column
Action85['total_num'] = total_num

#add the content_type and total_num together in order for calculating convenience
Action85['Content_Type_total'] = ''
content_type_total = Action85['Content Type'].add(Action85['total_num'])
Action85['Content_Type_total'] = content_type_total

#choose the unique UserID
df85 = pd.DataFrame({'User Id':Action85["User Id"].unique()})

#store all total_num of one user into one dataframe space
df85['total_num'] = [list(set(Action85['total_num'].loc[Action85['User Id'] == x['User Id']])) for _, x in
df85.iterrows()]

#delete the duplicates UserID
df = Action85.groupby('User Id').agg(lambda x: x.tolist())

#store the index which is thw userID into index_df
index_df = df.index.values

#add one column "count_total" in dataframe
df["count_total"] = ''

#create a function for counting same total_num
from collections import Counter
def count(x):
    Counter(df["total_num"][21]).most_common()

#count same total_num of one user and order from largest to smallest
for i in range(len(index_df)):
    df['count_total'][index_df[i]] = Counter(df['total_num'][index_df[i]]).most_common()

#create a function to transfer the count_total number of each user into data and store in the different
columns
def sort_arr(arr):
    res = [[],[],[],[],[],[],[]]
    for i in range(len(arr)):
        item = str(arr[i][0])
        if item[0] == "1":
            res[0].append(arr[i])
            #return res
        elif item[0] == "2":
            res[1].append(arr[i])
            #return res
        elif item[0] == "3":
            res[2].append(arr[i])
            #return res
        elif item[0] == "4":
            res[3].append(arr[i])
            #return res
        elif item[0] == "5":
            res[4].append(arr[i])
            #return res
        elif item[0] == "6":
            res[5].append(arr[i])
            #return res
        else:
            res[6].append(arr[i])
            #return res
    return res
import statistics

#in order to find the better user impression, create a function to choose the time which above the average
frequency
def get_above(arr):
    freq = []
    for i in range(len(arr)):
        ele = arr[i][-1]
        freq.append(ele)
    mean = statistics.mean(freq)
    l = []
    for i in range(len(arr)):
        if arr[i][-1] >= mean:
            l.append(arr[i])
    return l

# create a function to find the frequency of a selected user
def find_freq(arr):
    res = []
    sorted_arr = sort_arr(arr)
    for i in range(len(sorted_arr)):
        if len(sorted_arr[i]) == 0:
            res.append([])
        else:
            if len(sorted_arr[i]) == 1:
                res.append(sorted_arr[i])
            else:
                subarr = sorted_arr[i]
                freq = []
                for i in range(len(arr)):
                    ele = arr[i][-1]
                    freq.append(ele)
                l = []
                mean = statistics.mean(freq)
                for j in range(len(subarr)):
                    if subarr[j][-1] >= mean:
                        l.append(subarr[j])
                res.append(l)
    return res

#create a new column in dataframe
df["every_day_freq"] = "

#find a frequency for each user
for i in range(len(index_df)):
    arr = df["count_total"][index_df[i]]
    df["every_day_freq"][index_df[i]] = find_freq(arr)

#create new columns to dataframe
df['Mon'] = "
df['Tue'] = "
df['Wed'] = "
df['Thu'] = "
```

```
df['Fri'] = "
df['Sat'] = "
df['Sun'] = "

#generate hour with corresponding frequency on each day of one week
for i in range(len(index_df)):
    arr = df["every_day_freq"][index_df[i]]
    temp = []
    ### Monday
    if len(arr[0]) == 0:
        df['Mon'][index_df[i]] = [np.nan]
    else:
        if len(arr[0]) == 1:
            df['Mon'][index_df[i]] = [[arr[0][0][0]%100, arr[0][0][1]]]
        else:
            temp = []
            for j in range(len(arr[0])):
                ele = [arr[0][j][0]%100, arr[0][j][1]]
                temp.append(ele)

