Shopify Data Science Intern Challenge
Muhammad Sohail

1. Background: I downloaded the Google Spreadsheet dataset as a .csv file and imported it into a software I could use to run SQL queries (to make life easier).

    a) Running a simple average query on the order_amount column shows that the value $3,145.13 is the total revenue divided by the total number of orders—across all 100 shops.

    To better understand why this value is absurd, we can try to determine the AOV for each shop, and view them in a descending order (i.e., maybe some shop sells very expensive collectible sneakers!). For this, I ran the following query

```
SELECT
    shop_id,
    ROUND(AVG(order_amount) 2) AS AOV
FROM
    Data
GROUP BY
    shop_id
ORDER BY
    AOV DESC
```

And the following screenshot shows the first few results

shop_id	AOV
Search	Search
42	235101.49
78	49213.04
50	403.55
90	403.22
38	390.86
81	384
6	383.51
89	379.15
33	376.27
51	361.8
59	358.97
11	356.73
88	355.52

[1] https://www.shopify.ca/blog/average-order-value

And so, as can be seen, shops 42 and 78 greatly sway the overall AOV, whereas the remaining shops tend to have more reasonable values.

To get a better idea of *why* these values were so high, I ran a quick select query on the table, first restricting it to orders at shop 42 then at shop 78, and got the following

order_id	shop_id	user_id	order_amount	total_items	payment_method▲	created_at
Search	Search	Search	Search	Search	Search	Search
4295	42	859	704	2	cash	2017-03-24 20:50:40
4232	42	962	352	1	cash	2017-03-04 0:01:19
2988	42	819	1056	3	cash	2017-03-03 9:09:25
2004	42	934	704	2	cash	2017-03-26 9:21:26
1912	42	739	704	2	cash	2017-03-07 5:42:52
1368	42	926	1408	4	cash	2017-03-13 2:38:34
1365	42	797	1760	5	cash	2017-03-10 6:28:21
836	42	819	704	2	cash	2017-03-09 14:15:15
835	42	792	352	1	cash	2017-03-25 21:31:25
4883	42	607	704000	2000	credit_card	2017-03-25 4:00:00
4869	42	607	704000	2000	credit_card	2017-03-22 4:00:00
4768	42	720	704	2	credit_card	2017-03-14 10:26:08
4647	42	607	704000	2000	credit_card	2017-03-02 4:00:00

order_id	shop_id	user_id	order_amount	total_items	payment_method▲	created_at
Search	Search	Search	Search	Search	Search	Search
161	78	990	25725	1	credit_card	2017-03-12 5:56:57
491	78	936	51450	2	debit	2017-03-26 17:08:19
494	78	983	51450	2	cash	2017-03-16 21:39:35
512	78	967	51450	2	cash	2017-03-09 7:23:14
618	78	760	51450	2	cash	2017-03-18 11:18:42
692	78	878	154350	6	debit	2017-03-27 22:51:43
1057	78	800	25725	1	debit	2017-03-15 10:16:45
1194	78	944	25725	1	debit	2017-03-16 16:38:26
1205	78	970	25725	1	credit_card	2017-03-17 22:32:21
1260	78	775	77175	3	credit_card	2017-03-27 9:27:20
1385	78	867	25725	1	cash	2017-03-17 16:38:06
1420	78	912	25725	1	cash	2017-03-30 12:23:43
1453	78	812	25725	1	credit_card	2017-03-17 18:09:54

Immediately, the issue becomes apparent. For shop 42, it seems quite a few bulk orders were made, in the order of 2000 sneakers at a time. On the other hand, shop 72 either sells insanely expensive sneakers, or, more reasonably, the order amount was accidentally inputted in cents rather than dollars. In either case, we've found the outliers!

b) In a more general setting, where one is working with many datasets such as the one provided, nobody really has time to examine what's wrong with each and every outlier. In this general approach, the better metric to report would likely be one that simply ignores the outliers (i.e., using the $1.5 \times$ IQR method). For those curious, this approach yields an AOV of $300.16. However, I decided on a more holistic approach to determine the AOV. First, I updated all orders completed at shop 72 by dividing the order amount by 100

```
UPDATE
    Data
SET
    order_amount = order_amount / 100.0
WHERE
    shop_id = 78
```

Then, I proceeded by calculating the AOV for the entire data, *except*, that I restricted which orders to consider by the order_item column, taking only those with fewer than 10 items purchased. This accounted for the bulk orders by removing them from the examined dataset.

```
SELECT
    ROUND(AVG(order_amount), 2) as AOV
FROM
    Data
WHERE
    total_items <= 10
```

c) The value of the above query was $304.33, a much more reasonable value!

2.

a)

```
SELECT
    COUNT(*) as [Orders Shipped]
FROM
    Orders
INNER JOIN
    Shippers
ON
    Orders.ShipperID = Shippers.ShipperID
WHERE
    Shippers.ShipperName = "Speedy Express"
```

The above query returned that there were 54 orders shipped by Speedy Express.

b)

```
SELECT
    TOP 1 *
FROM
    (
        SELECT
            Employees.LastName,
            COUNT(Orders.OrderID) AS NumOfOrders
        FROM
            Orders
        INNER JOIN
            Employees
        ON
            Orders.EmployeeID = Employees.EmployeeID
        GROUP BY
            Employees.LastName
    )
ORDER BY
    NumOfOrders DESC
```

This query returned that the last name of the employee with the most orders was Peacock (40 orders).

c)

```
SELECT
    TOP 1 *
FROM
    (
        SELECT
            Products.ProductName,
            SUM(OrderDetails.Quantity) AS OrderCount
        FROM
            (
                (
                    OrderDetails
                    INNER JOIN
                        Orders
                    ON
                    OrderDetails.OrderID = Orders.OrderID
                )
                INNER JOIN
                    Customers
                ON
                    Customers.CustomerID = Orders.CustomerID
            )
            INNER JOIN
                Products
            ON
                Products.ProductID = OrderDetails.ProductID
        WHERE Customers.Country = "Germany"
        GROUP BY Products.ProductName
    )
ORDER BY
    OrderCount DESC
```

Note, for this query it was assumed that “ordered most” meant quantity and not number of times order. This query returned that Boston Crab Meat was the most ordered item by Germans, a grand total of 160 counts!