# Product Sales Analysis Using Python

**OVERVIEW**

In this post, I use Python Pandas & Python Matplotlib to analyze and answer business questions about 12 months worth of sales data. The data contains hundreds of thousands of electronics store purchases broken down by month, product type, cost, purchase address, etc. The dataset can be downloaded [here](https://github.com/alfifutuhi13/miniproject1/blob/main/Pandas-Data-Science-Tasks-master.zip). In this analysis, I’m using jupyter notebook.



Source: [https://www.freepik.com/photos/business](https://www.freepik.com/photos/business') Business photo created by cookie\_studio —

**PROBLEMS**

1. What was the best month for sales? How much was earned that month?
2. What city sold the most product?
3. What time should we display advertisements to maximize likelihood of customer’s buying products?
4. What Products are most often sold together?
5. What product sold the most? Why do you think it did?

**SOLUTION**

1. **What was the best month for sales? How much was earned that month?**

First of all, we need to see what kind of data we are trying to analyze. we can simply do this

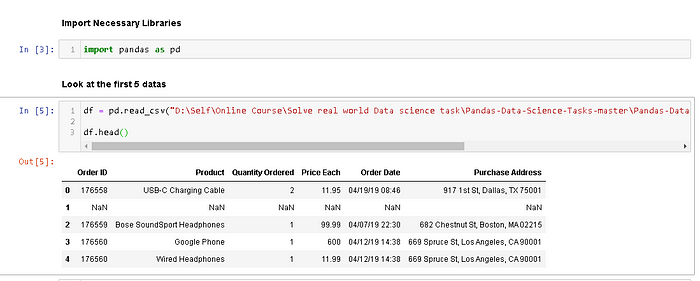


Figure 1. Code to show the top 5 data in Sales\_April\_2019.csv

We use panda to read the csv file and create a dataframe from it. The file’s directory can be put anywhere. I personally put it in D. You can copy the directory and paste it in the syntax. At Figure 1, we can see that we have 6 columns. Now the first task is to merge all 12 months worth of sales data (12 csv files) into a single csv file. To do that, we need to import new library called os.



Figure 2. Import new library os.

We need os library to read all csv files’ title and call it using *for loop*. As you can see from Figure 2, we successfully read all the csv files’ title and we’re ready to merge it. To do that, we can simply do

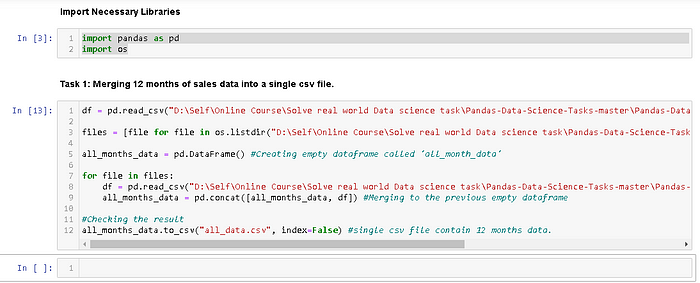


Figure 3. Creating a new file contains all 12 months data.

**— — — — — — — — — — — DETAIL CODE — — — — — — — — — — —**

import pandas as pd  
import os

df = pd.read\_csv(“D:\Self\Online Course\Solve real world Data science task\Pandas-Data-Science-Tasks-master\Pandas-Data-Science-Tasks-master\SalesAnalysis\Sales\_Data\Sales\_April\_2019.csv”)  
  
files = [file for file in os.listdir(“D:\Self\Online Course\Solve real world Data science task\Pandas-Data-Science-Tasks-master\Pandas-Data-Science-Tasks-master\SalesAnalysis\Sales\_Data”)]

all\_months\_data = pd.DataFrame() #Creating empty dataframe called ‘all\_month\_data’

for file in files:  
df = pd.read\_csv(“D:\Self\Online Course\Solve real world Data science task\Pandas-Data-Science-Tasks-master\Pandas-Data-Science-Tasks-master\SalesAnalysis\Sales\_Data/”+file)  
all\_months\_data = pd.concat([all\_months\_data, df]) #Merging to the previous empty dataframe

#Checking the result  
all\_months\_data.to\_csv(“all\_data.csv”, index=False) #single csv file contain 12 months data.

**— — — — — — — — — — — — — — — — — — — — — — — — — — — — —**

It will take a little longer time because of heavy computation. But once it’s done, you can open the same directory folder and check the folder called “Output”. You will see a new csv file contains all 12 months data.

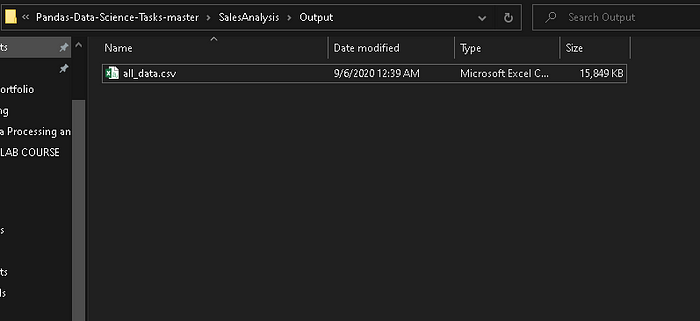


Figure 4. A new csv file contains all 12 months data.

After we create this new csv file, you can delete the previous code (if you want) and we will use this file to answer all of the problems.

Now we only use this code to read all of 12 months data.

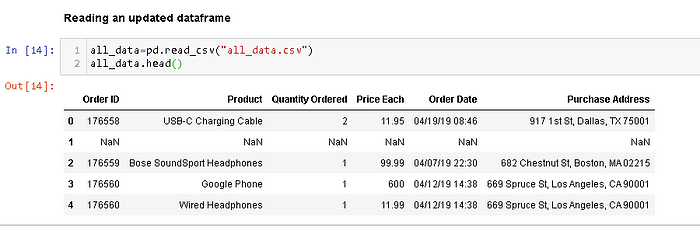


Figure 5. Reading an updated dataframe

Now we’re ready to answer the problem number 1. To remind you, the question is: **What was the best month for sales? How much was earned that month?**

To answer this problem, obviously we need an additional column called “Month”. If you look carefully at Figure 5, you will see the first 2 characters in “Order Date” values represent months. So the next task we will do is to add “Month” Column.



https://miro.medium.com/v2/resize:fit:700/1*winqW-qJ3a6IjFaC1ePnSw.png

Figure 6. Adding “Month” Column

Now, we get an issue here. There are NaN values in our data. You could spot on of NaN value at Figure 1 or Figure 5 in index 1. Now we need to clean up the data by dropping rows of NaN. Let’s spot more NaN value here. You don’t have to do this, I am just curious.

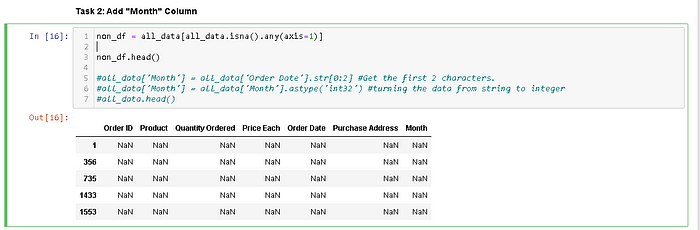
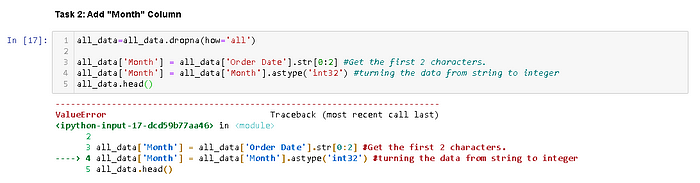


Figure 7. Spotting the NaN Values in our data.

We use *.isna().any(axis=1)* to spot rows containing the NaN values (axis = 0 to spot column containing NaN values). Now, we’re gonna remove it from our dataframe using *.dropna()* method.



https://miro.medium.com/v2/resize:fit:700/1*J8zN6VfV-_O0ImzOjwQP3w.png

Figure 8. Dropping the NaN values from our dataframe.

*.dropna()* method is successful, but we get a new issue here. There are values “Or” in our data. Let’s find it first.

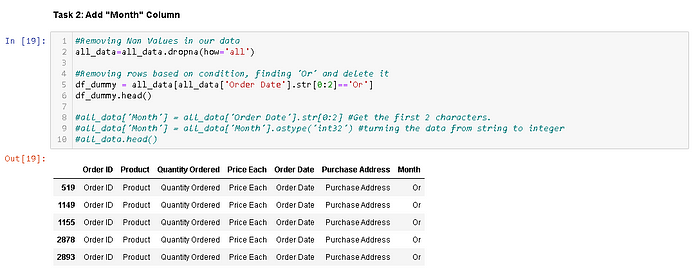


Figure 9. Finding “Or” values in our dataframe.

We can see clearly from Figure 9 that the issue is the rows contain the same words as the title rows. So clearly ‘Or’ is coming from ’Order Date’. We need to drop this “Or” rows just simply change the equal sign (“==”) to not equal sign (“!=”).

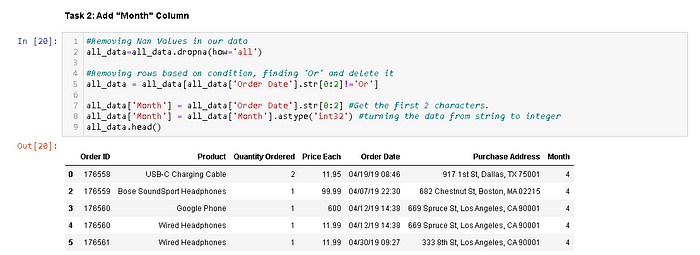
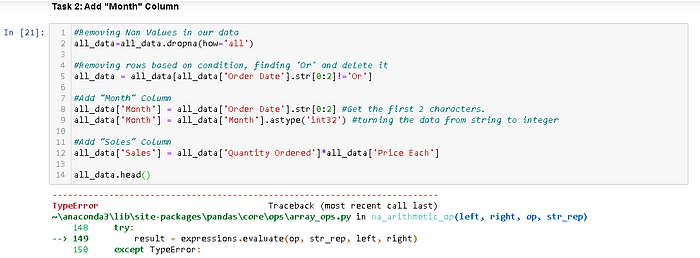


Figure 10. Dropping all unnecesary values in our dataframe.

