Problem 1:

Given some sample data, write a program to answer the following: [click here to access the required data set (https://docs.google.com/spreadsheets/d/16i38oonuX1y1g7C_UAmiK9GkY7cS-64DfiDMNiR41LM/edit#gid=0)](https://docs.google.com/spreadsheets/d/16i38oonuX1y1g7C_UAmiK9GkY7cS-64DfiDMNiR41LM/edit#gid=0)

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.

- Think about what could be going wrong with our calculation. Think about a better way to evaluate this data.
- What metric would you report for this dataset?
- What is its value?

Solution:

The Key Data columns to focus on in this dataset are the **order_amount** and **total_items** column. I imported the dataset to investigate further.

In [94]:
```
import pandas as pd
import numpy as np
csv = pd.read_csv('/Users/ameyadalvi/Downloads/shopifydata.csv')
shopify = pd.DataFrame(csv)
shopify
```

Out[94]:

	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

Checking some distribution metrics and their values for the order_amount

In [95]:
```
shopify["order_amount"].mean()
```

Out[95]:
```
3145.128
```

In [96]:
```
shopify["order_amount"].median()
```

Out[96]:
```
284.0
```

In [97]:
```
shopify["order_amount"].mode()
```

Out[97]:
```
0    153
dtype: int64
```

In [98]:
```
shopify["order_amount"].max()
```

Out[98]:
```
704000
```

The Issue:

- The problem statement's naive calculation of the average order value appears to be determined by averaging all order amounts in all stores over a 30-day period. (Sum of all order amounts / 5000 records)
- After digging deeper into the order amount column, I discovered a maximum value of **704000**, which is actually for 2000 pairs of sneakers, but it functions as an **outlier** in this naïve averaging computation, resulting in the inflated AOV of $3145.13.

My Solution:

- If we were to average all of the order amounts for all 5000 records, we should technically also take into account the number of the items sold to arrive at the particular order amount.
- The total_items column comes into play at this point.
- According to my understanding, **Average Order Value** must also take into account the **quantity** of each sold item for order amount provided by each shop.
- There's also two ways to do this

> 1. Sum all the order amounts and divide it by total number of items ordered.
> 2. Calculate order value for a single item for a record and then calculate the mean.

In [104]:
```
shopify["order_amount"].sum()/shopify["total_items"].sum()  # The first met
```

Out[104]:
```
357.92152221412965
```

In [107]:
```
shopify["ov"] = shopify["order_amount"]/shopify["total_items"] # The second
shopify["ov"].mean()
```

Out[107]:
```
387.7428
```

As you can see, in the first method i added all the order amounts and divided it by the total items ordered to get the AOV as $357.92

In the second method (more logical), I calculated the order value for a single item (single pair of sneakers) for each shop by adding the new column "ov" As a result, I determined the new mean by dividing the order amount by the total items for each order. The mean by this method - $387.75

Just to explore a bit more, I also tried to check the impact of the outlier shop 42 (order amount = 704000) on the mean calculated by the first method

In [109]:
```
shopify_2 = shopify[shopify.shop_id!=42]
shopify_2["order_amount"].mean()
```

Out[109]:
```
754.7916750858759
```

This shows how wrong the naive calculation was that removing a shop from consideration, which ideally shouldn't be done as the order amount is for a certain number of items, drastically changes the Average Order Value.

From the second method, we derived a value of 387.75 which although, is considerably lower than 3145.13 but still seems to be on a higher end considering it is just a sneaker and mentioned as affordable in the problem statement.

To analyze a little deeper, we can check the average order value for every single of the 100 shops.

