ReadMe for Glossier Problem Report:

Input data are 60 files in Json format.
They were downloaded to my local PC and put in following folder:
/Users/shufang/Documents/Insight_fellow/Glossier_oa/data/*.json

1. Pre-analysis

(for more pre-analysis detaisl, please check the Jupyter Notebook file: Glossier_data_pre_analysis.html)

There are 25,000 order records in total.
For each order record, there are 51 key-value pairs (saved in dictionary).

1) 9 of 51 key-value pairs are with None values for all 25,000 records.
2) 15 of 51 key-value pairs are with same values for all 25,000 records.

9 key-value pairs with None Values

closed_at	**None**
referring_site	None
landing_site	None
cancelled_at	None
cancel_reason	None
source_url	None
customer_locale	None
landing_site_ref	None
total_discount	None

15 key-value pairs with Same values

email	**censored@censored.com**
contact_email	censored@censored.com
phone	"555-555-5555"
token	censored
cart_token	censored
checkout_token	censored
financial_status	paid
confirmed	TRUE
taxes_included	False
currency	"USD"
browser_ip	127.0.0.1
source_name	pos
app_id	129785
tags	""
order_status_url	https://checkout.shopify.com/censored

3) 26 of 51 key-value pairs are with normal single values. I included them in **orders_table**.
4) 1 of 51 key-value pair(line_items) is with multiple values(list-type), which I saved in a different

table: **products_table.**

The key-value pair (line_items) contains the product information.
The number of the multiple values for each order is from 1~21. Most of the order are with 1~5 products.

# of line_items values	# of orders
1	8758
2	7205
3	4303
4	2197
5	1175
6	637
7	325
8	186
9	90
10	49
11	37
12	14
13	7
14	7
15	3
16	4
17	1
18	0
19	1
20	0
21	1

There are 42 distinct product_ids and 66306 products in total including the quantity information.

Distinct Product_id	# of product_id
9096535107	1465
9086876675	8616
9086871875	13716
10724240323	812
9251741507	1380
10892093059	1818
10776543363	1487
9681624963	128
9086876995	1224
10895762755	1399
9115063107	2856
9086873987	2239
9086871491	2648

9086872771	4771
9086873027	2201
9086872003	995
9086872195	1127
9591643907	4276
9110760835	589
10238031811	923
10724180163	847
9086877827	2179
9979804739	1820
9086873155	68
10873762691	517
9086871619	859
9086873091	406
9086876931	643
10873763459	672
9092600515	466
10974324355	997
9092606275	741
11015281475	924
9398113283	95
9086871427	71
10724180867	42
None	1
21495218179	271
10985482179	1
10592905539	12
10917699971	1
11023253507	2

2. Tables

Table a: orders_table:
33 columns, 25,000 rows
(including 26 from original input,
1 total_line_items_nums (the length of the value list)
3 dates from created_at, updated_at, processed_at, and
year, month, and day from processed_at.
The primary key is order_id from key ”id” of each order)

I kept most of the information in this table for further query or analysis.
Added the dates there because I am not sure witch date you prefer to use.
Saved year, month and day for my personal choice. It is easy to query and obtained results for certain month or day.

Table b: products_table:

9 columns, 60,555 rows
(including one serial_id as primary key,
order_id,
total_line_items_price,
total_line_items_nums,
location_id,
line_items_id,
line_items_vairant_id, l
ine_items_quantity, and
line_items_product_id

In fact, there are only 42 the distinct product_ids.
I kept everything here to have to connection between line_items_product_id, order_id and location_id. Then, I can know which product is popular on certain locations.

Table c: users_table

Based on the orders_table and products_table, I did some query and saved them in users_tables as following.

Columns	Values	Query
Serial_id	1	
Total_order_nums	25,000	`select count(order_id) from orders_table`
Total_day_nums	60	
Total_locations	5	`select count(distinct(location_id)) from orders_table`
Total_price	4,611,578.760	`select sum(CAST(total_price AS DOUBLE PRECISION)) as price from orders_table`
Total_discount	41,60.010	`select sum(CAST(total_discounts AS DOUBLE PRECISION)) as price from orders_table`
Total_tax	372,752.790	`select sum(CAST(total_tax AS DOUBLE PRECISION)) as price from orders_table`
Total_product_nums	66,360	`select sum(CAST(line_items_quantity AS INTEGER)) from products_table`
Total_dis_product_nums	42	`select count(distinct(line_items_product_id)) from products_table`
Most_populer_product	‘9086871875’	`select line_items_product_id, sum(CAST(line_items_quantity as INTEGER)) as con from products_table group by line_items_product_id order by con DESC limit 1`
Most_poluler_location	‘371291’	`select location_id, sum(CAST(line_items_quantity as INTEGER)) as con from products_table group by location_id order by con DESC limit 1`
Most_populer_order_day	‘2017-11-11’	`select date3, count(order_id) as order_con from orders_table group by date3 order by order_con DESC limit 1`
Average_price_loc_day	15,371.929	
Price_populer_loc_day	92,939.880	`select sum(CAST(total_price AS DOUBLE PRECISION)) as price from orders_table where location_id = '371291' and date3 = '2017-11-11’`
Average_products_loc_day	221	
Products_populer_loc_day	1307	`select sum(CAST(p.line_items_quantity AS INTEGER)) as con from products_table as p join orders_table as o on o.order_id = p.order_id where p.location_id = '371291' and o.date3 = '2017-11-11'`

3. Source code:

- Pre-analysis:
  Glossier_data_pre_analysis.html
- Generate table:
  creat_tables_pg.py
  order_table_insert.py
  product_table_insert.py
- Generate users table with summary metrices
  creat_users_table.py
  users_table_insert.py