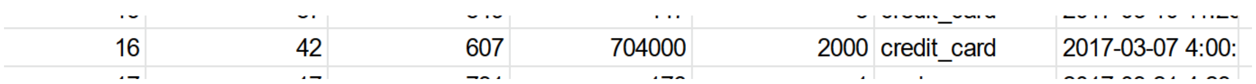
Shopify Data Science Intern Challenge – Summer 2022

Mark Harper
markjharper17@gmail.com | (905)-808-4008
1944 W 5TH AVE, Vancouver, BC

Question 1:

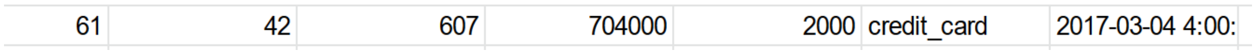
a. Think about what could be going wrong with our calculation. Think about a better way to evaluate this data.

The first thing that comes to mind when I notice any average value that is either significantly higher or lower than expected is to check for any data points that may be drastically skewing the value. In the *order_amount* column, I first browse for any obvious outliers. I immediately notice that entry #16 has an order amount significantly greater than the rest of the surrounding entries.



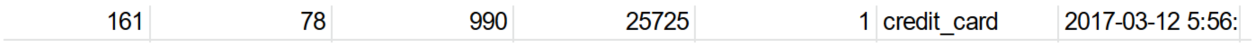
16	42	607	704000	2000	credit_card	2017-03-07 4:00:

It seems as though this order was for 2000 units and amounted to a total of $704,000.00. There could be a few explanations for this. Firstly, this could be a miss-entry. Perhaps the order amount was for 2 or 20 or even 200 pairs. However, further down the dataset I notice a similar entry. It appears as though the same user bought the same number of items from the same shop. However, this entry is on a different date and is under a different *order_id*. This leads me to believe these are not duplicate entries, and perhaps could signify that the user is a retailer and is ordering stock for their shop. These data points would require further investigation with the shop owner.



61	42	607	704000	2000	credit_card	2017-03-04 4:00:

A few more data points like above are also found elsewhere in the data set. Furthermore, it is possible there are miss-entered data points. For example, the data point below is an order for 1 item and supposedly cost the user $25,725.00.



161	78	990	25725	1	credit_card	2017-03-12 5:56:

Again, there could be multiple explanations for this. Perhaps the value of the order was actually $257.25, and the decimal point on the dollar amount was miss-entered. Another possible explanation however is that this shop sells a certain high-value, collectable sneaker. Myself not being a shoe collector I would not be too sure what a collectable shoe may sell for, however it would be worth a further investigation with the shop owner to determine what this sale was.

In conclusion, dirty data could be a result of the higher-than-expected average order value. However, assuming the data is clean, the 30-day average order value metric could be including one or a few of these much larger order values which is skewing the average.

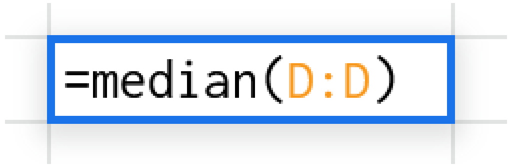
b. What metric would you report for this dataset?

The first metric that came to mind, as we were previously on the topic of average values, is we could report the average *order_amount* per shop and assess each shop individually. However, the dataset is only showing entries for the month of March 2017 and therefore may not shown enough data on each shop to be a valuable metric.

Ultimately, I believe we would like a metric that is less influenced by statistical outliers, yet still describes the data set well. I believe a better metric to report for this data would be the median order amount. The median is a metric that describes the middle value when a dataset is ordered in ascending or descending order. In the ordered dataset, any existing outliers will be appended to the start or the end of the dataset and therefore lengthen the list, but may only marginally affect the median/middle value.

c. What is its value?

To determine the median order amount, we can type the following into an empty cell.



