
bundle*notificationsDocumentation*

Release 0.1.0

Javier SG

Feb 09, 2020

Contents:

1	A simple tool to bundle notifications	1
1.1	About	1
1.2	Quickstart	2
1.3	Optimization problem & local search solution	4
1.4	Features: Current and future	5
2	Speed considertions	7
2.1	Optimal delay	7
2.2	Bundle functions	7
3	bundle_notifications	9
3.1	bundle_notifications package	9
4	History	15
4.1	0.1.0 (2020-01-29)	15
5	A simple tool to bundle notifications	17
5.1	About	17
5.2	Quickstart	18
5.3	Optimization problem & local search solution	20
5.4	Features: Current and future	21
6	Indices and tables	23
	Python Module Index	25
	Index	27

CHAPTER 1

A simple tool to bundle notifications

[Read the docs here](#)

1.1 About

This package contains a tool for bundling notifications in event streams. The goal is to minimize the number of notifications sent to users and to not send more than 4 notifications per day.

As an example, here it is the first couple of rows for a sample `user_id`:

=====	=====	=====	⌋
↪=====			
timestamp	user_id	friend_id	⌋
↪friend_name			
=====	=====	=====	⌋
↪=====			
2017-08-01 01:20:47	CFFEC5978B0A4A05FA6DCEFB2C82CC	2BB0471CAA78ED0FCEE143E175F034	⌋
↪Mona			
2017-08-01 02:28:27	CFFEC5978B0A4A05FA6DCEFB2C82CC	2BB0471CAA78ED0FCEE143E175F034	⌋
↪Mona			
2017-08-01 03:00:42	CFFEC5978B0A4A05FA6DCEFB2C82CC	74C09338D7CA031859AE26A1586692	⌋
↪Toomas			
2017-08-01 03:51:05	CFFEC5978B0A4A05FA6DCEFB2C82CC	F039A0F7A3245F7B2D7BD0942F3680	⌋
↪Sean			
2017-08-01 05:03:44	CFFEC5978B0A4A05FA6DCEFB2C82CC	DF6A386FE701217C2A12292DB8D142	⌋
↪Buse			
2017-08-01 05:08:29	CFFEC5978B0A4A05FA6DCEFB2C82CC	385308FE41CA0484E84B01D5EED659	⌋
↪Σωτριο			

(continues on next page)

(continued from previous page)

2017-08-01 05:59:33	CFFEC5978B0A4A05FA6DCEFB2C82CC	00A0ED2A6F99DE0E577C51FAEBF302	
2017-08-01 06:31:08	CFFEC5978B0A4A05FA6DCEFB2C82CC	72C688FB41B4EDE06DBC790020FBE7	↪
↪Victoria			
2017-08-01 06:45:44	CFFEC5978B0A4A05FA6DCEFB2C82CC	159854120B568D6449798289D97D64	↪
↪Franciso			
2017-08-01 06:53:51	CFFEC5978B0A4A05FA6DCEFB2C82CC	B5BA8FA5CF5342CBCC9CDAA427E058	↪
↪Δυκων			
2017-08-01 07:01:17	CFFEC5978B0A4A05FA6DCEFB2C82CC	9AB43D430EF8C4443FB8698EFD5092	↪
↪Δαμιαν			
2017-08-01 07:04:32	CFFEC5978B0A4A05FA6DCEFB2C82CC	B34CFFB5CA3C6EAA95991E35FA5066	↪
↪Bakos			
2017-08-01 07:19:44	CFFEC5978B0A4A05FA6DCEFB2C82CC	16CC2AA801B1F29D4991C947B8705A	↪
↪Rozalia			
2017-08-01 07:51:01	CFFEC5978B0A4A05FA6DCEFB2C82CC	268045C1DDB279D56F9873FCC5D2AA	↪
↪Marcu			
2017-08-01 08:38:00	CFFEC5978B0A4A05FA6DCEFB2C82CC	57AA5706AD9E5D051463DCEA8FD9BF	↪
↪Blanduzia			

Using bundle_notifications tool, we can easily compute a solution table with the following columns:

1. **notification_sent**: timestamp of the timestamp when the notification should have been sent
2. **timestamp_first_tour**: timestamp for the first tour amongst his/her friends
3. **tours**: number of friends that have gone on a tour since the last notification was sent
4. **receiver_id**: id of the receiver
5. **message**: custom notification message

