Shopify

May 22, 2022

1 Question 1

1.1 Introduction

Like any other python project, we begin by importing some important libraries and modules:

[2]:
```
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
```

We begin by reading the data:

[3]:
```
d1 = pd.read_csv("2019-Winter-Data-Science-Intern-Challenge-Data-Set-Sheet1.
  ↪csv")
d1.head(2)
```

[3]:
```
   order_id  shop_id  user_id  order_amount  total_items payment_method  \
0         1       53      746           224            2           cash
1         2       92      925            90            1           cash

            created_at
0  2017-03-13 12:36:56
1  2017-03-03 17:38:52
```

1.2 Exploratory Data Analysis:

It is a good practice to conducta basic exploratory analysis to get a better sense of data.

[4]:
```
d1.size # there are 35000 data entries
```

[4]:
```
35000
```

[5]:
```
d1.describe()# gives general statistics such as count, mean, standard␣
  ↪deivaiation, minimum, maximum value etc.
```

[5]:
```
          order_id      shop_id      user_id  order_amount  total_items
count  5000.000000  5000.000000  5000.000000   5000.000000   5000.00000
mean   2500.500000    50.078800   849.092400   3145.128000      8.78720
std    1443.520003    29.006118    87.798982  41282.539349    116.32032
```

```
min       1.000000     1.000000   607.000000      90.000000      1.00000
25%    1250.750000    24.000000   775.000000     163.000000      1.00000
50%    2500.500000    50.000000   849.000000     284.000000      2.00000
75%    3750.250000    75.000000   925.000000     390.000000      3.00000
max    5000.000000   100.000000   999.000000  704000.000000   2000.00000
```

order_amount seems to have the an unusually high standard deviation. the range is wide from 90 units to 704000 units. We could use this information later.

[6]:
```
d1.count()
```

[6]:
```
order_id          5000
shop_id           5000
user_id           5000
order_amount      5000
total_items       5000
payment_method    5000
created_at        5000
dtype: int64
```

[8]:
```
d1.isna().sum() # No null Values.
```

[8]:
```
order_id          0
shop_id           0
user_id           0
order_amount      0
total_items       0
payment_method    0
created_at        0
dtype: int64
```

[9]:
```
len(d1) # 5000 rows
```

[9]:
```
5000
```

[10]:
```
d1.mean() # all the means
```

```
C:\Users\prate\anaconda3\envs\shopify\lib\site-packages\ipykernel_launcher.py:1:
FutureWarning: Dropping of nuisance columns in DataFrame reductions (with
'numeric_only=None') is deprecated; in a future version this will raise
TypeError.  Select only valid columns before calling the reduction.
  """Entry point for launching an IPython kernel.
```

[10]:
```
order_id        2500.5000
shop_id           50.0788
user_id          849.0924
order_amount    3145.1280
total_items        8.7872
```

```
dtype: float64
```

Notice the average or order_amount. it is 3145.1280. The average of order_amount was considered as AOV (average order value). This is not considered an accurate calculator of AOV.

A more accurate approach for AOV = sum of order_amount / sum of order_amount

There are a couple of approaches that can be considered. * 1 - We could calculate the total order_amount and divide it by the sum of total_items. * 2 - We could also calculate the individual averages(create a new column avg_aovs = order_amount/total_amount) and then futher take the average of this new column. When we say 'averages' we could do 2 things: — 2a - mean of the avg_aovs — 2b - median of the avg_aovs

1.3 Analysis and Observation

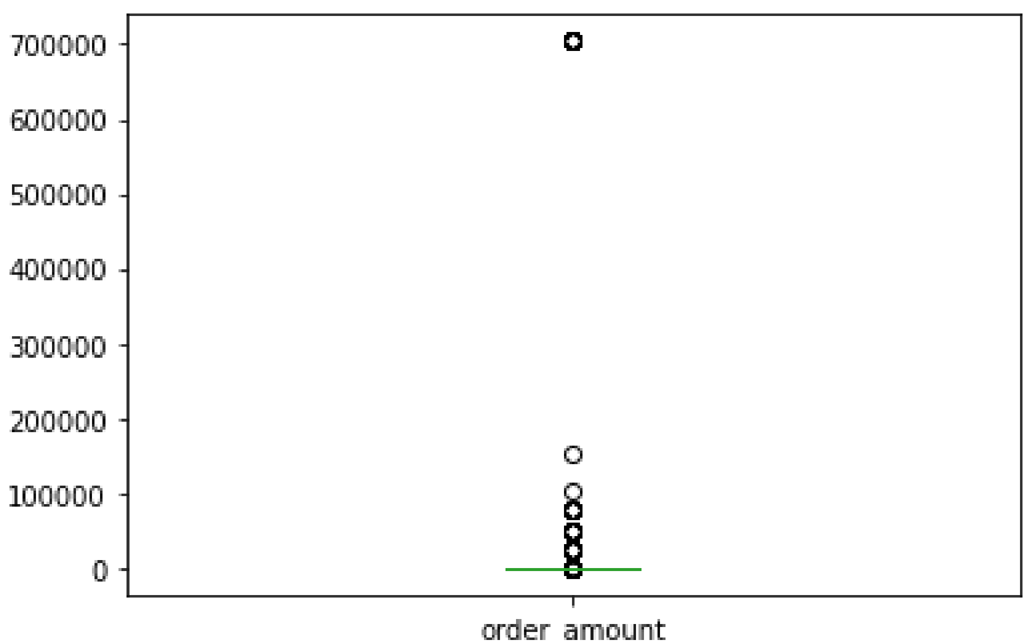
[11]:
```
AOV_1 = d1['order_amount'].sum() / d1['total_items'].sum()
AOV_1
```

[11]:
```
357.92152221412965
```

this is without tkaing outliers into consideration. Let us look for outliers in the order_amount and total_items features.

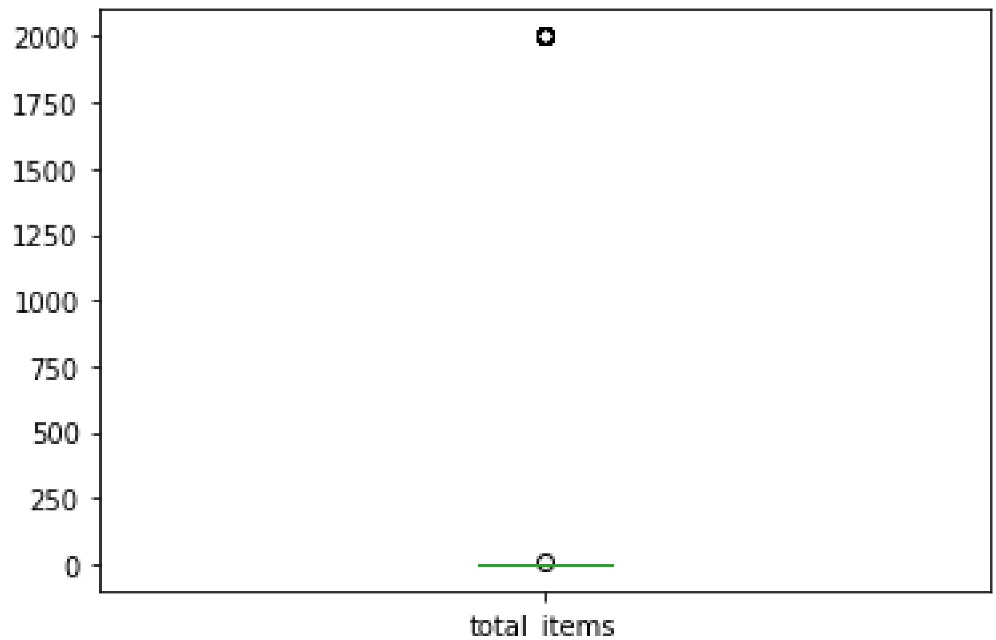
[12]:
```
# boxplots for order_amount and total_items
d1['order_amount'].plot(kind = 'box')
```

[12]:
```
<AxesSubplot:>
```



[13]:
```
d1['total_items'].plot(kind = 'box')
```

[13]: <AxesSubplot:>



The boxplots suggests presence of outliers. Its not a great practice to simply remove outliers as they can potentially showcase interesting insights of the situation. It is a good practice to always first investigate the outliers and then make a more informed decision as to how to preceed further. For the sake of this project, since we have limited information, let us attempt to remove the outliers and see.

We define a function 'outliers' which we will use to create a list of indcies that contain outliers for the respective feature. We are developing a function since we might need to perform this a number of times. this function will return a list of indicies that contain the outlier for respective feature.

In addition we develop another function, 'remove'. this function, will take returned list from 'outliers' function and return a clean dataframe without the outliers.

Observations that are significantly away from the rest of the data are called outliers. Generally, values beyond 3 standard deviations are considered oultier. We will follow this rule of thumb for this project.

[17]:
```
def outliers(df, ft):
    Q1 = df[ft].quantile(0.25)    # defining the 1st quantile
    Q3 = df[ft].quantile(0.75)    # defining the 3rd quantile
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR  # 1.5 + 1.5 = 3 standard deviation
    upper_bound = Q3 + 1.5 * IQR
```

```
    # getting the indexes              #  OR operater
    ls = df.index[(df[ft] < lower_bound) | (df[ft] > upper_bound)]
    # anything lower than the lower bound OR anything greater than the upper␣
 ↪bound

    return ls # list

def remove(df, ls):       # ls -> index list
    ls = sorted(set(ls)) # to get unique indices in ascending order.
    df2 = df.drop(ls)    # drop the respective indicies, df2 is the clean␣
 ↪dataframe.
    return df2
```

[19]:
```
ind_ls_1 = []
for i in ['order_amount','total_items']:
    ind_ls_1.extend(outliers(d1, i))
ind_ls_1 # these are the indices of the outliers
d1_clean = remove(d1, ind_ls_1)
d1_clean
```

[19]:

	order_id	shop_id	user_id	order_amount	total_items	payment_method	\
0	1	53	746	224	2	cash	
1	2	92	925	90	1	cash	
2	3	44	861	144	1	cash	
3	4	18	935	156	1	credit_card	
4	5	18	883	156	1	credit_card	
…	…	…	…	…	…	…	
4995	4996	73	993	330	2	debit	
4996	4997	48	789	234	2	cash	
4997	4998	56	867	351	3	cash	
4998	4999	60	825	354	2	credit_card	
4999	5000	44	734	288	2	debit	

	created_at
0	2017-03-13 12:36:56
1	2017-03-03 17:38:52
2	2017-03-14 4:23:56
3	2017-03-26 12:43:37
4	2017-03-01 4:35:11
…	…
4995	2017-03-30 13:47:17
4996	2017-03-16 20:36:16
4997	2017-03-19 5:42:42
4998	2017-03-16 14:51:18
4999	2017-03-18 15:48:18

```
[4859 rows x 7 columns]
```

[21]:
```
aov2 = d1_clean['order_amount'].sum() / d1_clean['total_items'].sum()
aov2
```

[21]:
```
150.60816800337696
```

[22]:
```
aov3 = d1_clean['order_amount'].mean() / d1_clean['total_items'].mean()
aov3
```

[22]:
```
150.60816800337696
```

[24]:
```
# aov4 = d1_clean['order_amount'].mode() / d1_clean['total_items'].mode()
# aov4
```

[24]:
```
0    76.5
dtype: float64
```

Another approach is to calcualate respective averages and then calculate the average of the average order value. for this we need to create a new column which is the ratio of order_amount and total_items. We will name the new column 'avg'.

it is generally prefered that the original imported datafame should not be manipulated. Since we are attempting to create a new column avg_aovs, let us first create a copy of the original dataframe and work on that one.

[27]:
```
d2 = d1.copy()
d2.head(2)
```

[27]:
```
   order_id  shop_id  user_id  order_amount  total_items payment_method  \
0         1       53      746           224            2           cash
1         2       92      925            90            1           cash

            created_at
0  2017-03-13 12:36:56
1  2017-03-03 17:38:52
```

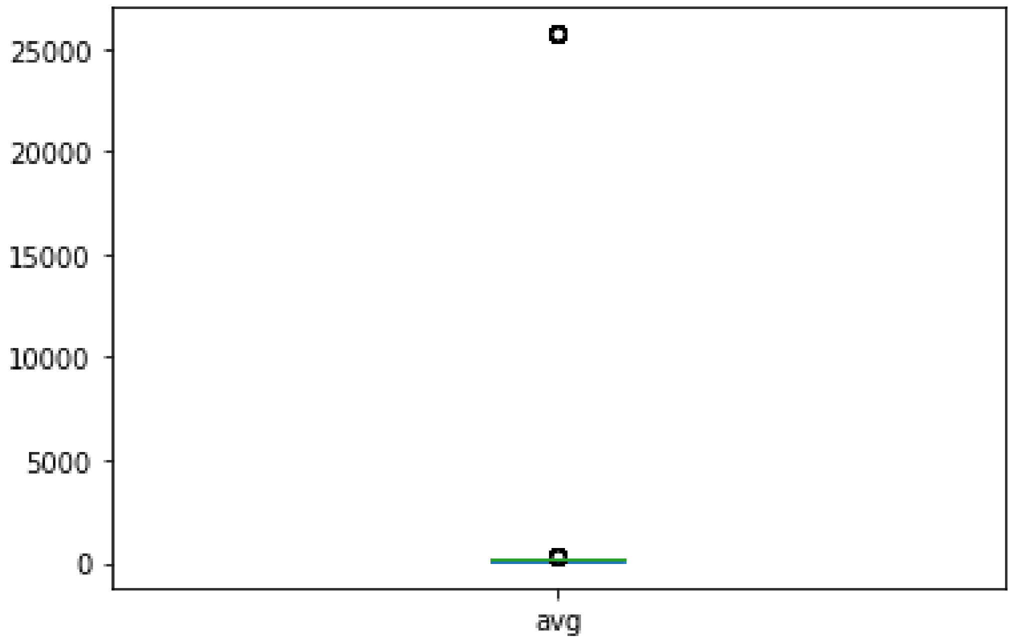
[28]:
```
d2['avg'] = d2['order_amount'] / d2['total_items']
d2.head(2)
```

[28]:
```
   order_id  shop_id  user_id  order_amount  total_items payment_method  \
0         1       53      746           224            2           cash
1         2       92      925            90            1           cash

            created_at    avg
0  2017-03-13 12:36:56  112.0
1  2017-03-03 17:38:52   90.0
```

[30]:
```python
# let us look for outliers of avg column. and attempt to remove them
d2['avg'].plot(kind = 'box')
```

[30]: <AxesSubplot:>



[31]:
```python
# removing outliers:
ind_l2_2 = []
ind_ls_2.extend(outliers(d2, 'avg'))
ind_ls_2 # these are the indices of the outliers
d2_clean = remove(d2, ind_ls_2)
d2_clean
```

[31]:

	order_id	shop_id	user_id	order_amount	total_items	payment_method \
0	1	53	746	224	2	cash
1	2	92	925	90	1	cash
2	3	44	861	144	1	cash
3	4	18	935	156	1	credit_card
4	5	18	883	156	1	credit_card
…	…	…	…	…	…	…
4995	4996	73	993	330	2	debit
4996	4997	48	789	234	2	cash
4997	4998	56	867	351	3	cash
4998	4999	60	825	354	2	credit_card
4999	5000	44	734	288	2	debit

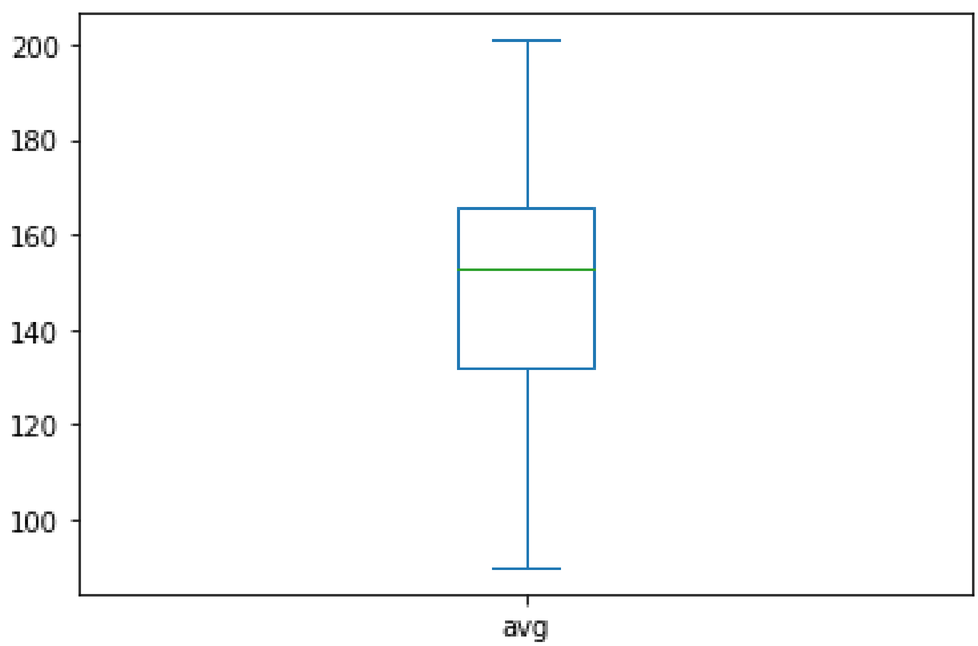
created_at avg

```
0     2017-03-13 12:36:56  112.0
1     2017-03-03 17:38:52   90.0
2      2017-03-14 4:23:56  144.0
3     2017-03-26 12:43:37  156.0
4      2017-03-01 4:35:11  156.0
…                    …     …
4995  2017-03-30 13:47:17  165.0
4996  2017-03-16 20:36:16  117.0
4997   2017-03-19 5:42:42  117.0
4998  2017-03-16 14:51:18  177.0
4999  2017-03-18 15:48:18  144.0

[4903 rows x 8 columns]
```

[32]:
```python
d2_clean['avg'].plot(kind = 'box')
```

[32]:
```
<AxesSubplot:>
```



[34]:
```python
aov5 = d2_clean['avg'].mean()
aov5
```

[34]:
```
150.40016316540894
```

[36]:
```python
aov6 = d2_clean['avg'].median()
aov6
```

[36]: 153.0

[37]:
```
# aov7 = d2_clean['avg'].mode()
# aov7
```

[37]:
```
0    153.0
dtype: float64
```

1.4 Conclusion:

We are only considering mean and median since the feature is quantitative. Had the feature been qualitative, we would have considered mode.

Based on our analysis, $150 dollar seem to be a more accurate Average of value for the given situation.

2 Question 2

2.0.1 Part A: How many orders were shipped by speedy express in total?

Orders table contain details of orders. This is our main table. We need to count all the orders shipped by 'Speedy Express'. The table consists of ShipperID. We need to identify which Shipper ID represents Speedy Express. That detail is in the 'Shipper' table. We join the From 'Shippers' table, we get that ShipperID = 1 is for Speedy Express.

[42]:
```
'''
SELECT ShipperName , COUNT(*)
    FROM [Orders]
    JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID
    WHERE ShipperName = 'Speedy Express';
'''
```

[42]:
```
"\nSELECT ShipperName , COUNT(*)\n    FROM [Orders]\n    JOIN Shippers ON
Orders.ShipperID = Shippers.ShipperID\n    WHERE ShipperName = 'Speedy
Express';\n"
```

ANSWER: Speedy Express shipped a total of **54 orders**.

2.0.2 Part B: What is the last name of the employee with the most orders?

This time we need to extract the last name of the employee. This information is available in 'Employees' table. The table also contains the 'EmployeeID' of the respective employee. We will match (join) it with the EmployeeID from 'Orders' table.

[44]:
```
'''
SELECT LastName, COUNT(LastName)
    FROM [Orders]
    JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID
    GROUP BY LastName
```

```
    ORDER BY COUNT(LastName) DESC
    LIMIT 1

'''
```

[44]:
```
'\nSELECT LastName, COUNT(LastName)\n    FROM [Orders]\n    JOIN Employees ON
Orders.EmployeeID = Employees.EmployeeID\n    GROUP BY LastName\n    ORDER BY
COUNT(LastName) DESC\n    LIMIT 1\n    \n'
```

ANSWER: The last name of the employee with the most orders is **'Peacock'**

2.0.3 Part C: What product was ordered the most by customers in Germany?

For this questions, we need to merge a number of tables since different tables contain different interconnected information. In total, we used Orders, OrderDetails, Products and Customers tables. We selected only Germany. Since we needs, maximum orders per country, we will group the ProductNames and add the quantity. To get the maximum Orders, we will sort the sum in descending order and extract only the 1st row.

[46]:
```
'''
SELECT ProductName, SUM(Quantity)
    FROM [Orders]
    JOIN OrderDetails ON Orders.OrderID = OrderDetails.OrderID
    JOIN Products ON Products.ProductID = OrderDetails.ProductID
    JOIN Customers ON Orders.CustomerID = Customers.CustomerID
    WHERE Country = 'Germany'
    GROUP BY ProductName
    ORDER BY SUM(Quantity) DESC
    LIMIT 1
'''
```

[46]:
```
"\nSELECT ProductName, SUM(Quantity)\n    FROM [Orders]\n    JOIN OrderDetails
ON Orders.OrderID = OrderDetails.OrderID\n    JOIN Products ON
Products.ProductID = OrderDetails.ProductID\n    JOIN Customers ON
Orders.CustomerID = Customers.CustomerID\n    WHERE Country = 'Germany'\n
GROUP BY ProductName\n    ORDER BY SUM(Quantity) DESC\n    LIMIT 1\n"
```

ANSWER: **Boston Crab Meat** was ordered the most by customers in Germany with a grand total of 160 orders.

[ ]:
```
```