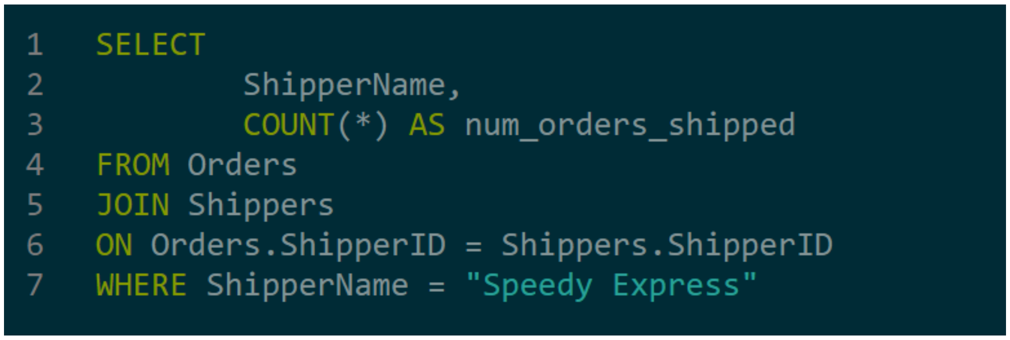
```
=median(D:D)
```

Therefore, for this dataset, the average *order_amount* is $284.00.

Question 2:

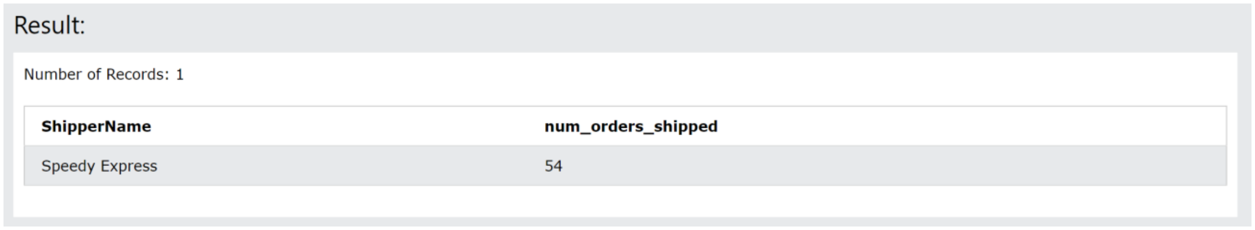
a. How many orders were shipped by Speedy Express in total?

From the *Orders* table, we are able to see the *ShipperID* for each order that was sent out. From the *Shippers* table, we can identify the name of each shipping company based on their unique *ShipperID*. The query below joins the *Orders* and *Shippers* tables on the key *ShipperID*. This will populate the Orders table with the name of the Shipper, as well as other information presented in the *Shippers* Table. We want to display the number of orders shipped by Speedy Express, therefore in our SELECT statement, we include the *ShipperName* and an expression to count the frequency of which each shipper shipped an order. We use the COUNT() function to do this and store the new variable as *num_orders_shipped*. Finally, in the WHERE clause, we specify that we are only looking for information where the *ShipperName* is equal to "Speedy Express".



```
SELECT
        ShipperName,
        COUNT(*) AS num_orders_shipped
FROM Orders
JOIN Shippers
ON Orders.ShipperID = Shippers.ShipperID
WHERE ShipperName = "Speedy Express"
```

Output:



Result:

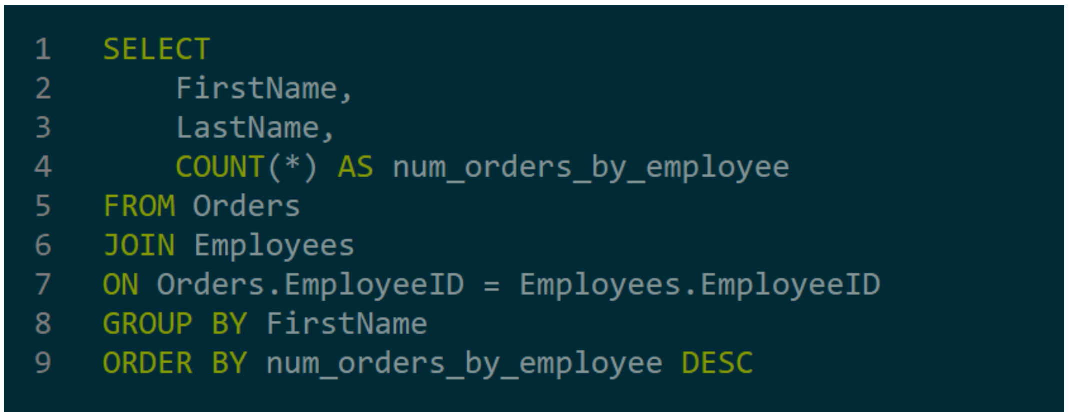
Number of Records: 1

ShipperName	num_orders_shipped
Speedy Express	54

From the output above the above query, it is determined that Speedy Express shipped 54 orders in total.

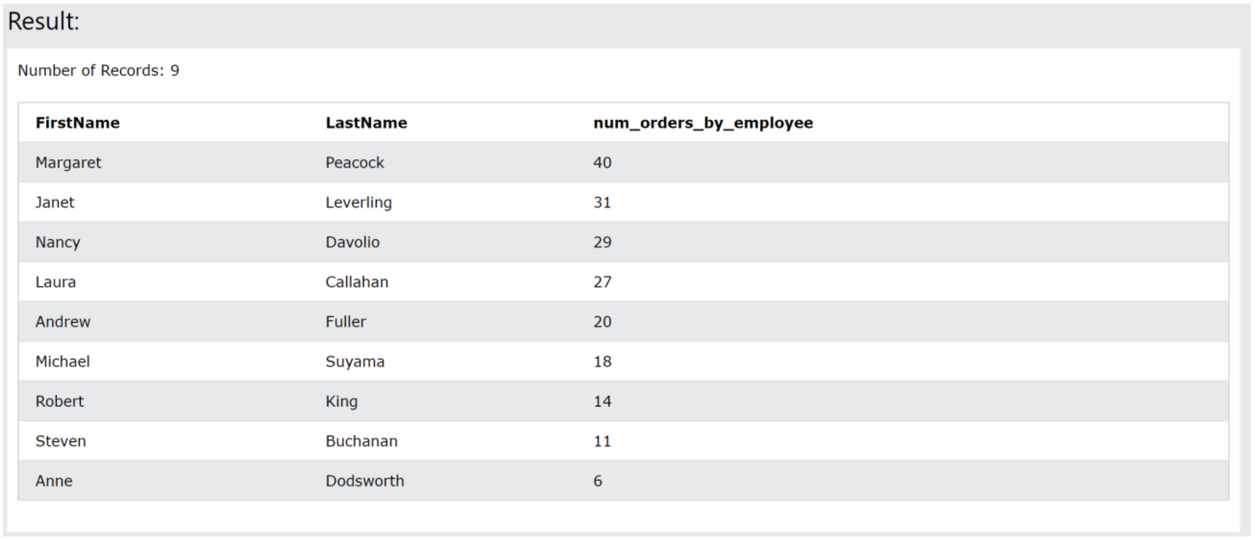
b. What is the last name of the employee with the most orders?

A similar thought process for question a) is applied here as well. Instead, we join the *Orders* and *Employees* tables on the *EmployeeID* key. want to summarize data per employee, therefore we include a GROUP BY clause to organize the data based on the employee's FirstName. Jumping back to the SELECT statement, we grab the *FirstName*, *LastName*, and frequency of which each employee completed an order by using the COUNT(*) function. The asterisk (*) indicates that we want to count the number of entries. We call this new variable *num_orders_by_employee*. Finally, we finish the query with an ORDER BY statement, to order by our new variable, *num_orders_by_employee* in descending order. This will present the employee with the most orders, and the number of orders they made in the first row of our query.



```sql
SELECT
    FirstName,
    LastName,
    COUNT(*) AS num_orders_by_employee
FROM Orders
JOIN Employees
ON Orders.EmployeeID = Employees.EmployeeID
GROUP BY FirstName
ORDER BY num_orders_by_employee DESC
```

Output:



Result:

Number of Records: 9

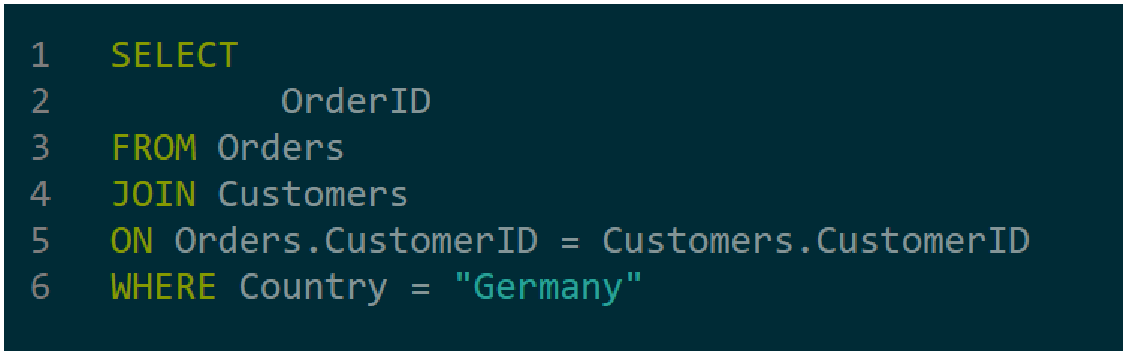
FirstName	LastName	num_orders_by_employee
Margaret	Peacock	40
Janet	Leverling	31
Nancy	Davolio	29
Laura	Callahan	27
Andrew	Fuller	20
Michael	Suyama	18
Robert	King	14
Steven	Buchanan	11
Anne	Dodsworth	6

From the output above, it appears as though Margaret Peacock completed the most with a total of 40 orders completed.

c. What product was ordered the most by customers in Germany?

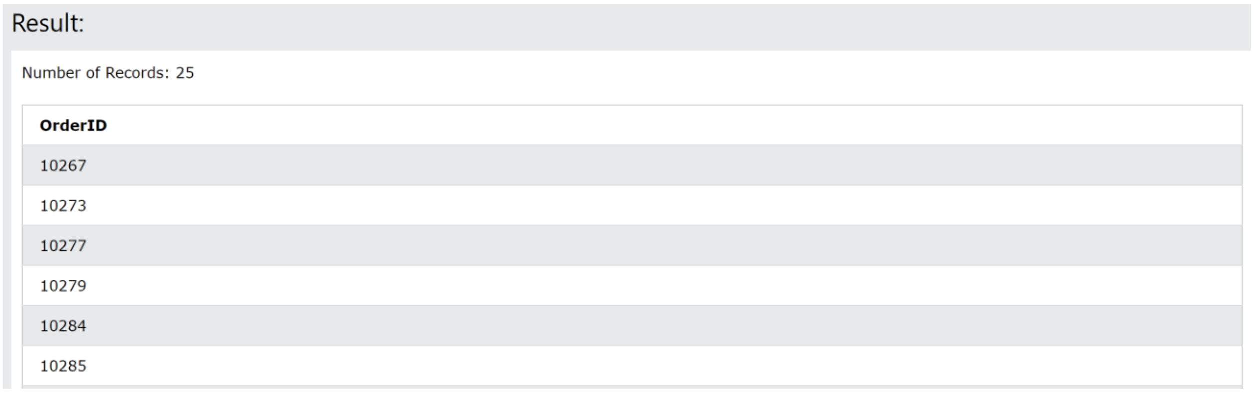
Assumption: I will interpret this to mean which product had the greatest frequency of being ordered by companies in Germany. Therefore, I will aim to product a summary of all the products that came from *OrderID's* associated with German customers. I will accompany this result with the frequency in which each product was ordered.

First, from the Orders table, it is important to determine which *OrderID's* are from customers in Germany. To do this, we join the *Orders* and *Customers* tables on the key *CustomerID*. In the WHERE clause, we search for only entries where *Country* is equal to Germany. The query to do this and it's output is shown below.



```sql
SELECT
        OrderID
FROM Orders
JOIN Customers
ON Orders.CustomerID = Customers.CustomerID
WHERE Country = "Germany"
```

Output: (First 6 of 25 Results)

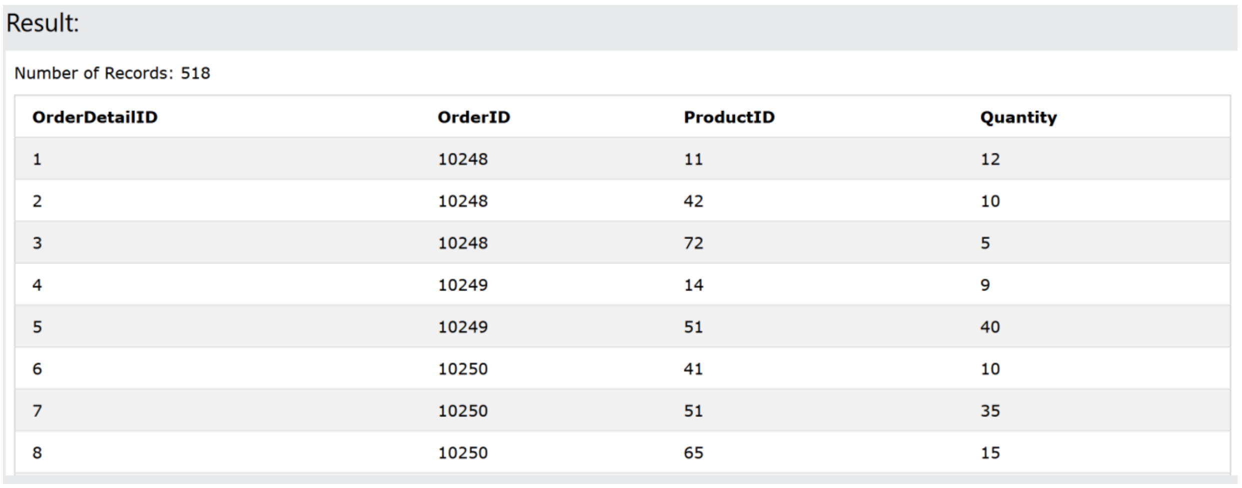


Result:

Number of Records: 25

OrderID
10267
10273
10277
10279
10284
10285

Next, we can look at the *OrderDetails* table. The first 8 results are shown below.

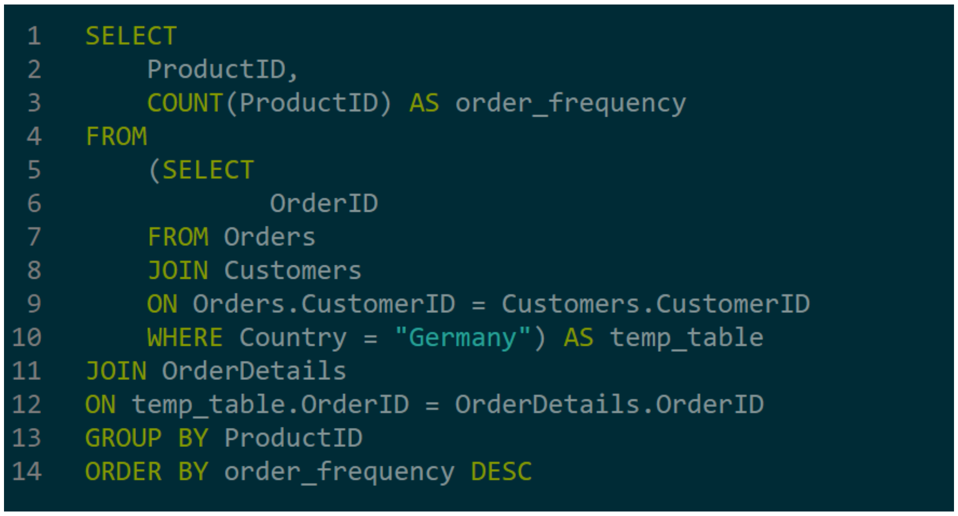


Result:

Number of Records: 518

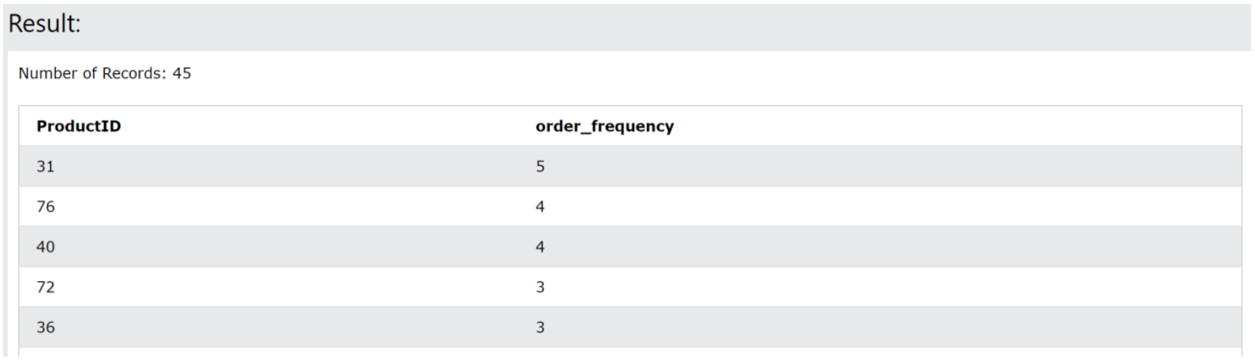
OrderDetailID	OrderID	ProductID	Quantity
1	10248	11	12
2	10248	42	10
3	10248	72	5
4	10249	14	9
5	10249	51	40
6	10250	41	10
7	10250	51	35
8	10250	65	15

Looking at the *OrderDetails* table, we see our previous query and this table share the same key of *OrderID*. If we join our previous query with this table, we should be left with a table that contains only *OrderDetails* that are relevant to German customers. We will use the above query as a temporary table / subquery in an outer FROM clause, and join the two table on *OrderID*. We are interested in the frequency of which each product ID was ordered. In our outer SELECT clause, we will include *ProductID* and the frequency of which each *ProductID* showed up. We will do this by including a COUNT() function to count the number of times each *ProductID* occurred. Finally, we will group by *ProductID* and put the order frequency in descending order to determine the product ID most frequently ordered. The final query and output are shown below.



```
SELECT
    ProductID,
    COUNT(ProductID) AS order_frequency
FROM
    (SELECT
            OrderID
    FROM Orders
    JOIN Customers
    ON Orders.CustomerID = Customers.CustomerID
    WHERE Country = "Germany") AS temp_table
JOIN OrderDetails
ON temp_table.OrderID = OrderDetails.OrderID
GROUP BY ProductID
ORDER BY order_frequency DESC
```

Output:


Result:

Number of Records: 45

ProductID	order_frequency
31	5
76	4
40	4
72	3
36	3

From the table above, it appears as though *ProductID* 31 was the most reoccurring product purchased. Manually checking the *Products* table shows that this ID is associated with the product name “Goronzola Telino”.