            df['Mon'][index_df[i]] = temp
    ### Tuesday
    if len(arr[1]) == 0:
        df['Tue'][index_df[i]] = [np.nan]
    else:
        if len(arr[1]) == 1:
            df['Tue'][index_df[i]] = [[arr[1][0][0]%100, arr[1][0][1]]]
        else:
            temp = []
            for j in range(len(arr[1])):
                ele = [arr[1][j][0]%100, arr[1][j][1]]
                temp.append(ele)
            df['Tue'][index_df[i]] = temp
    ### Wednesday
    if len(arr[2]) == 0:
        df['Wed'][index_df[i]] = [np.nan]
    else:
        if len(arr[2]) == 1:
            df['Wed'][index_df[i]] = [[arr[2][0][0]%100, arr[2][0][1]]]
        else:
            temp = []
            for j in range(len(arr[2])):
                ele = [arr[2][j][0]%100, arr[2][j][1]]
                temp.append(ele)
            df['Wed'][index_df[i]] = temp
    ### Thursday
    if len(arr[3]) == 0:
        df['Thu'][index_df[i]] = [np.nan]
    else:
        if len(arr[3]) == 1:
            df['Thu'][index_df[i]] = [[arr[3][0][0]%100, arr[3][0][1]]]
        else:
            temp = []
            for j in range(len(arr[3])):
                ele = [arr[3][j][0]%100, arr[3][j][1]]
                temp.append(ele)
            df['Thu'][index_df[i]] = temp
    ### Friday
    if len(arr[4]) == 0:
        df['Fri'][index_df[i]] = [np.nan]
    else:
        if len(arr[4]) == 1:
            df['Fri'][index_df[i]] = [[arr[4][0][0]%100, arr[4][0][1]]]
        else:
            temp = []
            for j in range(len(arr[4])):
                ele = [arr[4][j][0]%100, arr[4][j][1]]
                temp.append(ele)
            df['Fri'][index_df[i]] = temp
    ### Saturday
    if len(arr[5]) == 0:
        df['Sat'][index_df[i]] = [np.nan]
    else:
        if len(arr[5]) == 1:
            df['Sat'][index_df[i]] = [[arr[5][0][0]%100, arr[5][0][1]]]
        else:
            temp = []
            for j in range(len(arr[5])):
                ele = [arr[5][j][0]%100, arr[5][j][1]]
                temp.append(ele)
            df['Sat'][index_df[i]] = temp
    ### Sunday
    if len(arr[6]) == 0:
        df['Sun'][index_df[i]] = [np.nan]
    else:
        if len(arr[6]) == 1:
            df['Sun'][index_df[i]] = [[arr[6][0][0]%100, arr[6][0][1]]]
        else:
            temp = []
            for j in range(len(arr[6])):
                ele = [arr[6][j][0]%100, arr[6][j][1]]
                temp.append(ele)
            df['Sun'][index_df[i]] = temp

#convert userID into index
for i in range(len(index_df)):
    df['Content_Type_total'][index_df[i]] = list(set(df['Content_Type_total'][index_df[i]]))

#create a function to find the nth_digit of one number
def nth_digit(number, digit):
    return abs(number) // (10**(digit-1)) % 10

#add Content_Type into each space in the dataframe after hour and frequency
#Monday
for i in range(len(index_df)):
    if df['Mon'][index_df[i]] == [np.nan]:
        pass
    else:
        for j in range(len(df['Mon'][index_df[i]])):
            for n in range(len(df['Content_Type_total'][index_df[i]])):
                if nth_digit(df['Content_Type_total'][index_df[i]][n],3) == 1 and df['Mon'][index_df[i]][j][0] ==
df['Content_Type_total'][index_df[i]][n] % 100:
                    df['Mon'][index_df[i]][j].append(df['Content_Type_total'][index_df[i]][n] // 1000)

