Summer 2022 Shopify Data Science Intern Challenge

Anh Vu Pham
anhvupham.ma@gmail.com

1 Question 1: Investigate Sneaker Shops AOV

1.1 Data Exploration

Before starting, let us understand what is being asked. We are asked to calculate the Average Order Value over 100 sneaker shops. Each sneaker shop only sells one model of shoe. There are different orders made from different shop, but each order should have the same price per shoe. A sneaker can ***maybe*** differ in price at the same shop on different days (if there are discounts or coupon codes).

Given these assumptions, let's load necessary libraries and load the dataset using Python:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
df = pd.read_csv("2019 Winter Data Science Intern Challenge Data Set - Sheet1.csv")
df
```

	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
...	...	...	...	...	...	...	...
4995	4996	73	993	330	2	debit	2017-03-30 13:47:17
4996	4997	48	789	234	2	cash	2017-03-16 20:36:16
4997	4998	56	867	351	3	cash	2017-03-19 5:42:42
4998	4999	60	825	354	2	credit_card	2017-03-16 14:51:18
4999	5000	44	734	288	2	debit	2017-03-18 15:48:18

5000 rows × 7 columns

We will now verify that there are no null values in the dataset and look at the summary of statistics pertaining to the dataset columns:

```
df.isna().sum()
```

```
order_id          0
shop_id           0
user_id           0
order_amount      0
total_items       0
payment_method    0
created_at        0
dtype: int64
```

```
df.describe()
```

	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

Let us look at the "order_amount" column. The mean is in fact $3145.13 as stated in the question. Let us now look at the different quartile rows (25%, 50% and 75%). Their respective values in the order_amount and total_items seem to have nothing wrong considering sneakers can have those price ranges and people can order more than one pair of sneakers ($163; one pair of sneakers, $284; two pairs of sneakers and $390; 3 pairs of sneakers).

Now looking at the min and max values, the min is not abnormal, but the max looks like an outlier. By dividing the "order_amount" by "total_items", we get:
$704000/2000 = $352 per pair of sneakers
This price is expensive, but possible if the sneaker is from a high-end collection or brand. It is highly unlikely that a sneaker store holds 2000 pairs of the same sneaker.

Let us now verify if there are any other outliers in the dataset:

```
sorted_data = df.sort_values(by=['order_amount', 'total_items'], ascending=False)
sorted_data.head(25)
```

	order_id	shop_id	user_id	order_amount	total_items	payment_method	created_at
15	16	42	607	704000	2000	credit_card	2017-03-07 4:00:00
60	61	42	607	704000	2000	credit_card	2017-03-04 4:00:00
520	521	42	607	704000	2000	credit_card	2017-03-02 4:00:00
1104	1105	42	607	704000	2000	credit_card	2017-03-24 4:00:00
1362	1363	42	607	704000	2000	credit_card	2017-03-15 4:00:00
1436	1437	42	607	704000	2000	credit_card	2017-03-11 4:00:00
1562	1563	42	607	704000	2000	credit_card	2017-03-19 4:00:00
1602	1603	42	607	704000	2000	credit_card	2017-03-17 4:00:00
2153	2154	42	607	704000	2000	credit_card	2017-03-12 4:00:00
2297	2298	42	607	704000	2000	credit_card	2017-03-07 4:00:00
2835	2836	42	607	704000	2000	credit_card	2017-03-28 4:00:00
2969	2970	42	607	704000	2000	credit_card	2017-03-28 4:00:00
3332	3333	42	607	704000	2000	credit_card	2017-03-24 4:00:00
4056	4057	42	607	704000	2000	credit_card	2017-03-28 4:00:00
4646	4647	42	607	704000	2000	credit_card	2017-03-02 4:00:00
4868	4869	42	607	704000	2000	credit_card	2017-03-22 4:00:00
4882	4883	42	607	704000	2000	credit_card	2017-03-25 4:00:00
691	692	78	878	154350	6	debit	2017-03-27 22:51:43
2492	2493	78	834	102900	4	debit	2017-03-04 4:37:34
1259	1260	78	775	77175	3	credit_card	2017-03-27 9:27:20
2564	2565	78	915	77175	3	debit	2017-03-25 1:19:35
2690	2691	78	962	77175	3	debit	2017-03-22 7:33:25
2906	2907	78	817	77175	3	debit	2017-03-16 3:45:46
3403	3404	78	928	77175	3	debit	2017-03-16 9:45:05
3724	3725	78	766	77175	3	credit_card	2017-03-16 14:13:26

There seems to be duplicates of the high price orders, but if we look at the "created_at" column, we can see these orders were not all made at the same time. Let's explore more before assuming that these orders are duplicate errors.

To dive deeper into the pricing of the sneakers, let us create a new column to see the price per pair of sneakers and look at the different unique values.

```
sorted_data['price_per_item'] = sorted_data.order_amount/sorted_data.total_items
sorted_data.price_per_item.unique()
```

```
array([  352., 25725.,   181.,   133.,   196.,   193.,   160.,   158.,
         187.,   184.,   178.,   177.,   176.,   173.,   169.,   166.,
         136.,   163.,   162.,   161.,   201.,   131.,   130.,   156.,
         195.,   129.,   153.,   190.,   149.,   148.,   147.,   146.,
         145.,   142.,   171.,   168.,   134.,   132.,   165.,   164.,
         155.,   154.,   118.,   116.,   144.,   112.,   140.,   138.,
         172.,   128.,   127.,   122.,    94.,   117.,   114.,    90.,
         111.,   101.])
```

The first two values are $352 and $25725. They are way above the price range of the other pairs of sneakers. We will definitely remove the $25725 values since sneakers are reasonably affordable, but what about the $352? Let's see which orders contain this value. (see on next page)

```
sorted_data.loc[sorted_data['price_per_item'] == 352]
```

	order_id	shop_id	user_id	order_amount	total_items	payment_method	created_at	price_per_pair
15	16	42	607	704000	2000	credit_card	2017-03-07 4:00:00	352.0
60	61	42	607	704000	2000	credit_card	2017-03-04 4:00:00	352.0
520	521	42	607	704000	2000	credit_card	2017-03-02 4:00:00	352.0
1104	1105	42	607	704000	2000	credit_card	2017-03-24 4:00:00	352.0
1362	1363	42	607	704000	2000	credit_card	2017-03-15 4:00:00	352.0
1436	1437	42	607	704000	2000	credit_card	2017-03-11 4:00:00	352.0
1562	1563	42	607	704000	2000	credit_card	2017-03-19 4:00:00	352.0
1602	1603	42	607	704000	2000	credit_card	2017-03-17 4:00:00	352.0
2153	2154	42	607	704000	2000	credit_card	2017-03-12 4:00:00	352.0
2297	2298	42	607	704000	2000	credit_card	2017-03-07 4:00:00	352.0
2835	2836	42	607	704000	2000	credit_card	2017-03-28 4:00:00	352.0
2969	2970	42	607	704000	2000	credit_card	2017-03-28 4:00:00	352.0
3332	3333	42	607	704000	2000	credit_card	2017-03-24 4:00:00	352.0
4056	4057	42	607	704000	2000	credit_card	2017-03-28 4:00:00	352.0
4646	4647	42	607	704000	2000	credit_card	2017-03-02 4:00:00	352.0
4868	4869	42	607	704000	2000	credit_card	2017-03-22 4:00:00	352.0
4882	4883	42	607	704000	2000	credit_card	2017-03-25 4:00:00	352.0
1364	1365	42	797	1760	5	cash	2017-03-10 6:28:21	352.0
1367	1368	42	926	1408	4	cash	2017-03-13 2:38:34	352.0
1471	1472	42	907	1408	4	debit	2017-03-12 23:00:22	352.0
938	939	42	808	1056	3	credit_card	2017-03-13 23:43:45	352.0
2987	2988	42	819	1056	3	cash	2017-03-03 9:09:25	352.0
3513	3514	42	726	1056	3	debit	2017-03-24 17:51:05	352.0
409	410	42	904	704	2	credit_card	2017-03-04 14:32:58	352.0
835	836	42	819	704	2	cash	2017-03-09 14:15:15	352.0
1520	1521	42	756	704	2	debit	2017-03-22 13:10:31	352.0
1911	1912	42	739	704	2	cash	2017-03-07 5:42:52	352.0
2003	2004	42	934	704	2	cash	2017-03-26 9:21:26	352.0
2273	2274	42	747	704	2	debit	2017-03-27 20:48:19	352.0
2491	2492	42	868	704	2	debit	2017-03-01 18:33:33	352.0
2609	2610	42	868	704	2	debit	2017-03-23 18:10:14	352.0
2766	2767	42	970	704	2	credit_card	2017-03-05 10:45:42	352.0
4294	4295	42	859	704	2	cash	2017-03-24 20:50:40	352.0
4326	4327	42	788	704	2	debit	2017-03-16 23:37:57	352.0
4421	4422	42	736	704	2	credit_card	2017-03-01 12:19:49	352.0
4767	4768	42	720	704	2	credit_card	2017-03-14 10:26:08	352.0
40	41	42	793	352	1	credit_card	2017-03-24 14:15:41	352.0
308	309	42	770	352	1	credit_card	2017-03-11 18:14:39	352.0
834	835	42	792	352	1	cash	2017-03-25 21:31:25	352.0
979	980	42	744	352	1	debit	2017-03-12 13:09:04	352.0
1512	1513	42	946	352	1	debit	2017-03-24 13:35:04	352.0
1929	1930	42	770	352	1	credit_card	2017-03-17 8:11:13	352.0
2018	2019	42	739	352	1	debit	2017-03-01 12:42:26	352.0
2053	2054	42	951	352	1	debit	2017-03-19 11:49:12	352.0
3651	3652	42	830	352	1	credit_card	2017-03-24 22:26:58	352.0
3697	3698	42	839	352	1	debit	2017-03-12 2:45:09	352.0
3903	3904	42	975	352	1	debit	2017-03-12 1:28:31	352.0
3998	3999	42	886	352	1	debit	2017-03-09 20:10:41	352.0
4231	4232	42	962	352	1	cash	2017-03-04 0:01:19	352.0
4625	4626	42	809	352	1	credit_card	2017-03-11 8:21:26	352.0
4745	4746	42	872	352	1	debit	2017-03-24 0:57:24	352.0

The $352 sneaker is plausible with the variety of orders from different users. We can therefore conclude that the only outliers for the $352 sneaker are the 2000 total items orders.

1.2 Takeaways

We will remove the outliers in the dataset. These outliers were the orders which contained 2000 total items as well as the orders where the price per item is $25725. Here is the dataset with the dropped values:

```
#add price per item in original df
df['price_per_item'] = df.order_amount/df.total_items
df = df[df['total_items'] != 2000]
df = df[df['price_per_item'] != 25725]
df
```

	order_id	shop_id	user_id	order_amount	total_items	payment_method	created_at	price_per_item
0	1	53	746	224	2	cash	2017-03-13 12:36:56	112.0
1	2	92	925	90	1	cash	2017-03-03 17:38:52	90.0
2	3	44	861	144	1	cash	2017-03-14 4:23:56	144.0
3	4	18	935	156	1	credit_card	2017-03-26 12:43:37	156.0
4	5	18	883	156	1	credit_card	2017-03-01 4:35:11	156.0
...	...	...	...	...	...	...	...	...
4995	4996	73	993	330	2	debit	2017-03-30 13:47:17	165.0
4996	4997	48	789	234	2	cash	2017-03-16 20:36:16	117.0
4997	4998	56	867	351	3	cash	2017-03-19 5:42:42	117.0
4998	4999	60	825	354	2	credit_card	2017-03-16 14:51:18	177.0
4999	5000	44	734	288	2	debit	2017-03-18 15:48:18	144.0

4937 rows × 8 columns

We went from 5000 to 4937 rows.

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

There were multiple orders with a high order amount. These orders affected the Average Order Value by inflating the price. To better evaluate this data, we must remove these high order amount outliers as stated in the Takeaways subsection.

1.4 b) What metric would you report for this dataset?

I would report the order ID accompanied by the order amount, total items and price per item. The order ID is to verify uniqueness of order, while the other columns give us the important information on the order with its prices.
We can now compute the Average Order Value as well as compare prices of orders between shops.

1.5 c) What is its value?

```
df['order_amount'].mean()
```

```
302.58051448247926
```

```
df['order_amount'].median()
```

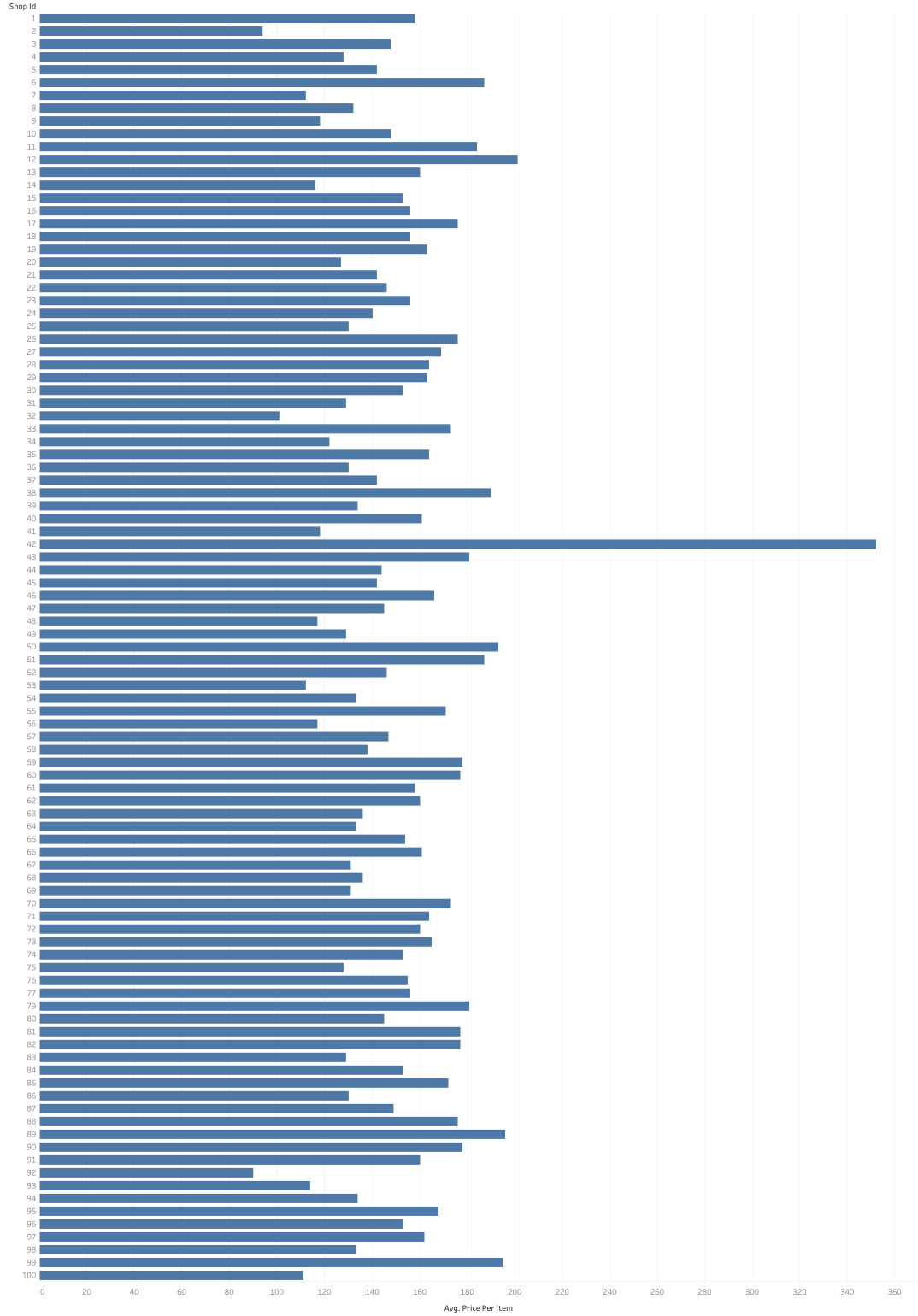
```
284.0
```

The new mean is around $302.58, which is close to the median of $284. We can conclude that this new AOV is more accurate to the value we are looking for. (See next page for extra graphs)

See next page for extra plots using Tableau that helped me understand and visualize this problem better (after removing outliers):

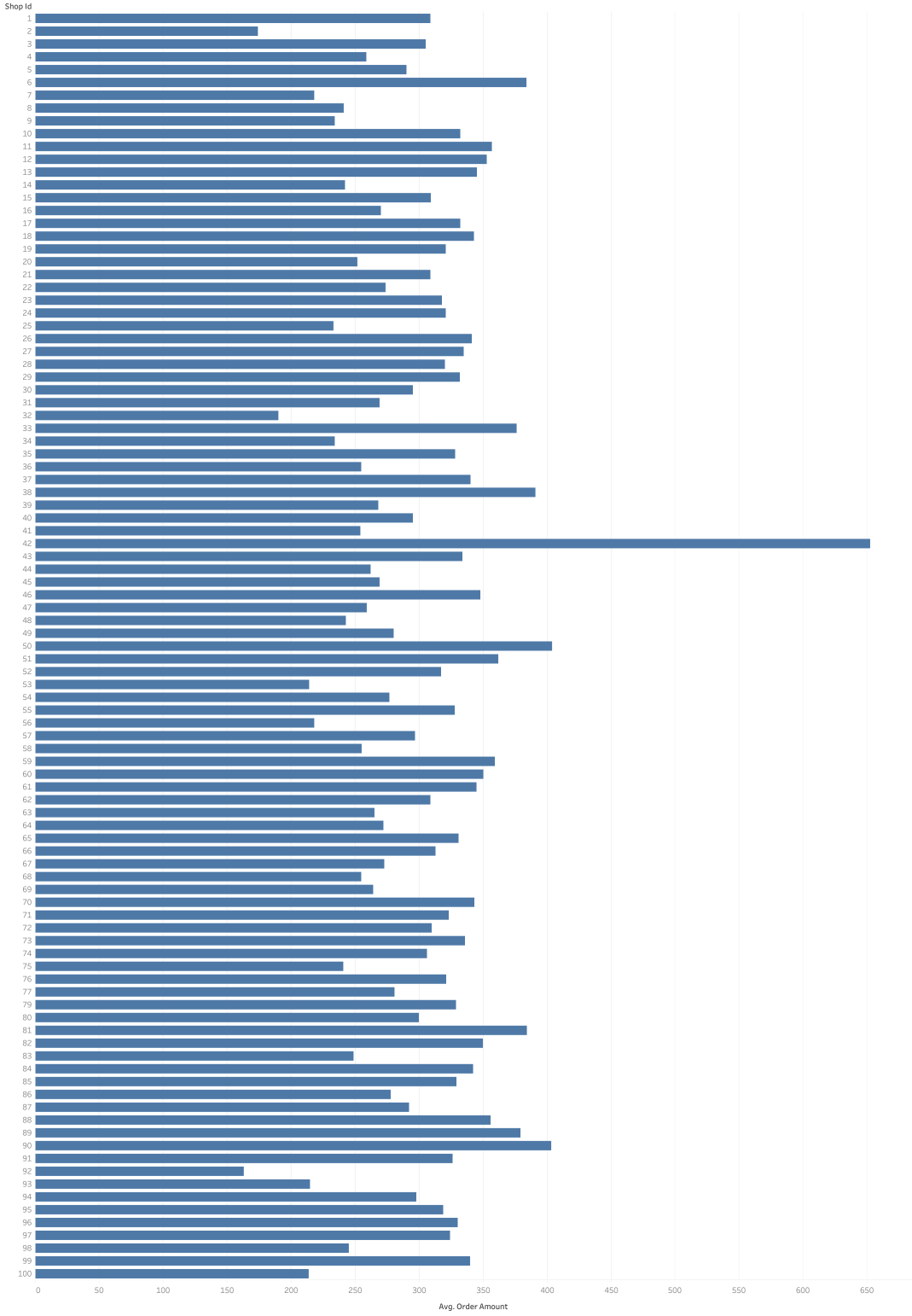
1.5.1 Sneaker Price per Shop

<Sneaker Price per Shop>

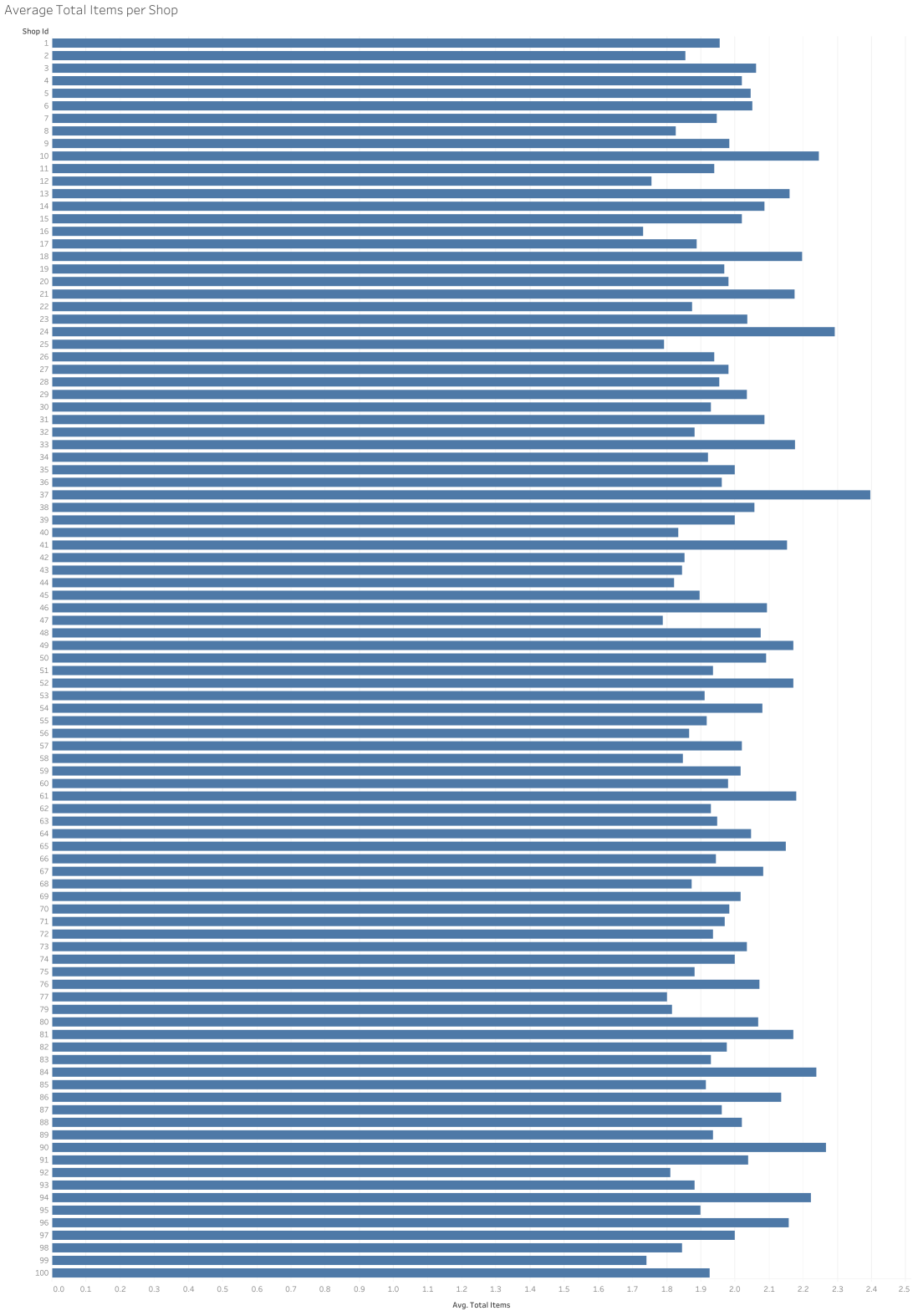


1.5.2 Average Order Amount per Shop

<Average Order Amount per Shop>



1.5.3 Average Total Items per Shop



2 Question 2: SQL

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

SQL Statement:

```
SELECT COUNT(ShipperID)
FROM Orders
WHERE ShipperID = 1;
```

Edit the SQL Statement, and click "Run SQL" to see the result.

Run SQL »

Result:

Number of Records: 1

COUNT(ShipperID)
54

Here ShipperID = 1 is Speedy Express.

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

SQL Statement:

```
SELECT a.LastName
FROM Employees AS a, Orders AS b
WHERE a.employeeID=b.EmployeeID
GROUP BY a.employeeID
ORDER BY COUNT(a.employeeID) DESC LIMIT 1
;
```

Edit the SQL Statement, and click "Run SQL" to see the result.

Run SQL »

Result:

Number of Records: 1

LastName
Peacock

We had to join the Employees table with the Orders table to be able to link the number of orders with the employees last name. Each row is an order, therefore

we can group by EmployeeID to count the number of orders per employee since the EmployeeID appears on each order.

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

SQL Statement:

```
SELECT b.ProductName, b.ProductID, SUM(c.Quantity) AS AmountOrdered
FROM Customers as a, Products as b, OrderDetails as c, Orders AS d
WHERE a.customerID=d.customerID AND b.ProductID=c.ProductID AND d.OrderID=c.OrderID AND a.Country='Germany'
GROUP BY b.ProductID
ORDER BY SUM(c.Quantity) DESC LIMIT 1;
```

Edit the SQL Statement, and click "Run SQL" to see the result.

Run SQL »

Result:

Number of Records: 1

ProductName	ProductID	AmountOrdered
Boston Crab Meat	40	160

c) is similarly to b). After joining the necessary tables, we group by ProductID (or ProductName) and order by the sum of the quantity ordered to find the product ordered most.