Shopify Challenge

August 23, 2017

1 Shopify Data Analytics Development Problem

100 shops and every shop sells a unique model of shoe. We want to find out the Average Order Value for a given window of 30 days. Let's see what we can do.

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
```

Starting off with importing the necessary libraries and the given data.

```
In [2]: shop_data = pd.read_excel('desktop/data.xlsx')
```

```
In [3]: shop_data.columns.values
```

```
Out[3]: array(['order_id', 'shop_id', 'user_id', 'order_amount', 'total_items',
               'payment_method', 'created_at'], dtype=object)
```

```
In [4]: shop_data.describe()
```

```
Out[4]:           order_id      shop_id      user_id   order_amount  total_items
        count  5000.000000  5000.000000  5000.000000    5000.000000   5000.00000
        mean   2500.500000    50.078800   849.092400    3145.128000      8.78720
        std    1443.520003    29.006118    87.798982   41282.539349    116.32032
        min       1.000000     1.000000   607.000000      90.000000      1.00000
        25%    1250.750000    24.000000   775.000000     163.000000      1.00000
        50%    2500.500000    50.000000   849.000000     284.000000      2.00000
        75%    3750.250000    75.000000   925.000000     390.000000      3.00000
        max    5000.000000   100.000000   999.000000  704000.000000   2000.00000
```

Using describe, we get a summary of the dataset's distribution. We can also observe that 'payment_method' and 'created_at' are not included due their column values not being numerical.

```
In [5]: shop_data['order_amount'].mean()
```

```
Out[5]: 3145.128
```

As we can see, the given AOV was calculated by taking the mean 'order_amount' of the dataset.

Right off the bat there are a few things that we are able to notice. Since we are looking at AOV, 'order_amount' will be the column we will want to keep an eye on.

We should also keep in mind that Average Order Value is different from Average Item Value. Since AOV can include more than one item purchased, it is not unreasonable to have a number that is slightly bigger than the average market value of a sneaker.

```
In [6]: shop_data['order_amount'].describe()
```

```
Out[6]: count      5000.000000
        mean       3145.128000
        std       41282.539349
        min          90.000000
        25%         163.000000
        50%         284.000000
        75%         390.000000
        max      704000.000000
        Name: order_amount, dtype: float64
```

We can see, however, that the Max of 'order_amount' is disproportionate, thus skewing the data far to the right. This will be something we will look to fix when cleaning the data.

Before we begin the cleaning, we will be adding a new column, 'per_item', which will give us a per item cost. This will help us better understand the dataset by allowing us (in the end) to not only see AOV, but also Average Item Value.

```
In [7]: shop_data['per_item'] = np.divide(shop_data.order_amount, shop_data.total_items)
```

Now, we can take another look with describe.

```
In [8]: shop_data.describe()
```

```
Out[8]:           order_id      shop_id      user_id   order_amount  total_items  \
        count  5000.000000  5000.000000  5000.000000    5000.000000   5000.00000
        mean   2500.500000    50.078800   849.092400    3145.128000      8.78720
        std    1443.520003    29.006118    87.798982   41282.539349    116.32032
        min       1.000000     1.000000   607.000000      90.000000      1.00000
        25%    1250.750000    24.000000   775.000000     163.000000      1.00000
        50%    2500.500000    50.000000   849.000000     284.000000      2.00000
        75%    3750.250000    75.000000   925.000000     390.000000      3.00000
        max    5000.000000   100.000000   999.000000  704000.000000   2000.00000

                  per_item
        count   5000.000000
        mean     387.742800
        std     2441.963725
        min       90.000000
        25%      133.000000
        50%      153.000000
        75%      169.000000
        max    25725.000000
```

After adding in the column, we can see that the mean 'per_item' cost is $387.74.

For now, let's start working with the 'per_item' cost a bit. Since 'order_amount' is very much dependent on the number of items bought, it is harder to gauge what an ideal 'order_amount' would be. What is easier to evaluate is the 'per_item' cost. Since we know a sneaker is around the $150 mark, we can start by trimming down the mean of 'per_item'.

For our next step, we are going to make a quick initial assumption. In my opinion, anything past the $500 mark is nearing the 'damn, that's some rare hypebeast Yeezys' type of shoes. In other words, I feel that anything over the 'per_item' price of $500 might prove to be an outlier.

```
In [9]: shop_data.query('per_item > 500')
```

Out[9]:

	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	per_item
160	2017-03-12 05:56:57	25725.0
490	2017-03-26 17:08:19	25725.0
493	2017-03-16 21:39:35	25725.0
511	2017-03-09 07:23:14	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 09:27:20	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 07:12:01	25725.0
2270	2017-03-14 23:58:22	25725.0
2452	2017-03-27 11:04:04	25725.0
2492	2017-03-04 04:37:34	25725.0
2495	2017-03-26 04:38:52	25725.0
2512	2017-03-18 18:57:13	25725.0
2548	2017-03-17 19:36:00	25725.0
2564	2017-03-25 01:19:35	25725.0
2690	2017-03-22 07:33:25	25725.0
2773	2017-03-26 10:36:43	25725.0
2818	2017-03-17 06:25:51	25725.0
2821	2017-03-02 17:13:25	25725.0
2906	2017-03-16 03:45:46	25725.0
2922	2017-03-12 20:10:58	25725.0
3085	2017-03-26 01:59:27	25725.0
3101	2017-03-21 05:10:34	25725.0
3151	2017-03-18 13:13:07	25725.0
3167	2017-03-12 12:23:08	25725.0
3403	2017-03-16 09:45:05	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 09:25:32   25725.0
4311 2017-03-01 03:02:10   25725.0
4412 2017-03-02 04:13:39   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 05:10:44   25725.0
4918 2017-03-15 13:26:46   25725.0
```

There we have it. Immidiately we can see that we ran into something odd. It seems like 'shop_id' 78 is selling shoes made out of diamonds. At $25725 per item, this is definitely an outlier in our data.

We should make it a point to show this data to the appropriate team/manager, but for now, let's remove it and keep on going.

```
In [10]: shop_data_v2 = shop_data[shop_data.shop_id != 78]

In [11]: shop_data_v2.describe()
```

Out[11]:

	order_id	shop_id	user_id	order_amount	total_items	per_item
count	4954.000000	4954.000000	4954.000000	4954.000000	4954.000000	4954.000000
mean	2498.990916	49.819540	848.919257	2717.367784	8.851029	152.475575
std	1444.498907	29.014845	87.846007	41155.996469	116.857286	31.260218
min	1.000000	1.000000	607.000000	90.000000	1.000000	90.000000
25%	1248.250000	24.000000	775.000000	163.000000	1.000000	132.000000
50%	2494.500000	50.000000	849.000000	284.000000	2.000000	153.000000
75%	3750.750000	74.000000	925.000000	390.000000	3.000000	168.000000
max	5000.000000	100.000000	999.000000	704000.000000	2000.000000	352.000000

After the removal of 'shop_id' 78, we are much closer to an actual AOV. With the AOV at $2717.37, we can also see that the 'per_item' price also accurately reflects the average price for a quality sneaker. The Max of 'order_amount', 'total_items', and 'per_item' are still high. Let's see if we can clean them up in the next steps.

One thing that we have not yet checked for in the dataset is if there are any duplicate data. This might be one of the reasons for the data skew.

```
In [12]: dupes = shop_data_v2[shop_data_v2.duplicated(['created_at', 'shop_id'],keep = False)]

         dupes.sort_values(['created_at'], ascending = True)
```

```
Out[12]:       order_id  shop_id  user_id  order_amount  total_items payment_method  \
520         521       42      607        704000         2000    credit_card
4646       4647       42      607        704000         2000    credit_card
15           16       42      607        704000         2000    credit_card
2297       2298       42      607        704000         2000    credit_card
1104       1105       42      607        704000         2000    credit_card
3332       3333       42      607        704000         2000    credit_card
2835       2836       42      607        704000         2000    credit_card
2969       2970       42      607        704000         2000    credit_card
4056       4057       42      607        704000         2000    credit_card

                created_at  per_item
520  2017-03-02 04:00:00     352.0
4646 2017-03-02 04:00:00     352.0
15   2017-03-07 04:00:00     352.0
2297 2017-03-07 04:00:00     352.0
1104 2017-03-24 04:00:00     352.0
3332 2017-03-24 04:00:00     352.0
2835 2017-03-28 04:00:00     352.0
2969 2017-03-28 04:00:00     352.0
4056 2017-03-28 04:00:00     352.0
```

From the data shown, we can see that there are indeed duplicates for 'shop_id' 42. There are two different conclusions that this data may lead to. Since we do not have more data to further validate, the most we can do right now is to consider the different conclusions.

- one: these are indeed duplicates, and thus one entry has to be removed
- two: these are bulk orders, but the order cap is set at 2000. This forces the buyer to make multiple 2000 item orders

After evaluating the two options, personally, I find that this must be a bulk order from the user. However, we will still consider these transactions as outliers, as the 'per_item' cost and 'total_items' count are too far out from our mean (it is actually our Max value for both 'total_items' and 'per_item' cost).

```
In [13]: shop_data_v3 = shop_data_v2[shop_data_v2.shop_id != 42]

In [14]: shop_data_v3.describe()

Out[14]:           order_id      shop_id      user_id  order_amount  total_items  \
count  4903.000000  4903.000000  4903.000000   4903.000000  4903.000000
mean   2499.584540    49.900877   849.858862    300.155823     1.995717
std    1444.221163    29.154367    86.887947    155.941112     0.982602
min       1.000000     1.000000   700.000000     90.000000     1.000000
25%    1246.500000    24.000000   776.000000    163.000000     1.000000
50%    2499.000000    50.000000   850.000000    284.000000     2.000000
75%    3750.500000    74.000000   925.000000    386.500000     3.000000
max    5000.000000   100.000000   999.000000   1086.000000     8.000000
```

```
                per_item
count  4903.000000
mean    150.400163
std      23.851202
min      90.000000
25%     132.000000
50%     153.000000
75%     166.000000
max     201.000000
```

```
In [15]: shop_data_v3.std()
```

```
Out[15]: order_id        1444.221163
         shop_id           29.154367
         user_id           86.887947
         order_amount     155.941112
         total_items        0.982602
         per_item          23.851202
         dtype: float64
```

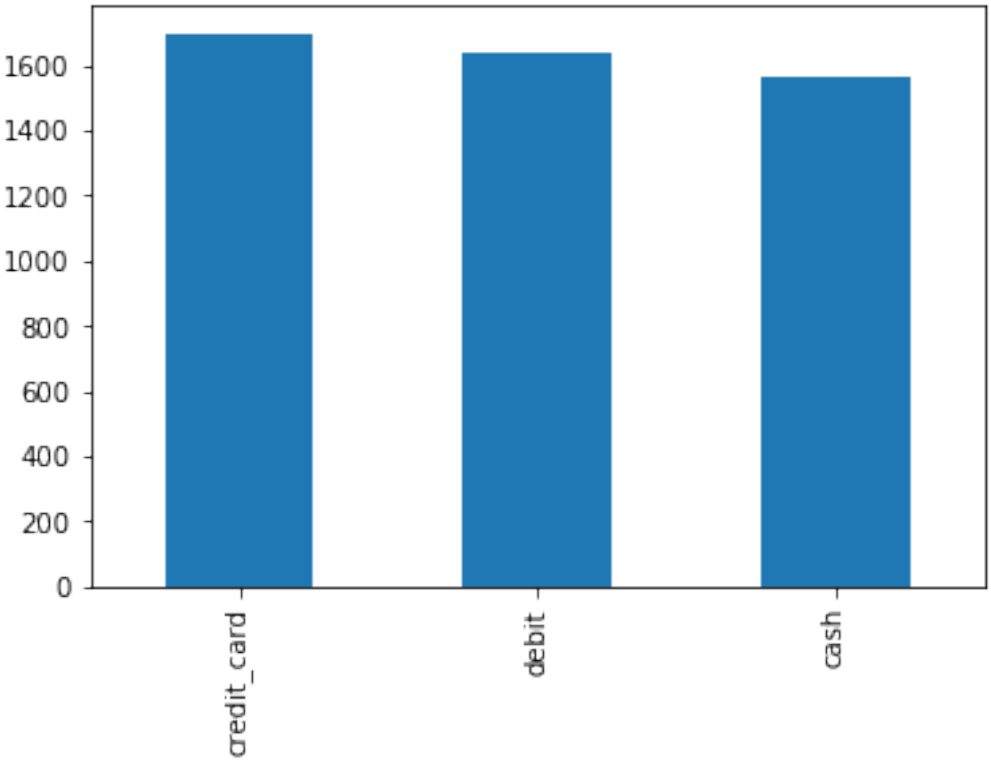
Now that we have removed 'shop_id' 48, our Max value for 'order_amount', 'total_items', and 'per_item' is no longer skewed to the right.

For our final step, let's check if 'payment_method' affects the AOV in any significant way.

```
In [16]: pay = shop_data_v3['payment_method'].value_counts()
```

```
In [17]: print(shop_data_v3['payment_method'].describe())
         pay.plot.bar()
         plt.show()
         pay
```

```
count            4903
unique              3
top       credit_card
freq             1698
Name: payment_method, dtype: object
```




```
Out[17]: credit_card    1698
         debit          1638
         cash           1567
         Name: payment_method, dtype: int64

In [18]: credit = shop_data_v3[shop_data_v3.payment_method == 'credit_card']
         debit = shop_data_v3[shop_data_v3.payment_method == 'debit']
         cash = shop_data_v3[shop_data_v3.payment_method == 'cash']

In [19]: credit.mean()

Out[19]: order_id        2505.371614
         shop_id           49.960542
         user_id          849.382803
         order_amount     299.873380
         total_items        1.991166
         per_item         150.553004
         dtype: float64

In [20]: debit.mean()

Out[20]: order_id        2472.727717
         shop_id           49.179487
```

```
user_id          850.199634
order_amount     305.625763
total_items        2.028694
per_item         150.700244
dtype: float64
```

```
In [21]: cash.mean()
```

```
Out[21]: order_id        2521.387364
         shop_id           50.590300
         user_id          850.018507
         order_amount     294.744097
         total_items        1.966177
         per_item         149.920868
         dtype: float64
```

After plotting and segmenting our data by payment method, we can see that there is actualy no major influence of AOV by this variable. The AOV ranges from $294.74 to $305.623. All three payment methods are used almost equally (with credit being on top), every order consists (on average) of two items, and the average per item price roughly stands at $150 for all three methods.

2 Conclusion

In conclusion, the final AOV should be set to $300.16. We have reached this price by removing the outliers within the dataset, while also keeping most of the data to maintain sample size (4903 out of the initial 5000).

We have also concluded that payment method does not play a big role in affecting the adjustment of the AOV. Also, we have found two stores that we should further investigate (store 78 and 42), as they have proved to be extreme outliers.