#Tuesday
for i in range(len(index_df)):
    if df['Tue'][index_df[i]] == [np.nan]:
        pass
    else:
        for j in range(len(df['Tue'][index_df[i]])):
            for n in range(len(df['Content_Type_total'][index_df[i]])):
                if nth_digit(df['Content_Type_total'][index_df[i]][n],3) == 2 and df['Tue'][index_df[i]][j][0] ==
df['Content_Type_total'][index_df[i]][n] % 100:
                    df['Tue'][index_df[i]][j].append(df['Content_Type_total'][index_df[i]][n] // 1000)

#Wednesday
for i in range(len(index_df)):
    if df['Wed'][index_df[i]] == [np.nan]:
        pass
    else:
        for j in range(len(df['Wed'][index_df[i]])):
            for n in range(len(df['Content_Type_total'][index_df[i]])):
                if nth_digit(df['Content_Type_total'][index_df[i]][n],3) == 3 and df['Wed'][index_df[i]][j][0] == df['Content_Type_total'][index_df[i]][n] % 100:
                    df['Wed'][index_df[i]][j].append(df['Content_Type_total'][index_df[i]][n] // 1000)

#Thursday
for i in range(len(index_df)):
    if df['Thu'][index_df[i]] == [np.nan]:
        pass
    else:
        for j in range(len(df['Thu'][index_df[i]])):
            for n in range(len(df['Content_Type_total'][index_df[i]])):
                if nth_digit(df['Content_Type_total'][index_df[i]][n],3) == 4 and df['Thu'][index_df[i]][j][0] == df['Content_Type_total'][index_df[i]][n] % 100:
                    df['Thu'][index_df[i]][j].append(df['Content_Type_total'][index_df[i]][n] // 1000)

#Friday
for i in range(len(index_df)):
    if df['Fri'][index_df[i]] == [np.nan]:
        pass
    else:
        for j in range(len(df['Fri'][index_df[i]])):
            for n in range(len(df['Content_Type_total'][index_df[i]])):
                if nth_digit(df['Content_Type_total'][index_df[i]][n],3) == 5 and df['Fri'][index_df[i]][j][0] == df['Content_Type_total'][index_df[i]][n] % 100:
                    df['Fri'][index_df[i]][j].append(df['Content_Type_total'][index_df[i]][n] // 1000)

#Saturday
for i in range(len(index_df)):
    if df['Sat'][index_df[i]] == [np.nan]:
        pass
    else:
        for j in range(len(df['Sat'][index_df[i]])):
            for n in range(len(df['Content_Type_total'][index_df[i]])):
                if nth_digit(df['Content_Type_total'][index_df[i]][n],3) == 6 and df['Sat'][index_df[i]][j][0] == df['Content_Type_total'][index_df[i]][n] % 100:
                    df['Sat'][index_df[i]][j].append(df['Content_Type_total'][index_df[i]][n] // 1000)

#Sunday
for i in range(len(index_df)):
    if df['Sun'][index_df[i]] == [np.nan]:
        pass
    else:
        for j in range(len(df['Sun'][index_df[i]])):
            for n in range(len(df['Content_Type_total'][index_df[i]])):
                if nth_digit(df['Content_Type_total'][index_df[i]][n],3) == 7 and df['Sun'][index_df[i]][j][0] ==
df['Content_Type_total'][index_df[i]][n] % 100:
                    df['Sun'][index_df[i]][j].append(df['Content_Type_total'][index_df[i]][n] // 1000)

#Final result
customer85_100 = pd.DataFrame()
customer85_100 = df.loc[:,('Mon','Tue','Wed','Thu','Fri','Sat','Sun')]

#Final result store in csv file
customer85_100.to_csv('customer85_100_1206.csv')

# # Action > 0 and Action < 85

# **Same steps and functions as action >= 85**

Action0_85 = open_time[(open_time.loc[:,("Action")] < 85) & (open_time.loc[:,("Action")] > 0)]
Action = clean_data[["Content Type","Content ID","Device","Channel","Action"]]
Action.isna().sum()
Action["Content ID"] = Action["Content ID"].astype("int")
day_num = open_time.loc[:,('Activity Day')].map(lambda x: tonum(x))
Action0_85["day_num"] = day_num
hr_num = Action0_85['hour'].map(lambda x: int(x))
Action0_85["hour"] = hr_num
total_num = Action0_85['hour'].add(Action0_85['day_num'])
Action0_85['total_num'] = total_num
df0_85 = pd.DataFrame({'User Id':Action0_85["User Id"].unique()})
df0_85['total_num'] = [list(set(Action0_85['total_num'].loc[Action0_85['User Id'] == x['User Id']])) for _,
x in df0_85.iterrows()]
df0_85_res = Action0_85.groupby('User Id').agg(lambda x: x.tolist())
index_df0_85_res = df0_85_res.index.values
df0_85_res["count_total"] = ''
for i in range(len(index_df0_85_res)):
    df0_85_res['count_total'][index_df0_85_res[i]] =
Counter(df0_85_res['total_num'][index_df0_85_res[i]]).most_common()
df0_85_res["every_day_freq"] = ''
for i in range(len(index_df0_85_res)):
    arr = df0_85_res["count_total"][index_df0_85_res[i]]
    df0_85_res["every_day_freq"][index_df0_85_res[i]] = find_freq(arr)
df0_85_res.head()
df0_85_res['Mon'] = ''
df0_85_res['Tue'] = ''
df0_85_res['Wed'] = ''
df0_85_res['Thu'] = ''
df0_85_res['Fri'] = ''
```

```
df0_85_res['Sat'] = ''
df0_85_res['Sun'] = ''

for i in range(len(index_df0_85_res)):
    arr = df0_85_res["every_day_freq"][index_df0_85_res[i]]
    temp = []
    ### Monday
    if len(arr[0]) == 0:
        df0_85_res['Mon'][index_df0_85_res[i]] = [np.nan]
    else:
        if len(arr[0]) == 1:
            df0_85_res['Mon'][index_df0_85_res[i]] = [arr[0][0][0]%100, arr[0][0][1]]
        else:
            temp = []
            for j in range(len(arr[0])):
                ele = [arr[0][j][0]%100, arr[0][j][1]]
                temp.append(ele)

            df0_85_res['Mon'][index_df0_85_res[i]] = temp
    ### Tuesday
    if len(arr[1]) == 0:
        df0_85_res['Tue'][index_df0_85_res[i]] = [np.nan]
    else:
        if len(arr[1]) == 1:
            df0_85_res['Tue'][index_df0_85_res[i]] = [arr[1][0][0]%100, arr[1][0][1]]
        else:
            temp = []
            for j in range(len(arr[1])):
                ele = [arr[1][j][0]%100, arr[1][j][1]]
                temp.append(ele)
            df0_85_res['Tue'][index_df0_85_res[i]] = temp
    ### Wednesday
    if len(arr[2]) == 0:
        df0_85_res['Wed'][index_df0_85_res[i]] = [np.nan]
    else:
        if len(arr[2]) == 1:
            df0_85_res['Wed'][index_df0_85_res[i]] = [arr[2][0][0]%100, arr[2][0][1]]
        else:
            temp = []
            for j in range(len(arr[2])):
                ele = [arr[2][j][0]%100, arr[2][j][1]]
                temp.append(ele)
            df0_85_res['Wed'][index_df0_85_res[i]] = temp
    ### Thursday
```

