The Process of Sales Analysis using Python

by Trung Bui

22.02.2023

This is the document of the process of sales analysis using Python and Pandas library for the case study.

Import necessary Libraries:

We import the pandas library for data manipulation, cleaning and analysis. After that, we import OS to gain access to the computer's file system, in order to read the files.

```
import pandas as pd
import os
```

Merging 12 months of sales data into a single CSV file:

We want to work with one unified file; hence we must merge the 12 files together:

- Firstly, we need to find the path to the files.
- Then we make a list with a list comprehension to go through each file and put them in this new list of files.
- We create an object 'all_months_data' and assign this object as the data frame.
- Then we go through each file again, this time read each file and to add them all in to a temporary data frame call 'df'.
- After that, we concatenate them in to one single data frame all_months_data.
- Finally, we convert this new concatenated data frame into a csv file call 'all_data.csv'.

```
files = [file for file in os.listdir('/Users/trungbui0177/Desktop/Data-Analyst/
/SalesAnalysis/Sales_Data')]

all_months_data = pd.DataFrame()

for file in files:
    df = pd.read_csv('/Users/trungbui0177/Desktop/Data-
Analyst/SalesAnalysis/Sales_Data/'+file)
    all_months_data = pd.concat([all_months_data, df])

all_months_data.to_csv('all_data.csv', index=False)
```

Read in updated data frame:

We need to assign the newly form ‘all_data’ to the csv file that we have created above and make sure everything is correct.

```
all_data = pd.read_csv("all_data.csv")
all_data.head()
```

Cleaning up data:

1. Drop rows of NAN:
   We remove the rows where no data was collected (NAN).

```
nan_df = all_data[all_data.isna().any(axis=1)]
all_data = all_data.dropna(how='all')
```

2. Find ‘Or’ and delete it:
   Since we found some Month that has ‘0r’ in it, maybe an error or missing data, so we must leave these out of ‘all_data’.

```
all_data = all_data[all_data['Order Date'].str[0:2] != 'Or']
```

3. Convert columns to the correct types:
   Some values series is not recognized as number, so we need to turn them in to numeric types (either as Integer or Float).

```
all_data['Quantity Ordered'] = pd.to_numeric(all_data['Quantity Ordered'])
all_data['Price Each'] = pd.to_numeric(all_data['Price Each'])
```

Augment data with additional columns:

As we are answering the questions, we will need to add more column which serve some purpose in the analysis.

1. Add ‘Month’ column (Question 1):
   By reading the csv data we see that the date format of this file is MM/DD/YY, so we need to extract the first two indexes of the ‘Order Date’ column to form a ‘Month’ column.

```
all_data['Month'] = all_data['Order Date'].str[0:2]
all_data['Month'] = all_data['Month'].astype('int32')
```

2. Add ‘Sales’ column (Question 2):
   We need a total number of each item that we’ve sold, because we only have the price for each item and how many time, they were sold. Now we need to multiply them together to get the total sales.

```
all_data['Sales'] = all_data['Quantity Ordered'] * all_data['Price Each']
```

3. Add ‘City’ column (Question 2):
   We need to extract from the ‘Purchase address’ column the cities’ name and since there could be repeated name for a city in different states, we need to get the states as well.

```
get_city(adress):
    return adress.split(',')[1]

def get_state(adress):
    return adress.split(',')[2].split(' ')[1]

all_data['City'] = all_data['Purchase Address'].apply(lambda x: f"{get_city(x)}
({get_state(x)})")
```

4. Add ‘Hour’ column (Question 3):
   Since we must work with date and time in this question, we need to set them into the right format for Pandas. This will give us a more comprehensive view of the date and time correct to the index to use later. We need the information of date time in the ‘Order Date’ column and create a new ‘Hour’ column base on this information.

```
all_data['Order Date'] = pd.to_datetime(all_data['Order Date'])
all_data['Hour'] = all_data['Order Date'].dt.hour
```

Business Questions:

Question 1: What was the best month for sales? How much did we earn?

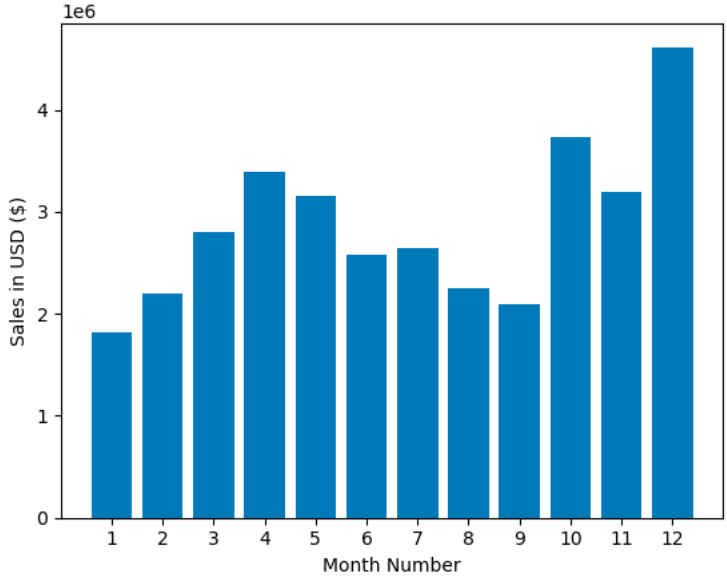
Firstly, we need to get a sum of sales for each month and then group them into a temporal table.

```
sum_month = all_data.groupby('Month').sum()
```

Then we import matpltlib.pyplot to plot our results to a more readable chart of bars.

```
import matplotlib.pyplot as plt
months = range(1,13)
plt.bar(months, sum_month['Sales'])
plt.xticks(months)
plt.ylabel('Sales in USD ($)')
plt.xlabel('Month Number')
```

After implement these we have a chart as follow:



Conclusion, we have the highest sales in December, maybe because of holidays, end of year sales, … If we want to know exactly why we would need to collect more data.

Question 2: Which city had the best sales?

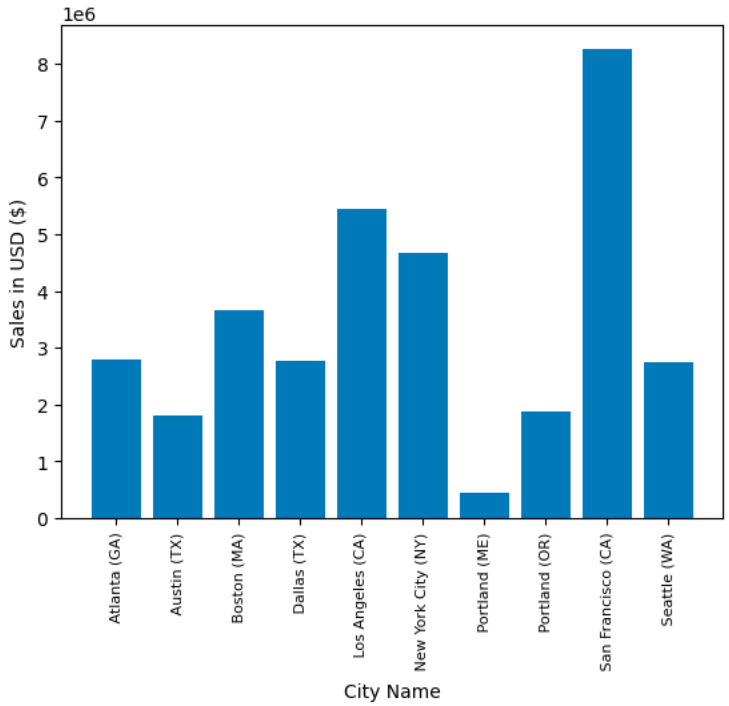
We need this information find out what we have done right to get such high sales and set example to replicate these in other cities.

First, we need the total of sales sorted by each city.

```
sum_ciy = all_data.groupby('City').sum()
```

Then we can draw a chart using the same method we have used for Question 1, but instead of using monthly sales, we will use cities' sales as parameter for our chart.

```
cities = [city for city, df in all_data.groupby('City')]

plt.bar(cities, sum_city["Sales"])
plt.xticks(cities, rotation='vertical', size=8)
plt.ylabel('Sales in USD ($)')
plt.xlabel('City Name')

plt.show()
```

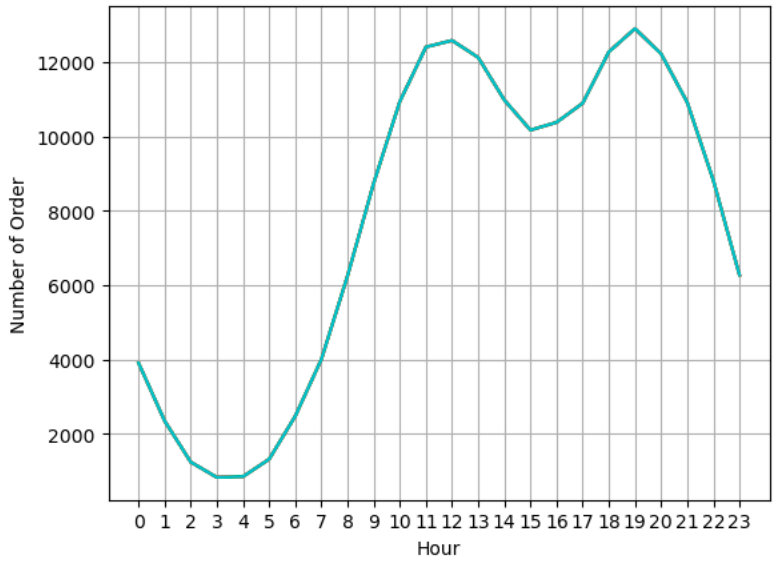


Conclusion, San Francisco has the best sales of all the cities. It could be that people in this city have bigger spending compacity as others, we need extra data on that to prove this. If this is true, then we need to make this city our priority and focusing on pushing products.

Question 3: What time should we display advertisements to maximize likelihood of customer buying the products?

Since we must work with date and time in this question, we need to set them into the right format for Pandas. This will give us a more comprehensive view of the date and time correct to the index to use later.
Finally, we do the same as for other chart, but we need to add a grid for visual and count the number of sales according to each hour.

```
hours = [hour for hour, df in all_data.groupby('Hour')]

plt.xticks(hours)
plt.grid()
plt.ylabel('Number of Order')
plt.xlabel('Hour')
plt.plot(hours, all_data.groupby(['Hour']).count())

plt.show()
```



Conclusion, based on the chart my recommendation is around 11am (11) or 7pm (19).

Question 4: What products are most often sold together?

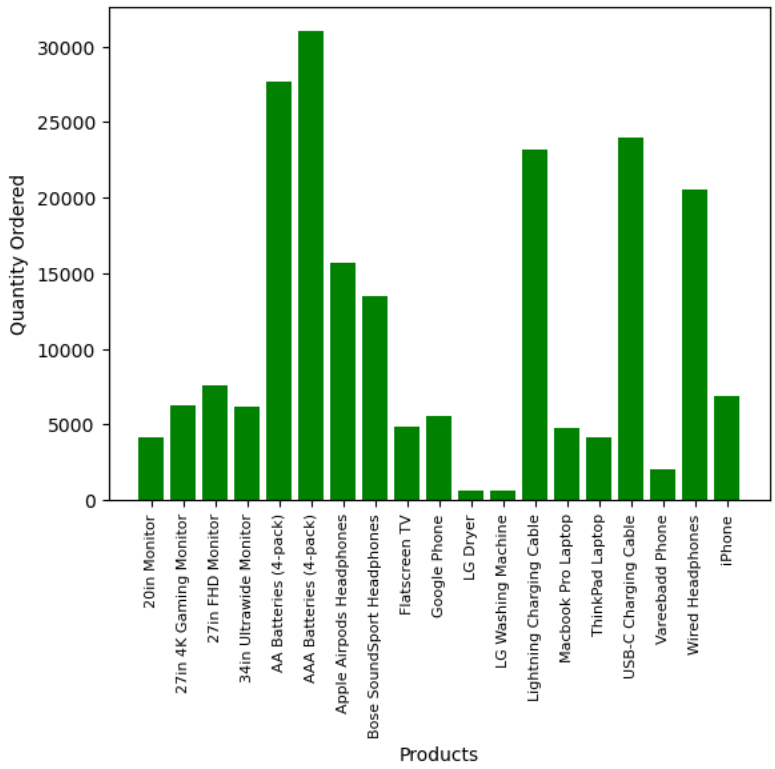
To identify the products that are sold together in an order, we need to sort out the order ID that two or more 'purchased orders' shared. Then we create a temporal data frame to test and see if we gave what we want. This data frame should be group by their Order ID and contain the item that matched each similar ID. After that we need to extract from the data frame how many times, they were sold together in order to come up with the right business strategy.

```
from itertools import combinations
from collections import Counter

df = all_data[all_data['Order ID'].duplicated(keep=False)]
df['Grouped'] = df.groupby('Order ID')['Product'].transform(lambda x: ','.join(x))
df = df[['Order ID', 'Grouped']].drop_duplicates()

count = Counter()
for row in df['Grouped']:
    row_list = row.split(',')
    count.update(Counter(combinations(row_list, 2))) #change the parameter here for
the 3,4 items

for key, value in count.most_common(20):
    print(key, value)
```

Question 5: Which product sold the most and why?

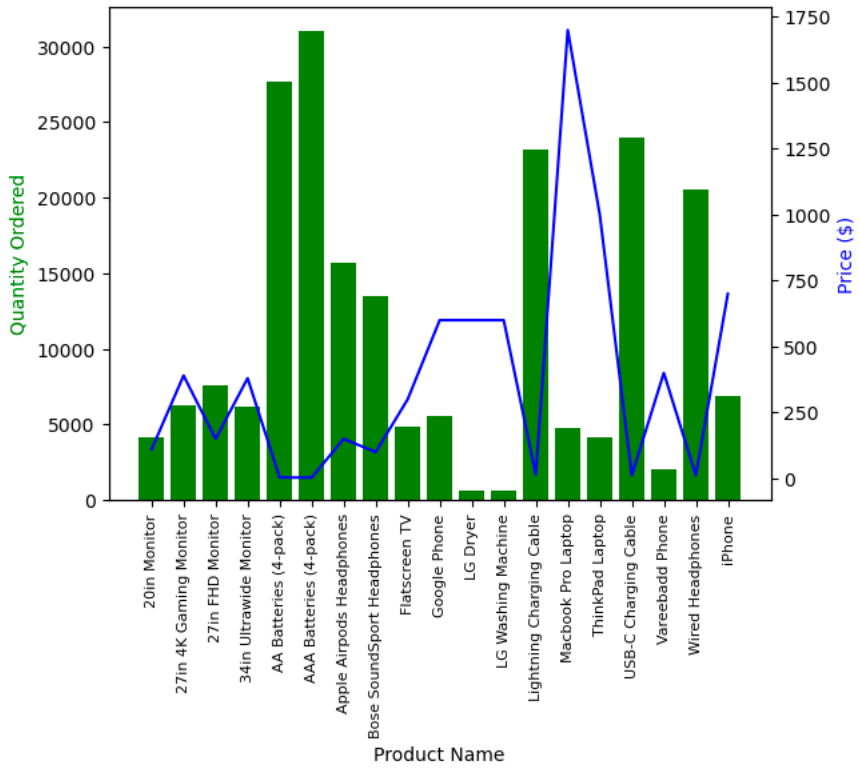
To find out which product sold the most we do the same as for question 2.

```
product_ordered = all_data.groupby('Product').sum()
products = [pro for pro, df in all_data.groupby('Product')]
quantity_ordered = product_ordered["Quantity Ordered"]
plt.bar(products, product_ordered["Quantity Ordered"], color='g')
plt.xticks(products, rotation='vertical', size=8)
plt.ylabel('Quantity Ordered')
plt.xlabel('Products')
```



We can answer the 'Why' part by analysing the chart above. One hypothesis is that cheaper products tend to sale better than more expensive ones. To prove this, we can overlay a price chart of the same product over the existing chart to see if the corelation is there.

```
prices = all_data.groupby('Product').mean()['Price Each']
print(prices)

fig, ax1 = plt.subplots()

ax2 = ax1.twinx()
ax1.bar(products, quantity_ordered, color='g')
ax2.plot(products, prices, 'b-')

ax1.set_xlabel('Product Name')
ax1.set_ylabel('Quantity Ordered', color='g')
ax2.set_ylabel('Price ($)', color='b')
ax1.set_xticklabels(products, rotation='vertical', size=8)
```



With this chart we can see that the cheapest products such as AA or AAA Batteries, cables and headphones sale well, while expensive products such as TV or dryer sale much less. An exception here is Apple's products still sale relatively well, given the high prices. Such as the case with the MacBook laptop or the iPhone.