Question 1: Given some sample data, write a program to answer the following: [click here to access the required data set]

On Shopify, we have exactly 100 sneaker shops, and each of these shops sells only one model of shoe. We want to do some analysis of the average order value (AOV). When we look at orders data over a 30 day window, we naively calculate an AOV of $3145.13. Given that we know these shops are selling sneakers, a relatively affordable item, something seems wrong with our analysis.

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

Solution: I have initially checked the statistics of the data such as mean, median and mode of order_data column to get better understanding of the data. Below is the snippet of Python code where I have used pandas describe function to find the statistics.

Importing the required libraries

```
[1] import numpy as np
    import pandas as pd
```

Reading the dataset

```
[2] url="https://raw.githubusercontent.com/bmounikareddy98/Shopify/main/2019%20Winter%20Data%20Science%20Intern%20Challenge%20Data%20Set%20-%20Sheet1.csv"

    dataset= pd.read_csv(url)
```

Checking the statistics of order_amount

```
dataset.order_amount.describe()

count      5000.000000
mean       3145.128000
std       41282.539349
min          90.000000
25%         163.000000
50%         284.000000
75%         390.000000
max      704000.000000
Name: order_amount, dtype: float64
```

As we can see, the AOV is 3145.13 , which is high for a relatively affordable product like Sneakers. If we observe the statistics, the maximum order amount is 704000 and minimum is 90 and the standard deviation is 41282.54. The standard deviation is very high and it implies that on average, the values vary 41282.54 from the mean, which makes mean or AOV not a useful metric. Also, we can observe that 25% Quartile, median and 75% Quartile values vary hugely from maximum. We can infer that , there are few outliers with high order amounts in our dataset that are dragging up the mean to a higher value.

In order to understand the data distribution, I have plotted a box plot.

```
[ ]  dataset.boxplot(column='order_amount')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f290868c250>
```

We can observe that the box portion of the box plot is a line along 0 and there are lots of outliers. In order to find about these outliers, I have considered to group the data by order_amount and the count the number of entries in each group and sort it in descending order.

```
[ ]  unique_amount_data = dataset.groupby(['order_amount']).size().reset_index(name='count').sort_values(by='order_amount', ascending=False)
     unique_amount_data.head(10)
```

	order_amount	count
257	704000	17
256	154350	1
255	102900	1
254	77175	9
253	51450	16
252	25725	19
251	1760	1
250	1408	2
249	1086	1
248	1064	1

We can see that some of the higher order amounts such as 704000, 154350, 102900 are repeated. Let’s look at the rows from our dataset as well.

```
dataset.loc[dataset['order_amount'].isin([704000, 51450, 25725])].sort_values(by='order_amount', ascending=False)
```

	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
1362	1363	42	607	704000	2000	credit_card	2017-03-15 4:00:00
2969	2970	42	607	704000	2000	credit_card	2017-03-28 4:00:00
2835	2836	42	607	704000	2000	credit_card	2017-03-28 4:00:00
4056	4057	42	607	704000	2000	credit_card	2017-03-28 4:00:00
60	61	42	607	704000	2000	credit_card	2017-03-04 4:00:00
2297	2298	42	607	704000	2000	credit_card	2017-03-07 4:00:00
2153	2154	42	607	704000	2000	credit_card	2017-03-12 4:00:00
1562	1563	42	607	704000	2000	credit_card	2017-03-19 4:00:00
1436	1437	42	607	704000	2000	credit_card	2017-03-11 4:00:00
1602	1603	42	607	704000	2000	credit_card	2017-03-17 4:00:00
3332	3333	42	607	704000	2000	credit_card	2017-03-24 4:00:00
1104	1105	42	607	704000	2000	credit_card	2017-03-24 4:00:00
4882	4883	42	607	704000	2000	credit_card	2017-03-25 4:00:00
4868	4869	42	607	704000	2000	credit_card	2017-03-22 4:00:00
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
511	512	78	967	51450	2	cash	2017-03-09 7:23:14
3167	3168	78	927	51450	2	cash	2017-03-12 12:23:08
3705	3706	78	828	51450	2	credit_card	2017-03-14 20:43:15
3101	3102	78	855	51450	2	credit_card	2017-03-21 5:10:34
490	491	78	936	51450	2	debit	2017-03-26 17:08:19
2821	2822	78	814	51450	2	cash	2017-03-02 17:13:25
2818	2819	78	869	51450	2	debit	2017-03-17 6:25:51
493	494	78	983	51450	2	cash	2017-03-16 21:39:35
2495	2496	78	707	51450	2	cash	2017-03-26 4:38:52
2512	2513	78	935	51450	2	debit	2017-03-18 18:57:13
2452	2453	78	709	51450	2	cash	2017-03-27 11:04:04
4079	4080	78	946	51450	2	cash	2017-03-20 21:14:00
617	618	78	760	51450	2	cash	2017-03-18 11:18:42
1529	1530	78	810	51450	2	cash	2017-03-29 7:12:01
4311	4312	78	960	51450	2	debit	2017-03-01 3:02:10
4412	4413	78	756	51450	2	debit	2017-03-02 4:13:39

3440	3441	78	982	25725	1	debit	2017-03-19 19:02:54
4040	4041	78	852	25725	1	cash	2017-03-02 14:31:12
3780	3781	78	889	25725	1	cash	2017-03-11 21:14:50
4505	4506	78	866	25725	1	debit	2017-03-22 22:06:01
4584	4585	78	997	25725	1	cash	2017-03-25 21:48:44
2548	2549	78	861	25725	1	cash	2017-03-17 19:36:00
3151	3152	78	745	25725	1	credit_card	2017-03-18 13:13:07
3085	3086	78	910	25725	1	cash	2017-03-26 1:59:27
2922	2923	78	740	25725	1	debit	2017-03-12 20:10:58
2773	2774	78	890	25725	1	cash	2017-03-26 10:36:43
2270	2271	78	855	25725	1	credit_card	2017-03-14 23:58:22
1452	1453	78	812	25725	1	credit_card	2017-03-17 18:09:54
1419	1420	78	912	25725	1	cash	2017-03-30 12:23:43
1384	1385	78	867	25725	1	cash	2017-03-17 16:38:06
1204	1205	78	970	25725	1	credit_card	2017-03-17 22:32:21
1193	1194	78	944	25725	1	debit	2017-03-16 16:38:26
1056	1057	78	800	25725	1	debit	2017-03-15 10:16:45
160	161	78	990	25725	1	credit_card	2017-03-12 5:56:57
4918	4919	78	823	25725	1	cash	2017-03-15 13:26:46

We can observe that the order amounts of 704000 occur at the same time each day between the same store and users. Also, the data data for order amounts of 51450 and 25725 is similar. We can also infer that the orders of 51450 are just transactions that bought two items worth 25725 as they are from store id 78.

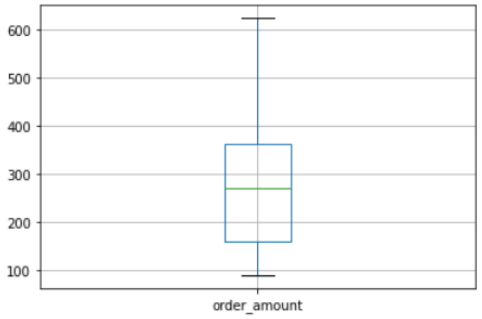
We can also consider that for the high order value 704000 amounts, the transactions are done by a supplier who is purchasing many shoes at once as the order amount is constantly 2000.

In order to evaluate the data well, we have to eliminate some of the values. We can try plotting only the values that are median +/- 1.5 IQR.

```
[ ]  q1 = dataset.order_amount.quantile(q=0.25)
     q2 = dataset.order_amount.quantile(q=0.5)
     q3 = dataset.order_amount.quantile(q=0.75)
     IQR = q3 - q1

     dataset_truncated = dataset[(dataset.order_amount < q2 + IQR * 1.5) & (dataset.order_amount > q2 - IQR * 1.5)]
     dataset_truncated.boxplot(column='order_amount')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f29080713d0>
```



Using dataset_truncated will provide a more accurate representation of the typical order data.

b. What metric would you report for this dataset?

By observing the box plot for dataset_truncated, we can see that the distribution is skewed towards lower values. Considering this, we can report that the **median value of the truncated dataset is a good metric**, because the values above the median have increased the mean(AOV) by a higher amount.

c. What is its value?

I have observed that the value is **$272** by using the describe( ) function on truncated dataset. The standard deviation is 132.06, which is also a reasonable result.

```
[ ]  dataset_truncated.order_amount.describe()
```

```
count    4738.000000
mean      283.814268
std       132.061996
min        90.000000
25%       161.000000
50%       272.000000
75%       362.000000
max       624.000000
Name: order_amount, dtype: float64
```

Question 2: For this question you'll need to use SQL. Follow this link to access the data set required for the challenge. Please use queries to answer the following questions. Paste your queries along with your final numerical answers below.

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

We have to join the tables Shippers and Orders to get Order details according to each Shipper. I have inner joined on ShipperID. On this joined relation, I have applied filter on ShipperName as "Speedy Express" which resulted in the **total number of orders as 54.**

SQL Query: SELECT count(OrderID) from Orders O, Shippers S

where O.ShipperID= S.ShipperID and ShipperName="Speedy Express"

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

In order to solve this problem, we have to count the number of orders associated with each employee, and find the employee who has ordered most and then retrieve their last name.

Using SQL, I counted number of orders in the [Orders] table and grouped by EmployeeID. We can get the employee ID of the person with the most orders, but not their last name. We can use the JOIN expression and merge the [Orders] table and the [Employees] on employee ID and group by their last name to retrieve the results.

SQL Query:

```
SELECT [Employees].LastName, COUNT(*) AS Number_Of_Orders
FROM [Orders]
JOIN [Employees]
ON [Orders].EmployeeID = [Employees].EmployeeID
GROUP BY [Employees].LastName
ORDER BY Number_Of_Orders DESC
LIMIT 1
```

We have got the result as Peacock with number of orders as 40.

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

The data required to solve this problem is present across multiple tables. Our output should show a list of products that are ordered by customers in Germay and number of orders of that product.

We have to break the problem into smaller sections and tackle them one at a time. Initially, let's view all the orders *from* Germany.

SQL Query:

```
SELECT [Orders].OrderID,[Customers].Country
FROM [Orders]
JOIN [Customers]
ON [Customers].CustomerID = [Orders].CustomerID
WHERE [Customers].Country = 'Germany'
```

From the above query, we have got list of all orders to customers in Germany.

The next part of the problem is figuring out *which item was ordered the most*. We need to incorporate information such as product ID and quantity from the [OrderDetails] table to do this. By joining [OrderDetails] through the OrderID column, we can find the total quantity for each product by summing the quantity column and grouping by product id.

```
SELECT [Customers].Country,[OrderDetails].ProductID,SUM([OrderDetails].Quantity) AS "Total_Ordered"
FROM [Orders]
JOIN [Customers]
ON [Customers].CustomerID = [Orders].CustomerID
JOIN [OrderDetails]
ON [OrderDetails].OrderID = [Orders].OrderID
WHERE [Customers].Country = 'Germany'
GROUP BY [OrderDetails].ProductID
ORDER BY Total_lOrdered DESC
```

After executing this query, we can see the most ordered item has a product ID of 40 with 160 orders. In order to get the name of the most ordered product, we have to join the [Products] table on the ProductID column to get the name of the most ordered product.

SQL Query:

```
SELECT [Products].ProductName, SUM([OrderDetails].Quantity) AS "Total_Ordered"

FROM [Orders]

JOIN [Customers]

ON [Customers].CustomerID = [Orders].CustomerID

JOIN [OrderDetails]

ON [OrderDetails].OrderID = [Orders].OrderID

JOIN [Products]

ON [Products].ProductID = [OrderDetails].ProductID

WHERE [Customers].Country = 'Germany'

GROUP BY [OrderDetails].ProductID

ORDER BY Total_Ordered DESC
```

After executing this query, the result turned out to be **Boston Crab Meat** has the most orders with total of **160.**