We can see clearly from Figure 10 that we successfully created “Month” column and make its data type to integer.

Now, are we ready to answer the question? Not yet, we need obviously one more column called “Sales” Column. How can we get that? We get “Sales” by multiplying “Quantity Ordered” and “Price Each” values. Let’s create it.



https://miro.medium.com/v2/resize:fit:700/1*qvm7TJJJemAPuBlYweHg-A.png

Figure 11. Adding “Sales” Column in our dataframe.

Now we encounter a new issue. The values of the column “Quantity Ordered” and “Price Each” are strings. So the next task is to convert these columns to the correct type (“Quantity Ordered” is integer and “Price Each” is float). We’re gonna use *pd.to\_numeric()* method to convert them to numeric.

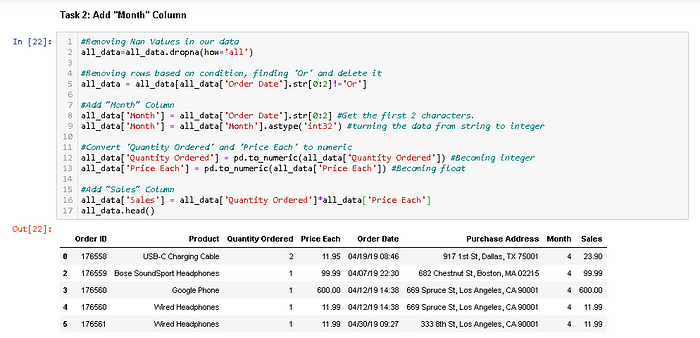


Figure 12. Converting “Quantity Ordered” and “Price Each” Columns to Numeric

Now the “Sales” Column is successfully created, we can answer the first question. What was the best month for sales? How much was earned that month? We can easily answer it by using*groupby(‘Month’).sum()* method.

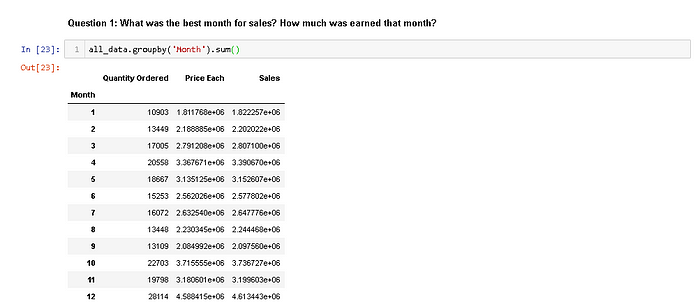


Figure 13. Grouping by month and summing the Sales.

Look carefully at Figure 13. We can clearly see that month 12 (December) is the highest sales in 2019 with approximately $4,810,000. But we need to visualize it to make our bussiness partner easier to understand. So we’re gonna import matplotlib and visualizing our results with bar chart.

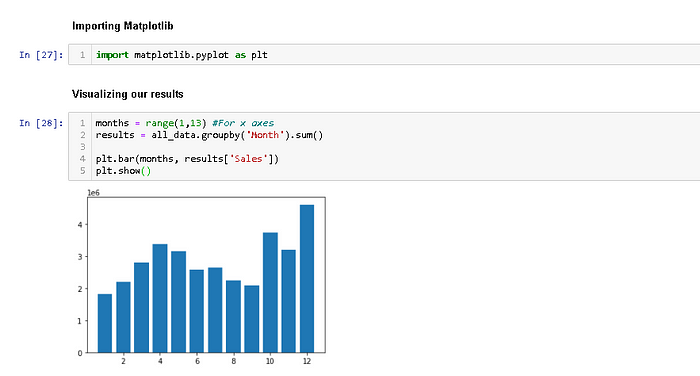


Figure 14. Visualizing our results using matplotlib library.

It’s good. But we need to make it looks neater. So we’re just gonna add a little code.

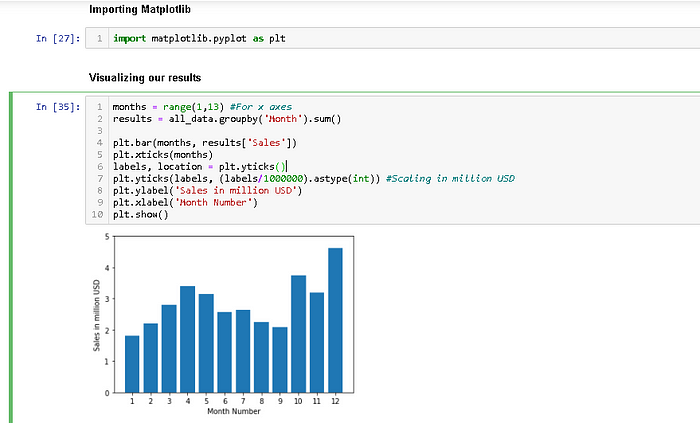


Figure 15. Improving the visualization.

