Winter_2021_DS_Internship_Challenge_Shopify_Question_1

September 3, 2020

1 Problem Statement

On Shopify, we have exactly 100 sneaker shops, and each of these shops sells only one model of shoe. **We want to do some analysis of the average order value (AOV).** When we look at orders data over a 30 day window, we naively calculate an AOV of $3145.13. Given that we know these shops are selling sneakers, a relatively affordable item, something seems wrong with our analysis.

1. Think about what could be going wrong with our calculation. Think about a better way to evaluate this data.
2. What metric would you report for this dataset?
3. What is its value?

2 Summary of The Answers

Through investigating the dataset, I found that the abnormal AOV was caused by the existence of some extremely large order amount values. These values come from two different sources:

1. A very large unit price of sneaker ($25725) for shop No. 78
2. A very large number of items ordered (2000 items in each order) for shop No. 42

While a sneaker of $25725 seems unreasonable, an item order of 2000 can be possible. Therefore, after careful considerations, I decided not to simply drop the large order amount values. **Instead, I used the median order value (MOV) instead of (AOV) as the metric for the dataset. The MOV value is $284.0, which is more reasonable to describe the dataset and more helpful for the analysis**.

In addition, I would recommend checking the data acquisition process for shop No.78, which has all its sneaker at an abnormal price of $25725, to ensure there is no mistake in the data.

3 Solution Walkthrough

[20]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

[5]:
```
# Read csv file
df_raw = pd.read_csv(r'C:\DataScience\Jupyter Files\Internship␣
↪Applications\2019 Winter Data Science Intern Challenge Data Set - Sheet1.
↪csv')
```

[6]:
```
df_raw.head()
```

[6]:

	order_id	shop_id	user_id	order_amount	total_items	payment_method	created_at
0	1	53	746	224	2	cash	2017-03-13 12:36:56
1	2	92	925	90	1	cash	2017-03-03 17:38:52
2	3	44	861	144	1	cash	2017-03-14 4:23:56
3	4	18	935	156	1	credit_card	2017-03-26 12:43:37
4	5	18	883	156	1	credit_card	2017-03-01 4:35:11

[7]:
```
df_raw.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 7 columns):
order_id          5000 non-null int64
shop_id           5000 non-null int64
user_id           5000 non-null int64
order_amount      5000 non-null int64
total_items       5000 non-null int64
payment_method    5000 non-null object
created_at        5000 non-null object
dtypes: int64(5), object(2)
memory usage: 273.6+ KB
```

First, we need to understand the definition of the average order value (AOV):

"Average order value (AOV) is the average amount of money each customer spends per transaction with your store. You can calculate your average order value using this simple formula: **Total revenue / number of orders = average order value**" (from shopify blog)

After a quick view on the dataset, the columns that are relevent for calculating AOV are:

- order_id: this column describes the individual order ID. Ideally, every order should generate a distinct order ID. Total number of orders can thus be found by calculating the total number of order IDs.
- order_amount: this column describes the value of each order. Total revenue can be found by sum up all the values in this column.

We can verify that the thought process is correct by performing the calculation:

[14]:
```
aov_naive = sum(df_raw['order_amount']) / len(df_raw['order_id'])
print("Naive AOV: ${0:.2f}".format(aov_naive))
```

```
Naive AOV: $3145.13
```

The naive AOV calculated above matches the value given in the problem statement. The process of calculation AOV seems correct. However, **the problem is that an AOV of $3145.13 seems to be too big for sneaker shops**. So we need to find out the plausible causes.

There are two hypotheses that may contribute to the problem:

1. The dataset contains duplicates which have large *order_amount* values.
2. The dataset contains outliers in either *order_amount* values or sneaker unit prices.

Let's verify each hypothesis:

Hypothesis 1. the dataset contains duplicates which have large order_amount values

[18]:
```
df_raw.duplicated().value_counts()
```

[18]:
```
False    5000
dtype: int64
```

As no "True" is present in our analysis above, there is no duplicate in the dataset. However, to be more confident about the result, we can further verify by checking if there is any duplicate in the order ID, as the order ID should be distinct for each order.

[19]:
```
df_raw['order_id'].duplicated().value_counts()
```

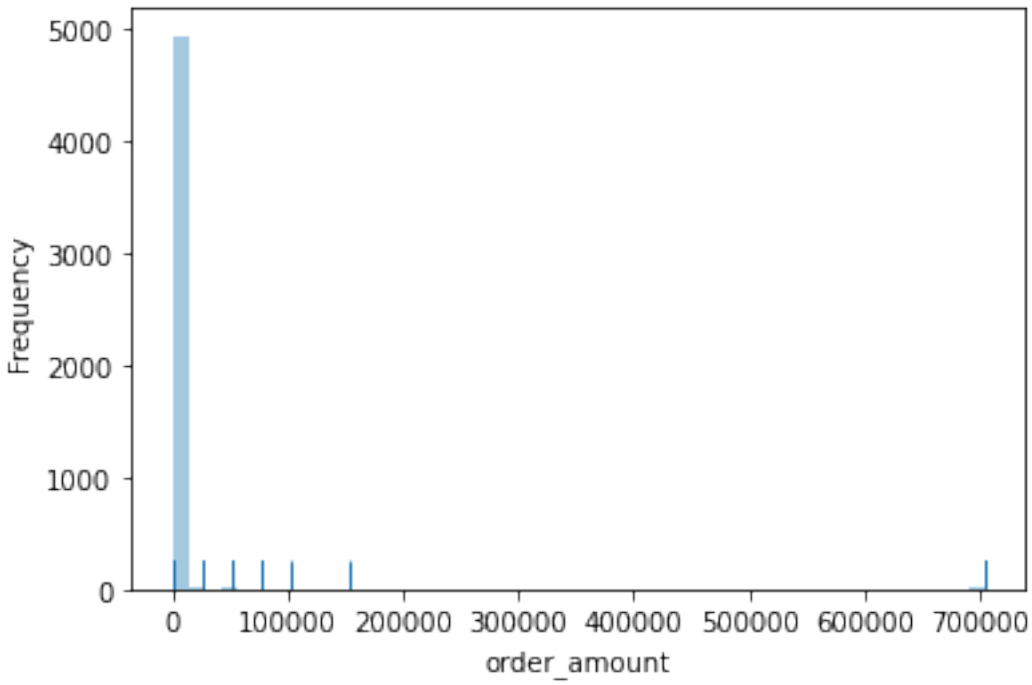
[19]:
```
False    5000
Name: order_id, dtype: int64
```

The order ID also does not have duplicate. So the problem should not be due to the presence of the duplicate.

Hypothesis 2. the dataset contains outliers in either *order_amount* values or sneaker unit prices

Let's first check the *order_amount* column:

[25]:
```
ax = sns.distplot(df_raw['order_amount'], kde=False, rug=True)
ax.set(ylabel = 'Frequency')
plt.show()
```



The histograph above shows both the frequency of the order amount, represented by the thick bar, and where the order amount occurs, represented by the thin bars. It can be noticed that the majority of the order amount are small, while several order amount is very large. These large order amount can significantly skew the data distribution and greatly impact the average. **In this situation, we can either drop the large order amount, or use median instead of mean to describe the order value.** To make the desicion, let's take a closer look at what causes the order amount to be so large. If the causes are not reasonable, we should drop the data. Otherwise we should still maintain the data and choose to use median value for our analysis.

[35]:
```
df_big_order_amount = df_raw[df_raw['order_amount'] > 10000]
print("Total big order amount: {}".
 ↪format(len(df_big_order_amount['order_amount'])))
print("Big order amount: ")
print(df_big_order_amount['order_amount'].value_counts())
```

```
Total big order amount: 63
Big order amount:
25725     19
704000    17
51450     16
77175      9
102900     1
154350     1
Name: order_amount, dtype: int64
```

[36]:
```
df_big_order_amount[df_big_order_amount['order_amount'] == 25725]
```

[36]:

	order_id	shop_id	user_id	order_amount	total_items	payment_method	created_at
160	161	78	990	25725	1	credit_card	2017-03-12 5:56:57
1056	1057	78	800	25725	1	debit	2017-03-15 10:16:45
1193	1194	78	944	25725	1	debit	2017-03-16 16:38:26
1204	1205	78	970	25725	1	credit_card	2017-03-17 22:32:21
1384	1385	78	867	25725	1	cash	2017-03-17 16:38:06
1419	1420	78	912	25725	1	cash	2017-03-30 12:23:43
1452	1453	78	812	25725	1	credit_card	2017-03-17 18:09:54
2270	2271	78	855	25725	1	credit_card	2017-03-14 23:58:22
2548	2549	78	861	25725	1	cash	2017-03-17 19:36:00
2773	2774	78	890	25725	1	cash	2017-03-26 10:36:43
2922	2923	78	740	25725	1	debit	2017-03-12 20:10:58
3085	3086	78	910	25725	1	cash	2017-03-26 1:59:27
3151	3152	78	745	25725	1	credit_card	2017-03-18 13:13:07
3440	3441	78	982	25725	1	debit	2017-03-19 19:02:54
3780	3781	78	889	25725	1	cash	2017-03-11 21:14:50
4040	4041	78	852	25725	1	cash	2017-03-02 14:31:12
4505	4506	78	866	25725	1	debit	2017-03-22 22:06:01
4584	4585	78	997	25725	1	cash	2017-03-25 21:48:44
4918	4919	78	823	25725	1	cash	2017-03-15 13:26:46

