

Data Analyst – Internship

Task 2: SuperMarket Sales Analysis

Step-1: Data Extraction

```
Python Console (1) x +
import sys; print('Python %s on %s' % (sys.version, sys.platform))
sys.path.extend(['C:\\Users\\lenovo\\PyCharmMiscProject'])

Python Console
...: import matplotlib.pyplot as plt
>>> ...: from matplotlib.pyplot import xlabel
...: from matplotlib.pyplot import ylabel
...: from matplotlib.pyplot import show
...: from matplotlib.pyplot import title
...: import seaborn as sns

Backend tkagg is interactive backend. Turning interactive mode on.
```

```
Python Console (1) x +
In [3]: data = pd.read_csv("SuperMarket Analysis.csv")
In [4]: data.head()
Out[4]:
   Invoice ID Branch      City  ... gross margin percentage gross income Rating
0  750-67-8428  Alex   Yangon  ...           4.761905         26.1415      9.1
1  226-31-3081  Giza  Naypyitaw ...           4.761905          3.8200      9.6
2  631-41-3108  Alex   Yangon  ...           4.761905         16.2155      7.4
3  123-19-1176  Alex   Yangon  ...           4.761905         23.2880      8.4
4  373-73-7910  Alex   Yangon  ...           4.761905         30.2085      5.3

[5 rows x 17 columns]
In [5]: |
```

```
Python Console (1) x +
In [5]: data.info()
<class 'pandas.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 17 columns):
#   Column                Non-Null Count  Dtype
---  -
0   Invoice ID             1000 non-null  str
1   Branch                1000 non-null  str
2   City                  1000 non-null  str
3   Customer type          1000 non-null  str
4   Gender                 1000 non-null  str
In [6]:
```

```
Python Console (1) × +
5 Product line 1000 non-null str
6 Unit price 1000 non-null float64
7 Quantity 1000 non-null int64
8 Tax 5% 1000 non-null float64
9 Sales 1000 non-null float64
10 Date 1000 non-null str
11 Time 1000 non-null str
12 Payment 1000 non-null str
13 cogs 1000 non-null float64
14 gross margin percentage 1000 non-null float64
15 gross income 1000 non-null float64
16 Rating 1000 non-null float64
dtypes: float64(7), int64(1), str(9)
memory usage: 132.9 KB
In [6]: |
```

Step-2: Data Cleaning

```
Python Console (1) × +
In [6]: data.isnull().sum()
Out[6]:
Invoice ID 0
Branch 0
City 0
Customer type 0
Gender 0
Product line 0
Unit price 0
Quantity 0
Tax 5% 0
```

```
Sales      0
Date       0
Time       0
Payment    0
cogs       0
gross margin percentage  0
gross income  0
Rating     0
dtype: int64

In [7]: |
```

```
Python Console (1) x +
In [7]: data = data.drop_duplicates()
In [8]: data.info()
<class 'pandas.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 17 columns):
#   Column              Non-Null Count  Dtype
---  -
0   Invoice ID           1000 non-null   str
1   Branch              1000 non-null   str
2   City                1000 non-null   str
3   Customer type       1000 non-null   str
4   Gender              1000 non-null   str
5   Product line        1000 non-null   str
```

```

6 Unit price      1000 non-null float64
7 Quantity       1000 non-null int64
8 Tax 5%         1000 non-null float64
9 Sales          1000 non-null float64
10 Date          1000 non-null str
11 Time          1000 non-null str
12 Payment       1000 non-null str
13 cogs          1000 non-null float64
14 gross margin percentage 1000 non-null float64
15 gross income  1000 non-null float64
16 Rating        1000 non-null float64

dtypes: float64(7), int64(1), str(9)
memory usage: 132.9 KB

```

Step-3: Basic Statistical Summary

```

Python Console (1) x +
In [9]: data.describe()
Out[9]:
      Unit price  Quantity  ...  gross income  Rating
count  1000.000000  1000.000000  ...  1000.000000  1000.000000
mean    55.672130    5.510000  ...    15.379369    6.97270
std     26.494628    2.923431  ...    11.708825    1.71858
min     10.080000    1.000000  ...     0.508500    4.00000
25%     32.875000    3.000000  ...     5.924875    5.50000
50%     55.230000    5.000000  ...    12.088000    7.00000
75%     77.935000    8.000000  ...    22.445250    8.50000
max     99.960000   10.000000  ...    49.650000   10.00000

[8 rows x 8 columns]

```

I have created the variable named **meta_gross_income**, but I mistakenly typed wrong variable name.

```

In [10]: meta_gross_income = data['gross income'].mean()
In [11]: print(f"Average Gross Income: {mean_gross_income:.2f}")
Traceback (most recent call last):
  File "C:\Users\lenovo\PyCharmMiscProject\.venv\Lib\site-packages\IPython\core\interactiveshell.py", line 370
    exec(code_obj, self.user_global_ns, self.user_ns)
  File "<ipython-input-11-9c2a4ec10b2d>", line 1, in <module>
    print(f"Average Gross Income: {mean_gross_income:.2f}")
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
NameError: name 'mean_gross_income' is not defined. Did you mean: 'meta_gross_income'?
In [12]: print(f"Average Gross Income: {meta_gross_income:.2f}")
Average Gross Income: 15.38

```

Step-4: Branch-wise Sales Analysis

```
In [13]: branch_sales = data.groupby('Branch')['gross income'].sum()
>>> In [14]: print("Total Gross Income by Branch:\n", branch_sales)
Total Gross Income by Branch:
      Branch
Alex      5057.1605
Cairo     5057.0320
Giza      5265.1765
Name: gross income, dtype: float64

In [15]: branch_mean = data.groupby('Branch')['gross income'].mean()
>>> In [16]: print("Mean Gross Income by Branch:\n", branch_mean)
Mean Gross Income by Branch:
      Branch
Alex      14.874001
Cairo     15.232024
Giza      16.052367
Name: gross income, dtype: float64

In [17]: best_branch = branch_mean.idxmax()
In [18]: print(f"The best performing branch is: {best_branch}")
The best performing branch is: Giza
```

Step-5: Category-wise Sales Analysis

```
In [19]: category_sales = data.groupby('Product line')['gross income'].sum().sort_values(ascending=False)
In [20]: print("Gross Income by Product Line:\n", category_sales)
Gross Income by Product Line:
      Product line
Food and beverages      2673.5640
Sports and travel       2624.8965
Electronic accessories  2587.5015
Fashion accessories     2585.9950
Home and lifestyle      2564.8530
Health and beauty       2342.5590
Name: gross income, dtype: float64

In [21]: top_categories = category_sales.head(3)
>>> In [22]: print("Top 3 Product Lines:\n", top_categories)
Top 3 Product Lines:
      Product line
Food and beverages      2673.5640
Sports and travel       2624.8965
Electronic accessories  2587.5015
Name: gross income, dtype: float64

In [23]:
```

Step-6: Customer and City Sales Analysis

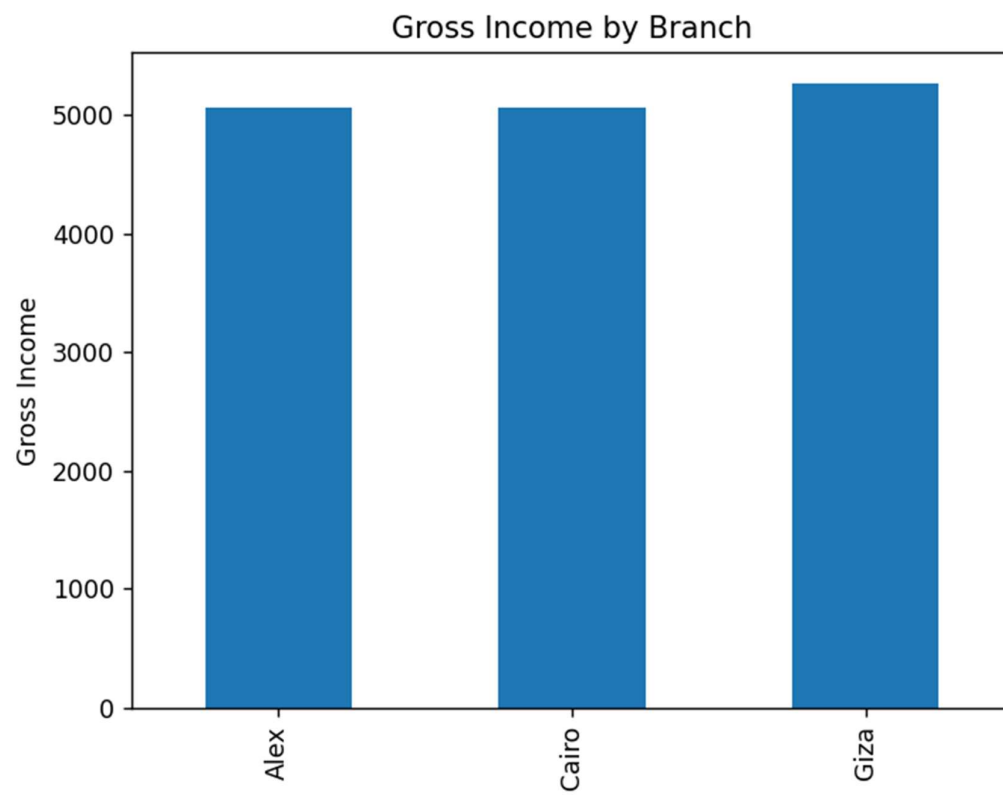
```
>>> In [23]: customer_type_sales = data.groupby('Customer type')['gross income'].sum()
In [24]: print("Gross Income by Customer Type:\n", customer_type_sales)
Gross Income by Customer Type:
  Customer type
Member      9033.084
Normal     6346.285
Name: gross income, dtype: float64
```

```
In [25]: city_sales = data.groupby('City')['gross income'].sum()
>>> In [26]: print("Gross Income by City:\n", city_sales)
Gross Income by City:
  City
Mandalay      5057.0320
Naypyitaw     5265.1765
Yangon        5057.1605
Name: gross income, dtype: float64
```

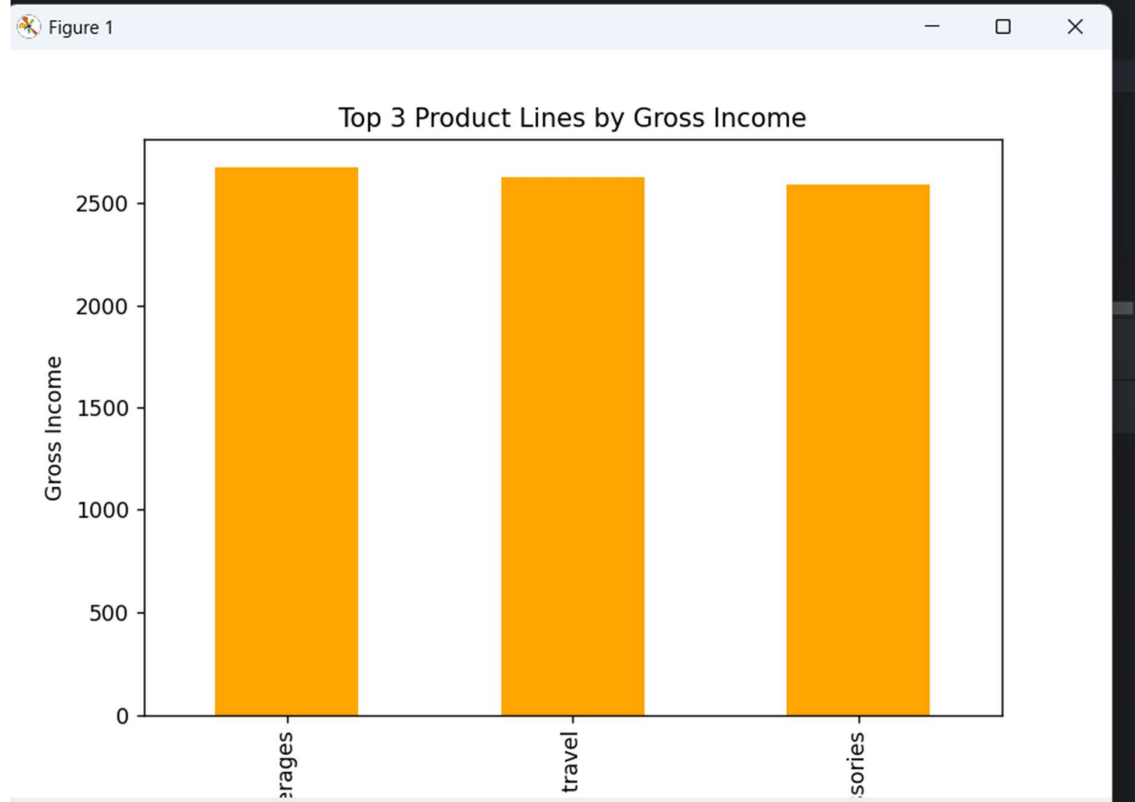
Step-7: Data Visualization

```
branch_sales.plot(kind='bar', title='Gross Income by Branch')  
plt.ylabel('Gross Income')  
plt.show()
```

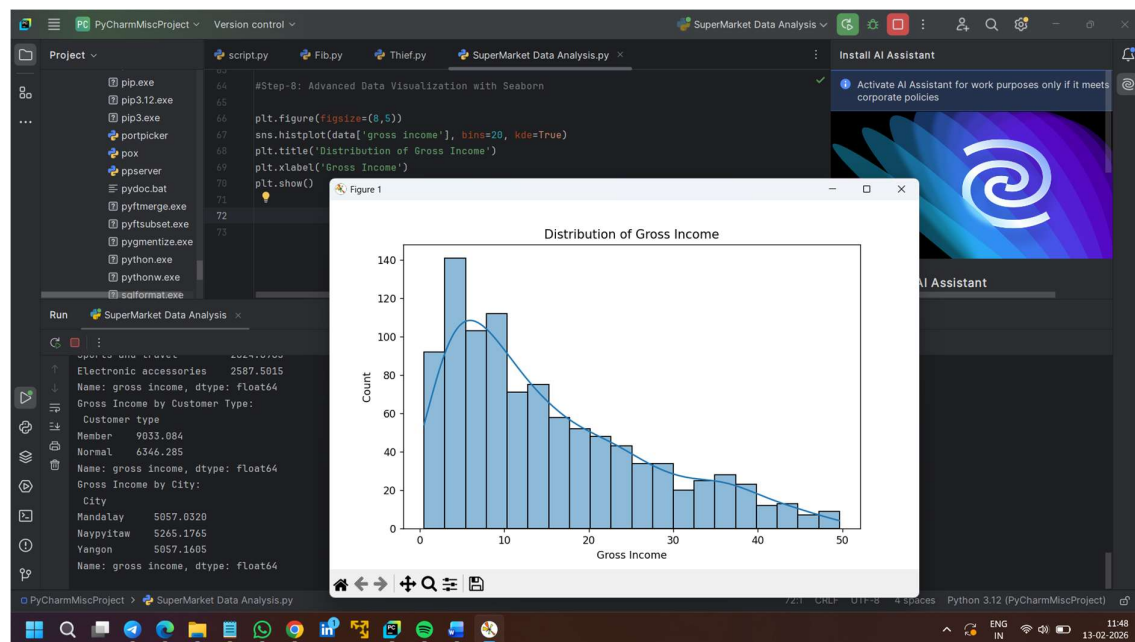
Figure 1

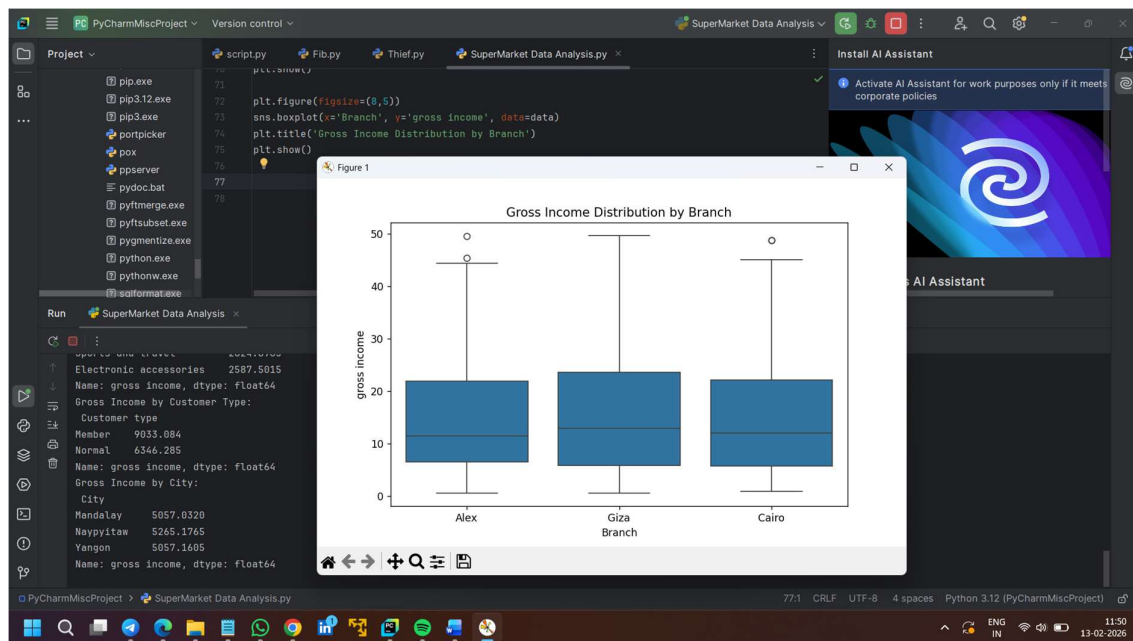



```
top_categories.plot(kind='bar', color='orange', title='Top 3 Product Lines by Gross Income')
plt.ylabel('Gross Income')
plt.show()
```



Step-8: Advanced Data Visualization with Seaborn

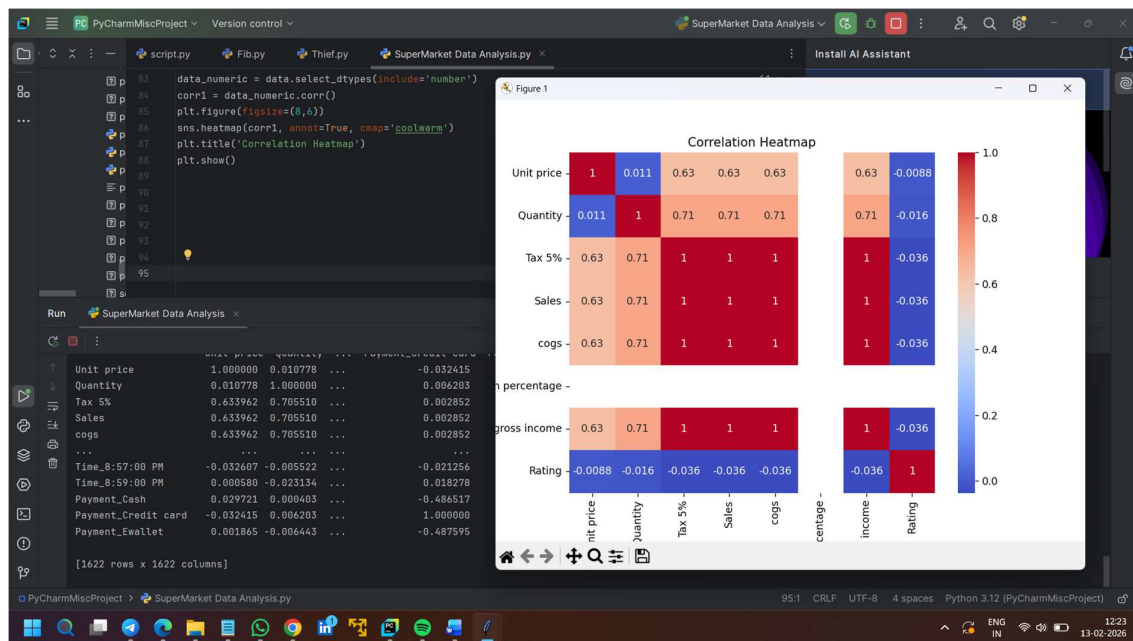




Step-9: Correlation Analysis

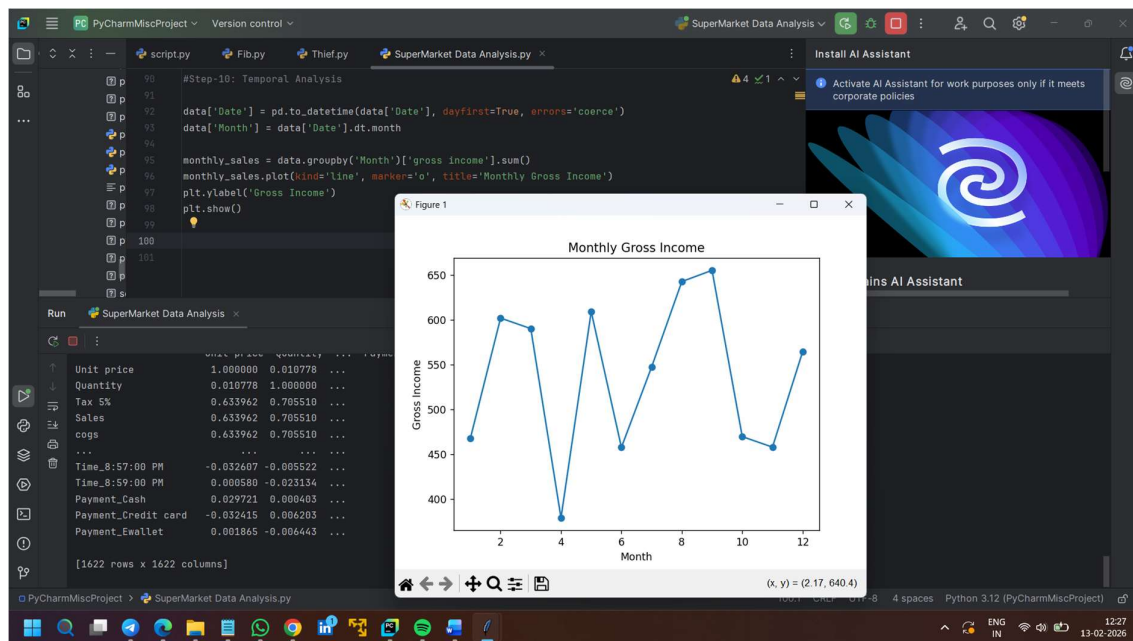
```
Python Console (1) x +
In [6]: data_encoded = pd.get_dummies(data)
...: corr = data_encoded.corr()
...: print("Correlation matrix:\n", corr)
Correlation matrix:
>>>
      Unit price  Quantity  ...  Payment_Credit card  Payment_Ewallet
Unit price      1.000000  0.010778  ...             -0.032415           0.001865
Quantity         0.010778  1.000000  ...             0.006203          -0.006443
Tax 5%           0.633962  0.705510  ...             0.002852          -0.012244
Sales            0.633962  0.705510  ...             0.002852          -0.012244
cogs             0.633962  0.705510  ...             0.002852          -0.012244
...              ...           ...           ...             ...             ...
Time_8:57:00 PM -0.032607 -0.005522  ...             -0.021256          -0.022962
Time_8:59:00 PM  0.000580 -0.023134  ...             0.018278           0.014597
In [7]:
>>>
Time_8:57:00 PM -0.032607 -0.005522  ...             -0.021256          -0.022962
Time_8:59:00 PM  0.000580 -0.023134  ...             0.018278           0.014597
Payment_Cash     0.029721  0.000403  ...             -0.486517          -0.525553
Payment_Credit card -0.032415  0.006203  ...             1.000000          -0.487595
Payment_Ewallet  0.001865 -0.006443  ...             -0.487595           1.000000

[1622 rows x 1622 columns]
```



Step-10: Temporal Analysis

```
Python Console (1) x +
In [8]: print(data['Date'].head(10))
0      1/5/2019
1      3/8/2019
2      3/3/2019
3      1/27/2019
4      2/8/2019
5      3/25/2019
6      2/25/2019
7      2/24/2019
8      1/10/2019
9      2/20/2019
Name: Date, dtype: str
```



Step-11: Pivot Table Analysis

```
>>> pivot_table = pd.pivot_table(data, values='gross income', index='Branch', columns='Customer type', aggfunc='sum')
>>> print(pivot_table)
```

	Member	Normal
Alex	2995.037	2062.1235
Cairo	2848.770	2208.2620
Giza	3189.277	2075.8995

