EDA supermarket case study

June 5, 2023

1 Description:

In this project, Python was used to do exploratory data analysis on a dataset from a supermarket. Significant conclusions were derived through investigating the data, cleaning the data, and doing statistical analysis and vizualisation. Actionable suggestions to enhance product ratings, pricing tactics, marketing initiatives and inventory control were developed as a result of these findings. The study shows how to find trends and make data-driven decisions for store improvement in a methodical manner.

1. Imported necessary libraries for data analysis and visualization in Python.
2. Imported the supermarket dataset and converted it into a data frame for further analysis.
3. Conducted an initial overview of the data to understand its structure and content.
4. Explored the dataset by examining the rows and columns, assessing its dimensions and characteristics.
5. Calculated statistical properties of each field, gaining insights into the distribution and variability of the data.
6. Determined the count of non-null values in each column and verified the data types for accurate analysis.
7. Plotted bar graphs to visualize columns with null values and stored the data frame into another variable for data manipulation and cleaning.
8. Replaced null values in the rating column with the mean of the existing ratings to ensure completeness and consistency.
9. Filled null values in the tax column by calculating 5% of the total amount, maintaining data integrity.
10. Replaced missing values in other columns (product line, gender, customer type, payment method) with the mode of their respective data to ensure representative values.
11. Imputed missing values in the branch column using city data, resulting in a complete and reliable dataset.
12. Modified the data types of the date and time columns to datetime format and added various features (e.g., month, year, days, weekend, quarter) using the date column.
13. Developed a function to analyze numeric columns (e.g., unit price, quantity) using swarm plots, strip plots, KDE plots, and volume plots to understand their spread and distribution.
14. Analyzed gender distribution through paragraph and pie charts, providing insights into customer demographics.
15. Created heatmaps to examine the relationships between numerical columns and identify correlations and patterns.
16. Generated pie charts to analyze sales across different stores and cities, allowing for market

performance assessment.
17. Implemented a comprehensive function to analyze total sales, mean sales, mean rating, and total quantity across categorical columns, providing insights into branch, city, customer type, time, gender, product line, and payment method.
18. Documented findings and actionable insights.

==

1. Imported necessary libraries for data analysis and visualization in Python.

[78]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

2. Imported the supermarket dataset and converted it into a data frame for further analysis.

[79]:
```
df=pd.read_csv('D:/DS/resume projects/supermarket EDA/columns_description.csv')
dfdict=pd.read_csv('D:/DS/resume projects/supermarket EDA/data_dict.csv')
```

3. Conducted an initial overview of the data to understand its structure and content.

[80]:
```
df.head()
```

[80]:
```
    Invoice ID Branch CustomerID       City Customer type  Gender  \
0  750-67-8428      A      C1888     Yangon        Member  Female
1  226-31-3081      C      C1475  Naypyitaw        Normal  Female
2  631-41-3108      A      C1746     Yangon        Normal    Male
3  123-19-1176      A      C1896     Yangon        Member    Male
4  373-73-7910      A      C1790     Yangon        Normal    Male

        Product line  Unit price  Quantity   Tax 5%   Total        Date  \
0  Health and beauty       74.69        10  37.3450  746.90  21-02-2019
1  Health and beauty       15.28         6   4.5840   91.68  27-05-2019
2  Health and beauty       46.33         7  16.2155  324.31  27-12-2019
3  Health and beauty       58.22        11  32.0210  640.42  15-11-2019
4  Health and beauty       86.31         7  30.2085  604.17  31-03-2019

    Time      Payment        cogs  gross margin percentage  gross income  \
0  13:08      Ewallet  711.333333                 4.761905     35.566667
1  10:29         Cash   76.400000                 4.761905     15.280000
2  13:23  Credit card  324.310000                 4.761905      0.000000
3  20:33      Ewallet  465.760000                 4.761905    174.660000
4  10:37      Ewallet  604.170000                 4.761905      0.000000

   Rating  Longitude  Latitude
0     9.1    96.1735   16.8409
1    10.0    96.0785   19.7633
2     7.4    96.1735   16.8409
3     8.4    96.1735   16.8409
```

```
4     NaN    96.1735   16.8409
```

[81]:
```
dfdict
```

[81]:

	Field	Description
0	Invoice ID	Invoice ID of the transaction
1	Branch	One out of 3 branches. Every city belongs to a…
2	CustomerID	Customer ID of the cutomer doing transaction
3	City	City where the tx took place. The chain has st…
4	Customer Type	Where a member or normal customer
5	Gender	Male or Female
6	Product Line	Product line of the product purchased
7	Unit Price	Unit price of product purchased
8	Quantity	Qty purchased
9	Tax 5%	Tax as a fixed % of invoice
10	Total	Total Invoice amount
11	Date	Date of tx
12	Time	Time of tx
13	Payment	Payment mode
14	cogs	Cost of goods sold
15	gross margin	Margin %
16	gross income	Margin amount
17	Rating	Rating given by cx out of 10
18	Longitude	Geo field for tx location
19	Latitude	Geo field for tx location

[116]:
```
df.style.background_gradient(cmap='cool')
```

<pandas.io.formats.style.Styler object at 0x000001FF980652D0>

4. Explored the dataset by examining the rows and columns, assessing its dimensions and characteristics.

[83]:
```
l=df.shape
print('The supermak=rket data has',l[0],'rows and',l[1],'columns')
```

```
The supermak=rket data has 1000 rows and 20 columns
```

5. Calculated statistical properties of each field, gaining insights into the distribution and variability of the data.

[84]:
```
df.describe().style.background_gradient(cmap='hot')
```

[84]: <pandas.io.formats.style.Styler at 0x1ff97d7d030>

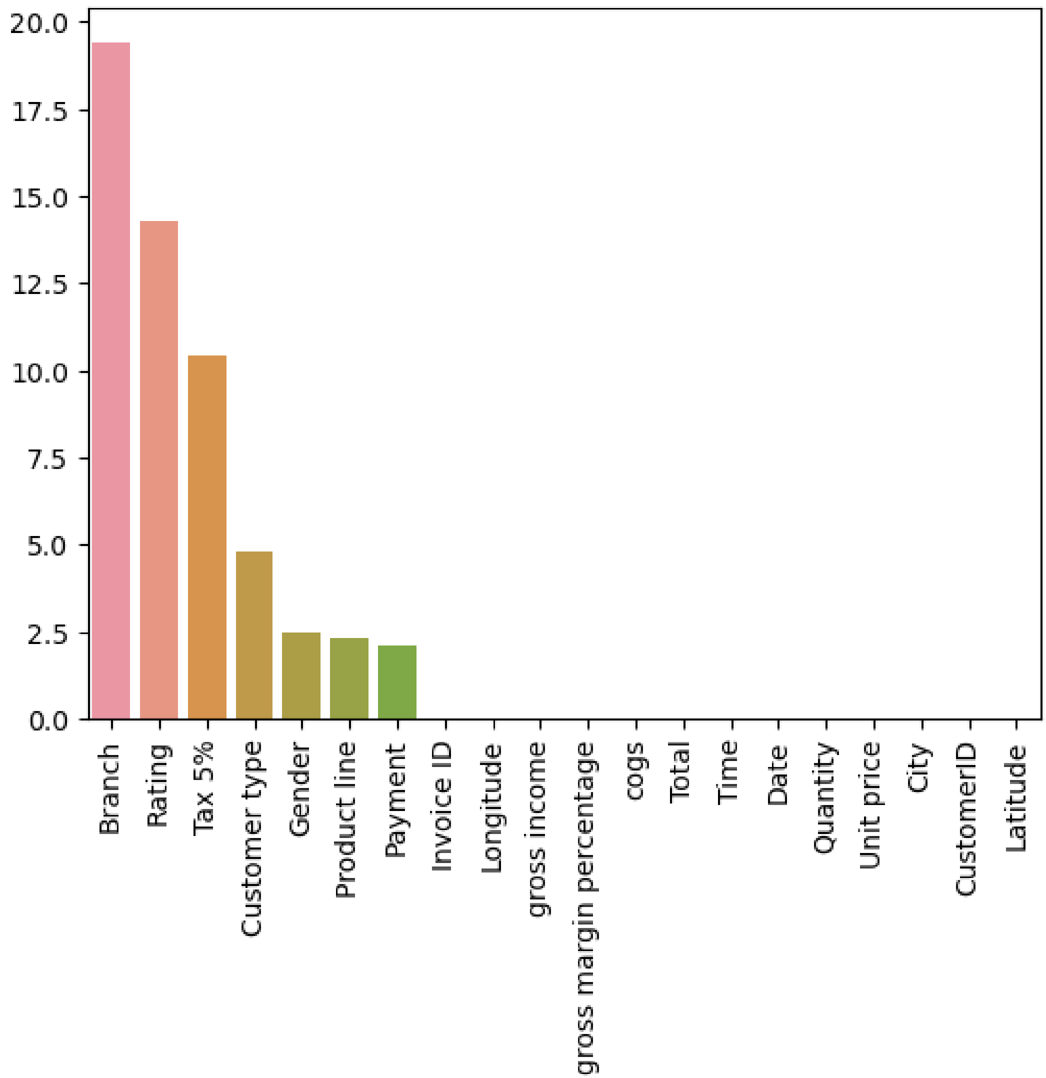
6. Determined the count of non-null values in each column and verified the data types for accurate analysis.

[85]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 20 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Invoice ID               1000 non-null   object
 1   Branch                   806 non-null    object
 2   CustomerID               1000 non-null   object
 3   City                     1000 non-null   object
 4   Customer type            952 non-null    object
 5   Gender                   975 non-null    object
 6   Product line             977 non-null    object
 7   Unit price               1000 non-null   float64
 8   Quantity                 1000 non-null   int64
 9   Tax 5%                   896 non-null    float64
 10  Total                    1000 non-null   float64
 11  Date                     1000 non-null   object
 12  Time                     1000 non-null   object
 13  Payment                  979 non-null    object
 14  cogs                     1000 non-null   float64
 15  gross margin percentage  1000 non-null   float64
 16  gross income             1000 non-null   float64
 17  Rating                   857 non-null    float64
 18  Longitude                1000 non-null   float64
 19  Latitude                 1000 non-null   float64
dtypes: float64(9), int64(1), object(10)
memory usage: 156.4+ KB
```

7. Plotted bar graphs to visualize columns with null values and stored the data frame into another variable for data manipulation and cleaning.

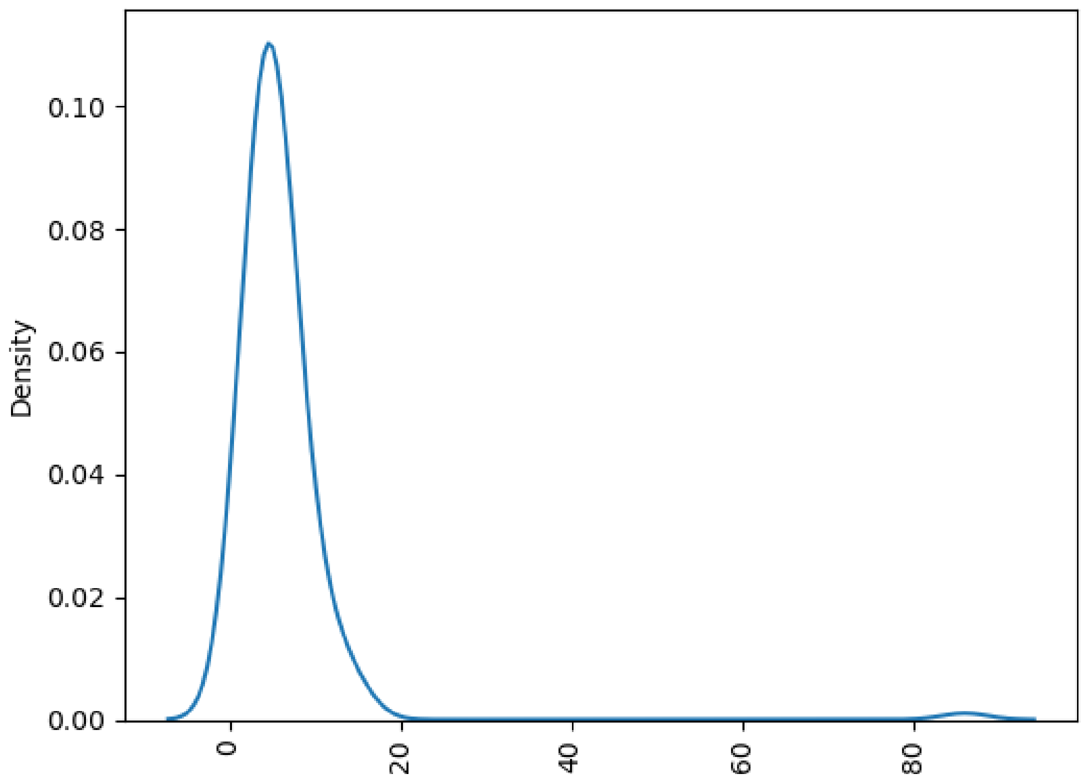
[86]:
```
f=(df.isna().mean()*100).sort_values(ascending=False)
sns.barplot(x=f.index,y=f.values)
plt.xticks(rotation=90)
plt.show()
```



[87]:
```
#orignal data in df and manupulation in sdf
sdf=df
```

8. Replaced null values in the rating column with the mean of the existing ratings to ensure completeness and consistency.

[88]:
```
#Rating is summeticaly distributed replacing null with mean
b=df.Rating.value_counts()
sns.kdeplot(b.values)
plt.xticks(rotation=90)
plt.show()
```



[89]:
```
df.Rating.describe()
```

[89]:
```
count    857.000000
mean       7.462625
std        1.776179
min        4.000000
25%        5.900000
50%        7.455000
75%        9.100000
max       10.000000
Name: Rating, dtype: float64
```

[90]:
```
sdf['Rating']=df['Rating'].fillna(df.Rating.mean())
```

[91]:
```
sdf.Rating.describe()
```

[91]:
```
count    1000.000000
mean        7.462625
std         1.644148
min         4.000000
25%         6.200000
50%         7.462625
```

```
75%           8.820000
max          10.000000
Name: Rating, dtype: float64
```

9. Filled null values in the tax column by calculating 5% of the total amount, maintaining data integrity.

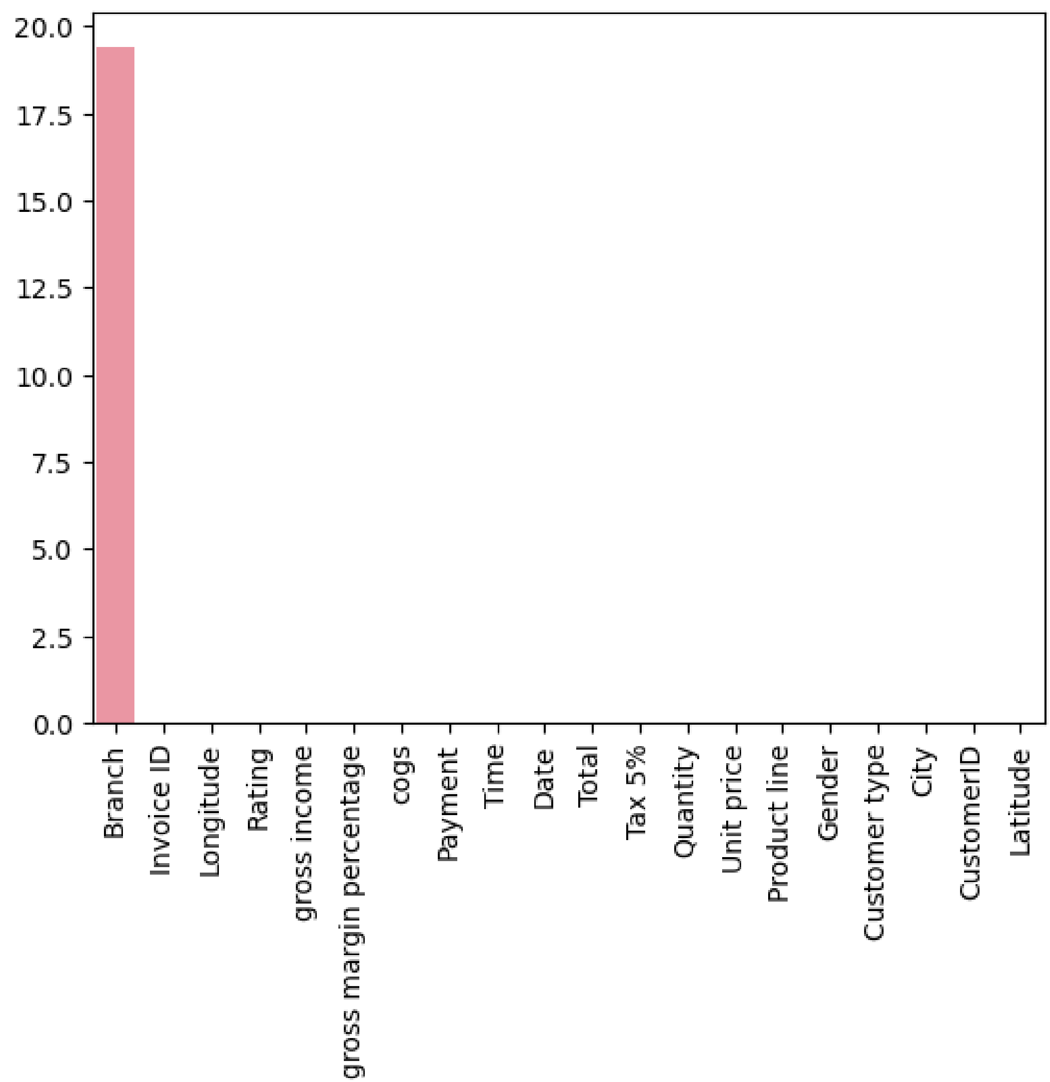
[92]:
```
# filling null of tax column by calculating it from total column
sdf['Tax 5%']=df['Tax 5%'].fillna(df['Total']*.05)
```

10. Replaced missing values in other columns (product line, gender, customer type, payment method) with the mode of their respective data to ensure representative values.

[93]:
```
# filling null values in categorical column by mode
def repmode(col):
    sdf[col]=df[col].fillna(df[col].mode()[0])

repmode('Product line')
repmode('Gender')
repmode('Customer type')
repmode('Payment')
```

11. Imputed missing values in the branch column using city data, resulting in a complete and reliable dataset.

[94]:
```
# Branch column need to be cleaned
f=(df.isna().mean()*100).sort_values(ascending=False)
sns.barplot(x=f.index,y=f.values)
plt.xticks(rotation=90)
plt.show()
```



[95]:
```
sdf.groupby(['City','Branch'])['Invoice ID'].count()
```

[95]:
```
City       Branch
Mandalay   B         265
Naypyitaw  C         260
Yangon     A         281
Name: Invoice ID, dtype: int64
```

[96]:
```
sdf.loc[(sdf['Branch'].isnull()) & (sdf['City'] == 'Mandalay'), 'Branch'] = 'B'
sdf.loc[(sdf['Branch'].isnull()) & (sdf['City'] == 'Naypyitaw'), 'Branch'] = 'C'
sdf.loc[(sdf['Branch'].isnull()) & (sdf['City'] == 'Yangon'), 'Branch'] = 'A'
```

[97]:
```
sdf.isna().mean()*100
```

[97]:
```
Invoice ID                 0.0
Branch                     0.0
CustomerID                 0.0
City                       0.0
Customer type              0.0
Gender                     0.0
Product line               0.0
Unit price                 0.0
Quantity                   0.0
Tax 5%                     0.0
Total                      0.0
Date                       0.0
Time                       0.0
Payment                    0.0
cogs                       0.0
gross margin percentage    0.0
gross income               0.0
Rating                     0.0
Longitude                  0.0
Latitude                   0.0
dtype: float64
```

12. Modified the data types of the date and time columns to datetime format and added various features (e.g., month, year, days, weekend, quarter) using the date column.

[98]:
```
sdf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 20 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Invoice ID               1000 non-null   object
 1   Branch                   1000 non-null   object
 2   CustomerID               1000 non-null   object
 3   City                     1000 non-null   object
 4   Customer type            1000 non-null   object
 5   Gender                   1000 non-null   object
 6   Product line             1000 non-null   object
 7   Unit price               1000 non-null   float64
 8   Quantity                 1000 non-null   int64
 9   Tax 5%                   1000 non-null   float64
 10  Total                    1000 non-null   float64
 11  Date                     1000 non-null   object
 12  Time                     1000 non-null   object
 13  Payment                  1000 non-null   object
 14  cogs                     1000 non-null   float64
 15  gross margin percentage  1000 non-null   float64
```

```
 16   gross income              1000 non-null   float64
 17   Rating                    1000 non-null   float64
 18   Longitude                 1000 non-null   float64
 19   Latitude                  1000 non-null   float64
dtypes: float64(9), int64(1), object(10)
memory usage: 156.4+ KB
```

[99]:
```
#correcing datatype of time,date column
sdf['Date']=sdf['Date'].astype('datetime64')
sdf['Time']=sdf['Time'].astype('datetime64')
sdf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 20 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Invoice ID                1000 non-null   object
 1   Branch                    1000 non-null   object
 2   CustomerID                1000 non-null   object
 3   City                      1000 non-null   object
 4   Customer type             1000 non-null   object
 5   Gender                    1000 non-null   object
 6   Product line              1000 non-null   object
 7   Unit price                1000 non-null   float64
 8   Quantity                  1000 non-null   int64
 9   Tax 5%                    1000 non-null   float64
 10  Total                     1000 non-null   float64
 11  Date                      1000 non-null   datetime64[ns]
 12  Time                      1000 non-null   datetime64[ns]
 13  Payment                   1000 non-null   object
 14  cogs                      1000 non-null   float64
 15  gross margin percentage   1000 non-null   float64
 16  gross income              1000 non-null   float64
 17  Rating                    1000 non-null   float64
 18  Longitude                 1000 non-null   float64
 19  Latitude                  1000 non-null   float64
dtypes: datetime64[ns](2), float64(9), int64(1), object(8)
memory usage: 156.4+ KB
```

```
C:\Users\Kundan Mourya\AppData\Local\Temp\ipykernel_19992\846853813.py:2:
UserWarning: Parsing dates in DD/MM/YYYY format when dayfirst=False (the
default) was specified. This may lead to inconsistently parsed dates! Specify a
format to ensure consistent parsing.
  sdf['Date']=sdf['Date'].astype('datetime64')
```

[100]:
```
sdf['Month']=sdf['Date'].dt.month
sdf['Year']=sdf['Date'].dt.year
```

```
sdf['Day']=sdf['Date'].dt.day
sdf['Weekday']=sdf['Date'].dt.day_name()
sdf['Qruarter']=sdf['Date'].dt.quarter
#df['week_number'] = df['Date'].dt.week
df['annual_week_number'] = df['Date'].dt.isocalendar().week

sdf['Hour']=sdf['Time'].dt.hour
sdf['Minute']=sdf['Time'].dt.minute
```

[101]:
```
sdf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 28 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Invoice ID               1000 non-null   object
 1   Branch                   1000 non-null   object
 2   CustomerID               1000 non-null   object
 3   City                     1000 non-null   object
 4   Customer type            1000 non-null   object
 5   Gender                   1000 non-null   object
 6   Product line             1000 non-null   object
 7   Unit price               1000 non-null   float64
 8   Quantity                 1000 non-null   int64
 9   Tax 5%                   1000 non-null   float64
 10  Total                    1000 non-null   float64
 11  Date                     1000 non-null   datetime64[ns]
 12  Time                     1000 non-null   datetime64[ns]
 13  Payment                  1000 non-null   object
 14  cogs                     1000 non-null   float64
 15  gross margin percentage  1000 non-null   float64
 16  gross income             1000 non-null   float64
 17  Rating                   1000 non-null   float64
 18  Longitude                1000 non-null   float64
 19  Latitude                 1000 non-null   float64
 20  Month                    1000 non-null   int64
 21  Year                     1000 non-null   int64
 22  Day                      1000 non-null   int64
 23  Weekday                  1000 non-null   object
 24  Qruarter                 1000 non-null   int64
 25  annual_week_number       1000 non-null   UInt32
 26  Hour                     1000 non-null   int64
 27  Minute                   1000 non-null   int64
dtypes: UInt32(1), datetime64[ns](2), float64(9), int64(7), object(9)
memory usage: 215.9+ KB
```

[102]:
```
sdf.head()
```

[102]:
```
       Invoice ID Branch CustomerID       City Customer type  Gender  \
0  750-67-8428      A      C1888     Yangon        Member  Female
1  226-31-3081      C      C1475  Naypyitaw        Normal  Female
2  631-41-3108      A      C1746     Yangon        Normal    Male
3  123-19-1176      A      C1896     Yangon        Member    Male
4  373-73-7910      A      C1790     Yangon        Normal    Male

         Product line  Unit price  Quantity   Tax 5%  …  Longitude Latitude  \
0  Health and beauty       74.69        10  37.3450  …    96.1735  16.8409
1  Health and beauty       15.28         6   4.5840  …    96.0785  19.7633
2  Health and beauty       46.33         7  16.2155  …    96.1735  16.8409
3  Health and beauty       58.22        11  32.0210  …    96.1735  16.8409
4  Health and beauty       86.31         7  30.2085  …    96.1735  16.8409

  Month  Year  Day   Weekday  Qruarter  annual_week_number  Hour  Minute
0      2  2019   21  Thursday         1                   8    13       8
1      5  2019   27    Monday         2                  22    10      29
2     12  2019   27    Friday         4                  52    13      23
3     11  2019   15    Friday         4                  46    20      33
4      3  2019   31    Sunday         1                  13    10      37

[5 rows x 28 columns]
```

[103]:
```
sdf.info()
```

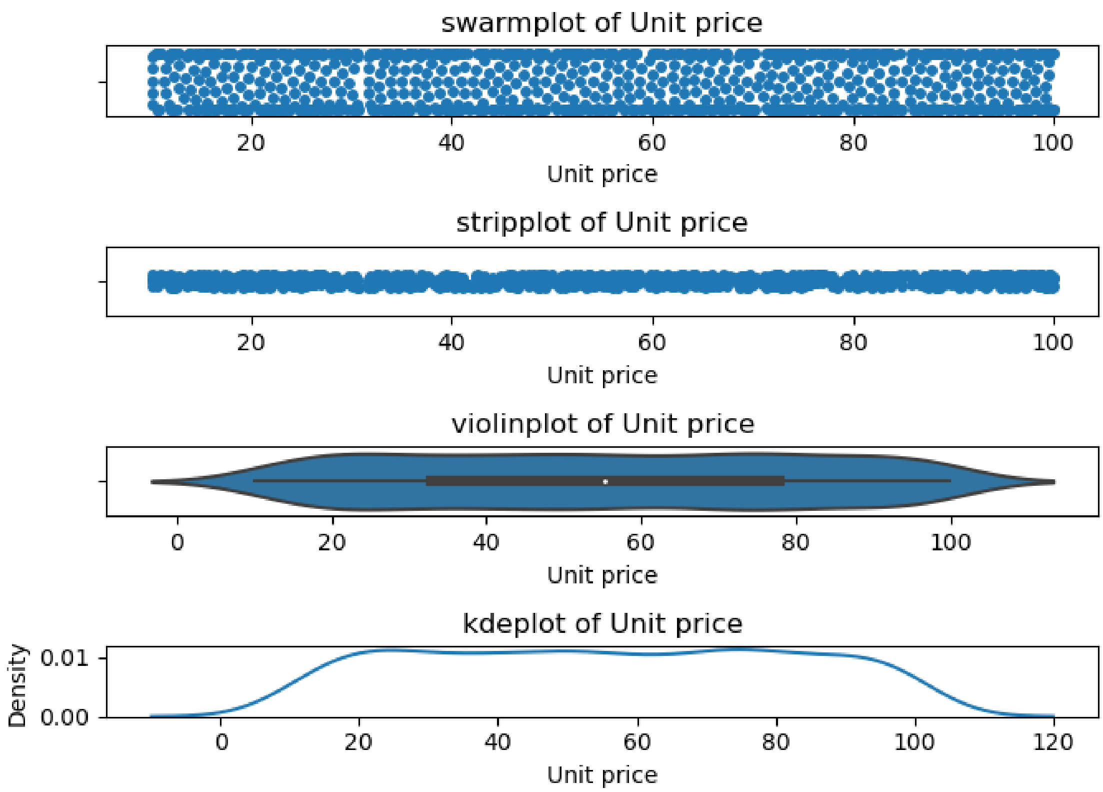
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 28 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Invoice ID               1000 non-null   object
 1   Branch                   1000 non-null   object
 2   CustomerID               1000 non-null   object
 3   City                     1000 non-null   object
 4   Customer type            1000 non-null   object
 5   Gender                   1000 non-null   object
 6   Product line             1000 non-null   object
 7   Unit price               1000 non-null   float64
 8   Quantity                 1000 non-null   int64
 9   Tax 5%                   1000 non-null   float64
 10  Total                    1000 non-null   float64
 11  Date                     1000 non-null   datetime64[ns]
 12  Time                     1000 non-null   datetime64[ns]
 13  Payment                  1000 non-null   object
 14  cogs                     1000 non-null   float64
 15  gross margin percentage  1000 non-null   float64
 16  gross income             1000 non-null   float64
 17  Rating                   1000 non-null   float64
```

```
 18  Longitude                1000 non-null   float64
 19  Latitude                 1000 non-null   float64
 20  Month                    1000 non-null   int64
 21  Year                     1000 non-null   int64
 22  Day                      1000 non-null   int64
 23  Weekday                  1000 non-null   object
 24  Qruarter                 1000 non-null   int64
 25  annual_week_number       1000 non-null   UInt32
 26  Hour                     1000 non-null   int64
 27  Minute                   1000 non-null   int64
dtypes: UInt32(1), datetime64[ns](2), float64(9), int64(7), object(9)
memory usage: 215.9+ KB
```

13. Developed a function to analyze numeric columns (e.g., unit price, quantity) using swarm plots, strip plots, KDE plots, and volume plots to understand their spread and distribution.

[104]:
```
def fun(col):
    fig,ax=plt.subplots(4,1,figsize=(7,5))
    sns.swarmplot(data=sdf, x=col, ax=ax[0])
    ax[0].set_title('swarmplot of '+col)

    sns.stripplot(data=sdf, x=col, ax=ax[1])
    ax[1].set_title('stripplot of '+col)

    sns.violinplot(data=sdf, x=col, ax=ax[2])
    ax[2].set_title('violinplot of '+col)

    sns.kdeplot(sdf[col],ax=ax[3])
    ax[3].set_title('kdeplot of '+col)
    plt.tight_layout()
    plt.show()

fun('Unit price')
```

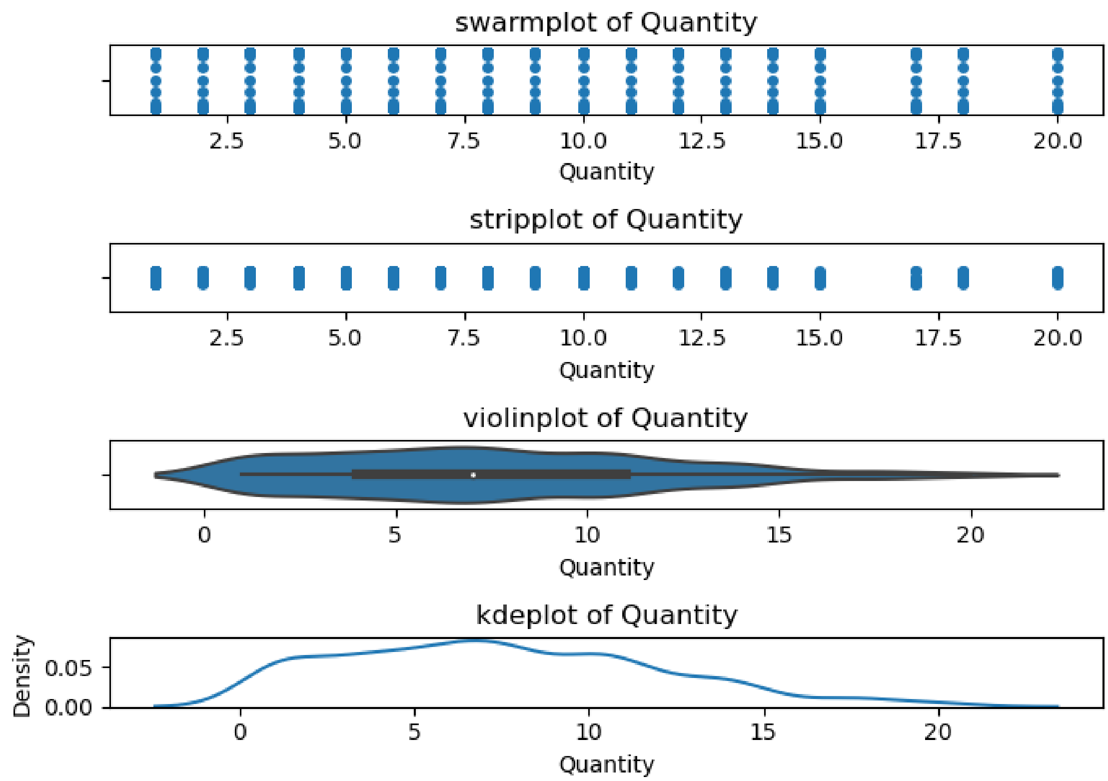
```
C:\Users\Kundan Mourya\anaconda3\lib\site-packages\seaborn\categorical.py:3544:
UserWarning: 64.4% of the points cannot be placed; you may want to decrease the
size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
```



[105]:

```
fun('Quantity')
```

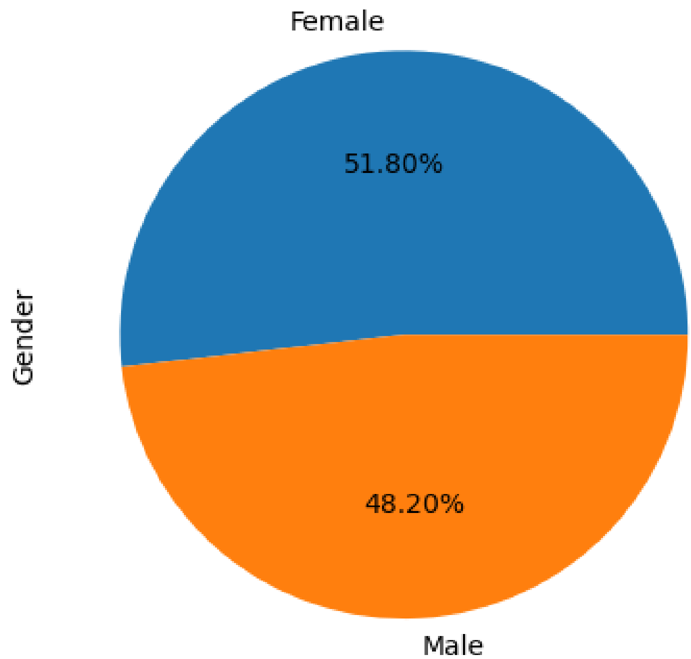
```
C:\Users\Kundan Mourya\anaconda3\lib\site-packages\seaborn\categorical.py:3544:
UserWarning: 7.1% of the points cannot be placed; you may want to decrease the
size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
C:\Users\Kundan Mourya\anaconda3\lib\site-packages\seaborn\categorical.py:3544:
UserWarning: 91.0% of the points cannot be placed; you may want to decrease the
size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
```



14. Analyzed gender distribution through pie charts, providing insights into customer demographics.

[106]:
```
sdf.Gender.value_counts().plot.pie(autopct='%1.2f%%')
```

[106]:
```
<Axes: ylabel='Gender'>
```



15. Created heatmaps to examine the relationships between numerical columns and identify correlations and patterns.

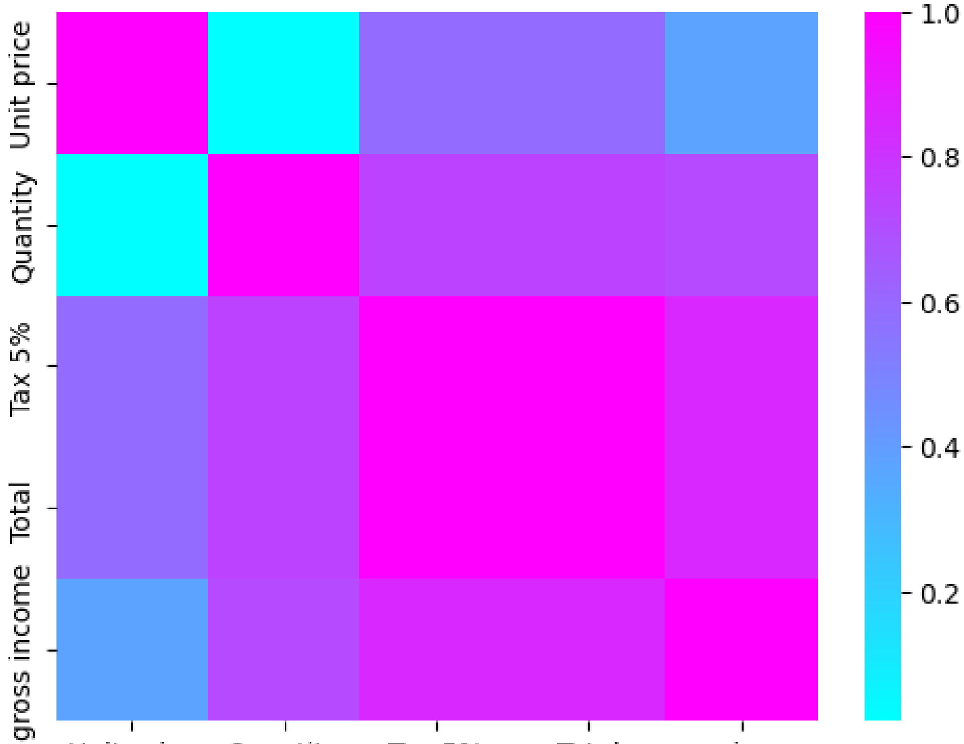
[107]:
```
sdf[['City','Longitude','Latitude']].drop_duplicates()
```

[107]:

	City	Longitude	Latitude
0	Yangon	96.1735	16.8409
1	Naypyitaw	96.0785	19.7633
9	Mandalay	96.0891	21.9588

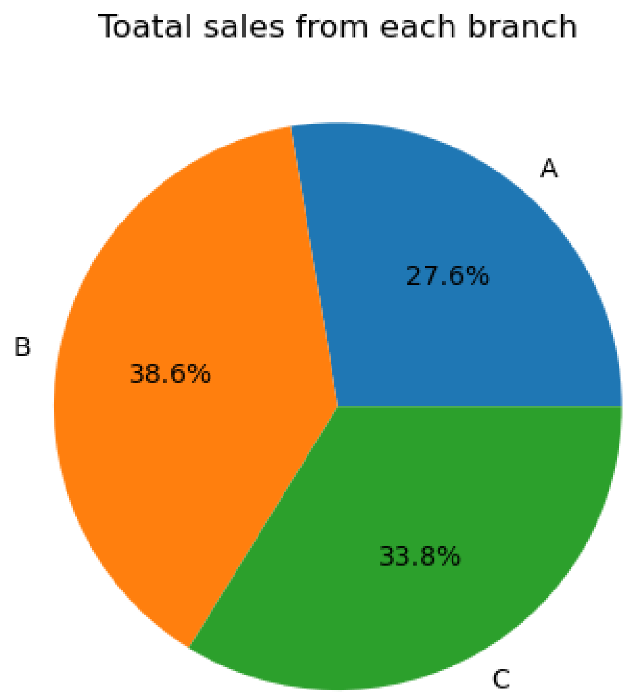
[108]:
```
sns.heatmap(sdf[['Unit price','Quantity','Tax 5%','Total','gross income']].
↪corr(),cmap='cool')
```

[108]: <Axes: >



16. Generated pie charts to analyze sales across different stores/cities, allowing for market performance assessment.

[109]:
```
h=sdf.groupby(['Branch'])['Total'].sum()
plt.pie(h.values,labels=h.index,autopct='%1.1f%%')
plt.title('Toatal sales from each branch')
plt.show()
```



17. Implemented a comprehensive function to analyze total sales, mean sales, mean rating, and total quantity across categorical columns, providing insights into branch, city, customer type, time, gender, product line, and payment method.

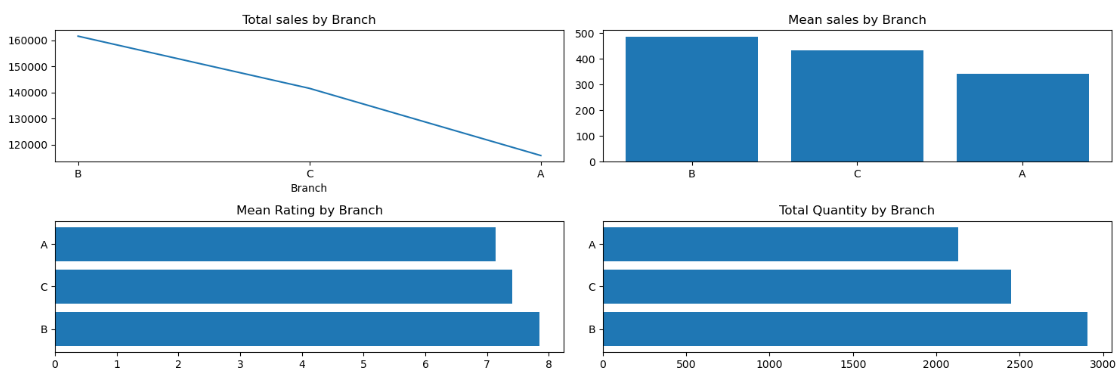
[110]:
```
catcol=list(sdf.columns[sdf.dtypes=='O'])
catcol.remove('Invoice ID')
catcol.remove('CustomerID')
catcol
```

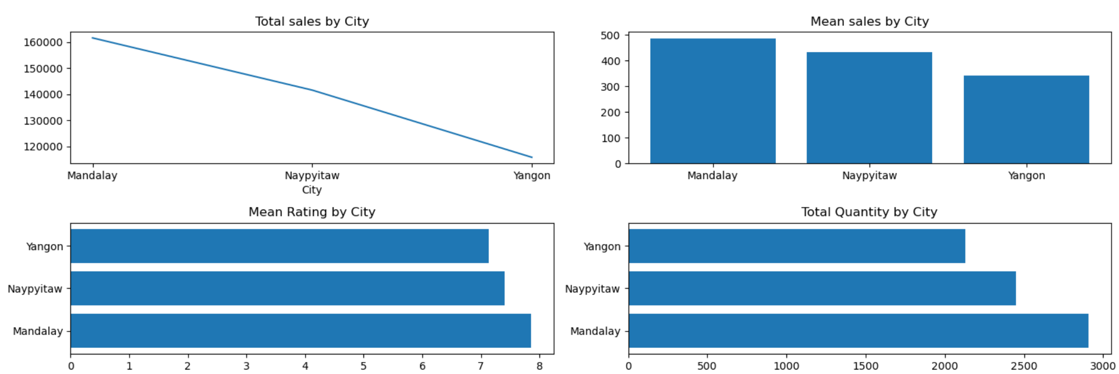
[110]:
```
['Branch',
 'City',
 'Customer type',
 'Gender',
 'Product line',
 'Payment',
 'Weekday']
```

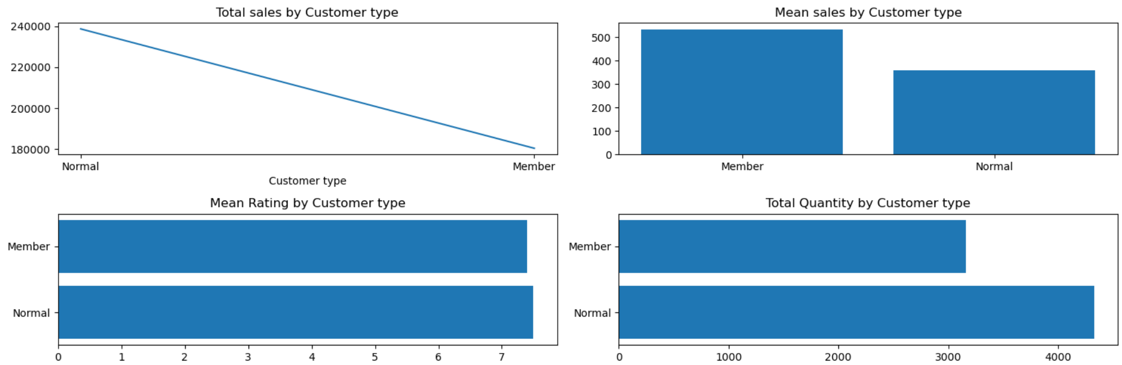
[111]:
```
def ubit(col):

    a=(sdf.groupby(sdf[col])['Total'].sum()).sort_values(ascending=False)
    b=(sdf.groupby([sdf[col]])['Total'].mean()).sort_values(ascending=False)
    c=(sdf.groupby([sdf[col]])['Rating'].mean()).sort_values(ascending=False)
    d=(sdf.groupby([sdf[col]])['Quantity'].sum()).sort_values(ascending=False)
```

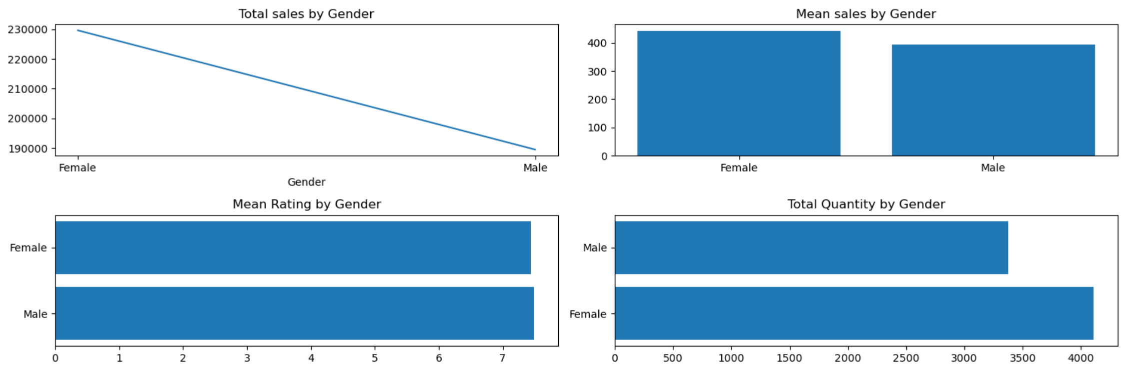
```
    fig,ax=plt.subplots(2,2,figsize=(15,5))

    sns.lineplot(x=a.index, y=a.values, ax=ax[0, 0])
    ax[0,0].set_title('Total sales by '+col)

    ax[0,1].bar(height=b.values,x=b.index)
    ax[0,1].set_title('Mean sales by '+col)

    ax[1,0].barh(y=c.index, width=c.values)
    ax[1,0].set_title('Mean Rating by '+col)

    ax[1,1].barh(y=d.index, width=d.values)
    ax[1,1].set_title('Total Quantity by '+col)

    plt.tight_layout()
```

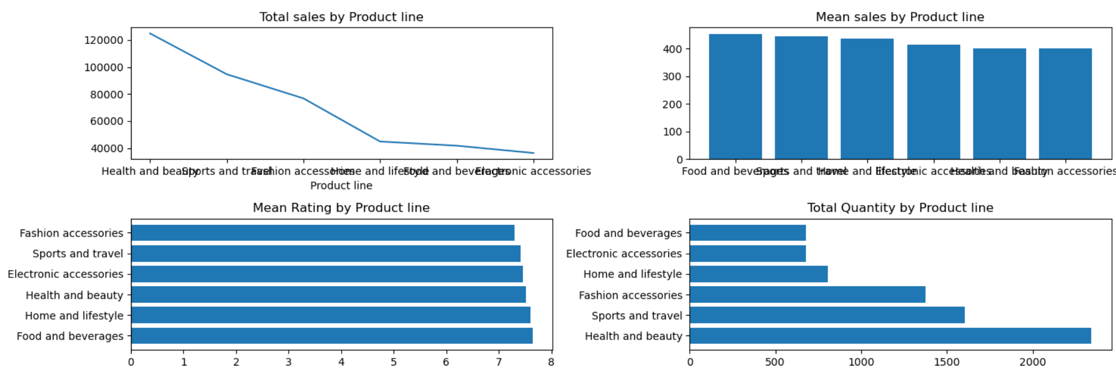
[112]:
```
for i in catcol:
    ubit(i)
```

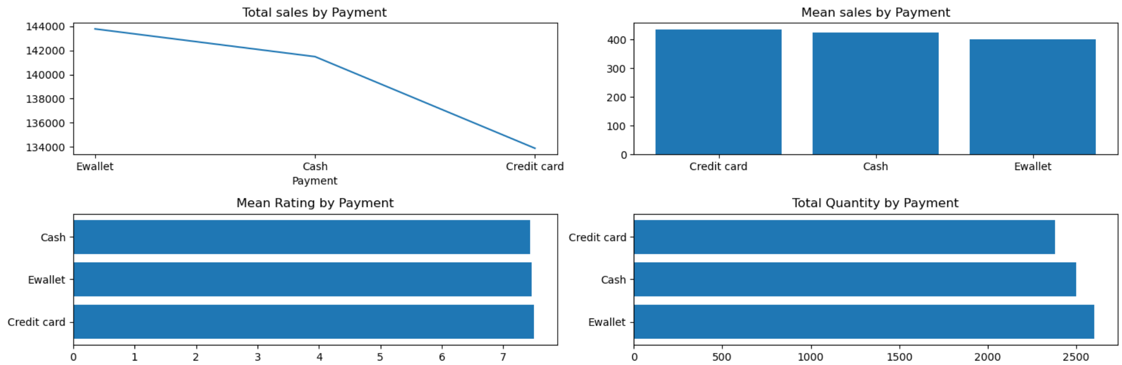


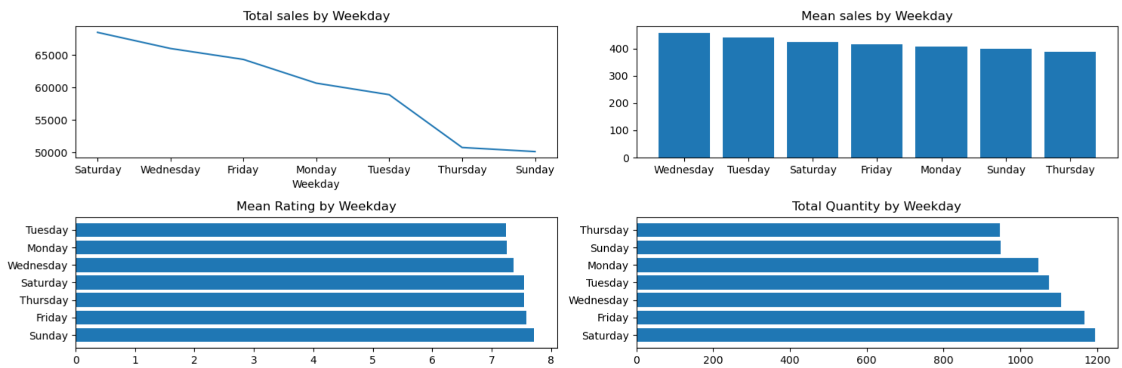



Total sales by Customer type
240000
220000
200000
180000
Normal
Member
Customer type
Mean sales by Customer type
500
400
300
200
100
0
Member
Normal
Mean Rating by Customer type
Member
Normal
0 1 2 3 4 5 6 7
Total Quantity by Customer type
Member
Normal
0 1000 2000 3000 4000


Total sales by Gender
230000
220000
210000
200000
190000
Female
Male
Gender
Mean sales by Gender
400
300
200
100
0
Female
Male
Mean Rating by Gender
Female
Male
0 1 2 3 4 5 6 7
Total Quantity by Gender
Male
Female
0 500 1000 1500 2000 2500 3000 3500 4000


Total sales by Product line
120000
100000
80000
60000
40000
Health and beauty
Sports and travel
Fashion accessories
Home and lifestyle
Food and beverages
Electronic accessories
Product line
Mean sales by Product line
400
300
200
100
0
Food and beverages
Sports and travel
Home and lifestyle
Electronic accessories
Health and beauty
Fashion accessories
Mean Rating by Product line
Fashion accessories
Sports and travel
Electronic accessories
Health and beauty
Home and lifestyle
Food and beverages
0 1 2 3 4 5 6 7 8
Total Quantity by Product line
Food and beverages
Electronic accessories
Home and lifestyle
Fashion accessories
Sports and travel
Health and beauty
0 500 1000 1500 2000





2 Insights

- Improve product ratings: With the knowledge that missing ratings were replaced with the mean rating, the supermarket can focus on improving the quality and customer happiness of items with lower ratings in order to better the overall shopping experience for customers.
- Modify pricing strategies: By evaluating the variation in unit prices and quantities, the supermarket may spot patterns in pricing and make necessary price adjustments. Reevaluating pricing tactics for particular products, for instance, may be advantageous if their prices vary widely.
- Boost marketing efforts: Bar graphs showing gender distribution can be utilised for more accurate marketing activities. In order to increase consumer engagement and sales, the supermarket can customise promotional events, product displays, and adverts to individual gender preferences.
- The supermarket can pinpoint failing locations and take action to improve inventory levels, product selection, and operational efficiency in those regions through analysing sales distribution across different stores and cities.
- Optimising product line offerings: Supermarkets may determine the best-performing product categories and make informed decisions about inventory management, marketing strategies,

and possible growth of profitable product lines via the insights gained from analysing the total sales, mean sales, mean rating, and total quantity across different product lines.

- Determine peak sales times: The supermarket can identify periods of highest sales and make suitable preparations by using the data gathered from the date column, such as month, year, days, weekend, and quarter. To maximise sales at times of high demand, this could involve altering employee numbers, inventory restocking, and promotional efforts.

2.0.1 Based on the data that was gathered, these choices might result in greater customer happiness, more sales, improved management of inventory, and overall company success for the supermarket.