```
if len(arr[3]) == 0:
    df0_85_res['Thu'][index_df0_85_res[i]] = [np.nan]
else:
    if len(arr[3]) == 1:
        df0_85_res['Thu'][index_df0_85_res[i]] = [arr[3][0][0]%100, arr[3][0][1]]
    else:
        temp = []
        for j in range(len(arr[3])):
            ele = [arr[3][j][0]%100, arr[3][j][1]]
            temp.append(ele)
        df0_85_res['Thu'][index_df0_85_res[i]] = temp
### Friday
if len(arr[4]) == 0:
    df0_85_res['Fri'][index_df0_85_res[i]] = [np.nan]
else:
    if len(arr[4]) == 1:
        df0_85_res['Fri'][index_df0_85_res[i]] = [arr[4][0][0]%100, arr[4][0][1]]
    else:
        temp = []
        for j in range(len(arr[4])):
            ele = [arr[4][j][0]%100, arr[4][j][1]]
            temp.append(ele)
        df0_85_res['Fri'][index_df0_85_res[i]] = temp
### Saturday
if len(arr[5]) == 0:
    df0_85_res['Sat'][index_df0_85_res[i]] = [np.nan]
else:
    if len(arr[5]) == 1:
        df0_85_res['Sat'][index_df0_85_res[i]] = [arr[5][0][0]%100, arr[5][0][1]]
    else:
        temp = []
        for j in range(len(arr[5])):
            ele = [arr[5][j][0]%100, arr[5][j][1]]
            temp.append(ele)
        df0_85_res['Sat'][index_df0_85_res[i]] = temp
### Sunday
if len(arr[6]) == 0:
    df0_85_res['Sun'][index_df0_85_res[i]] = [np.nan]
else:
    if len(arr[6]) == 1:
        df0_85_res['Sun'][index_df0_85_res[i]] = [arr[6][0][0]%100, arr[6][0][1]]
    else:
        temp = []
        for j in range(len(arr[6])):
                ele = [arr[6][j][0]%100, arr[6][j][1]]
                temp.append(ele)
            df0_85_res['Sun'][index_df0_85_res[i]] = temp

customer0_85 = df0_85_res.loc[:,('Mon','Tue','Wed','Thu','Fri','Sat','Sun')]
#customer0_85.head()

# # Delete duplicated user id from Action 85-100

customer0_85_deleted = customer0_85.loc[customer0_85.index.difference(customer85_100.index), ]

customer0_85_deleted.to_csv('customer0_85_deleted.csv')

# # Action == 0

# **Same steps and functions as Action >= 85**

Action0 = open_time[(open_time.loc[:,("Action")] == 0)]
Action = clean_data[["Content Type","Content ID","Device","Channel","Action"]]
Action.isna().sum()
Action["Content ID"] = Action["Content ID"].astype("int")
day_num = open_time.loc[:,('Activity Day')].map(lambda x: tonum(x))
Action0["day_num"] = day_num
hr_num = Action0['hour'].map(lambda x: int(x))
Action0["hour"] = hr_num
total_num = Action0['hour'].add(Action0['day_num'])
Action0['total_num'] = total_num
df0 = pd.DataFrame({'User Id':Action0["User Id"].unique()})
df0['total_num'] = [list(set(Action0['total_num'].loc[Action0['User Id'] == x['User Id']])) for _, x in
df0.iterrows()]
df0_res = Action0.groupby('User Id').agg(lambda x: x.tolist())
index_df0_res = df0_res.index.values
df0_res["count_total"] = ''
for i in range(len(index_df0_res)):
    df0_res['count_total'][index_df0_res[i]] =
Counter(df0_res['total_num'][index_df0_res[i]]).most_common()
df0_res["every_day_freq"] = ''
for i in range(len(index_df0_res)):
    arr = df0_res["count_total"][index_df0_res[i]]
    df0_res["every_day_freq"][index_df0_res[i]] = find_freq(arr)
df0_res.head()
df0_res['Mon'] = ''
df0_res['Tue'] = ''
df0_res['Wed'] = ''
```

```
df0_res['Thu'] = ''
df0_res['Fri'] = ''
df0_res['Sat'] = ''
df0_res['Sun'] = ''

for i in range(len(index_df0_res)):
    arr = df0_res["every_day_freq"][index_df0_res[i]]
    temp = []
    ### Monday
    if len(arr[0]) == 0:
        df0_res['Mon'][index_df0_res[i]] = [np.nan]
    else:
        if len(arr[0]) == 1:
            df0_res['Mon'][index_df0_res[i]] = [arr[0][0][0]%100, arr[0][0][1]]
        else:
            temp = []
            for j in range(len(arr[0])):
                ele = [arr[0][j][0]%100, arr[0][j][1]]
                temp.append(ele)

            df0_res['Mon'][index_df0_res[i]] = temp
    ### Tuesday
    if len(arr[1]) == 0:
        df0_res['Tue'][index_df0_res[i]] = [np.nan]
    else:
        if len(arr[1]) == 1:
            df0_res['Tue'][index_df0_res[i]] = [arr[1][0][0]%100, arr[1][0][1]]
        else:
            temp = []
            for j in range(len(arr[1])):
                ele = [arr[1][j][0]%100, arr[1][j][1]]
                temp.append(ele)
            df0_res['Tue'][index_df0_res[i]] = temp
    ### Wednesday
    if len(arr[2]) == 0:
        df0_res['Wed'][index_df0_res[i]] = [np.nan]
    else:
        if len(arr[2]) == 1:
            df0_res['Wed'][index_df0_res[i]] = [arr[2][0][0]%100, arr[2][0][1]]
        else:
            temp = []
            for j in range(len(arr[2])):
                ele = [arr[2][j][0]%100, arr[2][j][1]]
                temp.append(ele)
            df0_res['Wed'][index_df0_res[i]] = temp
    ### Thursday
    if len(arr[3]) == 0:
        df0_res['Thu'][index_df0_res[i]] = [np.nan]
    else:
        if len(arr[3]) == 1:
            df0_res['Thu'][index_df0_res[i]] = [arr[3][0][0]%100, arr[3][0][1]]
        else:
            temp = []
            for j in range(len(arr[3])):
                ele = [arr[3][j][0]%100, arr[3][j][1]]
                temp.append(ele)
            df0_res['Thu'][index_df0_res[i]] = temp
    ### Friday
    if len(arr[4]) == 0:
        df0_res['Fri'][index_df0_res[i]] = [np.nan]
    else:
        if len(arr[4]) == 1:
            df0_res['Fri'][index_df0_res[i]] = [arr[4][0][0]%100, arr[4][0][1]]
        else:
            temp = []
            for j in range(len(arr[4])):
                ele = [arr[4][j][0]%100, arr[4][j][1]]
                temp.append(ele)
            df0_res['Fri'][index_df0_res[i]] = temp
    ### Saturday
    if len(arr[5]) == 0:
        df0_res['Sat'][index_df0_res[i]] = [np.nan]
    else:
        if len(arr[5]) == 1:
            df0_res['Sat'][index_df0_res[i]] = [arr[5][0][0]%100, arr[5][0][1]]
        else:
            temp = []
            for j in range(len(arr[5])):
                ele = [arr[5][j][0]%100, arr[5][j][1]]
                temp.append(ele)
            df0_res['Sat'][index_df0_res[i]] = temp
    ### Sunday
    if len(arr[6]) == 0:
        df0_res['Sun'][index_df0_res[i]] = [np.nan]
    else:
        if len(arr[6]) == 1:
            df0_res['Sun'][index_df0_res[i]] = [arr[6][0][0]%100, arr[6][0][1]]
        else:
            temp = []
            for j in range(len(arr[6])):
                ele = [arr[6][j][0]%100, arr[6][j][1]]
                temp.append(ele)
            df0_res['Sun'][index_df0_res[i]] = temp

    customer0 = df0_res.loc[:,('Mon','Tue','Wed','Thu','Fri','Sat','Sun')]
    #customer0.head()

    customer0.to_csv('customer0.csv')
    customer0_85.to_csv('customer0_85.csv')
    customer85_100.to_csv('customer85_100.csv')
```

7.2 R

```
data <- read.csv("/Users/rqshi/Desktop/Project/out.csv")
hist(data$Action, main = "frequency of actions", xlab = "Action", xlim = c(0, 100), col = c("#B2182B",
"#D6604D", "#F4A582", "#FDDBC7", "#D1E5F0", "#92C5DE", "#4393C3", "#2166AC"))
```