Here it is the outcome:

notification_sent	timestamp_first_tour	tours	receiver_id	
↪message				↪
2017-08-01 07:51:01	2017-08-01 01:20:47	13	CFFEC5978B0A4A05FA6DCEFB2C82CC	↪
↪Mona and 12 others went on a tour				
2017-08-01 17:44:37	2017-08-01 08:38:00	9	CFFEC5978B0A4A05FA6DCEFB2C82CC	↪
↪Blanduzia and 8 others went on a tour				
2017-08-01 20:18:46	2017-08-01 17:56:46	14	CFFEC5978B0A4A05FA6DCEFB2C82CC	↪
↪ and 13 others went on a tour				
2017-08-01 21:59:55	2017-08-01 20:29:26	15	CFFEC5978B0A4A05FA6DCEFB2C82CC	↪
↪Rozalia and 14 others went on a tour				
2017-08-02 06:49:19	2017-08-02 02:25:24	14	CFFEC5978B0A4A05FA6DCEFB2C82CC	↪
↪Buse and 13 others went on a to				

Note that Mona went on 2 tours in a row. The tool takes this into account this so that Mona's friend receive a single notification that Mona went on a tour.

1.2 Quickstart

1. Option 1: using pip to directly install the package to a virtual enviroment:

```
# Create a virtual enviroment
$ python -m venv .venv_bundle_notifications

# Activate it
$ source .venv_bundle_notifications/bin/activate

# Install using pip + git
$ pip install git+https://github.com/jsga/bundle_notifications.git
```

2. Option 2: clone the repository from Github:

```
$ git clone https://github.com/jsga/bundle_notifications.git
```

Alternatively, you can manually download the repository as a zip file.

Make sure the terminal is at the root of the package:

```
$ cd bundle_notifications
$ pwd
>/Users/myuser/Documents/GitHub/bundle_notifications
```

You should see something like the above

In case you want to install this package in a virtual enviroment, create one and activate it:

```
$ python -m venv .venv_bundle_notifications
$ source .venv_bundle_notifications/bin/activate
```

Install the package:

```
$ python setup.py install
```

3. Ready! Bundle your first notifications! Using an example dataset, printing only 10 rows:

```
$ bundle_notifications -p "https://static-eu-komoot.s3.amazonaws.com/backend/
↳challenge/notifications.csv" -n 10
```

It takes about 5 or 6 minutes. The output is as follows:

```
Downloading data...
Bundling notifications... (Estimated time: 368.35 seconds for 337657 rows)
Saving to csv: bundle_notifications.csv
Great! Here there are the first 10 bundled notifications
notification_sent    timestamp_first_tour    tours    receiver_id
↳message
-----
↳-----
2017-09-05 12:54:49    2017-09-05 12:54:49    1    00013DA3ABDE2F0771AB56A53A9AA3
↳Amelia went on a tour
2017-09-05 13:28:31    2017-09-05 13:28:31    1    00013DA3ABDE2F0771AB56A53A9AA3
↳Amelia went on a tour
2017-09-24 11:20:48    2017-09-24 11:20:48    1    000367CD43072A5C649AD27FAC6479
↳Magdaléna went on a tour
2017-08-01 12:34:51    2017-08-01 12:09:25    1    0005BDD51B0185DCF1A4932CEB8437
↳Sara went on a tour
2017-08-01 13:01:50    2017-08-01 12:55:33    1    0005BDD51B0185DCF1A4932CEB8437
↳Sara went on a tour
2017-08-01 14:26:25    2017-08-01 13:45:58    1    0005BDD51B0185DCF1A4932CEB8437
↳Sara went on a tour
```

(continues on next page)

(continued from previous page)

2017-08-01 15:31:46	2017-08-01 14:58:13	1	0005BDD51B0185DCF1A4932CEB8437	↳
↳ Sara went on a tour				
2017-08-02 09:25:01	2017-08-02 09:25:01	1	0005BDD51B0185DCF1A4932CEB8437	↳
↳ Bonifác went on a tour				
2017-08-03 11:00:03	2017-08-03 11:00:03	1	0005BDD51B0185DCF1A4932CEB8437	↳
↳ Bonifác went on a tour				
2017-08-04 13:26:34	2017-08-04 13:26:34	1	0005BDD51B0185DCF1A4932CEB8437	↳
↳ Rameshwor went on a tour				

1.3 Optimization problem & local search solution

There are two goals:

1. To not send more than 4 notifications a day to a user (should happen only a few times)
2. To keep sending delay minimal

These goals can be translated into an optimization problem, where the decision variable is $\mathbf{x} = [x_1, x_2, x_3, x_4]$ representing indexes, each one corresponding to the timestamp when the notification should have been sent. The function to minimize is, therefore, the total delay incurred by sending the notifications at \mathbf{x} .

Let's formulate an example. Say there are 11 events with timestamps $\mathbf{t} = [t_0, t_2, \dots, t_{10}]$ and that we decide to send notification at indexes $\mathbf{x} = [0, 2, 7, 10]$. The total delay \mathbf{D} is then calculated as:

$$\mathbf{D} = (t_2 - t_1) + (t_7 - t_6) + (t_7 - t_4) + (t_7 - t_3) + (t_7 - t_6) + (t_{10} - t_8) + (t_{10} - t_9)$$

Even though the intuition behind this problem is quite simple, this optimization problem is unfortunately not linear and not straight-forward to formulate. For this reason, in this tool, we use a heuristic optimization approach (local search), which, in practice, seems to work well. See [speed considerations](#) section for further information.

1.3.1 Implementation

In short, the strategy is as follows.

First, we group the users by *user_id* and *day*. For each of those groups, do:

1. If the number of notifications for a user is lower or equal than 4, we simply decide to send all notifications.
2. **If the number of notifications is greater than 4:**
 1. **Obtain an optimal notification schedule:**
 1. We first obtain an initial solution, simply by distributing the notifications at indexes $\mathbf{x} = [\text{int}(N/4), \text{int}(N/2), \text{int}(0.75*N), N-1]$ where N is the number of events for that user and day.
 2. From the initial solution we do a negative local search, checking 1 step a time if the total delay decreases by subtracting one of the indexes in \mathbf{x} . We repeat this until the total delay function stops decreasing, or until a maximum number of iterations is reached
 3. We do a positive local search (similar as above)
 2. **We proceed to bundle the notifications:**
 1. Count how many unique friends are active in between two notifications
 2. Discard rows that do not correspond to \mathbf{x}

3. Create a custom message
4. Gather relevant columns and delete intermediate ones

Finally, we consolidate all the datasets (one per group) into one.

1.4 Features: Current and future

The tool relies on two main pandas functionalities: reading CSV files and group-apply functions. Applying a custom function to a pandas groupby element is known to be rather slow - it is even mentioned in the [documentation](#). However, it is flexible and easy to work with, so for this reason, this was my initial approach.

The advantage of using pandas over custom-made tools is its simplicity. The initial version of the functions was quite simple and it was also quick to develop. However, the computing time was too high and the tool would become unusable: 1h30min for 330k rows of data. By iteratively analyzing the bottlenecks and coding equivalent custom functions, the time is now reduced to 5% of what it used to be. I am sure that with some more effort it could go down to 1%.

The tool is built as a Python package. Tests have been implemented and the coding style is PEP8 consistent, checked with *flake8*. I have based the project on this [cookie-cutter](#).

Here there are some possible future improvements:

1. Implement speed enhancements, focusing on translating the groupby step in function *bundle_func()* to Numpy and Numba.
2. Implement a parameter to set the maximum number of notifications. Currently, 4 is hardcoded.
3. Encapsulating this tool in a Docker image would make it much easier to move from development to a production server.
4. Add an option to read the data directly from a database, so that this tool can be run periodically without human supervision
5. If data grows, we could parallelize the computation using [Dask](#). If the docker image is in place we could scale this up to many threads quite easily.

1.4.1 Note

This tool could be used to analyze what *could* have been the optimal notification schedule. As of version V.01, it cannot be used to predict *when* is the best time to send a notification.

This tool could be used as a basis for further analysis: once we know what was optimal in the past, we can create rules for future decisions. This functionality falls out of the scope of the assignment.

Speed considerations

Below there are two notebooks I used for comparing speeds of several functions that I have developed iteratively. Generally speaking, the conclusions are:

1. Pandas-based functions are quick to develop and easy to understand
2. Numpy-based functions run significantly faster than pandas-based functions
3. Numba-compiled functions run significantly faster than Numpy-based functions

2.1 Optimal delay

2.2 Bundle functions

3.1 bundle_notifications package

3.1.1 Submodules

3.1.2 bundle_notifications.bundle_notifications module

Main module.