From the above table, it is very interesting to see that the shop No.78 sells its sneakers at a price of $25725!! Although there exists shoes that cost thousands of dollars, they are usually shoes either with limited edition or with special values. **Generally speaking, this data does not seem**

normal so might worth more digging to ensure there is no mistake of how the data was acquired and processed. To check if others have similar abnormal unit prices, we can add a column to list the unit prices for each sneaker.

[41]:
```
df_big_order_amount.loc[:, 'unit_price'] = df_big_order_amount.loc[:,␣
↪'order_amount'] / df_big_order_amount.loc[:, 'total_items']
```

Interestingly, after a further investigation, we can see that most of the big order amount (46 out of 63) is caused by this expensive sneaker sold by shop No. 78:

[49]:
```
print("Total expensive sneakers: {}".
↪format(len(df_big_order_amount[df_big_order_amount['unit_price']== 25725.
↪0])))
df_big_order_amount[df_big_order_amount['unit_price'] == 25725.0]
```

```
Total expensive sneakers: 46
```

[49]:

	order_id	shop_id	user_id	order_amount	total_items	payment_method	\
160	161	78	990	25725	1	credit_card	
490	491	78	936	51450	2	debit	
493	494	78	983	51450	2	cash	
511	512	78	967	51450	2	cash	
617	618	78	760	51450	2	cash	
691	692	78	878	154350	6	debit	
1056	1057	78	800	25725	1	debit	
1193	1194	78	944	25725	1	debit	
1204	1205	78	970	25725	1	credit_card	
1259	1260	78	775	77175	3	credit_card	
1384	1385	78	867	25725	1	cash	
1419	1420	78	912	25725	1	cash	
1452	1453	78	812	25725	1	credit_card	
1529	1530	78	810	51450	2	cash	
2270	2271	78	855	25725	1	credit_card	
2452	2453	78	709	51450	2	cash	
2492	2493	78	834	102900	4	debit	
2495	2496	78	707	51450	2	cash	
2512	2513	78	935	51450	2	debit	
2548	2549	78	861	25725	1	cash	
2564	2565	78	915	77175	3	debit	
2690	2691	78	962	77175	3	debit	
2773	2774	78	890	25725	1	cash	
2818	2819	78	869	51450	2	debit	
2821	2822	78	814	51450	2	cash	
2906	2907	78	817	77175	3	debit	
2922	2923	78	740	25725	1	debit	
3085	3086	78	910	25725	1	cash	
3101	3102	78	855	51450	2	credit_card	
3151	3152	78	745	25725	1	credit_card	

3167	3168	78	927	51450	2	cash
3403	3404	78	928	77175	3	debit
3440	3441	78	982	25725	1	debit
3705	3706	78	828	51450	2	credit_card
3724	3725	78	766	77175	3	credit_card
3780	3781	78	889	25725	1	cash
4040	4041	78	852	25725	1	cash
4079	4080	78	946	51450	2	cash
4192	4193	78	787	77175	3	credit_card
4311	4312	78	960	51450	2	debit
4412	4413	78	756	51450	2	debit
4420	4421	78	969	77175	3	debit
4505	4506	78	866	25725	1	debit
4584	4585	78	997	25725	1	cash
4715	4716	78	818	77175	3	debit
4918	4919	78	823	25725	1	cash

	created_at	unit_price
160	2017-03-12 5:56:57	25725.0
490	2017-03-26 17:08:19	25725.0
493	2017-03-16 21:39:35	25725.0
511	2017-03-09 7:23:14	25725.0
617	2017-03-18 11:18:42	25725.0
691	2017-03-27 22:51:43	25725.0
1056	2017-03-15 10:16:45	25725.0
1193	2017-03-16 16:38:26	25725.0
1204	2017-03-17 22:32:21	25725.0
1259	2017-03-27 9:27:20	25725.0
1384	2017-03-17 16:38:06	25725.0
1419	2017-03-30 12:23:43	25725.0
1452	2017-03-17 18:09:54	25725.0
1529	2017-03-29 7:12:01	25725.0
2270	2017-03-14 23:58:22	25725.0
2452	2017-03-27 11:04:04	25725.0
2492	2017-03-04 4:37:34	25725.0
2495	2017-03-26 4:38:52	25725.0
2512	2017-03-18 18:57:13	25725.0
2548	2017-03-17 19:36:00	25725.0
2564	2017-03-25 1:19:35	25725.0
2690	2017-03-22 7:33:25	25725.0
2773	2017-03-26 10:36:43	25725.0
2818	2017-03-17 6:25:51	25725.0
2821	2017-03-02 17:13:25	25725.0
2906	2017-03-16 3:45:46	25725.0
2922	2017-03-12 20:10:58	25725.0
3085	2017-03-26 1:59:27	25725.0
3101	2017-03-21 5:10:34	25725.0

```
3151  2017-03-18 13:13:07     25725.0
3167  2017-03-12 12:23:08     25725.0
3403   2017-03-16 9:45:05     25725.0
3440  2017-03-19 19:02:54     25725.0
3705  2017-03-14 20:43:15     25725.0
3724  2017-03-16 14:13:26     25725.0
3780  2017-03-11 21:14:50     25725.0
4040  2017-03-02 14:31:12     25725.0
4079  2017-03-20 21:14:00     25725.0
4192   2017-03-18 9:25:32     25725.0
4311   2017-03-01 3:02:10     25725.0
4412   2017-03-02 4:13:39     25725.0
4420  2017-03-09 15:21:35     25725.0
4505  2017-03-22 22:06:01     25725.0
4584  2017-03-25 21:48:44     25725.0
4715   2017-03-05 5:10:44     25725.0
4918  2017-03-15 13:26:46     25725.0
```

While the rest of the big order amount is caused by large total items orderd in each order from shop No.42:

[51]:
```
df_big_order_amount[df_big_order_amount['unit_price'] != 25725.0]
```

[51]:
```
      order_id  shop_id  user_id  order_amount  total_items payment_method  \
15          16       42      607        704000         2000    credit_card
60          61       42      607        704000         2000    credit_card
520        521       42      607        704000         2000    credit_card
1104      1105       42      607        704000         2000    credit_card
1362      1363       42      607        704000         2000    credit_card
1436      1437       42      607        704000         2000    credit_card
1562      1563       42      607        704000         2000    credit_card
1602      1603       42      607        704000         2000    credit_card
2153      2154       42      607        704000         2000    credit_card
2297      2298       42      607        704000         2000    credit_card
2835      2836       42      607        704000         2000    credit_card
2969      2970       42      607        704000         2000    credit_card
3332      3333       42      607        704000         2000    credit_card
4056      4057       42      607        704000         2000    credit_card
4646      4647       42      607        704000         2000    credit_card
4868      4869       42      607        704000         2000    credit_card
4882      4883       42      607        704000         2000    credit_card

              created_at  unit_price
15    2017-03-07 4:00:00       352.0
60    2017-03-04 4:00:00       352.0
520   2017-03-02 4:00:00       352.0
1104  2017-03-24 4:00:00       352.0
```

```
1362  2017-03-15 4:00:00       352.0
1436  2017-03-11 4:00:00       352.0
1562  2017-03-19 4:00:00       352.0
1602  2017-03-17 4:00:00       352.0
2153  2017-03-12 4:00:00       352.0
2297  2017-03-07 4:00:00       352.0
2835  2017-03-28 4:00:00       352.0
2969  2017-03-28 4:00:00       352.0
3332  2017-03-24 4:00:00       352.0
4056  2017-03-28 4:00:00       352.0
4646  2017-03-02 4:00:00       352.0
4868  2017-03-22 4:00:00       352.0
4882  2017-03-25 4:00:00       352.0
```

Large total items in each order is not a very abnormal situation, as it might just because the shop is popular. Thus we should maintain these order amount values. Considering that some of the large order amount values have to be maintained (which will still skew the distribution), **it is thus better to use median value instead of mean value for the order value analysis. In addition, we should check if there is any mistake in the data acquisition process for shop No. 78.**

3.0.1 Calculate the median order value (MOV)

[53]:
```
mov = df_raw['order_amount'].median()
print("The median order value is: ${}".format(mov))
```

```
The median order value is: $284.0
```