In [102]:
```
shopify.sort_values('shop_id',ascending = 'True')
```

Out[102]:

	order_id	shop_id	user_id	order_amount	total_items	payment_method	created_at	ov
4515	4516	1	797	316	2	cash	2017-03-17 16:43:51	158.0
4657	4658	1	886	316	2	debit	2017-03-02 19:20:24	158.0
1190	1191	1	734	316	2	debit	2017-03-03 3:40:15	158.0
347	348	1	907	158	1	credit_card	2017-03-03 6:59:35	158.0
2587	2588	1	756	316	2	debit	2017-03-30 12:27:19	158.0
...	...	...	...	...	...	...	...	...
649	650	100	851	222	2	debit	2017-03-07 6:02:56	111.0
3981	3982	100	818	111	1	cash	2017-03-23 21:34:27	111.0
4204	4205	100	847	333	3	debit	2017-03-18 11:13:20	111.0
3989	3990	100	889	333	3	debit	2017-03-05 0:13:44	111.0
1210	1211	100	765	111	1	debit	2017-03-30 8:31:13	111.0

5000 rows × 8 columns

In [91]:
```
shopify_mini = shopify[['shop_id','order_amount','total_items']].copy()
shopify_mini = shopify_mini.sort_values('shop_id',ascending = 'True')
shopify_mini
```

Out[91]:

	shop_id	order_amount	total_items
4515	1	316	2
4657	1	316	2
1190	1	316	2
347	1	158	1
2587	1	316	2
...	...	...	...
649	100	222	2
3981	100	111	1
4204	100	333	3
3989	100	333	3
1210	100	111	1

5000 rows × 3 columns

In [69]:
```
shopify_mini = shopify_mini.groupby('shop_id')['order_amount','total_items'
shopify_mini
```

```
<ipython-input-69-a9e6f66f7449>:1: FutureWarning: Indexing with multiple
keys (implicitly converted to a tuple of keys) will be deprecated, use a
list instead.
  shopify_mini = shopify_mini.groupby('shop_id')['order_amount','total_it
ems'].apply(np.sum).reset_index()
```

Out[69]:

	shop_id	order_amount	total_items
0	1	13588	86
1	2	9588	102
2	3	14652	99
3	4	13184	103
4	5	13064	92
...	...	...	...
95	96	16830	110
96	97	15552	96
97	98	14231	107
98	99	18330	94
99	100	8547	77

100 rows × 3 columns

In [70]:
```
shopify_mini["ov"] = shopify_mini["order_amount"]/shopify_mini["total_items
shopify_mini
```

Out[70]:

	shop_id	order_amount	total_items	ov
0	1	13588	86	158.0
1	2	9588	102	94.0
2	3	14652	99	148.0
3	4	13184	103	128.0
4	5	13064	92	142.0
...	...	...	...	...
95	96	16830	110	153.0
96	97	15552	96	162.0
97	98	14231	107	133.0
98	99	18330	94	195.0
99	100	8547	77	111.0

100 rows × 4 columns

In [71]:
```
shopify_mini['ov'].mean()
```

Out[71]:
```
407.99
```

In [72]:
```
shopify_mini['ov'].max()
```

Out[72]:
```
25725.0
```

In [74]:
```
shopify_mini.loc[shopify_mini['ov'] == shopify_mini['ov'].max()]
```

Out[74]:

	shop_id	order_amount	total_items	ov
77	78	2263800	88	25725.0

- As you can see in the cell above, shop number 78 sells the same pair of shoes at an average order value of $25725.0 which is a crazy amount for a single pair :P
- The shop number 78 is the reason for a higher AOV. Calculating the mean of all the shops gives us $408, for a single pair of shoes :D, all thanks to the outlier.
- To check the extent of it's impact, we can drop the particular shop to see how the mean re-adjusts itself.

In [75]:
```
shopify_mini1 = shopify_mini.drop([77])
shopify_mini1
```

Out[75]:

	shop_id	order_amount	total_items	ov
0	1	13588	86	158.0
1	2	9588	102	94.0
2	3	14652	99	148.0
3	4	13184	103	128.0
4	5	13064	92	142.0
...	...	...	...	...
95	96	16830	110	153.0
96	97	15552	96	162.0
97	98	14231	107	133.0
98	99	18330	94	195.0
99	100	8547	77	111.0