Now, not only we can get the highest sales, but we can also get the lowest sales just by looking it for a few seconds. As a data scientist, we have to figure out why a certain month is better than others. Maybe the company spend more money on April so the product sales are increasing. Maybe the best product sales are on December because it’s holiday and Christmas. Those are just my hyphothesis, right now we don’t have enough data to prove that hypothesis. But we can take these as a consideration if you want to decide something that relates to product sales.

**2. What city sold the most product?**

To answer this question, obviously we need to create a new column called “City” column. How do we get that? As usual, we’re gonna check the top 5 data in our dataframe to figure out where can we get our “City” column using *.head()* method.

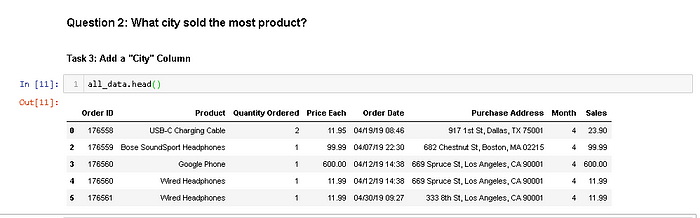


Figure 16. Showing Our Top 5 Updated Dataframe

As you can see at Figure 16, the “Purchase Address” Column contain the city. We can’t get it directly, we need to extract the data. We can use one of most useful function in pandas, *.apply()* method.

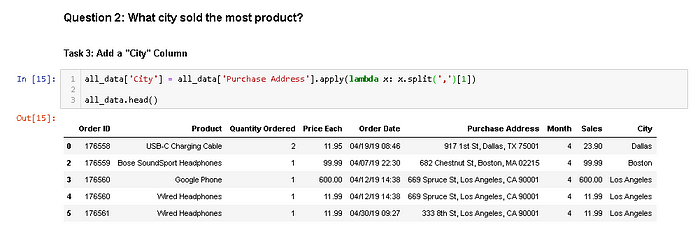


Figure 17. Using .apply() Method to Extract The Data

To make it neater, we can use this

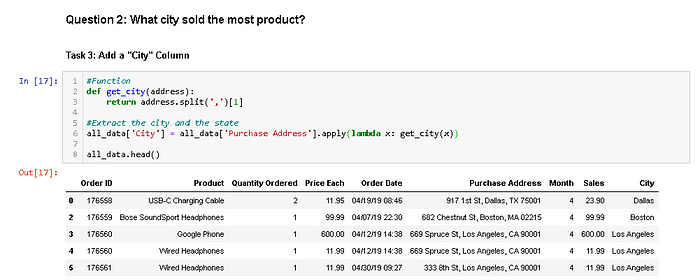


Figure 18. Using def function to Make The Code Neater

apply and lambda usually used to create new column based on other column (example .apply(lambda x: x\*2. It means every input x in other column will be changed to x\*2 in a new column). In this case we create “City” column based on “Purchase Address” column and we split the data into 3 part. The first one is before the first comma (index = 0), the second one is between the commas (index = 1), and the third on is after the last comma (index = 2). As we need to extract the city data, we use [1] to state it to index 1.

As you can see at Figure 17, we successfully created a “City” column. So are we ready to answer the second question? Not yet. We get an issue here. It’s not error, it’s the value of the “City” Column. This is just a rare case when there are 2 cities are named exactly the same. Example someone in New England and someone in West Coast would think Portland in different way. Someone in New England thinks Portland as Portland Maine and someone in West Coast thinks Portland as Portland Oregon. So in our dataset we actually had the overlapping cities between these two. So, we should also grab the state.

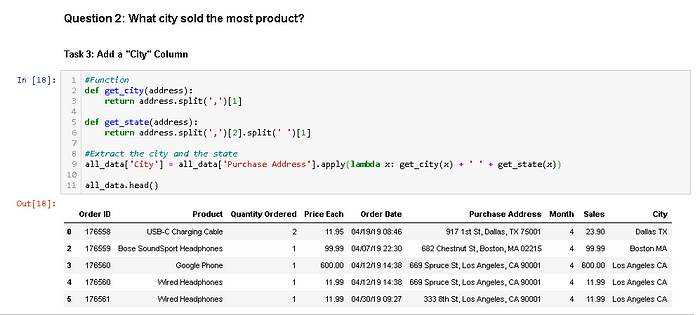


Figure 19. Extracting the state to “City” Column

The function get\_state() basically works as explained before. But in this function, we seperate the data again become three parts. The first one is before the whitespace (index = 0), the second one is between the whitespaces (index = 1), and the third one is after the last whitespace (index = 2). So that’s why we use *.split(‘ ‘)[1*] in the second split.

Now we’re ready to answer the second question, **what city sold the most product**? As we did before, we’re gonna group it by the city and summing all the values based on the group.

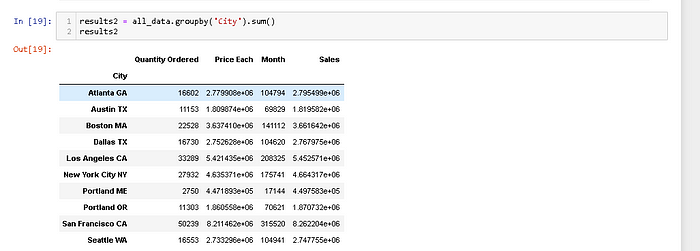


Figure 20. Grouping The Dataframe by The City

It’s too messy, but if you look carefully you can see that San Fransisco is the highest sold product of all cities with approximately $8,200,000. We clearly need to visualize it because it’s so hard to conclude anything just based on that numbers and also it will make our bussiness partner easier to understand.

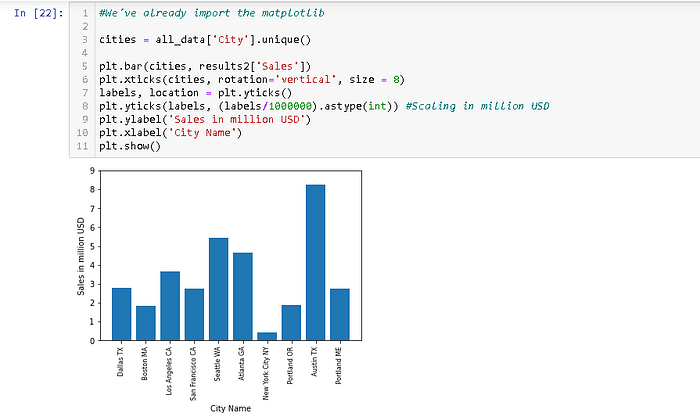


Figure 21. Plotting the Sales Grouped by Cities.

Now we successfully plot it. But there is a big issue here. If you notice that the values (Figure 20) and the plot (Figure 21) are not synchronized. The highest sales should be San Fransisco. What’s wrong with our code?

There’s an issue between *.unique()* method and *plt.bar()*. Their cities order are different. we’re gonna syncronized the order by simply fixing the variable ‘cities’.

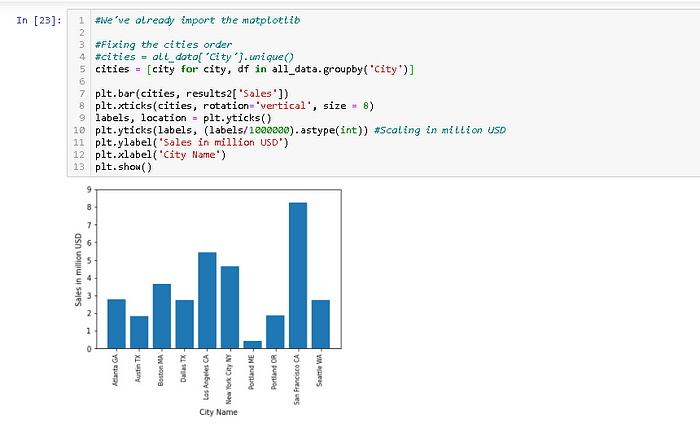


Figure 22. Fixing The Code

Now, we fix the issue and successfully plot it. As a data scientist, we need to figure out why San Fransisco is the highest sale compare to other cities. Maybe Sillicon Valley need more electronic products. Maybe the advertisement is better in San Fransisco. We can use this data to improve the sales of bussiness.

**3. What Time Should We Display Advertisements to Maximize Likelihood of Customer’s Buying Product?**

As usual, to remind what our data look like, we use the .head() method to show the top 5 of our updated dataframe.

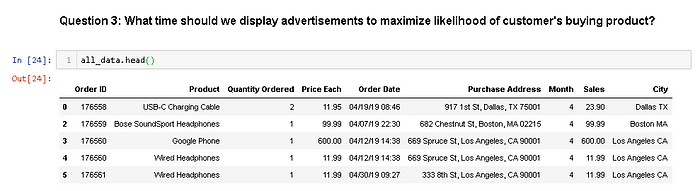


Figure 23. Showing our Top 5 updated dataframe

If we’re gonna use our data to answer this question, we need to aggregate the period in 24 hours distribution. Look carefully at Figure 23. In “Order Date” column, there are times data. We could extract it like we did before. But to make it more consistent, we need to convert the “Order Date” Column into date time object. We’re gonna use *pd.to\_datetime()* method.

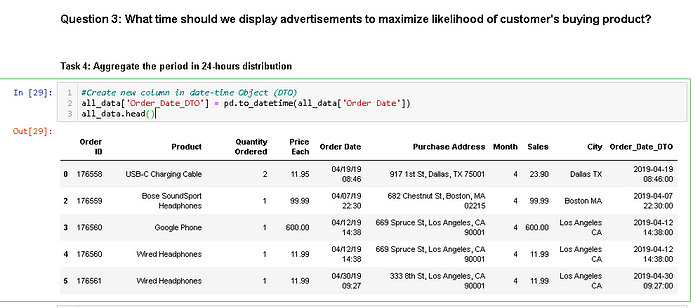


Figure 24. Converting the “Order Date” Column into Date-Time Object

It will take a little bit longer because of the heavy calculation. Now we can create a new column called “Hour” contain the extraction of “Order\_Date\_DTO” data. We only need the hours data, so we can extract them by doing this.

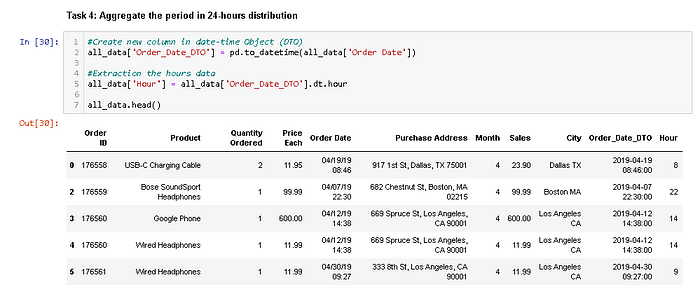
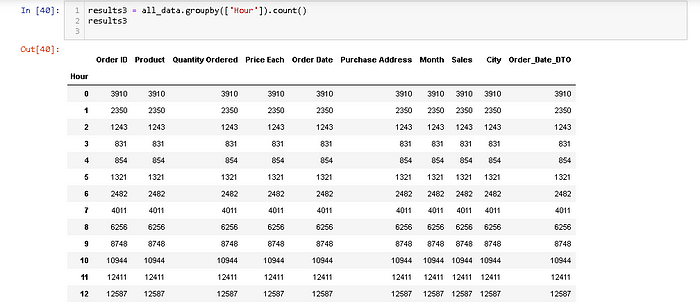


Figure 25. Extracting The Hours Data Into The New Column

Now we can answer the third question, **what time should we display advertisements to maximize likelihood of customer’s buying product?**To answer this, we’re gonna group it by the hours and counting all of the orders.



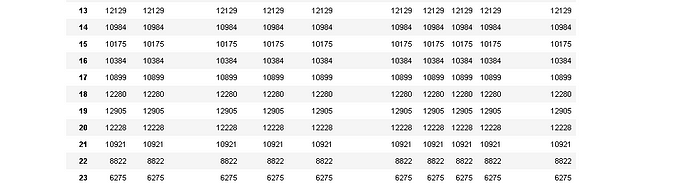


Figure 26. Grouping the data by the hours

If we want to answer the third question, we only need the “Quantity Ordered” column. Now let’s visualize it. We want it to be the line chart because this spesific data (hours) are more logical to show using line chart than bar chart because the data has to be continue.

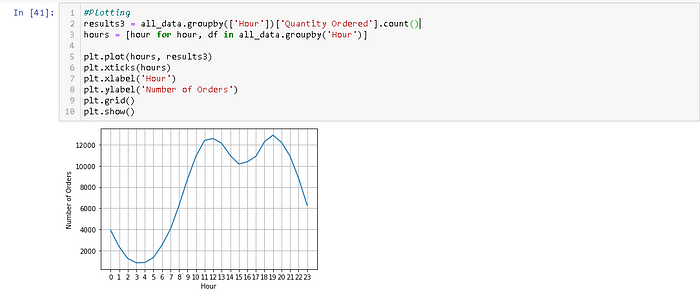


Figure 27. Visualizing the Number of Orders in 24 hours format

As you can see from Figure 27, there are approximately 2 peaks at the data. They are 12 (12 PM) and 19 (7 PM). It makes sense since most people shopping during the day. From this data, we can suggest to our bussiness partner to advertise their product right before 12 PM and/or 7 PM. It could be 11.30 AM and/or 6.30 PM.

Remember, this chart is the total orders of **all cities**. Maybe you could make a spesific chart for a spesific city and planning the advertisement better for that city.

**4. What Products Are Most Often Sold Together?**

We’re gonna take a look to our top 5 data as usual.



Figure 28. Showing Our Top 5 Updated Dataframe

Look carefully at Figure 28. We can see that “Order ID” indicate the transaction. So by grouping the product by the Order ID, we are able to know what products are often sold together. We’re gonna use .duplicated() method to find a duplicate values of “Order ID”.

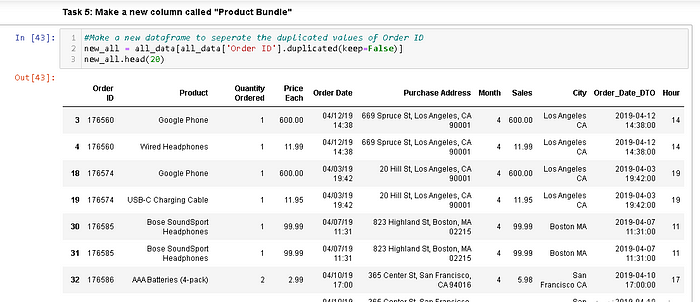




Figure 29. Showing Our Top 20 Duplicated Dataframe

Now we want to create a column called “Product Bundle” that contain example *Google Phone* and *Wired Headphone* (transaction 17650) at the same line. We’re gonna use the *.transform()* method to join values from two rows into a single row.

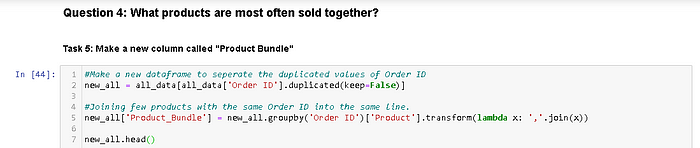




Figure 30. Joining Few Products With The Same Order ID Into The Same Line

It’s good, but we have an issue here. We have the same order at least twice because we did merge them in every situation in groupby without dropping the duplicate values. Now let’s drop the rows with duplicate values.

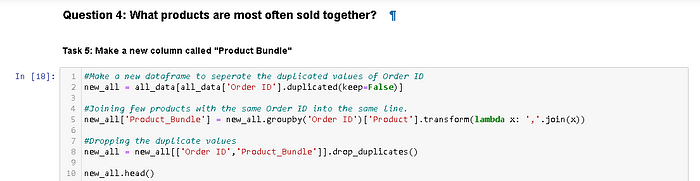




Figure 31. Dropping rows with duplicate values

Now, we need to count the pair of products. We need new libraries because they have all we need to count all the combination of products bundle. We’re gonna use *itertools* and *collections* libraries.

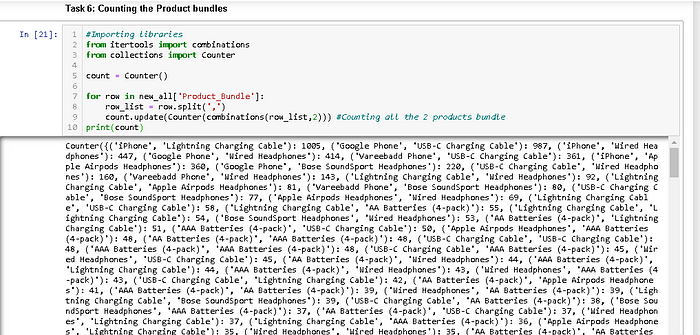


Figure 32. Counting the Product Bundle

It’s too messy, let’s just showing the top 10 data

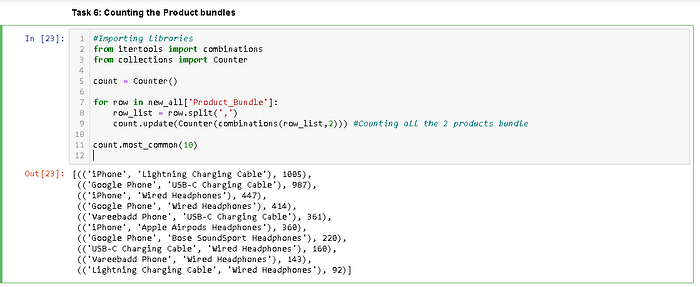


Figure 33. Showing The Top 10 2-Product Bundles

Now we can clearly see that the most often products sold together are iPhone and Lightning Charging Cable with 1005 transactions. We could count the 3 product bundles by just changing the *count.update* index into 3.



Figure 34. Showing The Top 10 3-Product Bundles

We can see the most often sold products (3 products) together are Google Phone, USB-C Charging Cable, and Wired Headphones with 87 transactions. It’s not really significant compare to the 2-Product Bundle. So we’re gonna ignore the 3-Product bundle.

What would we do with this data? Well, we could offer a smart deal to the customer that buy iPhone, you could recommend the charging cable with discount. That’s one of the possibility and you can bundle the remaining products if you need to.

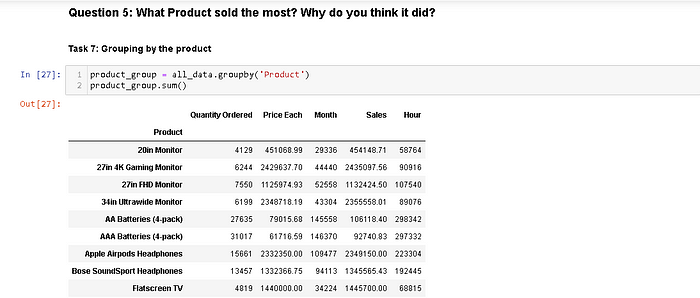
**5. What product sold the most? Why do you think it did?**

As usual, let’s see our data looks like again.



Figure 35. Showing Out Top 5 Updated Dataframe

We need to sum up the “Quantity Ordered” based on grouping the Product. So let’s do it.



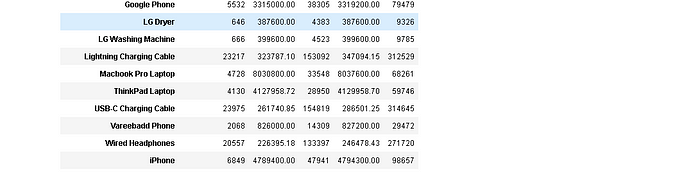
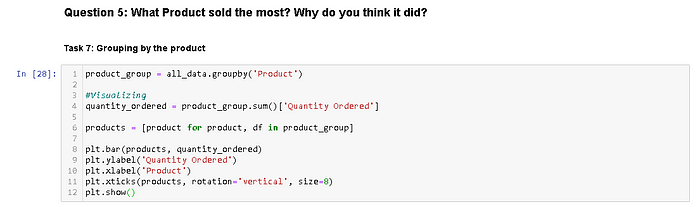


Figure 36. Grouping by the Product

To make it easier to understand, let’s visualize it.



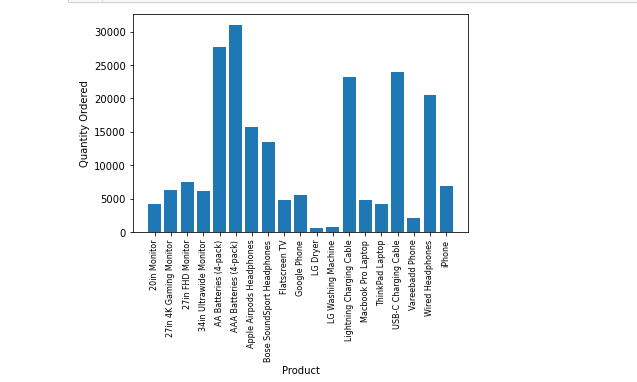
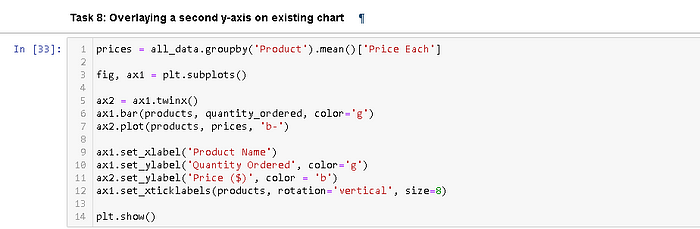


Figure 37. Visualizing The Grouped Product.

Now we can see what product sold the most, it’s AAA Batteries(4 pack). We can also see that AA Batteries (4 pack), Lightning Charging Cable, USB-C Charging Cable, and Wired Headphones are sold more than other products. Why are they sold the most? The first impression is that they are cheaper than other products. As a data scientist, let’s prove this hypothesis. We could do it by overlaying the graph by their actual price and see if they have direct correlation.



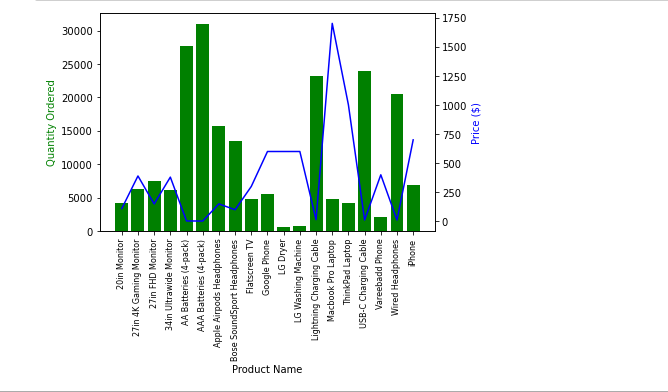


Figure 38. Overlaying The Second y-axis

Now we will interprate our results. Our hypothesis is true if the high sold products have low price. From the graph we can see it is the case for AAA Batteries and all products except the Macbook Pro Laptop and ThinkPad Laptop. They have decent orders eventhough they are expensive. We can say that there are many people in the world need laptops. So the laptops are the exception because the laptops have high demand.

**CONCLUSION**

1. **What was the best month for sales? How much was earned that month?**

The best month for sales is **December**. The company earned approximately $4,810,000.

**2. What city sold the most product?**

**San Fransisco** is the city with the highest sales.

**3. What time should we display advertisements to maximize likelihood of customer’s buying products?**

We can suggest to advertise the products right before 12 PM and/or 7 PM. It could be **11.30 AM and/or 6.30 PM**.

**4. What Products are most often sold together?**

The most often products sold together are **iPhone and Lightning Charging Cable** with 1005 transactions.

**5. What product sold the most? Why do you think it did?**

**AAA Batteries(4 pack)** is the most sold product. Because it’s cheaper than other products and has high demand.