`bundle_notifications.bundle_notifications.add_notif_counter(x)`

Creates a counter given a solution x

Equivalent function to:

```
df_g['notification_bool'] = False
df_g.notification_bool.iloc[x] = True

# Now do a cumsum counter
df_g['notification_counter'] =
    df_g.notification_bool.cumsum().shift()+1
df_g.notification_counter.iloc[0] = 1
```

Computationally it is 50x faster to do it this way:

```
print('With @jit:')
%time n1 = add_notif_counter_j(x,np.zeros(x[-1]+1,dtype='int'))
print('Without @jit:')
%time n2 = add_notif_counter(x)
print('With pandas:')
%time n3 = notif_counter_pandas(df_g)
# Recall that 1ms = 1000 ms. So it is 95x faster

np.allclose(n1,n2.astype("int") )
np.allclose(n2,n3.astype("int") )
```

(continues on next page)

(continued from previous page)

```

>> With @jit:
>> CPU times: user 22 µs, sys: 0 ns, total: 22 µs
>> Wall time: 26.9 µs
>> Without @jit:
>> CPU times: user 25 µs, sys: 1 µs, total: 26 µs
>> Wall time: 29.1 µs
>> With pandas:
>> CPU times: user 2.62 ms, sys: 705 µs, total: 3.33 ms
>> Wall time: 2.73 ms

```

Parameters *x* (*np.array of int*) – Array of length 4 with. Each element is an index, indicating the timestamp when the notification should be sent. Example: *x* = *np.array*([0, 1, 2, 5])

Returns Numpy array containing a counter, starting from 1 up to 4

Return type *np.array of int*

bundle_notifications.bundle_notifications.bundle(df)

Bundles the notifications given a *pd.DataFrame* of events

Parameters *df* (*pd.DataFrame*) – *DataFrame* containing 4 columns: ['timestamp', 'user_id', 'friend_id', 'friend_name']

Returns Contains 4 derived columns: ['notification_sent', 'timestamp_first_tour', 'tours', 'receiver_id', 'message']

Return type *pd.DataFrame*

bundle_notifications.bundle_notifications.bundle_func(df_g)

Bundles notifications for a *user_id*

This function is meant to be used after a pandas grouping (or manual filtering) of *user_ids*.

Parameters *df_g* (*pd.DataFrame*) – *DataFrame* containing 4 columns: ['timestamp', 'user_id', 'friend_id', 'friend_name']

Returns *DataFrame* containing 5 extra columns: ['notification_bool', 'tours', 'notification_counter', 'message', 'timestamp_first_tour']

Return type *pd.DataFrame*

bundle_notifications.bundle_notifications.count_tours_per_notif(notification_counter, friend_id, friend_name, timestamp)

Count number of friends that went on a tour during a given time, indicated by a counter.

Equivalent to:

```

df_g['tours'] =
    df_g.groupby('notification_counter')['friend_id'].apply(
        lambda x: (1- x.duplicated()).cumsum()
    )

```

In pseudo-code, this is equivalent to:

- As an input, we have a dataset filtered by *user_id* and day of the year

- Each of the inputs of this function are numpy arrays, corresponding to a column of the dataset. Doing basic in numpy is much faster, especially if we manage to use a @jit compiler (TODO)
- **Let us call the solution `_tours_`. For each element in `friend_id` we do:**
 - Start `tours = 1` at iteration `i=0`
 - We add `tours += 1` if the `friend_id` is new.
 - **We continue until `i` in `notification_counter`**
 - * Reset `tours = 1`
 - * **Keep track of the name and timestamp of the first element**
(`name_first`, `timestamp_first_tour`)

Parameters

- **`notification_counter`** (*np.array*) – notification counter. Could be the output of an `optimal_delay` method. For example, `np.array([1, 2, 3, 10])` for a 10 element array
- **`friend_id`** (*np.array of str*) – array containing names of the friends
- **`friend_name`** (*np.array of str*) – array containing names of the friends
- **`timestamp`** (*np.array of datetime64[ns]*) – timestamps when the notifications are generated

Returns

- **`tours`** (*np.array of int*) – Count of the number of tours done since the last notification was sent, for unique friends-id
- **`name_first`** (*np.array of <U256*) – Names of the friend who first did a tour since the last notification was sent. `len(name_first) <= 4`
- **`timestamp_first_tour`** (*np.array of datetime64[ns]*) – Timestamp of the first tour done by a friend, since the last notification was sent. `len(timestamp_first_tour) <= 4`
- **`message`** (*np.array of np.array of <U256*) – Message to be sent. `len(message) <= 4`

`bundle_notifications.bundle_notifications.create_message(tours, name_first)`

Returns the notification message as a numpy array

Parameters

- **`tours`** (*np.array of in*) – array of integers representing the number of tours
- **`name_first`** (*np.array*) – array of names

Returns array with the message like “Mona and 12 others went on a tour”

Return type np.array

`bundle_notifications.bundle_notifications.create_message_single(t, n)`

Creates custom message based on the number of tours and the friend name

Parameters

- **`t`** (*int*) – Number of tours
- **`n`** (*str*) – Name of the friend

Returns Notification message

Return type str

`bundle_notifications.bundle_notifications.load_data(path_csv, nrows=None)`

Loads the notification csv file

Parameters

- **path_csv** (*str*) – Path or url to the csv file containing the data. It should have 4 comma-separated columns without header.
- **nrows** (*int, optional*) – Number of rows of file to read. Useful for reading pieces of large files or for testing this function.

Returns DataFrame containing the stream of data. It has 4 columns: 'timestamp', 'user_id', 'friend_id', 'friend_name'. The column named 'timestamp' is cast as a datetime64[ns] type.

Return type pd.DataFrame

3.1.3 bundle_notifications.cli module

Console script for bundle_notifications package.

3.1.4 bundle_notifications.optimal_delay module

This module compiles a few functions used to compute the optimal notification times.

Functions decorated with @jit are not included in the coverage report.

`bundle_notifications.optimal_delay.delay`

Calculates delay if notifications are sent at indexes indicated by notification_idx

Parameters **t** (*np.array*) – Array containing the timestamps of the events

Returns Sum of total delay

Return type datetime[ns]

`bundle_notifications.optimal_delay.local_search(timestamp)`

Heuristic optimization of the notification schedule

Parameters **t** (*np.array (int)*) – Array of integer values. These could correspond to datetime64[ns]. The inputs needs to be an integer as it is better supported by the JIT compiler (TODO: allow for datetime64 type)

Returns Optimized notification schedule. Each item corresponds to an index of t where the notification should have been sent.

Return type np.array

`bundle_notifications.optimal_delay.local_search_negative`

Local search negative step

Parameters

- **t** (*np.array (DateTime)*) – array of timestamps
- **x** (*list of int*) – Indicate indexes where the notification is sent
- **max_iter** (*int*) – Maximum number of local search steps to be performed

Returns optimized notification schedule. Each item corresponds to an index of t

Return type np.array

`bundle_notifications.optimal_delay.local_search_positive`

Local search negative step

Parameters

- **t** (*np.array (DateTime)*) – array of timestamps
- **x** (*list of int*) – Indicate indexes where the notification is sent
- **max_iter** (*int*) – Maximum number of local search steps to be performed

Returns optimized notification schedule. Each item corresponds to an index of t

Return type np.array

`bundle_notifications.optimal_delay.total_delay_brute (timestamp)`

Given a Series of Timestamps, calculate total delay using brute force: try out all possible combinations

This function is kept here for reference and possible future implementations.

Parameters **t** (*np.array*) – Array containing the timestamps of the events

Returns array of length 4. Each element indicates an index where the optimal notification should be sent.

Return type np.array

`bundle_notifications.optimal_delay.total_delay_initial`

Given a Series of Timestamps, sample 4 points equally distributed index-wise

Parameters **t** (*np.array*) – Array containing the timestamps of the events

Returns array of length 4. Each element indicates an index where the initial optimal notification should be sent.

Return type np.array

3.1.5 Module contents

Top-level package for bundle_notifications.

CHAPTER 4

History

4.1 0.1.0 (2020-01-29)

- First release.

A simple tool to bundle notifications

[Read the docs here](#)

5.1 About

This package contains a tool for bundling notifications in event streams. The goal is to minimize the number of notifications sent to users and to not send more than 4 notifications per day.

As an example, here it is the first couple of rows for a sample `user_id`:

timestamp	user_id	friend_id
↪ friend_name		
2017-08-01 01:20:47	CFFEC5978B0A4A05FA6DCEFB2C82CC	2BB0471CAA78ED0FCEE143E175F034
↪ Mona		
2017-08-01 02:28:27	CFFEC5978B0A4A05FA6DCEFB2C82CC	2BB0471CAA78ED0FCEE143E175F034
↪ Mona		
2017-08-01 03:00:42	CFFEC5978B0A4A05FA6DCEFB2C82CC	74C09338D7CA031859AE26A1586692
↪ Toomas		
2017-08-01 03:51:05	CFFEC5978B0A4A05FA6DCEFB2C82CC	F039A0F7A3245F7B2D7BD0942F3680
↪ Sean		
2017-08-01 05:03:44	CFFEC5978B0A4A05FA6DCEFB2C82CC	DF6A386FE701217C2A12292DB8D142
↪ Buse		
2017-08-01 05:08:29	CFFEC5978B0A4A05FA6DCEFB2C82CC	385308FE41CA0484E84B01D5EED659
↪ Σωτριο		

(continues on next page)

(continued from previous page)

2017-08-01 05:59:33	CFFEC5978B0A4A05FA6DCEFB2C82CC	00A0ED2A6F99DE0E577C51FAEBF302	
2017-08-01 06:31:08	CFFEC5978B0A4A05FA6DCEFB2C82CC	72C688FB41B4EDE06DBC790020FBE7	↪
↪Victoria			
2017-08-01 06:45:44	CFFEC5978B0A4A05FA6DCEFB2C82CC	159854120B568D6449798289D97D64	↪
↪Franciso			
2017-08-01 06:53:51	CFFEC5978B0A4A05FA6DCEFB2C82CC	B5BA8FA5CF5342CBCC9CDAA427E058	↪
↪Δυκων			
2017-08-01 07:01:17	CFFEC5978B0A4A05FA6DCEFB2C82CC	9AB43D430EF8C4443FB8698EFD5092	↪
↪Δαμιαν			
2017-08-01 07:04:32	CFFEC5978B0A4A05FA6DCEFB2C82CC	B34CFFB5CA3C6EAA95991E35FA5066	↪
↪Bakos			
2017-08-01 07:19:44	CFFEC5978B0A4A05FA6DCEFB2C82CC	16CC2AA801B1F29D4991C947B8705A	↪
↪Rozalia			
2017-08-01 07:51:01	CFFEC5978B0A4A05FA6DCEFB2C82CC	268045C1DDB279D56F9873FCC5D2AA	↪
↪Marcu			
2017-08-01 08:38:00	CFFEC5978B0A4A05FA6DCEFB2C82CC	57AA5706AD9E5D051463DCEA8FD9BF	↪
↪Blanduzia			

Using bundle_notifications tool, we can easily compute a solution table with the following columns:

1. **notification_sent**: timestamp of the timestamp when the notification should have been sent
2. **timestamp_first_tour**: timestamp for the first tour amongst his/her friends
3. **tours**: number of friends that have gone on a tour since the last notification was sent
4. **receiver_id**: id of the receiver
5. **message**: custom notification message

Here it is the outcome:

notification_sent	timestamp_first_tour	tours	receiver_id	
↪message				↪
2017-08-01 07:51:01	2017-08-01 01:20:47	13	CFFEC5978B0A4A05FA6DCEFB2C82CC	↪
↪Mona and 12 others went on a tour				
2017-08-01 17:44:37	2017-08-01 08:38:00	9	CFFEC5978B0A4A05FA6DCEFB2C82CC	↪
↪Blanduzia and 8 others went on a tour				
2017-08-01 20:18:46	2017-08-01 17:56:46	14	CFFEC5978B0A4A05FA6DCEFB2C82CC	↪
↪ and 13 others went on a tour				
2017-08-01 21:59:55	2017-08-01 20:29:26	15	CFFEC5978B0A4A05FA6DCEFB2C82CC	↪
↪Rozalia and 14 others went on a tour				
2017-08-02 06:49:19	2017-08-02 02:25:24	14	CFFEC5978B0A4A05FA6DCEFB2C82CC	↪
↪Buse and 13 others went on a to				

Note that Mona went on 2 tours in a row. The tool takes this into account this so that Mona's friend receive a single notification that Mona went on a tour.

5.2 Quickstart

1. Option 1: using pip to directly install the package to a virtual enviroment:

```
# Create a virtual enviroment
$ python -m venv .venv_bundle_notifications

# Activate it
$ source .venv_bundle_notifications/bin/activate

# Install using pip + git
$ pip install git+https://github.com/jsga/bundle_notifications.git
```

2. Option 2: clone the repository from Github:

```
$ git clone https://github.com/jsga/bundle_notifications.git
```

Alternatively, you can manually download the repository as a zip file.

Make sure the terminal is at the root of the package:

```
$ cd bundle_notifications
$ pwd
>/Users/myuser/Documents/GitHub/bundle_notifications
```

You should see something like the above

In case you want to install this package in a virtual enviroment, create one and activate it:

```
$ python -m venv .venv_bundle_notifications
$ source .venv_bundle_notifications/bin/activate
```

Install the package:

```
$ python setup.py install
```

3. Ready! Bundle your first notifications! Using an example dataset, printing only 10 rows:

```
$ bundle_notifications -p "https://static-eu-komoot.s3.amazonaws.com/backend/
↳challenge/notifications.csv" -n 10
```

It takes about 5 or 6 minutes. The output is as follows:

```
Downloading data...
Bundling notifications... (Estimated time: 368.35 seconds for 337657 rows)
Saving to csv: bundle_notifications.csv
Great! Here there are the first 10 bundled notifications
notification_sent    timestamp_first_tour    tours    receiver_id
↳message
-----
↳-----
2017-09-05 12:54:49    2017-09-05 12:54:49    1    00013DA3ABDE2F0771AB56A53A9AA3
↳Amelia went on a tour
2017-09-05 13:28:31    2017-09-05 13:28:31    1    00013DA3ABDE2F0771AB56A53A9AA3
↳Amelia went on a tour
2017-09-24 11:20:48    2017-09-24 11:20:48    1    000367CD43072A5C649AD27FAC6479
↳Magdaléna went on a tour
2017-08-01 12:34:51    2017-08-01 12:09:25    1    0005BDD51B0185DCF1A4932CEB8437
↳Sara went on a tour
2017-08-01 13:01:50    2017-08-01 12:55:33    1    0005BDD51B0185DCF1A4932CEB8437
↳Sara went on a tour
2017-08-01 14:26:25    2017-08-01 13:45:58    1    0005BDD51B0185DCF1A4932CEB8437
↳Sara went on a tour
```

(continues on next page)

(continued from previous page)

2017-08-01 15:31:46	2017-08-01 14:58:13	1	0005BDD51B0185DCF1A4932CEB8437	↳ Sara went on a tour
2017-08-02 09:25:01	2017-08-02 09:25:01	1	0005BDD51B0185DCF1A4932CEB8437	↳ Bonifác went on a tour
2017-08-03 11:00:03	2017-08-03 11:00:03	1	0005BDD51B0185DCF1A4932CEB8437	↳ Bonifác went on a tour
2017-08-04 13:26:34	2017-08-04 13:26:34	1	0005BDD51B0185DCF1A4932CEB8437	↳ Rameshwor went on a tour

5.3 Optimization problem & local search solution

There are two goals:

1. To not send more than 4 notifications a day to a user (should happen only a few times)
2. To keep sending delay minimal

These goals can be translated into an optimization problem, where the decision variable is $\mathbf{x} = [x_1, x_2, x_3, x_4]$ representing indexes, each one corresponding to the timestamp when the notification should have been sent. The function to minimize is, therefore, the total delay incurred by sending the notifications at \mathbf{x} .

Let's formulate an example. Say there are 11 events with timestamps $\mathbf{t} = [t_0, t_2, \dots, t_{10}]$ and that we decide to send notification at indexes $\mathbf{x} = [0, 2, 7, 10]$. The total delay \mathbf{D} is then calculated as:

$$\mathbf{D} = (t_2 - t_1) + (t_7 - t_6) + (t_7 - t_4) + (t_7 - t_3) + (t_7 - t_6) + (t_{10} - t_8) + (t_{10} - t_9)$$

Even though the intuition behind this problem is quite simple, this optimization problem is unfortunately not linear and not straight-forward to formulate. For this reason, in this tool, we use a heuristic optimization approach (local search), which, in practice, seems to work well. See [speed considerations](#) section for further information.

5.3.1 Implementation

In short, the strategy is as follows.

First, we group the users by *user_id* and *day*. For each of those groups, do:

1. If the number of notifications for a user is lower or equal than 4, we simply decide to send all notifications.
2. **If the number of notifications is greater than 4:**
 1. **Obtain an optimal notification schedule:**
 1. We first obtain an initial solution, simply by distributing the notifications at indexes $\mathbf{x} = [\text{int}(N/4), \text{int}(N/2), \text{int}(0.75*N), N-1]$ where N is the number of events for that user and day.
 2. From the initial solution we do a negative local search, checking 1 step a time if the total delay decreases by subtracting one of the indexes in \mathbf{x} . We repeat this until the total delay function stops decreasing, or until a maximum number of iterations is reached
 3. We do a positive local search (similar as above)
 2. **We proceed to bundle the notifications:**
 1. Count how many unique friends are active in between two notifications
 2. Discard rows that do not correspond to \mathbf{x}

3. Create a custom message
4. Gather relevant columns and delete intermediate ones

Finally, we consolidate all the datasets (one per group) into one.

5.4 Features: Current and future

The tool relies on two main pandas functionalities: reading CSV files and group-apply functions. Applying a custom function to a pandas groupby element is known to be rather slow - it is even mentioned in the [documentation](#). However, it is flexible and easy to work with, so for this reason, this was my initial approach.

The advantage of using pandas over custom-made tools is its simplicity. The initial version of the functions was quite simple and it was also quick to develop. However, the computing time was too high and the tool would become unusable: 1h30min for 330k rows of data. By iteratively analyzing the bottlenecks and coding equivalent custom functions, the time is now reduced to 5% of what it used to be. I am sure that with some more effort it could go down to 1%.

The tool is built as a Python package. Tests have been implemented and the coding style is PEP8 consistent, checked with *flake8*. I have based the project on this [cookie-cutter](#).

Here there are some possible future improvements:

1. Implement speed enhancements, focusing on translating the groupby step in function *bundle_func()* to Numpy and Numba.
2. Implement a parameter to set the maximum number of notifications. Currently, 4 is hardcoded.
3. Encapsulating this tool in a Docker image would make it much easier to move from development to a production server.
4. Add an option to read the data directly from a database, so that this tool can be run periodically without human supervision
5. If data grows, we could parallelize the computation using [Dask](#). If the docker image is in place we could scale this up to many threads quite easily.

5.4.1 Note

This tool could be used to analyze what *could* have been the optimal notification schedule. As of version V.01, it cannot be used to predict *when* is the best time to send a notification.

This tool could be used as a basis for further analysis: once we know what was optimal in the past, we can create rules for future decisions. This functionality falls out of the scope of the assignment.

CHAPTER 6

Indices and tables

- `genindex`
- `modindex`
- `search`

b

`bundle_notifications`, [13](#)

`bundle_notifications.bundle_notifications`,
[9](#)

`bundle_notifications.cli`, [12](#)

`bundle_notifications.optimal_delay`, [12](#)

A

`add_notif_counter()` (in module `bundle_notifications.bundle_notifications`), 9

B

`bundle()` (in module `bundle_notifications.bundle_notifications`), 10

`bundle_func()` (in module `bundle_notifications.bundle_notifications`), 10

`bundle_notifications` (module), 13

`bundle_notifications.bundle_notifications` (module), 9

`bundle_notifications.cli` (module), 12

`bundle_notifications.optimal_delay` (module), 12

C

`count_tours_per_notif()` (in module `bundle_notifications.bundle_notifications`), 10

`create_message()` (in module `bundle_notifications.bundle_notifications`), 11

`create_message_single()` (in module `bundle_notifications.bundle_notifications`), 11

D

`delay` (in module `bundle_notifications.optimal_delay`), 12

L

`load_data()` (in module `bundle_notifications.bundle_notifications`), 12

`local_search()` (in module `bundle_notifications.optimal_delay`), 12

`local_search_negative` (in module `bundle_notifications.optimal_delay`), 12

`local_search_positive` (in module `bundle_notifications.optimal_delay`), 13

T

`total_delay_brute()` (in module `bundle_notifications.optimal_delay`), 13

`total_delay_initial` (in module `bundle_notifications.optimal_delay`), 13