Active Product Sales Analysis in Python with Matplotlib

Today, every forward-thinking company with an online sales presence or dedicated e-commerce platform is focused on optimizing its performance to pinpoint exactly what customers want, boosting the likelihood of successful sales. By thoroughly analyzing large datasets, we can uncover key insights—like the peak times of day when transaction activity is highest—empowering companies to connect more effectively with their audience.

Implementation in Steps

Step 1:

The dataset must first be converted into a Dataframe, and even before that, some libraries must be imported.

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

order_detail = pd.read_csv('Order_details(masked).csv')
print(order_detail)
```

Output:

```
          Name                       Email  \
0      PERSON_1         PERSON_1@gmail.com
1      PERSON_2  PERSON_2@tataprojects.com
2      PERSON_3         PERSON_3@gmail.com
3      PERSON_4         PERSON_4@gmail.com
4      PERSON_5         PERSON_5@gmail.com
..          ...                        ...
576  PERSON_522       PERSON_522@gmail.com
577  PERSON_523       PERSON_523@gmail.com
578  PERSON_523       PERSON_523@gmail.com
579  PERSON_524       PERSON_524@gmail.com
580  PERSON_525       PERSON_525@gmail.com

                                          Product     Transaction Date
0                                       PRODUCT_75  01/03/2021 00:47:26
1                                       PRODUCT_75  01/03/2021 02:04:07
2                                       PRODUCT_63  01/03/2021 09:10:43
3                                       PRODUCT_63  01/03/2021 09:49:48
4          PRODUCT_34,PRODUCT_86,PRODUCT_57,PRODUCT_89  01/03/2021 10:56:46
..                                              ...                  ...
576  PRODUCT_48,PRODUCT_80,PRODUCT_71,PRODUCT_68,PR...  07/03/2021 23:53:03
577                                       PRODUCT_8  07/03/2021 23:55:01
578  PRODUCT_36,PRODUCT_14,PRODUCT_64,PRODUCT_28,PR...  07/03/2021 23:58:24
579  PRODUCT_75,PRODUCT_71,PRODUCT_86,PRODUCT_63,PR...  07/03/2021 23:59:26
580                             PRODUCT_66,PRODUCT_34  07/03/2021 23:59:19

[581 rows x 4 columns]
```

Step 2:

Enhance your dataset by adding a new **Time** column, converting the **Transaction Date** into a customizable DateTime format (e.g., YYYY-MM-DD HH:MM).

To focus specifically on the hour of each transaction, we'll extract this information into a separate **Hour** column using a built-in function:

```
order_detail['Time'] = pd.to_datetime(order_detail['Transaction Date'])

order_detail['Hour'] = (order_detail)['Time'].dt.hour
print(order_detail['Hour'])
```

Output

```
0       0
1       2
2       9
3       9
4      10
       ..
576    23
577    23
578    23
579    23
580    23
Name: Hour, Length: 581, dtype: int32
```

Step 3:

Next, we need to identify the **'n' busiest hours**. To do this, we'll generate a list of the top 'n' hours by transaction frequency. Using `value_counts` in Python, we can easily calculate the hourly frequency, then convert it to a list with `tolist()` for streamlined data manipulation. We'll also create a separate list to capture the corresponding hour values

```
timemost1 = order_detail['Hour'].value_counts().index.tolist()[:24]

timemost2 = order_detail['Hour'].value_counts().values.tolist()[:24]

print(timemost1)
print(timemost2)
```

Output:

```
[23, 12, 22, 19, 21, 15, 20, 11, 13, 18, 16, 14, 17, 10, 0, 9, 8, 7, 1, 2, 5, 6, 3]
[51, 51, 45, 42, 41, 41, 39, 37, 33, 33, 29, 28, 27, 24, 17, 14, 10, 6, 4, 3, 3, 2, 1]
```

Step 4:

To complete the process, we pair the hours with their respective frequencies, combining them into a final result that clearly highlights the busiest times of day based on transaction activity. By stacking these indices alongside their occurrence rates, we create an insightful overview of peak transactional hours, ready for further analysis or visualization.

```
tmost = np.column_stack((timemost1,timemost2))

print(" Hour Of Day" + "\t" + "Cumulative Number of Purchases \n")
print('\n'.join('\t\t'.join(map(str, row)) for row in tmost))
```

Output:

```
 Hour Of Day    Cumulative Number of Purchases

23              51
12              51
22              45
19              42
21              41
15              41
20              39
11              37
13              33
18              33
16              29
14              28
17              27
10              24
0               17
9               14
8               10
7               6
1               4
2               3
5               3
6               2
3               1
```

Step 5:

To prepare for effective data visualization, we first need to make the list more adaptable. This involves gathering the transaction frequencies by hour and then applying the following adjustments for enhanced customization

```
timemost = order_detail['Hour'].value_counts()
timemost1 = []

for i in range(0,23):
    timemost1.append(i)

timemost2 = timemost.sort_index()
timemost2.tolist()
timemost2 = pd.DataFrame(timemost2)
```

Step 6:

For visualizing the data, we'll use Matplotlib for clarity, as it's one of the most popular and user-friendly libraries. However, feel free to choose any preferred library, like Matplotlib, Seaborn, or Ggplot, for plotting.

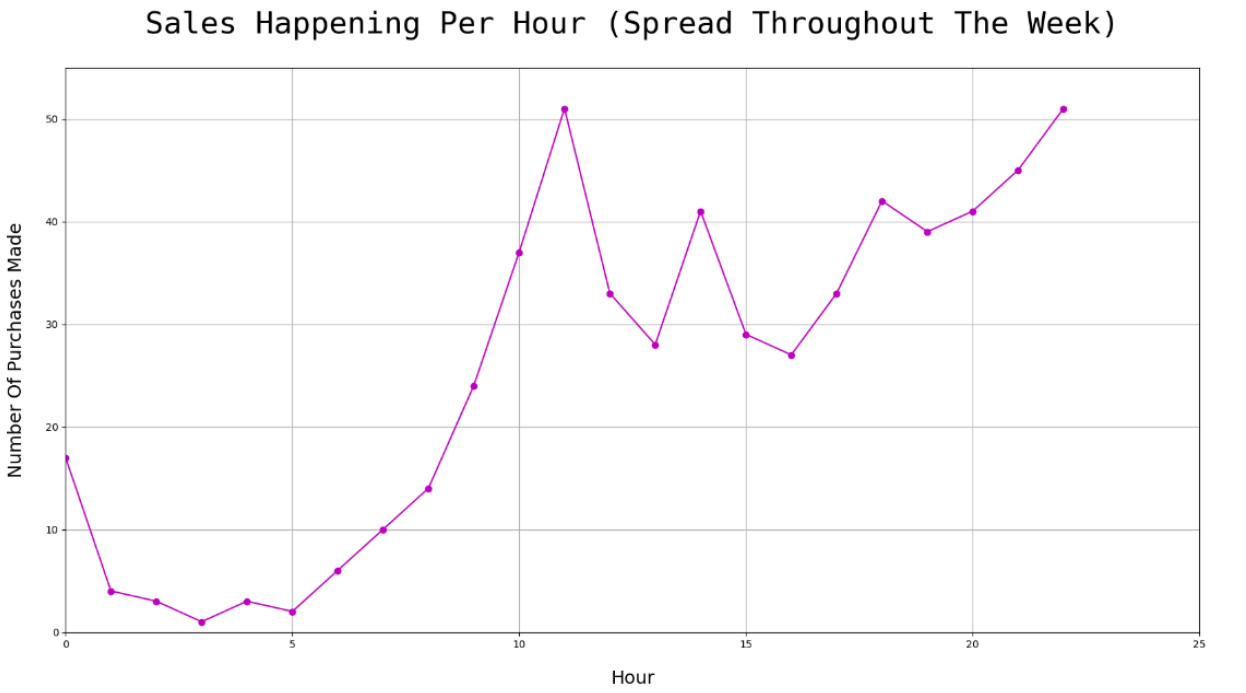
The following commands ensure that the X-axis represents the hours, while the Y-axis shows the transaction volumes. Additionally, we'll customize aspects of the line chart, such as color, font, and other stylistic elements, to make the data presentation as clear as possible.

```
plt.figure(figsize=(20, 10))

plt.title('Sales Happening Per Hour (Spread Throughout The Week)',
                fontdict={'fontname': 'monospace', 'fontsize': 30}, y=1.05)
plt.xlim(0, 25)  # Set x-axis limits
plt.ylim(0, 55)

plt.ylabel("Number Of Purchases Made", fontsize=18, labelpad=20)
plt.xlabel("Hour", fontsize=18, labelpad=20)
plt.plot(timemost1, timemost2, color='m', marker = 'o')
plt.grid()
plt.show()
```

Output:



Summary

Here's a summary of the steps outlined for analyzing and visualizing transactional data by hour:

1. **Convert Transaction Date to DateTime**: Create a new column, **Time**, by converting the **Transaction Date** column into a DateTime format. Extract the hour component to a separate **Hour** column for focused analysis.
2. **Identify Peak Hours**: Determine the top "n" busiest hours by counting transaction occurrences for each hour. Use `value_counts` to get the frequency of each hour, and then convert the results into a list.
3. **Combine Hours and Frequencies**: Pair the hours with their frequencies to generate a clear view of the most active transaction times, ready for further analysis.
4. **Prepare for Data Visualization**: For flexibility, ensure that the hourly transaction data can be adjusted as needed before visualization.
5. **Visualize with Matplotlib**: Use Matplotlib (or a preferred library like Seaborn or Ggplot) for graphical representation. Set up the X-axis to represent hours and the

Y-axis for transaction counts, adjusting elements like color and font to enhance clarity.

These steps create a detailed and customizable approach to identify and visualize peak transaction hours, providing valuable insights for optimizing customer engagement times.