Shopify Fall 2022 Data Science Intern Challenge

By Anjali Chauhan

Date: 16/05/2022

Question 1:

Given some sample data, write a program to answer the following:
(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.

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

(ii) What metric would you report for this dataset?

(iii) What is its value?

Import libraries

In [1]:

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

Reading the data

In [2]:

```
df = pd.read_excel('shopify_data.xlsx')
df.head()
```

Out[2]:

	order_id	shop_id	user_id	order_amount	total_items	payment_method	created
0	1.0	53.0	746.0	224.0	2.0	cash	2017-03 12:36:56.190(
1	2.0	92.0	925.0	90.0	1.0	cash	2017-03 17:38:51.999
2	3.0	44.0	861.0	144.0	1.0	cash	2017-03 04:23:55.594
3	4.0	18.0	935.0	156.0	1.0	credit_card	2017-03 12:43:36.648
4	5.0	18.0	883.0	156.0	1.0	credit_card	2017-03 04:35:10.772

In [3]:

```
# Converting identifier values from float to int
df['order_id'] = df['order_id'].astype('int')
df['shop_id'] = df['shop_id'].astype('int')
df['user_id'] = df['user_id'].astype('int')
df['total_items'] = df['total_items'].astype('int')

# Calculating price of one item
df['item_price'] = (df.order_amount/df.total_items)
```

In [4]:

```
df.shape
```

Out[4]:

```
(5000, 8)
```

In [5]:

```
# Number of distinct sneaker shops
len(df.shop_id.unique())
```

Out[5]:

```
100
```

In [6]:

```
# Number of distinct order_id
len(df.order_id.unique())
```

Out[6]:

```
5000
```

Exploratory Data Analysis

In [7]:

```
# We observe that no null values are present in the df

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 8 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   order_id        5000 non-null   int64
 1   shop_id         5000 non-null   int64
 2   user_id         5000 non-null   int64
 3   order_amount    5000 non-null   float64
 4   total_items     5000 non-null   int64
 5   payment_method  5000 non-null   object
 6   created_at      5000 non-null   datetime64[ns]
 7   item_price      5000 non-null   float64
dtypes: datetime64[ns](1), float64(2), int64(4), object(1)
memory usage: 312.6+ KB
```

In [8]:

```
# Summary statistics of the following columns in the df

df[['order_amount','total_items','item_price']].describe()
```

Out[8]:

	order_amount	total_items	item_price
count	5000.000000	5000.00000	5000.000000
mean	3145.128000	8.78720	387.742800
std	41282.539349	116.32032	2441.963725
min	90.000000	1.00000	90.000000
25%	163.000000	1.00000	133.000000
50%	284.000000	2.00000	153.000000
75%	390.000000	3.00000	169.000000
max	704000.000000	2000.00000	25725.000000

In [9]:

```
# AOV calculated in the question:
#       Total revenue/number of orders

df.order_amount.sum()/len(df.order_id.unique())
```

Out[9]:

```
3145.128
```

- We see the average of `order_amount` is 3145.128 and that's how AOV was calculated in the question. We also see why this is not a robust metric.
- The max value of order_amount is 704000 with min value of 90 and 3rd quartile of 390 (way smaller than the max value).
- Since mean is very sensitive to extreme values, the calculated value is not a very good AOV estimate due to skewness and that's why this amount is large.
- Usually in this case, we could check for data quality and make sure we don't have any errors in the data. If the extreme data points are data entry errors we can drop these rows and use the mean to calculate AOV.
- If these extreme data points are not incorrect data points, we should not be dropping these rows as that would mean excluding important subset of information. We could use median instead which is robust to outliers. Let's investigate a bit more.

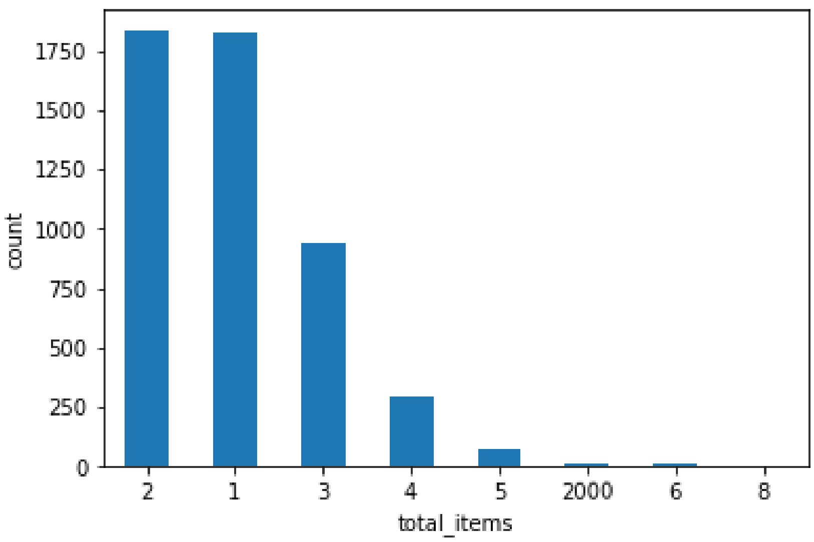
In [10]:

```
# Distribution of orders for different order sizes/total_items

fig1 = df.total_items.value_counts().plot.bar(rot = 1)
fig1.set_xlabel('total_items')
fig1.set_ylabel('count')
```

Out[10]:

```
Text(0, 0.5, 'count')
```



It's likely that these excessively large transactions (2000) are driving up the AOV.

In [11]:

```
# All orders with order_amount of 2000

df[df.total_items == 2000].sort_values('created_at', ascending = True)
```

Out[11]:

	order_id	shop_id	user_id	order_amount	total_items	payment_method	created_a
520	521	42	607	704000.0	2000	credit_card	2017-03 0 04:00:0
4646	4647	42	607	704000.0	2000	credit_card	2017-03 0 04:00:0
60	61	42	607	704000.0	2000	credit_card	2017-03 0 04:00:0
15	16	42	607	704000.0	2000	credit_card	2017-03 0 04:00:0
2297	2298	42	607	704000.0	2000	credit_card	2017-03 0 04:00:0
1436	1437	42	607	704000.0	2000	credit_card	2017-03 1 04:00:0
2153	2154	42	607	704000.0	2000	credit_card	2017-03 1 04:00:0
1362	1363	42	607	704000.0	2000	credit_card	2017-03 1 04:00:0
1602	1603	42	607	704000.0	2000	credit_card	2017-03 1 04:00:0
1562	1563	42	607	704000.0	2000	credit_card	2017-03 1 04:00:0
4868	4869	42	607	704000.0	2000	credit_card	2017-03 2 04:00:0
3332	3333	42	607	704000.0	2000	credit_card	2017-03 2 04:00:0
1104	1105	42	607	704000.0	2000	credit_card	2017-03 2 04:00:0
4882	4883	42	607	704000.0	2000	credit_card	2017-03 2 04:00:0
2835	2836	42	607	704000.0	2000	credit_card	2017-03 2 04:00:0
2969	2970	42	607	704000.0	2000	credit_card	2017-03 2 04:00:0
4056	4057	42	607	704000.0	2000	credit_card	2017-03 2 04:00:0

- We see that all of the orders of size 2000 occurred from the same user_id: 607 and from the same shop_id: 42.
- The order_amount is exactly the same too for each of these orders about 704000.
- Here's one instance of multiple identical transactions: From the first 5 rows, we can see that all the orders were made at exactly 4AM. on 2017-03-02.
- This might be a case of duplicate entries in the dataset, or this might just be the case where the customer might be buying sneakers in bulk.

In [12]:

```
# Average order amount for each shop (lowest 5)
df.groupby('shop_id')[['order_amount']].mean().sort_values('order_amount', a
scending = True).head()
```

Out[12]:

shop_id	order_amount
92	162.857143
2	174.327273
32	189.976190
100	213.675000
53	214.117647

In [13]:

```
# 5 shops based on LEAST total revenue
df.filter(['shop_id', 'order_amount'])\
        .groupby('shop_id')\
        .sum('order_amount')\
        .sort_values('order_amount', ascending = True)\
        .head(5)
```

Out[13]:

shop_id	order_amount
92	6840.0
32	7979.0
56	8073.0
100	8547.0
2	9588.0

In [14]:

```
# Average order amount for each shop (highest 5)
df.groupby('shop_id')[['order_amount']].mean().sort_values('order_amount', a
scending = False).head()
```

Out[14]:

	order_amount
shop_id	
42	235101.490196
78	49213.043478
50	403.545455
90	403.224490
38	390.857143

In [15]:

```
# Top 5 shops based on MOST total revenue
df.filter(['shop_id', 'order_amount'])\
        .groupby('shop_id')\
        .sum('order_amount')\
        .sort_values('order_amount', ascending = False)\
        .head(5)
```

Out[15]:

	order_amount
shop_id	
42	11990176.0
78	2263800.0
89	23128.0
81	22656.0
6	22627.0

Interesting observation: We see some inconsistency in the top 5 and the bottom 5 shops based on their average order amount and total revenue.

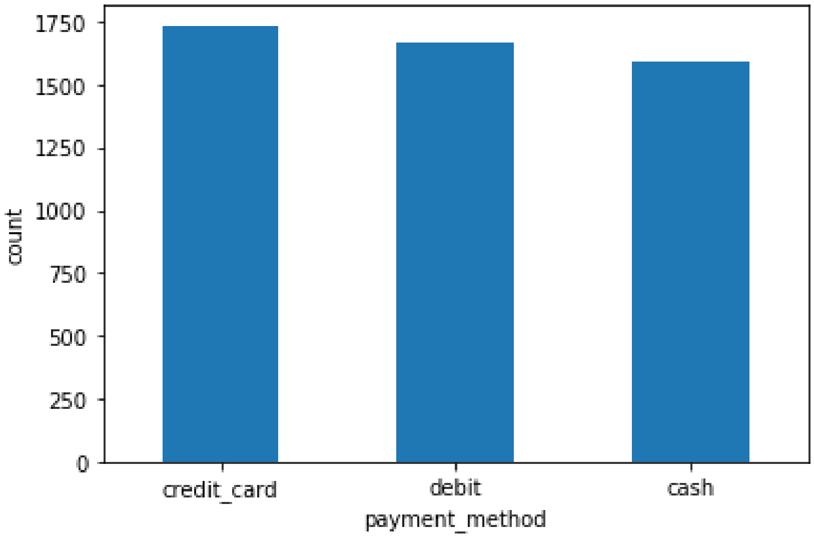
In [16]:

```
# Distribution of payment_method
# No imbalance

fig2 = df.payment_method.value_counts().plot.bar(rot = 0)
fig2.set_xlabel('payment_method')
fig2.set_ylabel('count')
```

Out[16]:

```
Text(0, 0.5, 'count')
```



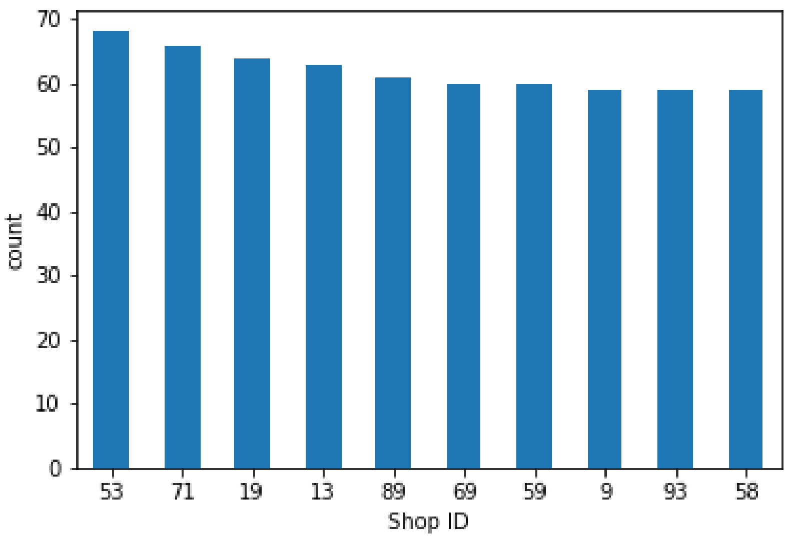
In [17]:

```
# Distribution of number of orders from the top 10 Shops
# No imbalance

fig3 = df.shop_id.value_counts()[0:10].plot.bar(rot = 1)
fig3.set_xlabel('Shop ID')
fig3.set_ylabel('count')
```

Out[17]:

```
Text(0, 0.5, 'count')
```



In [18]:

```
# Top 5 most expensive orders
df.filter(['order_id', 'order_amount'])\
        .groupby('order_id')\
        .sum('order_amount')\
        .sort_values('order_amount', ascending = False)\
        .head(5)
```

Out[18]:

	order_amount
order_id	
2154	704000.0
3333	704000.0
521	704000.0
1603	704000.0
61	704000.0

In [19]:

```
# 5 least expensive orders
df.filter(['order_id', 'order_amount'])\
        .groupby('order_id')\
        .sum('order_amount')\
        .sort_values('order_amount', ascending = True)\
        .head(5)
```

Out[19]:

	order_amount
order_id	
159	90.0
3872	90.0
4761	90.0
4924	90.0
4933	90.0

In [20]:

```
# Order history of shop 78
df[df.shop_id == 78].sort_values('order_amount',ascending = False)['order_am
ount'].value_counts()
```

Out[20]:

```
25725.0     19
51450.0     16
77175.0      9
154350.0     1
102900.0     1
Name: order_amount, dtype: int64
```

In [21]:

```
# Order history of shop 42
df[df.shop_id == 42].sort_values('order_amount',ascending = False)['order_am
ount'].value_counts()
```

Out[21]:

```
704000.0    17
352.0       15
704.0       13
1056.0       3
1408.0       2
1760.0       1
Name: order_amount, dtype: int64
```

In [22]:

```
new_df = df.groupby(['total_items']).agg({'order_amount': ['mean','max']})
new_df.columns = new_df.columns.droplevel(0)
new_df = new_df.reset_index()
new_df = new_df.rename(columns = {'mean': 'avg_order_amt',
                                  'max' : 'max_order_amt'})

new_df['per_item_amt'] = new_df.max_order_amt/new_df.total_items
new_df
```

Out[22]:

	total_items	avg_order_amt	max_order_amt	per_item_amt
0	1	417.364481	25725.0	25725.0
1	2	750.215066	51450.0	25725.0
2	3	1191.076514	77175.0	25725.0
3	4	947.686007	102900.0	25725.0
4	5	759.350649	1760.0	352.0
5	6	17940.000000	154350.0	25725.0
6	8	1064.000000	1064.0	133.0
7	2000	704000.000000	704000.0	352.0

Based on the df above, we see some of these max_order_amt are way above the average. We would never expect an average pair of shoes to cost 25725. This in addition to the duplicate transactions questions the correctness of these entries.

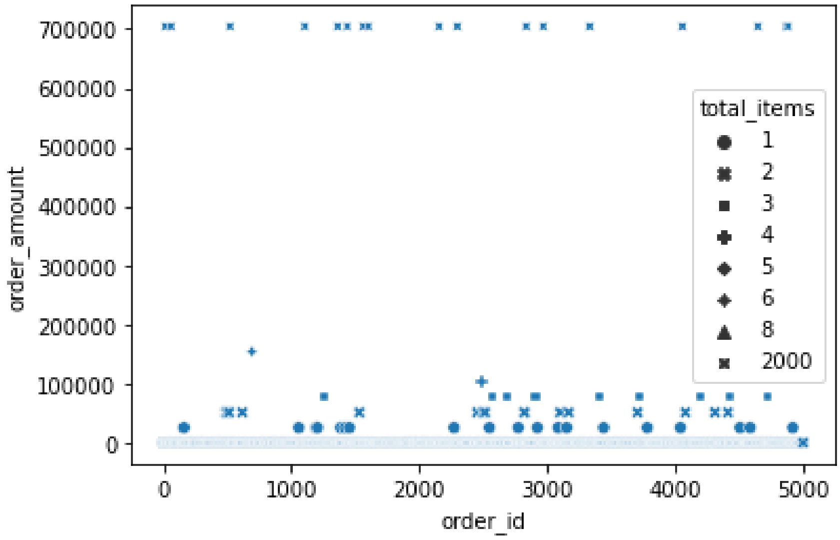
In [23]:

```
sns.scatterplot(data=df, x="order_id", y="order_amount",
                style="total_items",palette="deep")

# Here every point is one transaction
```

Out[23]:

```
&lt;matplotlib.axes._subplots.AxesSubplot at 0x7ff4ab8a7130&gt;
```



After a thorough exploration, we can say that the best option would be to use median as a metric.

In [24]:

```
df.order_amount.median()
```

Out[24]:

```
284.0
```

Question 2

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

(i) How many orders were shipped by Speedy Express in total?

Answer: 54 orders were shipped by Speedy Express in total

```
SELECT o.OrderID, o.ShipperID, s.ShipperID, s.ShipperName, COUNT(*) FR
OM Orders o
LEFT JOIN Shippers s ON o.ShipperID = s.ShipperID
WHERE s.ShipperName = 'Speedy Express'
```

(ii) What is the last name of the employee with the most orders?

Answer: Peacock is the last name of the employee with the most orders, about 40

```
SELECT e.LastName, COUNT(DISTINCT(o.OrderID)) AS total_orders FROM Emp
loyees e
LEFT JOIN Orders o
ON e.EmployeeID =  o.EmployeeID
GROUP BY e.LastName
ORDER BY  total_orders DESC
LIMIT 1
```

(iii) What product was ordered the most by customers in Germany?

Answer: Camembert Pierrot was the product ordered the most by customers in Germany.

```
WITH ProductQuantity AS (
    SELECT Orders.OrderID, Products.ProductName, OrderDetails.Quantity
    FROM OrderDetails, Orders
    JOIN Customers ON Customers.CustomerID = Orders.CustomerID
    JOIN Products ON OrderDetails.ProductID=Products.ProductID
    WHERE Country =='Germany'
),

ProductOrders as (
    SELECT ProductName, Quantity, COUNT(*) as 'Orders'
    FROM ProductQuantity
    GROUP BY ProductName
)

SELECT ProductName, Quantity, Orders, (Orders*Quantity) AS TotalQuanti
tyOrdered
FROM ProductOrders
ORDER BY TotalQuantityOrdered DESC
LIMIT 1;
```

ProductName	Quantity	Orders	TotalQuantityOrdered
Camembert Pierrot	40	300	12000

In [ ]: