Here are the answers and analyses to the questions which are part of Shopify's Data Science Internship Challenge. For the full code, please refer to the Jupyter Notebook in the same GitHub repository. Thank you!

Question 1

I am using pandas library in Python to help me analyze the given csv file and answer the provided question.

```
with open("2019 Winter Data Science Intern Challenge Data Set - Sheet1.csv", "r") as f:
    df = pd.read_csv(f)

df.head()
```

	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

1. What could be going wrong with the initial calculation and a better way to evaluate the data.

   What could have been wrong: If we look at the data closely, we can see that out of the 5000 orders listed, only 17 had 'total_items' of more than 2000. The remaining orders only had total items of less than 1000. This is shown by the code block below.

   ```
   subset_df = df[df["total_items"]>1000]
   subset_df.shape
   ```

   ```
   (17, 7)
   ```

   In fact, all those 17 orders were made by the same user, whose user_id is 607, as shown.

```
subset_df
```

	order_id	shop_id	user_id	order_amount	total_items	payment_method	created_at
4868	4869	42	607	704000	2000	credit_card	2017-03-22 4:00:00
1104	1105	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
3332	3333	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
4882	4883	42	607	704000	2000	credit_card	2017-03-25 4:00:00
1436	1437	42	607	704000	2000	credit_card	2017-03-11 4:00:00
2969	2970	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
2835	2836	42	607	704000	2000	credit_card	2017-03-28 4:00:00
2297	2298	42	607	704000	2000	credit_card	2017-03-07 4:00:00
520	521	42	607	704000	2000	credit_card	2017-03-02 4:00:00
2153	2154	42	607	704000	2000	credit_card	2017-03-12 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
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

Furthermore, those 17 orders of 2000 items each, were the only orders made by user 607, as shown below.

```
subset3 = df[df["user_id"]==607]
subset3.shape
```

```
(17, 7)
```

Furthermore, for the remaining 4,783 orders, the mean of 'total_items' is 1.993 which means that an order in average only consists of 2 items being bought.

```
subset2 = df[df["total_items"]<=1000]
subset2.shape
```

```
(4983, 7)
```

```
subset2["total_items"].mean()
```

```
1.9939795304033714
```

From this only, we can see that the 17 orders containing 2,000 items are essentially outliers. These outliers have an AOV of $704,000, while the rest of the orders have an AOV of about $754. This stark difference results in the total naive AOV of $3,145.

```
# AOV of 4983 orders with items < 2000 each
subset2["order_amount"].mean()
```

```
754.0919125025085
```

```
# AOV of 17 orders with items = 2000 each
subset_df["order_amount"].mean()
```

```
704000.0
```

```
# initial AOV of all orders
df["order_amount"].mean()
```

```
3145.128
```

Thus, I suggest we only look at the 4,983 orders with less than 2,000 items each (in fact, only 2 items in average for each order) to come up with the store's AOV. Doing so, will give us a value of $754 of AOV which makes more sense than the initial calculation. In the final report, however, we should still note that there are 17 orders (or about 0.34% of total orders) from one single customer that have an AOV of $704,000.

2. What metric would you report for this dataset?

   The number of unique consumers as well as the number of orders made by each customer.

   This will tell us how many consumers the shop has been able to reach out to. Knowing this will help us think about how relevant our shop is. We will also be able to identify our most loyal customers, so that we can reach out to them in the future for referrals to their network.

3. What is its value?

   The number of unique customers is the same as the number of 'user_id', because each 'user_id' identifies a unique customer. Counting the number of unique customers:

```
df['user_id'].nunique()
```

```
301
```

   Thus there are 301 unique customers who make up the total 5000 orders in March 2017.

   Counting the number of orders made by each customer can be done using the function below:

```
id_user = df.user_id.unique() # user id's
dd = {}
dd['user_id'] = id_user
total_order_amount_list = []
for uu in id_user:
    total_order_amount = df[df['user_id']==uu]['order_amount'].sum()
    total_order_amount_list.append(total_order_amount)
dd['total_order_amount'] = total_order_amount_list
dfin = pd.DataFrame(dd)
dfin.sort_values(by='total_order_amount', ascending=False)
```

	user_id	total_order_amount
300	607	11968000
202	878	156936
285	834	108342
245	787	85707
59	969	84269
...	...	...
213	750	2359
299	717	2337
274	719	2314
18	939	2196
81	902	2108

301 rows × 2 columns

Customer with the most order: customer with id 607, who is the same customer who orders 2,000 items per order for 17 times.

Question 2

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

From the 'Shipper' table, we know that the ShipperID of Speedy Express is 1. We can just eyeball the 'Shipper' table to get this information, or we can run:

```
select ShipperID from Shippers where ShipperName = "Speedy Express";
```

Returning:

Number of Records: 1

ShipperID
1

We then run the following query:

```
select sum(OrderDetails.Quantity)
from Orders
INNER JOIN OrderDetails
USING (OrderID)
where Orders.ShipperID = 1;
```

We need to use INNER JOIN because the information about which orders were shipped by which suppliers and the information about the quantity of each order are in two separate tables. The foreign key is OrderID.
Which returns:

Number of Records: 1

sum(OrderDetails.Quantity)
3575

Thus, Speedy Express shipped 3,575 orders in total.

b. What is the last name of the employee with the most orders?
There are 3 separate tables that we have to consult to answer this question. To know the OrderIDs and EmployeeIDs of the orders made, we must consult the Orders table. To know the sum of the quantity of each order, we must consult the OrderDetails table. To know the employee with a particular EmployeeID, we must consult the Employees table. In my solution, I will use two steps to answer this question instead of using multiple join or multiple select statements to maintain readability and to be clear. The steps are as follows:

   i. First, we find the EmployeeID that has the most orders by using the following query:

```
select o.EmployeeID, sum(d.Quantity)
from Orders o
INNER JOIN OrderDetails d USING (OrderID)
GROUP BY o.EmployeeID
ORDER BY sum(d.Quantity) DESC;
```

We group by the EmployeeID so that we know which employee is responsible for which subset of the orders. We then order the sum of quantities in a descending order so that the employee with the most orders will appear at the topmost position of the table.
The above query returns the following:

Number of Records: 9

EmployeeID	sum(d.Quantity)
4	3232
1	1924
3	1725
2	1315
8	1293
6	1094
5	778
7	733
9	649

We now know that the employee with the most orders has OrderID of 4. Now we look for the name of said employee by running the query below:

```
select LastName from Employees where EmployeeID = 4;
```

Which returns:

Number of Records: 1

LastName
Peacock

So, we know that **the last name of the employee with the most orders is Peacock.**

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

To answer this question, we need to look at 4 tables, i.e.

- Customers (to get the CustomerID of the customers in Germany),
- Orders (to get the OrderID of all orders made by the customers with the CustomerIDs we have collected),
- OrderDetails (to get the total quantities for certain ProductIDs based on the OrderIDs that we have collected), and
- Products (to get the ProductName of each of the ProductIDs).

With that in mind, we can write the query by joining the four tables using their own foreign key linking one to another. Thus, the query is now as follows:

```
select ProductID, ProductName, sum(Quantity) from Products
inner join OrderDetails using (ProductID)
inner join Orders using (OrderID)
inner join Customers using (CustomerID) where country = "Germany"
group by ProductID
order by sum(Quantity) desc;
```

And (a few first rows of) the returned table:

Number of Records: 45

ProductID	ProductName	sum(Quantity)
40	Boston Crab Meat	160
31	Gorgonzola Telino	125
23	Tunnbröd	105
35	Steeleye Stout	100
19	Teatime Chocolate Biscuits	95
72	Mozzarella di Giovanni	86

Thus, we can conclude that the product ordered the most by customers in Germany is Boston Crab Meat.