In [1]:
```
import pandas as pd
```

Question 1

> Data source: https://docs.google.com/spreadsheets/d/16i38oonuX1y1g7C_UAmiK9GkY7cS-64DfiDMNiR41LM/edit#gid=0
>
> Load the dataset in local, first quick glance at the dataset.

In [2]:
```
path = './2019 Winter Data Science Intern Challenge Data Set - Sheet1.csv'
df_raw = pd.read_csv(path)
```

In [3]:
```
df_raw.head(20)
```

Out[3]:

	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
5	6	58	882	138	1	credit_card	2017-03-14 15:25:01
6	7	87	915	149	1	cash	2017-03-01 21:37:57
7	8	22	761	292	2	cash	2017-03-08 2:05:38
8	9	64	914	266	2	debit	2017-03-17 20:56:50
9	10	52	788	146	1	credit_card	2017-03-30 21:08:26
10	11	66	848	322	2	credit_card	2017-03-26 23:36:40

11	12	40	983	322	2	debit	2017-03-12 17:58:30
12	13	54	799	266	2	credit_card	2017-03-16 14:15:34
13	14	100	709	111	1	cash	2017-03-22 2:39:49
14	15	87	849	447	3	credit_card	2017-03-10 11:23:18
15	16	42	607	704000	2000	credit_card	2017-03-07 4:00:00
16	17	17	731	176	1	cash	2017-03-21 4:23:38
17	18	28	752	164	1	credit_card	2017-03-21 12:09:07
18	19	83	761	258	2	cash	2017-03-17 13:18:47
19	20	63	898	408	3	credit_card	2017-03-29 15:11:52

In [4]:
```
len(df_raw)
```

Out[4]: 5000

> After taking a look at the dataset, I found out there might be some errors or outliers in this dataset. If the dataset is reliable, i.e., there are no errors in the dataset, then before further analysis, we probably should consider anomaly detection.
>
> For example, one unusual thing is for shop_id = 42 and user_id = 607, the shop sold 2000 pairs of sneakers on March 7th, 2017 and the total amount/value was 704000, and the same shop sold 2000 pairs of sneakers multiple times in the same month.These can be considered as outliers in our dataset.
>
> Another unusual thing that I noticed is that for shop_id = 78 and user_id = 760, the shop sold 2 pairs of shoes but the total amount/value was 51450. Since sneakers sold in these shops are relatively affordable items, my first guess is that it probably does not make sense that one pair of sneakers is 25725 dollars. However, there are other records for the same shop, and for both user_id = 867 and user_id = 912, one pair of sneakers was also sold for 25725 dollars. One possible reason is that there are some errors in this dataset, for example in shop 78, the price for each pair of sneakers is wrong. Another possible reason is that a few shops sell special edition sneakers and the prices for them are expensive. According to the given assumption: "Given that we know these shops are selling sneakers, a relatively affordable item, something seems wrong with our analysis. " We might consider that there are some (input record) errors for some shops in our dataset, for example, 25725 dollars per sneaker at shop 78 is the wrong data point.
>
> **Question 1(a):**
>
> The given calculation for an AOV is 3145.13 dollars, first let's find out how to get this number. Then we will be able to know where could be wrong.

In [5]:
```
total_value = df_raw['order_amount'].sum()
print(total_value)
naive_avg = total_value/len(df_raw)
naive_avg  #3145.128
```

```
15725640
```

Out[5]:
```
3145.128
```

> Now we know that 3145.13 is the result that we add all records in order_amount (15725640 dollars in total) and then divide it by the number of orders (5000 orders).
>
> Since there might be errors in this dataset as we mentioned before, we detect and remove the wrong data points before calcuting AOV.
>
> A better way we can do is to remove wrong data points first, then we group and analyze the dataset by different shops and calculate AOV. Here our assumption is that a relatively affordable prize for one pair of sneakers is in range [0, 500].

In [6]:
```
df_raw['price_per_pair'] = df_raw['order_amount']/df_raw['total_items']
df_raw
```

Out[6]:

	order_id	shop_id	user_id	order_amount	total_items	payment_method	created_at	p
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 × 8 columns

In [7]:

```
df = df_raw.copy(deep = True)
df
```

Out[7]:

	order_id	shop_id	user_id	order_amount	total_items	payment_method	created_at	p
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 × 8 columns

In [8]:

```
df = df.drop(df[df['price_per_pair'] >= 500].index)
df
```

Out[8]:

	order_id	shop_id	user_id	order_amount	total_items	payment_method	created_at	p
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	

4954 rows × 8 columns

In [9]:
```
df['order_id'].size
```

Out[9]:
```
4954
```

> **Question 1(b):**
>
> The metric we would report for this dataset is the average order value for each shop and the average of the average order value for each shop except for the shop(s) with wrong data. The reason why we calculate AOV by shops is that the AOV of shop 42 is an outlier, and when we do not want this outlier to make our result less accurate.

In [10]:
```
#convert int type to str type then use groupby for column shop_id in df
shop_list = df['shop_id'].tolist()

for i, ele in enumerate(shop_list):
    shop_list[i] = str(ele)

# type(shop_list[0])
df['shop_id'] = shop_list
```

In [11]:
```
groupby_shops_sum = df.groupby('shop_id', as_index = False)['order_amount'].s
groupby_shops_sum.head()
```

Out[11]:

	shop_id	order_amount
0	1	13588
1	10	17612
2	100	8547
3	11	17480
4	12	18693

In [12]:
```
groupby_shops_count = df.groupby('shop_id', as_index = False)['user_id'].size
groupby_shops_count.head()
```

Out[12]:

	shop_id	size
0	1	44
1	10	53
2	100	40
3	11	49
4	12	53

In [13]:
```
groupby_shops_avg = pd.DataFrame()
groupby_shops_avg['shop_id'] = groupby_shops_sum['shop_id']
groupby_shops_avg['total_order_value'] = groupby_shops_sum['order_amount']
groupby_shops_avg['total_number_of_orders'] = groupby_shops_count['size']
groupby_shops_avg['average_order_value(AOV)'] = groupby_shops_avg['total_orde
groupby_shops_avg.head()
```

Out[13]:

	shop_id	total_order_value	total_number_of_orders	average_order_value(AOV)
0	1	13588	44	308.818182
1	10	17612	53	332.301887
2	100	8547	40	213.675000
3	11	17480	49	356.734694
4	12	18693	53	352.698113

In [14]:
```
groupby_shops_avg
```

Out[14]:

	shop_id	total_order_value	total_number_of_orders	average_order_value(AOV)
0	1	13588	44	308.818182
1	10	17612	53	332.301887
2	100	8547	40	213.675000
3	11	17480	49	356.734694
4	12	18693	53	352.698113
...	...	...	...	...
94	95	12432	39	318.769231
95	96	16830	51	330.000000
96	97	15552	48	324.000000
97	98	14231	58	245.362069
98	99	18330	54	339.444444

99 rows × 4 columns

> The above dataframe groupby_shops_avg shows the the average order value (AOV) for each shop. Next, we calculate the average of AOV for each shop which is the answer for question 1(c).
>
> **Question 1(c)**
>
> The value of AOV is 299.68 dollars for 98 shops in March, 2017 except for shop 42 and shop 78.
>
> The AOV for shop 42 is 235101.490196 dollars in March, 2017.
>
> We do not have accurate AOV for shop 78 since we need to check again the price per pair of sneakers.

In [15]:
```
groupby_shops_avg = groupby_shops_avg.drop(groupby_shops_avg[groupby_shops_av
groupby_shops_avg
n = groupby_shops_avg['shop_id'].size #98
avg_aov = groupby_shops_avg['average_order_value(AOV)'].mean()
avg_aov
```

Out[15]: 299.6823991261548

Question 2

> **Question 2(a):**
>
> There are 54 orders were shipped by Speedy Express in total.
>
> Please refer SQL code below.

SELECT COUNT(o.OrderID)

FROM Orders AS o, Shippers AS s

WHERE s.ShipperName = "Speedy Express" AND o.ShipperID = s.ShipperID

GROUP BY o.ShipperID

> **Question 2(b):**
>
> The last name of the employee with the most orders is Peacock.
>
> Please refer SQL code below.

```
SELECT t2.LastName

FROM (SELECT EmployeeID,COUNT(*) AS a

FROM Orders

GROUP BY EmployeeID

HAVING COUNT(EmployeeID)

ORDER BY a DESC

LIMIT 1) AS t1, Employees AS t2

WHERE t1.EmployeeID = t2.EmployeeID
```

> **Question 2(c):**
>
> Boston Crab Meat was ordered the most by customers in Germany.
>
> Please refer SQL code below.

```
SELECT ProductName, Country, TotalCount

FROM(SELECT ProductId, Country, SUM(Quantity) AS TotalCount

FROM orderDetails AS od

LEFT JOIN(SELECT cc.Country, o.OrderID, o.CustomerId

FROM Orders AS o

LEFT JOIN(SELECT CustomerId, Country FROM Customers) cc

ON o.CustomerId = cc.CustomerId) occ

ON od.OrderID = occ.OrderID

GROUP BY Country, ProductId

HAVING Country='Germany') AS g

LEFT JOIN (SELECT ProductId, ProductName FROM Products) AS p

ON g.ProductId = p.ProductId

ORDER BY TotalCount DESC

LIMIT 1
```

In [ ]: