Shopify Summer 2022 Data Science Intern Challenge

Harry Qu

1 Question 1

I will use the pandas Python library as my tool of choice to tackle this sneaker data. We begin by loading in the data and taking a quick look at some summary statistics.

[1]:
```
import pandas as pd
```

[2]:
```
shoes = pd.read_csv("shoes.csv")
shoes.shape
```

[2]:
```
(5000, 7)
```

[3]:
```
shoes.describe()
```

[3]:

	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

The mean order amount is indeed $3145.13, so our calculation of Average Order Value (AOV) was not computationally wrong, but perhaps it is a little misleading. As noted, sneakers are a relatively affordable item. I hear a lot on the news nowadays about inflation but this is rather extreme! The mean is significantly higher than the median, suggesting the data is right-skewed. Chances are, there is a small proportion of orders with extremely large order amounts. In fact, the max order amount of 704000 and max total items of 2000 is immediately suspicious. We will take a quick look at some of the largest order amounts:

[4]:
```
shoes.sort_values(by=['order_amount'],ascending=False).head()
```

[4]:

	order_id	shop_id	user_id	order_amount	total_items	payment_method	created_at
2153	2154	42	607	704000	2000	credit_card	2017-03-12 4:00:00
3332	3333	42	607	704000	2000	credit_card	2017-03-24 4:00:00
520	521	42	607	704000	2000	credit_card	2017-03-02 4:00:00
1602	1603	42	607	704000	2000	credit_card	2017-03-17 4:00:00
60	61	42	607	704000	2000	credit_card	2017-03-04 4:00:00

It appears as though user 607 is the person from those math problems we got in elementary school, buying 2000 pairs of shoes in a single order. We will take a closer look at this user.

[5]:
```
shoes[shoes['user_id'] == 607].sort_values(by=['created_at']).head(10)
```

[5]:

	order_id	shop_id	user_id	order_amount	total_items	payment_method	created_at
520	521	42	607	704000	2000	credit_card	2017-03-02 4:00:00
4646	4647	42	607	704000	2000	credit_card	2017-03-02 4:00:00
60	61	42	607	704000	2000	credit_card	2017-03-04 4:00:00
15	16	42	607	704000	2000	credit_card	2017-03-07 4:00:00
2297	2298	42	607	704000	2000	credit_card	2017-03-07 4:00:00
1436	1437	42	607	704000	2000	credit_card	2017-03-11 4:00:00
2153	2154	42	607	704000	2000	credit_card	2017-03-12 4:00:00
1362	1363	42	607	704000	2000	credit_card	2017-03-15 4:00:00
1602	1603	42	607	704000	2000	credit_card	2017-03-17 4:00:00
1562	1563	42	607	704000	2000	credit_card	2017-03-19 4:00:00

User 607 is quite anomalous, ordering precisely at 4:00:00, sometimes with 2 orders in a single day. This anomaly may be worth investigating - perhaps it is fraud or erroneous - but that will be left for another day. For now we will exclude user 607 from our analysis and go back to looking at the

largest order amounts.

[6]:
```
shoes[shoes['user_id'] != 607].sort_values(by=['order_amount'],ascending=False).
↪head()
```

[6]:
```
      order_id  shop_id  user_id  order_amount  total_items payment_method  \
691        692       78      878        154350            6          debit
2492      2493       78      834        102900            4          debit
3724      3725       78      766         77175            3    credit_card
1259      1260       78      775         77175            3    credit_card
4420      4421       78      969         77175            3          debit

                created_at
691   2017-03-27 22:51:43
2492   2017-03-04 4:37:34
3724  2017-03-16 14:13:26
1259   2017-03-27 9:27:20
4420  2017-03-09 15:21:35
```

It appears that shop 78 is selling sneakers at $25725 a pair, so they are either signed by Justin Bieber or made of solid gold. Let us exclude shop 78 for now as well.

[7]:
```
shoes[(shoes['user_id'] != 607) & (shoes['shop_id'] != 78)].
↪sort_values(by=['order_amount'],ascending=False).head()
```

[7]:
```
      order_id  shop_id  user_id  order_amount  total_items payment_method  \
1364      1365       42      797          1760            5           cash
1367      1368       42      926          1408            4           cash
1471      1472       42      907          1408            4          debit
3538      3539       43      830          1086            6          debit
4141      4142       54      733          1064            8          debit

                created_at
1364   2017-03-10 6:28:21
1367   2017-03-13 2:38:34
1471  2017-03-12 23:00:22
3538  2017-03-17 19:56:29
4141  2017-03-07 17:05:18
```

These look a little more reasonable! Let's take a quick look at this reduced dataset.

[8]:
```
shoes_reduced = shoes[(shoes['user_id'] != 607) & (shoes['shop_id'] != 78)]
shoes_reduced.shape
```

[8]:
```
(4937, 7)
```

[9]:
```
shoes_reduced.describe()
```

[9]:

	order_id	shop_id	user_id	order_amount	total_items
count	4937.000000	4937.000000	4937.000000	4937.000000	4937.000000
mean	2499.551347	49.846465	849.752279	302.580514	1.994734
std	1444.069407	29.061131	86.840313	160.804912	0.982821
min	1.000000	1.000000	700.000000	90.000000	1.000000
25%	1248.000000	24.000000	775.000000	163.000000	1.000000
50%	2497.000000	50.000000	850.000000	284.000000	2.000000
75%	3751.000000	74.000000	925.000000	387.000000	3.000000
max	5000.000000	100.000000	999.000000	1760.000000	8.000000

The AOV of the reduced dataset is $302.58 which seems more reasonable. However, AOV doesn't need to be the metric we use to evaluate the dataset to begin with! Another simple metric we can report to help us understand the sizes of orders is the median order value, which I suppose could be called MOV. The MOV was seen in the summary at the beginning, and its value is $284. Interesting to note is that our reduced dataset shows the same MOV, highlighting that MOV as a metric is more resistant to outliers than AOV is. Finally, our analysis with the reduced dataset gives another interesting metric. Our reduced dataset has size 4937 as seen above. This represents $4937/5000 = 98.74\%$ of the data, i.e. with the top 1.26% of the order values removed. This gives us a metric of the 98th percentile or value-at-risk. By looking at the max of our reduced dataset, we can report that 98% of orders are below 1760.

2 Question 2

Part a: to find the number of orders shipped by Speedy Express, we join the table of all the orders (which includes the shipper ID for each order) to the table of shippers in order to map each shipper ID to the name of the shipper. This creates a table of every order, with the name of the shipper of each order included. Then, we simply count the entries that have shipper name Speedy Express. The query is below:

[ ]:
```
SELECT COUNT(*) FROM
        (SELECT o.OrderID, s.ShipperName
        FROM Orders AS o INNER JOIN Shippers AS s ON o.ShipperID = s.ShipperID)
WHERE ShipperName = 'Speedy Express';
```

The answer is 54.

Part b: to find the last name of the employee with the most orders, we join the table of all the orders (which includes the employee ID for each order) to the table of employees in order to map each employee ID to the last name of the employee. This creates a table of every order, with the name of the employee of each order included. So far, a similar approach to part a! Next, we select the count of of entries associated with each last name. This creates a table of each employee last name and the number of orders associated with them. Finally, we select the max order count from the table to find the employee with the most orders. The query is below:

[ ]:
```
SELECT MAX(OrderCount), LastName FROM
        (SELECT COUNT(OrderID) AS OrderCount, LastName FROM
            (SELECT o.OrderID, e.LastName
                FROM Orders AS o
                INNER JOIN Employees AS e ON o.EmployeeID = e.EmployeeID)
    GROUP BY LastName);
```

The answer is Peacock.

Part c: We once again use our mapping strategy, since names are much more fun to work with than just ID numbers. We will take the order details table, which contains a breakdown of products ordered in each order. This table contains the order ID as well as the product ID for each entry. Using the order ID, we join this with the orders table to find the customer ID who placed the order. Using the product ID, we join this the product table to get the names of the products. Now we have a table of each every product ordered with the customer ID, product name, and quantity ordered. We join this with the customer table in order to match each customer ID to the country it is ordered from. Now we have a table of each every product ordered with the customer country, product name, and quantity ordered. From this we select only the rows where the country is Germany, and sum the quantities for each product via grouping by the product name. Now we have a table of each product and the quantity of it ordered from customers in Germany. Finally we select the max quantity to find the product most ordered by customers in Germany. The query is below:

[ ]:
```
SELECT ProductName, MAX(GermanQuantity) FROM
  (SELECT odp.ProductName, SUM(odp.Quantity) AS GermanQuantity FROM
      (SELECT od.OrderDetailID, o.CustomerID, p.ProductName, od.Quantity
      FROM ((OrderDetails AS od
      INNER JOIN Orders AS o ON od.OrderID = o.OrderID)
      INNER JOIN Products AS p ON od.ProductID = p.ProductID)) AS odp
  INNER JOIN Customers AS c ON odp.CustomerID = c.CustomerID
  WHERE c.Country = 'Germany'
  GROUP BY odp.ProductName);
```

The answer is Boston Crab Meat. Delicious!