Deliverable 2

Vibon - Journey App - Team 2

Group members: Jiaqi Zhao, Lingyan Jiang, Ruoqi Shi

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, our primary goal is to find the send time optimization for each user of the Journey App, based on 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. Our secondary goal is to build a model to predict optimal notification sending time for future users.

2. Dataset and Representation

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 ask 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. Questions have been answered

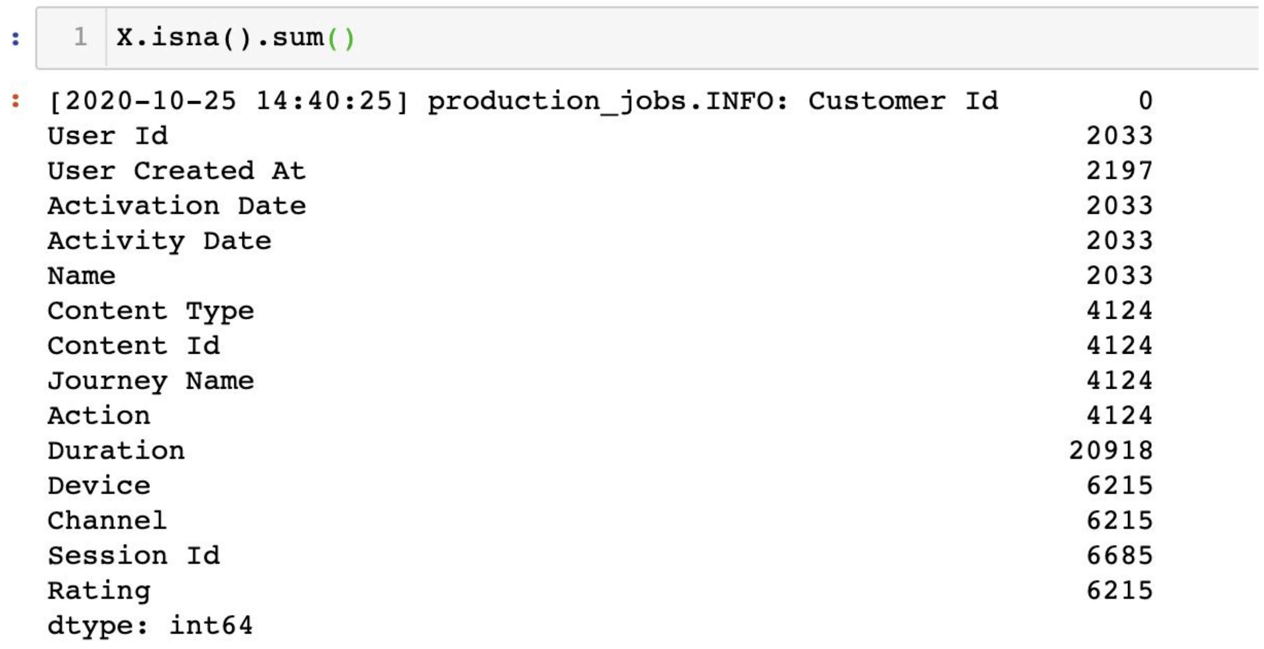
In this part, we will show how each question is answered through our project.

1) What research did the team do about optimizing the sending time of the notifications?

   From the project description that our client sent us, he linked a blog post inside it. In this blog post, the writer introduces the definition of sending time optimization and gives eight vendors with sending time optimization. This blog post gives us a brief introduction of what this project is doing and what we should achieve.

2) During the data cleaning step, how do we fill in the missing data, and how do we reduce the outliers?

   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 label it as zero. Then, we use 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 use dropna() method to drop the row if there is a NaN value for that user. At last, we delete “FLIPBOOK” entries inside the column “User Id”.

3) What data do we choose as the training data and what data do we abandon?

   We used all of the data given by our client to train our model. We filled the mode of abandoned rows that contain NaN values in all columns except for “Content Type”, “Device”, and “Channel”, and “FLIPBOOK” in the column “User Id”.

   According to our client, only actions above and including 85 are needed for training. We divided our dataset into 3 tables. The first one is all the entries that have an action above and including 85. The second one is all the entries that have an action above 0 and less than 85. The third one is all the entries that have an action of 0.

4) What algorithm do we choose to train the data, and what hyper-parameters and cross-validation do we choose?

   We created our own algorithm. We mainly focused on the first table mentioned above. To predict the optimal sending time, we analyzed the hour time of each activity day.

   First of all, we converted each date into the corresponding day of that week.

   Then, 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.

   Then, we added day_num and hour_num together as a new column called total_num. For instance, if a user opens a notification at 9 AM on Monday, the total_num is 109 for that user.

   Next, we aggregated each row by user id. The user id would be the index of each row. If a user has an ID of 21, and he or she finishes four times. Then the aggregated data frame would have an index of 21 and a total_num of [308, 209, 311, 415].

   Then, we went through the total number and found out the frequency for each day of the week. Our algorithm will give a list with seven elements corresponding from Monday to Sunday with the hour time and frequency as a list in it. To predict the optimal sending time, we always want the highest frequency. However, there might be a circumstance where a user has a close frequency for several hours. If a user has more than one activity hour on a certain day of the week, we would calculate the mean frequency of that day. If the hour has a frequency larger than or equal to the mean, we would remain that hour. Our result of this step was stored in a column called every_day_freq.

Later on, we created seven columns called "Mon", "Tue", "Wed", "Thu", "Fri", "Sat", and "Sun". We would store the elements in every_day_freq into the corresponding day of the week. Below is an output of our result:

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
User Id							
21	[16, 2]	[8, 1]	[15, 4]	[8, 3]	[nan]	[nan]	[nan]
38	[nan]	[nan]	[nan]	[nan]	[3, 2]	[nan]	[nan]
41	[[11, 14], [15, 11]]	[[15, 10], [10, 9]]	[[9, 17], [10, 13], [11, 10], [17, 8], [18, 6]]	[[10, 63], [11, 14], [16, 14], [15, 9], [21, 9...	[9, 12]	[17, 6]	[12, 6]
47	[nan]	[nan]	[14, 4]	[nan]	[nan]	[nan]	[nan]
53	[nan]	[nan]	[nan]	[nan]	[nan]	[13, 3]	[nan]

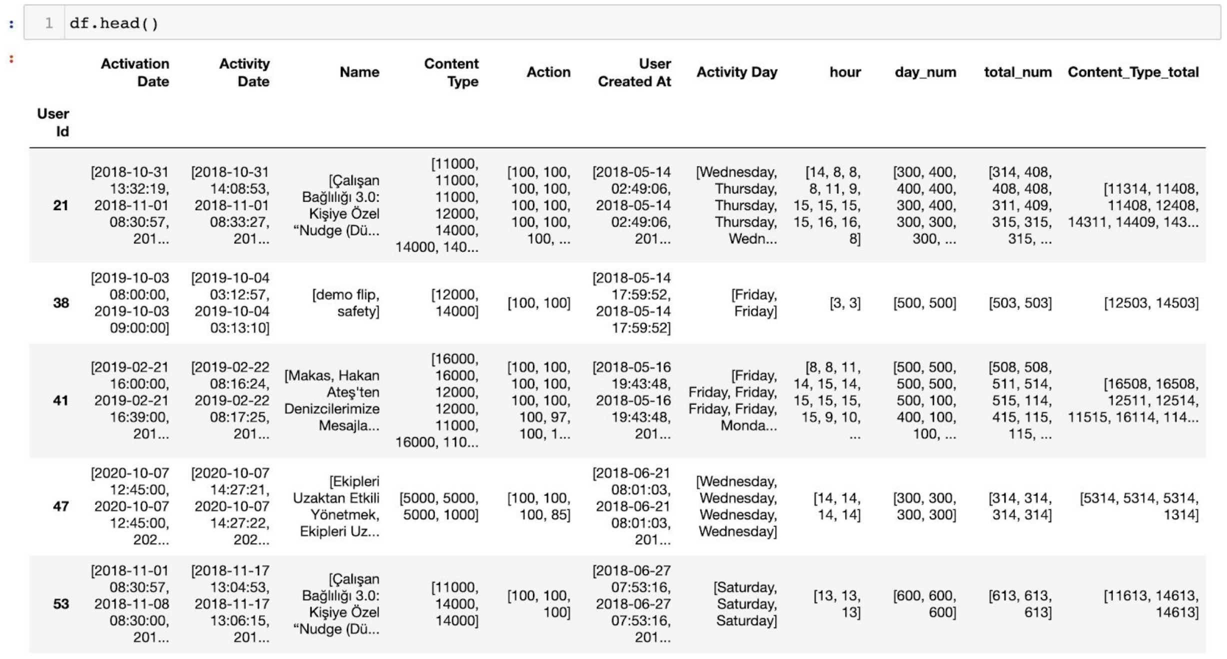
The same things were done with the other two tables. Below is the output:
Action larger than 0 and below 85:

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
User Id							
41	[[14, 11], [10, 11], [15, 8], [11, 4]]	[9, 4]	[[11, 15], [16, 5]]	[[14, 8], [21, 8], [11, 7]]	[[16, 9], [11, 4]]	[17, 4]	[15, 4]
47	[nan]	[nan]	[14, 2]	[nan]	[nan]	[nan]	[nan]
61	[[12, 1], [15, 1]]	[16, 1]	[nan]	[15, 1]	[nan]	[nan]	[nan]
62	[11, 5]	[nan]	[nan]	[nan]	[18, 1]	[nan]	[nan]
63	[11, 1]	[nan]	[nan]	[11, 1]	[nan]	[nan]	[nan]

Action equals 0:

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
User Id							
38	[nan]	[nan]	[nan]	[nan]	[3, 2]	[nan]	[nan]
41	[15, 8]	[15, 5]	[18, 1]	[[16, 19], [11, 10], [21, 7]]	[12, 7]	[nan]	[nan]
47	[nan]	[nan]	[14, 5]	[nan]	[nan]	[nan]	[nan]
61	[12, 6]	[16, 10]	[nan]	[15, 3]	[nan]	[nan]	[nan]
62	[11, 10]	[nan]	[nan]	[nan]	[18, 1]	[nan]	[nan]

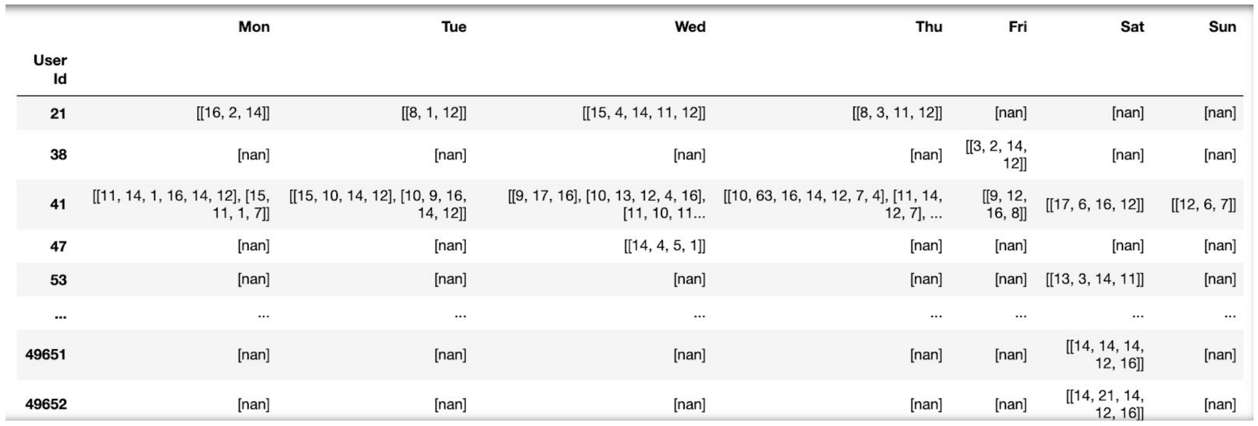
Later, we added the corresponding content type of each hour time on each day of the week. We first made changes to the categorical variable. At first, the numbers are stored as integers from 1 to 17. To make it easy to parse the day, hour, and content type, we multiply each integer by 100. We added hr_num, 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:



```
df.head()
```

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]

Next, we 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:



User Id	Mon	Tue	Wed	Thu	Fri	Sat	Sun
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]

Lastly, 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. For example, 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.

5) How to improve our algorithm so that our clients can use it in the future with the increase of application functions and users?

   We converted our result into CSV files so that our clients could check which time should he use for existing users. One problem is that we use the mean to delete low frequency hour time. We should find out if there exists any other optimal and accurate regulations for deleting low frequency hour time. Secondly, we have to find out which hour we should choose to send. For instance, if we have a result on Thursday that [[16, 8], [15,7]], we need to improve our algorithm to analyze either 16 or 15 is more optimal.

6) After completing this project, what do we still need to improve in the project?

   Although in this process, customer requirements are constantly changing and adding, our team still gives top priority to customer requirements. So far, our result can be used to predict for the existing users. We will hand in our results to our client for testing for the accuracy of the algorithm. We will use the feedback from our client to improve.

   Besides, while writing the code, we want to quickly find out an approach to solve the problem, but we do not consider if our code is succinct and the running time of it. Next, after we have made sure of the availability of algorithms, we will try to improve the performance of the code.

4. Questions need to be answered

Based on the questions that we mentioned in Deliverable 0, there might be two more questions that we may not be able to answer right now. The questions are below.

1) After the testing process, what accuracy do our predictions react to, and how do we adjust the algorithm to meet clients’ demanding click rate of more than 40%?
2) During the real users’ testing, what’s the real click rate we reach by using our algorithm?

For these two questions, since customer needs are constantly changing and increasing, our team still puts customer needs first. So we focus more on the current customer demands as we mentioned above. We will definitely communicate with the client to see if he still wants to increase the completion rate by more than 40% or not and may need the client's help for future testing on the real application and see our result's accuracy.

5. Summary and Reflection

For this project, our team has obtained the first result of the optimal notification sending time, especially for the users whose completion rate is over 85%.

Because our client values the completion rate more and given the requests that they want to get the optimal notification sending time for users whose completion rates were over 85% at first. And since there are different content types from different journey companies, the client also wants the optimal notification sending time to be classified according to that. Although in this process, customer requirements are constantly changing and adding, our team still gives top priority to customer requirements. While getting the first result in time, we also emailed the csv file of the result to the customer. In this way, customers can also check-in time, view their required results, and make changes while the project is in progress.

For our next step, as we mentioned before in deliverable 1 and presentation, we will 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.