99 rows × 4 columns

In [77]:
```
shopify_mini1['ov'].mean()
```

Out[77]:
```
152.26262626262627
```

- For a pair of sneakers, the average AOV is currently $152.26, which is a far more acceptable price than $408 and certainly much more affordable than $3145.13. :D
- Although this solves our problem, but omitting a whole shop from consideration is not a acceptable solution for the problem.
- We cannot omit a potentially important datapoint without a thorough reasoning to do so!
- Hence we can look into some alternative AOV metrics that don't take into account the average of the OVs

Median OV :

- The Median value of all the OVs of our 100 shops is 153, an affordable value that can be considered for analysis purposes and definitely much more dependable than the averaged out order value

In [79]:
```
shopify_mini1['ov'].median()
```

Out[79]:
```
153.0
```

Mode OV :

- The Mode value represents the most common value.
- When it comes to an item at sale, the mode order value will provide us with the price of the most commonly purchased item.
- As a result, the Mode order value might really assist you in better assessing your sales by highlighting the most popular product or the most sought-after goods.
- This shopify blog (https://www.shopify.com.ng/blog/average-order-value#averageorder) quotes Taylor Holiday - "Look at your modal, or your most frequent, order values as a starting point for your efforts to increase your overall revenue".
- Modal values might be a crucial statistic for developing sales strategies and increasing income.

In [78]:
```
shopify_mini1['ov'].mode()
```

Out[78]:
```
0    153.0
dtype: float64
```

B. What metric would you report for this dataset?

If the goal of this analysis is to check potential ways to improve the company revenue, I would prefer reporting the Mode Order Value metric rather than Average Order Value for the reasons mentioned above.

C. What is its value?

$153

Problem 2:

For this question you'll need to use SQL. Follow this link (https://www.w3schools.com/SQL/TRYSQL.ASP?FILENAME=TRYSQL_SELECT_ALL) 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.

- How many orders were shipped by Speedy Express in total?
- What is the last name of the employee with the most orders?
- What product was ordered the most by customers in Germany?

How many orders were shipped by Speedy Express in total?

In [ ]:
```sql
# Query 1

SELECT COUNT(ShipperID) FROM Orders
WHERE ShipperID =
(SELECT ShipperID FROM Shippers
 WHERE ShipperName =
 'Speedy Express')
```

SQL Statement:

```sql
SELECT COUNT(ShipperID) FROM Orders WHERE ShipperID = (SELECT ShipperID FROM Shippers WHERE ShipperName = 'Speedy Express')
```

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

Run SQL »

Result:

Number of Records: 1

COUNT(ShipperID)
54

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

In [ ]:
```sql
# Query 2

SELECT LastName FROM Orders o, Employees e
WHERE o.EmployeeID = e.EmployeeID
GROUP BY o.EmployeeID
HAVING count(o.EmployeeID) = (SELECT COUNT(EmployeeID) as "occurence" from
                              GROUP BY EmployeeID
                              ORDER BY "occurence" DESC limit 1)
```

SQL Statement:

```sql
SELECT LastName from Orders o, Employees e
WHERE o.EmployeeID = e.EmployeeID
GROUP BY o.EmployeeID
HAVING count(o.EmployeeID) = (SELECT COUNT(EmployeeID) as "occurence" from Orders GROUP BY EmployeeID ORDER BY "occurence" DESC limit 1)
```

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

Run SQL »

Result:

Number of Records: 1

LastName
Peacock

What product was ordered the most by customers in Germany?

In [ ]:
```sql
# Query 3

SELECT ProductName, SUM(Quantity) FROM Products p, Orders o, OrderDetails o
WHERE Country = "Germany" and c.CustomerID = o.CustomerID and p.ProductID =
GROUP BY ProductName
ORDER BY SUM(Quantity)
DESC Limit 1;
```

SQL Statement:

```sql
SELECT ProductName, SUM(Quantity) FROM Products p, Orders o, OrderDetails od, Customers c WHERE Country = "Germany" and c.CustomerID = o.CustomerID and p.ProductID =
od.ProductID and o.OrderID = od.OrderID
GROUP BY ProductName
ORDER BY SUM(Quantity)
DESC Limit 1;
```

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

Run SQL »

Result:

Number of Records: 1

ProductName	SUM(Quantity)
Boston Crab Meat	160

In [ ]: