Untitled

October 12, 2024

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import calmap
from ydata_profiling import ProfileReport
```

[2]:
```
df=pd.read_csv('supermarket_sales.csv')
```

[3]:
```
df.head(20)
```

[3]:

	Invoice ID	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%	Total	Date
0	750-67-8428	A	Yangon	Member	Female	Health and beauty	74.69	7.0	26.1415	548.9715	1/5/19
1	226-31-3081	C	Naypyitaw	Normal	Female	Electronic accessories	15.28	5.0	3.8200	80.2200	3/8/19
2	631-41-3108	A	Yangon	Normal	Male	Home and lifestyle	46.33	7.0	16.2155	340.5255	3/3/19
3	123-19-1176	A	Yangon	Member	Male						
4	373-73-7910	A	Yangon	Normal	Male						
5	699-14-3026	C	Naypyitaw	Normal	Male						
6	355-53-5943	A	Yangon	Member	Female						
7	315-22-5665	C	Naypyitaw	Normal	Female						
8	665-32-9167	A	Yangon	Member	Female						
9	692-92-5582	B	Mandalay	Member	Female						
10	351-62-0822	B	Mandalay	Member	Female						
11	529-56-3974	B	Mandalay	Member	Male						
12	365-64-0515	A	Yangon	Normal	Female						
13	252-56-2699	A	Yangon	Normal	Male						
14	829-34-3910	A	Yangon	Normal	Female						
15	299-46-1805	B	Mandalay	Member	Female						
16	656-95-9349	A	Yangon	NaN	Female						
17	765-26-6951	A	Yangon	Normal	Male						
18	329-62-1586	A	Yangon	Normal	Male						
19	319-50-3348	B	Mandalay	Normal	Female						

3	Health and beauty	58.22	8.0	23.2880	489.0480	1/27/19
4	Sports and travel	86.31	7.0	30.2085	634.3785	2/8/19
5	Electronic accessories	85.39	7.0	29.8865	627.6165	3/25/19
6	NaN	68.84	6.0	20.6520	433.6920	2/25/19
7	NaN	73.56	10.0	36.7800	772.3800	2/24/19
8	NaN	36.26	2.0	3.6260	76.1460	1/10/19
9	NaN	54.84	3.0	8.2260	172.7460	2/20/19
10	Fashion accessories	14.48	4.0	2.8960	60.8160	2/6/19
11	Electronic accessories	25.51	4.0	5.1020	107.1420	3/9/19
12	Electronic accessories	46.95	5.0	11.7375	246.4875	2/12/19
13	Food and beverages	43.19	10.0	21.5950	453.4950	2/7/19
14	Health and beauty	71.38	10.0	35.6900	749.4900	3/29/19
15	Sports and travel	93.72	6.0	28.1160	590.4360	1/15/19
16	Health and beauty	68.93	7.0	24.1255	506.6355	3/11/19
17	Sports and travel	72.61	6.0	21.7830	457.4430	1/1/19
18	Food and beverages	54.67	3.0	8.2005	172.2105	1/21/19
19	Home and lifestyle	40.30	2.0	4.0300	84.6300	3/11/19

	Time	Payment	cogs	gross margin percentage	gross income	Rating
0	13:08	Ewallet	522.83	4.761905	26.1415	9.1
1	10:29	Cash	76.40	4.761905	3.8200	9.6
2	13:23	Credit card	324.31	4.761905	16.2155	7.4
3	20:33	Ewallet	465.76	4.761905	23.2880	8.4
4	10:37	Ewallet	604.17	4.761905	30.2085	5.3
5	18:30	Ewallet	597.73	4.761905	29.8865	4.1
6	14:36	Ewallet	413.04	4.761905	20.6520	5.8
7	11:38	Ewallet	735.60	4.761905	36.7800	8.0
8	17:15	Credit card	72.52	4.761905	3.6260	7.2
9	13:27	Credit card	164.52	4.761905	8.2260	5.9
10	18:07	Ewallet	57.92	4.761905	2.8960	4.5
11	17:03	Cash	102.04	4.761905	5.1020	6.8
12	10:25	Ewallet	234.75	4.761905	11.7375	7.1
13	16:48	Ewallet	431.90	4.761905	21.5950	8.2
14	19:21	Cash	713.80	4.761905	35.6900	5.7
15	16:19	Cash	562.32	4.761905	28.1160	4.5
16	11:03	Credit card	482.51	4.761905	24.1255	4.6
17	10:39	Credit card	435.66	4.761905	21.7830	6.9
18	18:00	Credit card	164.01	4.761905	8.2005	8.6
19	15:30	Ewallet	80.60	4.761905	4.0300	4.4

[4]:
```
df.tail()
```

[4]:

	Invoice ID	Branch	City	Customer type	Gender	\
998	347-56-2442	A	Yangon	Normal	Male	
999	849-09-3807	A	Yangon	Member	Female	
1000	849-09-3807	A	Yangon	Member	Female	
1001	745-74-0715	A	Yangon	Normal	Male	

```
1002  452-04-8808      B  Mandalay        Normal    Male

                Product line  Unit price  Quantity  Tax 5%    Total     Date  \
998        Home and lifestyle       65.82       1.0   3.291   69.111  2/22/19
999       Fashion accessories       88.34       7.0  30.919  649.299  2/18/19
1000      Fashion accessories       88.34       7.0  30.919  649.299  2/18/19
1001  Electronic accessories         NaN       2.0   5.803  121.863  3/10/19
1002  Electronic accessories       87.08       NaN  30.478  640.038  1/26/19

       Time  Payment    cogs  gross margin percentage  gross income  Rating
998   15:33     Cash   65.82                 4.761905         3.291     4.1
999   13:28     Cash  618.38                 4.761905        30.919     6.6
1000  13:28     Cash  618.38                 4.761905        30.919     6.6
1001  20:46  Ewallet  116.06                 4.761905         5.803     8.8
1002  15:17     Cash  609.56                 4.761905        30.478     5.5
```

[5]:
```
df.columns
```

[5]:
```
Index(['Invoice ID', 'Branch', 'City', 'Customer type', 'Gender',
       'Product line', 'Unit price', 'Quantity', 'Tax 5%', 'Total', 'Date',
       'Time', 'Payment', 'cogs', 'gross margin percentage', 'gross income',
       'Rating'],
      dtype='object')
```

[6]:
```
df.dtypes
```

[6]:
```
Invoice ID                 object
Branch                     object
City                       object
Customer type              object
Gender                     object
Product line               object
Unit price                float64
Quantity                  float64
Tax 5%                    float64
Total                     float64
Date                       object
Time                       object
Payment                    object
cogs                      float64
gross margin percentage   float64
gross income              float64
Rating                    float64
dtype: object
```

[7]:
```
df['Date']
```

[7]:
```
0        1/5/19
1        3/8/19
2        3/3/19
3       1/27/19
4        2/8/19
         …
998     2/22/19
999     2/18/19
1000    2/18/19
1001    3/10/19
1002    1/26/19
Name: Date, Length: 1003, dtype: object
```

[8]:
```
df['Date']=pd.to_datetime(df['Date'])
```

[9]:
```
df['Date']
```

[9]:
```
0      2019-01-05
1      2019-03-08
2      2019-03-03
3      2019-01-27
4      2019-02-08
          …
998    2019-02-22
999    2019-02-18
1000   2019-02-18
1001   2019-03-10
1002   2019-01-26
Name: Date, Length: 1003, dtype: datetime64[ns]
```

[10]:
```
df.set_index('Date',inplace=True)
```

[11]:
```
df.head()
```

[11]:

Date	Invoice ID	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%	Total
2019-01-05	750-67-8428	A	Yangon	Member	Female	Health and beauty	74.69	7.0	26.1415	548.9715
2019-03-08	226-31-3081	C	Naypyitaw	Normal	Female	Electronic accessories	15.28	5.0	3.8200	80.2200
2019-03-03	631-41-3108	A	Yangon	Normal	Male	Home and lifestyle	46.33	7.0	16.2155	340.5255
2019-01-27	123-19-1176	A	Yangon	Member	Male					
2019-02-08	373-73-7910	A	Yangon	Normal	Male					

```
2019-01-27       Health and beauty       58.22       8.0  23.2880  489.0480
2019-02-08       Sports and travel       86.31       7.0  30.2085  634.3785

             Time      Payment    cogs  gross margin percentage  gross income  \
Date
2019-01-05  13:08      Ewallet  522.83                 4.761905       26.1415
2019-03-08  10:29         Cash   76.40                 4.761905        3.8200
2019-03-03  13:23  Credit card  324.31                 4.761905       16.2155
2019-01-27  20:33      Ewallet  465.76                 4.761905       23.2880
2019-02-08  10:37      Ewallet  604.17                 4.761905       30.2085

            Rating
Date
2019-01-05     9.1
2019-03-08     9.6
2019-03-03     7.4
2019-01-27     8.4
2019-02-08     5.3
```

[12]:
```
df.describe() # do for every numeric column
```

[12]:
```
       Unit price    Quantity       Tax 5%        Total         cogs  \
count  996.000000  983.000000  1003.000000  1003.000000  1003.000000
mean    55.764568    5.501526    15.400368   323.407726   308.007358
std     26.510165    2.924673    11.715192   246.019028   234.303836
min     10.080000    1.000000     0.508500    10.678500    10.170000
25%     33.125000    3.000000     5.894750   123.789750   117.895000
50%     55.420000    5.000000    12.096000   254.016000   241.920000
75%     78.085000    8.000000    22.539500   473.329500   450.790000
max     99.960000   10.000000    49.650000  1042.650000   993.000000

       gross margin percentage  gross income       Rating
count             1.003000e+03   1003.000000  1003.000000
mean              4.761905e+00     15.400368     6.972682
std               8.886215e-16     11.715192     1.717647
min               4.761905e+00      0.508500     4.000000
25%               4.761905e+00      5.894750     5.500000
50%               4.761905e+00     12.096000     7.000000
75%               4.761905e+00     22.539500     8.500000
max               4.761905e+00     49.650000    10.000000
```

[13]:
```
#EDA#Univariate Analysis = looking at  one variable at a time
```

Question 1:What does the distibution of customer ratings looks like?is it skewed?

[14]:
```
pip install --upgrade pandas seaborn
```

```
Requirement already satisfied: pandas in /opt/conda/lib/python3.10/site-packages
```

```
(2.2.2)
Requirement already satisfied: seaborn in /opt/conda/lib/python3.10/site-
packages (0.13.2)
Requirement already satisfied: numpy>=1.22.4 in /opt/conda/lib/python3.10/site-
packages (from pandas) (1.23.5)
Requirement already satisfied: python-dateutil>=2.8.2 in
/opt/conda/lib/python3.10/site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /opt/conda/lib/python3.10/site-
packages (from pandas) (2022.6)
Requirement already satisfied: tzdata>=2022.7 in /opt/conda/lib/python3.10/site-
packages (from pandas) (2024.1)
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in
/opt/conda/lib/python3.10/site-packages (from seaborn) (3.6.2)
Requirement already satisfied: contourpy>=1.0.1 in
/opt/conda/lib/python3.10/site-packages (from matplotlib!=3.6.1,>=3.4->seaborn)
(1.0.6)
Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.10/site-
packages (from matplotlib!=3.6.1,>=3.4->seaborn) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in
/opt/conda/lib/python3.10/site-packages (from matplotlib!=3.6.1,>=3.4->seaborn)
(4.38.0)
Requirement already satisfied: kiwisolver>=1.0.1 in
/opt/conda/lib/python3.10/site-packages (from matplotlib!=3.6.1,>=3.4->seaborn)
(1.4.4)
Requirement already satisfied: packaging>=20.0 in
/opt/conda/lib/python3.10/site-packages (from matplotlib!=3.6.1,>=3.4->seaborn)
(21.3)
Requirement already satisfied: pillow>=6.2.0 in /opt/conda/lib/python3.10/site-
packages (from matplotlib!=3.6.1,>=3.4->seaborn) (9.2.0)
Requirement already satisfied: pyparsing>=2.2.1 in
/opt/conda/lib/python3.10/site-packages (from matplotlib!=3.6.1,>=3.4->seaborn)
(3.0.9)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.10/site-
packages (from python-dateutil>=2.8.2->pandas) (1.16.0)

[notice] A new release of pip is
available: 24.0 -> 24.2
[notice] To update, run:
pip install --upgrade pip
Note: you may need to restart the kernel to use updated packages.
```

[15]:
```
import pandas as pd
import seaborn as sns
```

[16]:
```
# Clean the 'Rating' column: replace infinite values and drop NaNs
df['Rating'].replace([np.inf, -np.inf], np.nan, inplace=True)
df.dropna(subset=['Rating'], inplace=True)
```

[17]:
```
df
```

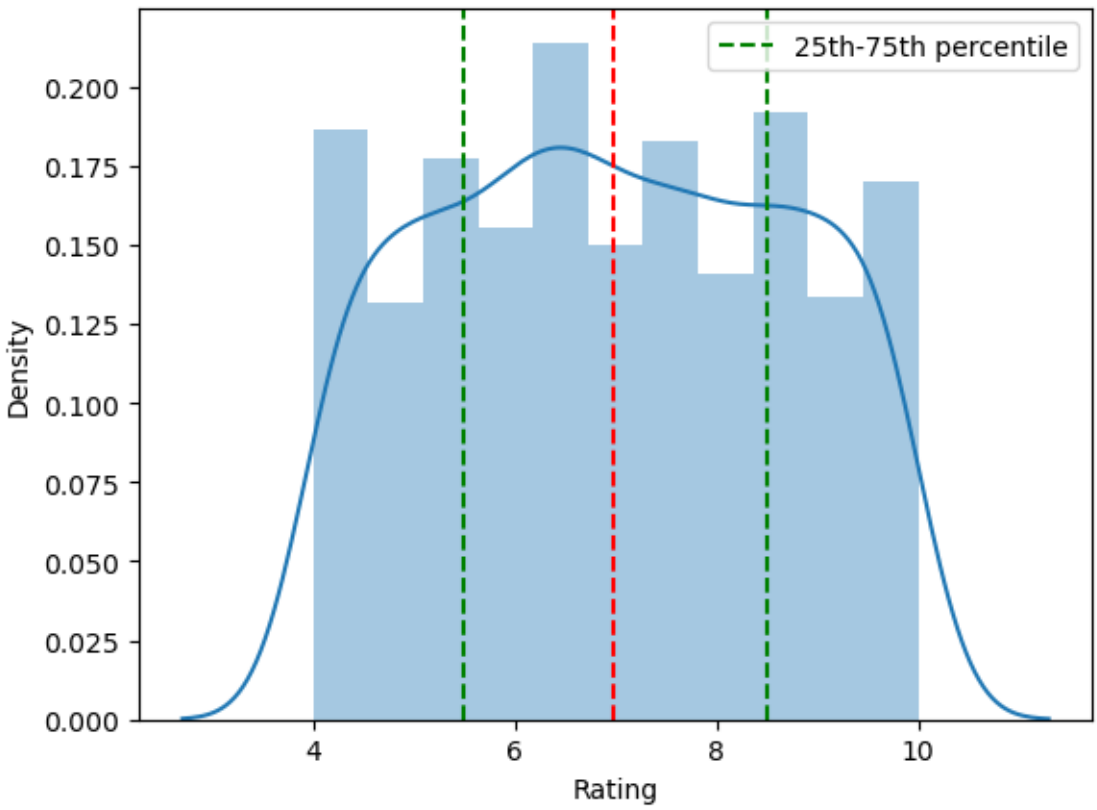
[17]:

Date	Invoice ID	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%	Total	Time	Payment	cogs	gross margin percentage	gross income	Rating
2019-01-05	750-67-8428	A	Yangon	Member	Female	Health and beauty	74.69	7.0	26.1415	548.9715	13:08	Ewallet	522.83	4.761905	26.1415	9.1
2019-03-08	226-31-3081	C	Naypyitaw	Normal	Female	Electronic accessories	15.28	5.0	3.8200	80.2200	10:29	Cash	76.40	4.761905	3.8200	
2019-03-03	631-41-3108	A	Yangon	Normal	Male	Home and lifestyle	46.33	7.0	16.2155	340.5255	13:23	Credit card	324.31	4.761905	16.2155	
2019-01-27	123-19-1176	A	Yangon	Member	Male	Health and beauty	58.22	8.0	23.2880	489.0480	20:33	Ewallet	465.76	4.761905	23.2880	
2019-02-08	373-73-7910	A	Yangon	Normal	Male	Sports and travel	86.31	7.0	30.2085	634.3785	10:37	Ewallet	604.17	4.761905	30.2085	
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	
2019-02-22	347-56-2442	A	Yangon	Normal	Male	Home and lifestyle	65.82	1.0	3.2910	69.1110	15:33	Cash	65.82	4.761905	3.2910	
2019-02-18	849-09-3807	A	Yangon	Member	Female	Fashion accessories	88.34	7.0	30.9190	649.2990	13:28	Cash	618.38	4.761905	30.9190	
2019-02-18	849-09-3807	A	Yangon	Member	Female	Fashion accessories	88.34	7.0	30.9190	649.2990	13:28	Cash	618.38	4.761905	30.9190	
2019-03-10	745-74-0715	A	Yangon	Normal	Male	Electronic accessories	NaN	2.0	5.8030	121.8630	20:46	Ewallet	116.06	4.761905	5.8030	
2019-01-26	452-04-8808	B	Mandalay	Normal	Male	Electronic accessories	87.08	NaN	30.4780	640.0380	15:17	Cash	609.56	4.761905	30.4780	

```
2019-03-08     9.6
2019-03-03     7.4
2019-01-27     8.4
2019-02-08     5.3
…              …
2019-02-22     4.1
2019-02-18     6.6
2019-02-18     6.6
2019-03-10     8.8
2019-01-26     5.5

[1003 rows x 16 columns]
```

[18]:
```
# Plotting the histogram with KDE
sns.distplot(df['Rating'], kde=True)
plt.axvline(x=np.mean(df['Rating']),c='red',ls='--')
plt.axvline(x=np.percentile(df['Rating'],25),c='green',ls='--',label='25th-75th␣
 ↪percentile')
plt.axvline(x=np.percentile(df['Rating'],75),c='green',ls='--')
plt.legend()
```

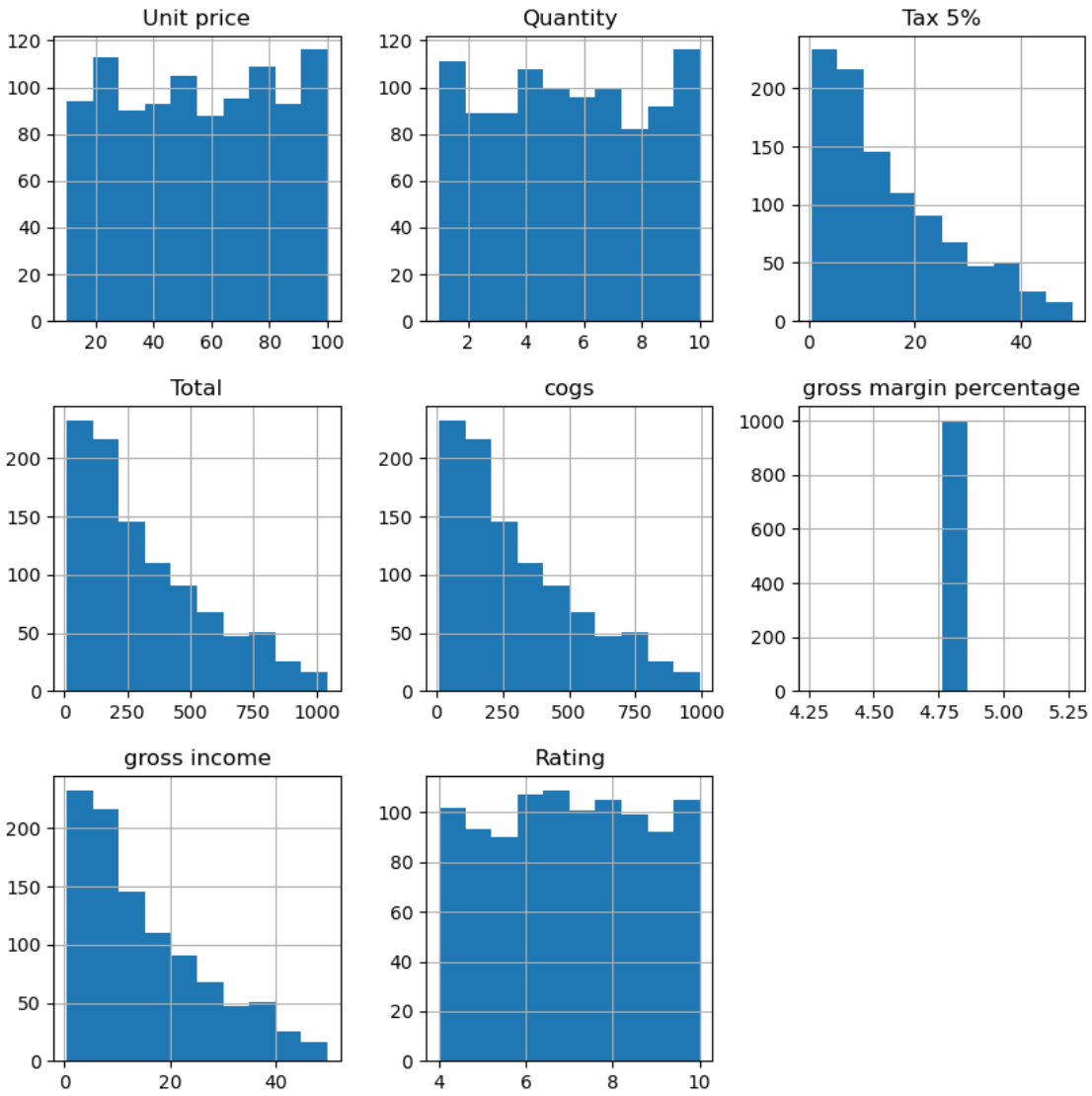
[18]:
```
<matplotlib.legend.Legend at 0x7aa5588bb670>
```



[19]:
```
# the distribution of user Rating looks relatively uniform and there doesn't
↪seem to be skewed in left or right direction
```

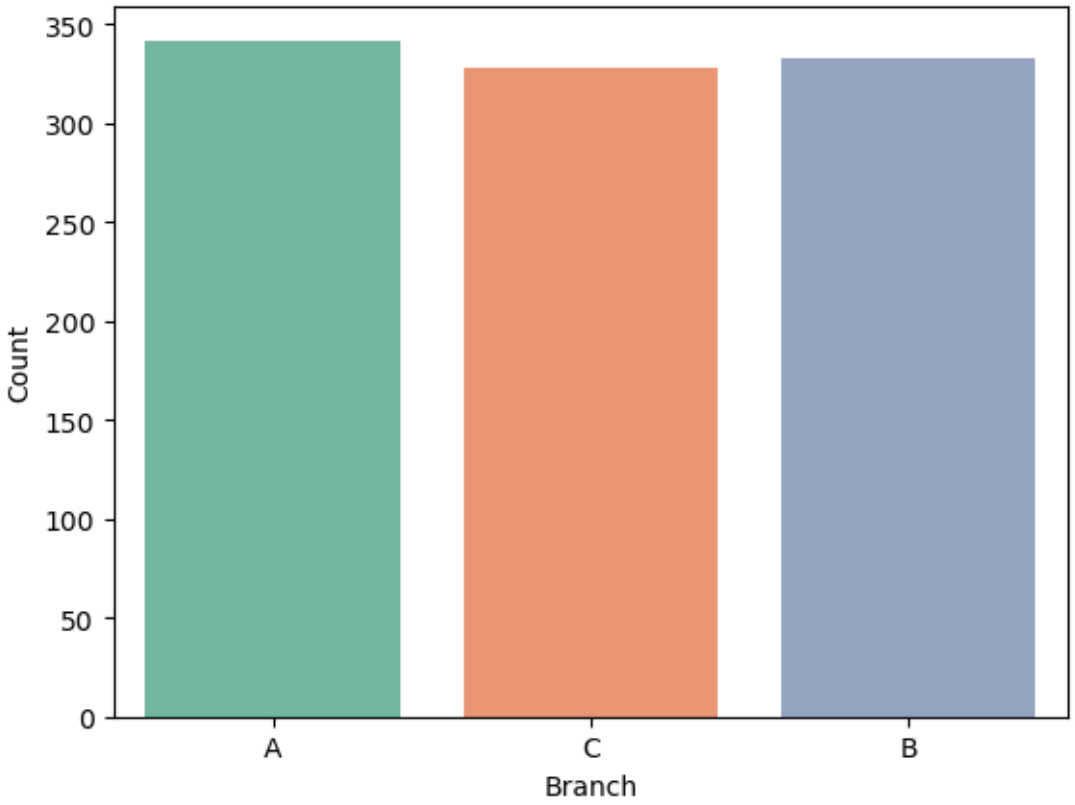
[20]:
```
df.hist(figsize=(10,10))
```

[20]:
```
array([[<AxesSubplot: title={'center': 'Unit price'}>,
        <AxesSubplot: title={'center': 'Quantity'}>,
        <AxesSubplot: title={'center': 'Tax 5%'}>],
       [<AxesSubplot: title={'center': 'Total'}>,
        <AxesSubplot: title={'center': 'cogs'}>,
        <AxesSubplot: title={'center': 'gross margin percentage'}>],
       [<AxesSubplot: title={'center': 'gross income'}>,
        <AxesSubplot: title={'center': 'Rating'}>, <AxesSubplot: >]],
      dtype=object)
```



Question 2: Do aggregate sales numbers differ by much between branches?

[21]:
```
# Create the count plot
sns.countplot(x='Branch', data=df, palette='Set2')

# Set the axis labels
plt.xlabel('Branch')
plt.ylabel('Count')

# Show the plot
plt.show()
```

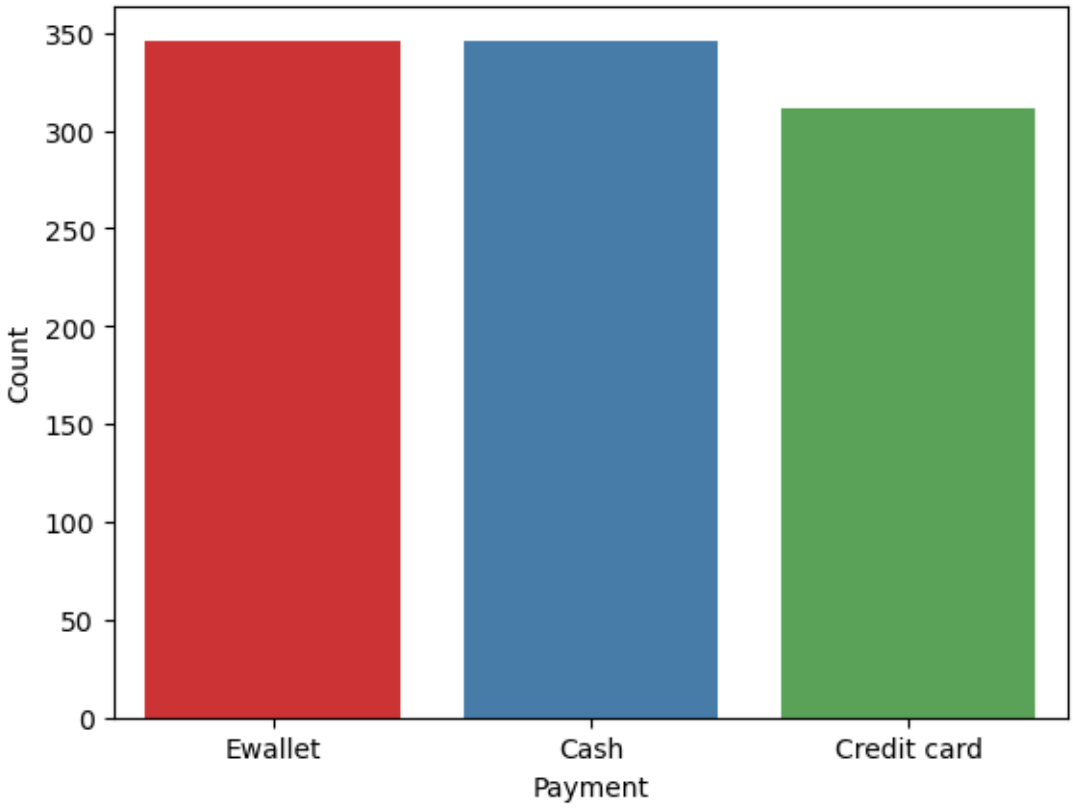


[22]:
```
df['Branch'].value_counts()
```

[22]:
```
Branch
A    342
B    333
C    328
Name: count, dtype: int64
```

[23]:
```
# To see user different Payment method
```

[24]:
```
# Create the count plot
sns.countplot(x='Payment', data=df, palette='Set1')
# Set the axis labels
plt.xlabel('Payment')
plt.ylabel('Count')

# Show the plot
plt.show()
```



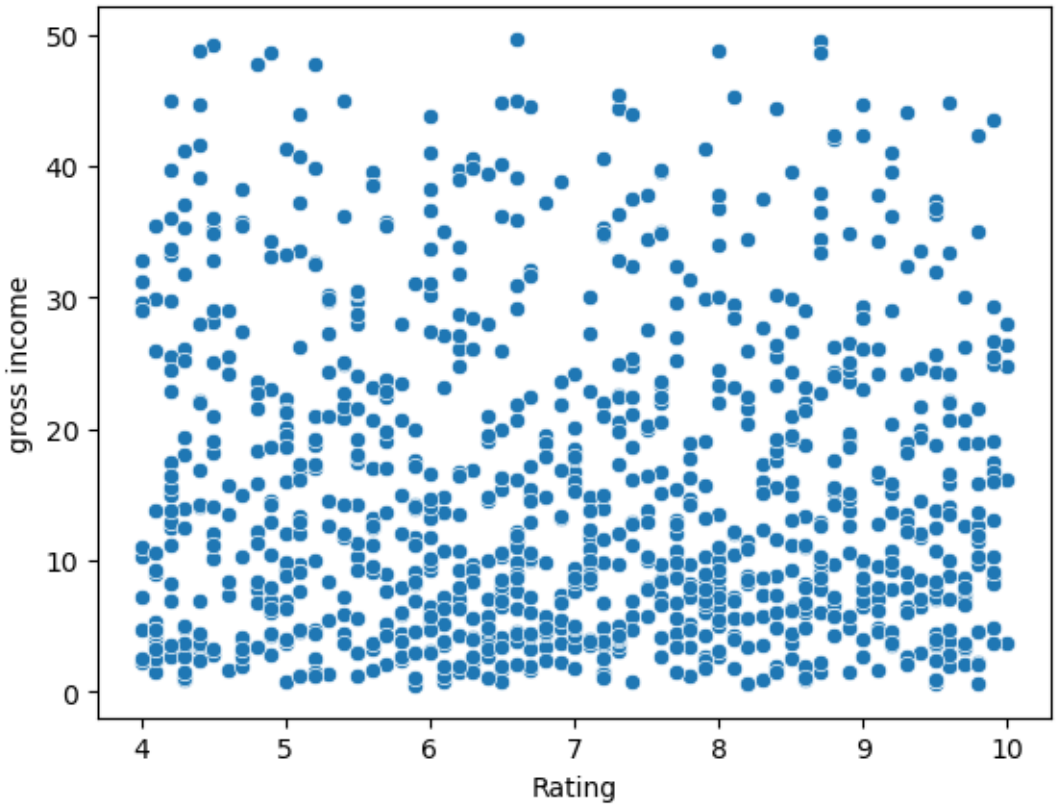
Task 3: Bivariate Analysis

[25]:
```
# looking at more than one variable at a time
```

Question 3: is there a relationship between gross income and customer Ratings?

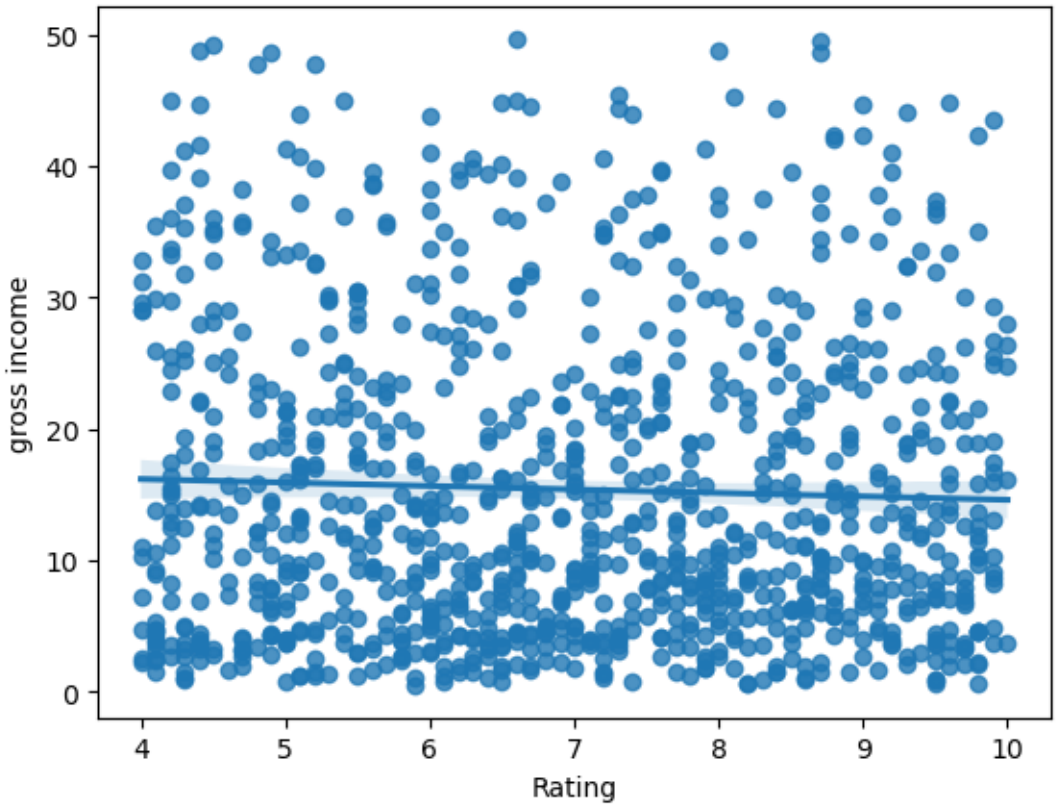
[26]:
```
sns.scatterplot(x=df['Rating'], y=df['gross income'])
```

[26]:
```
<AxesSubplot: xlabel='Rating', ylabel='gross income'>
```



[27]:
```
sns.regplot(x=df['Rating'], y=df['gross income'])
```

[27]: <AxesSubplot: xlabel='Rating', ylabel='gross income'>

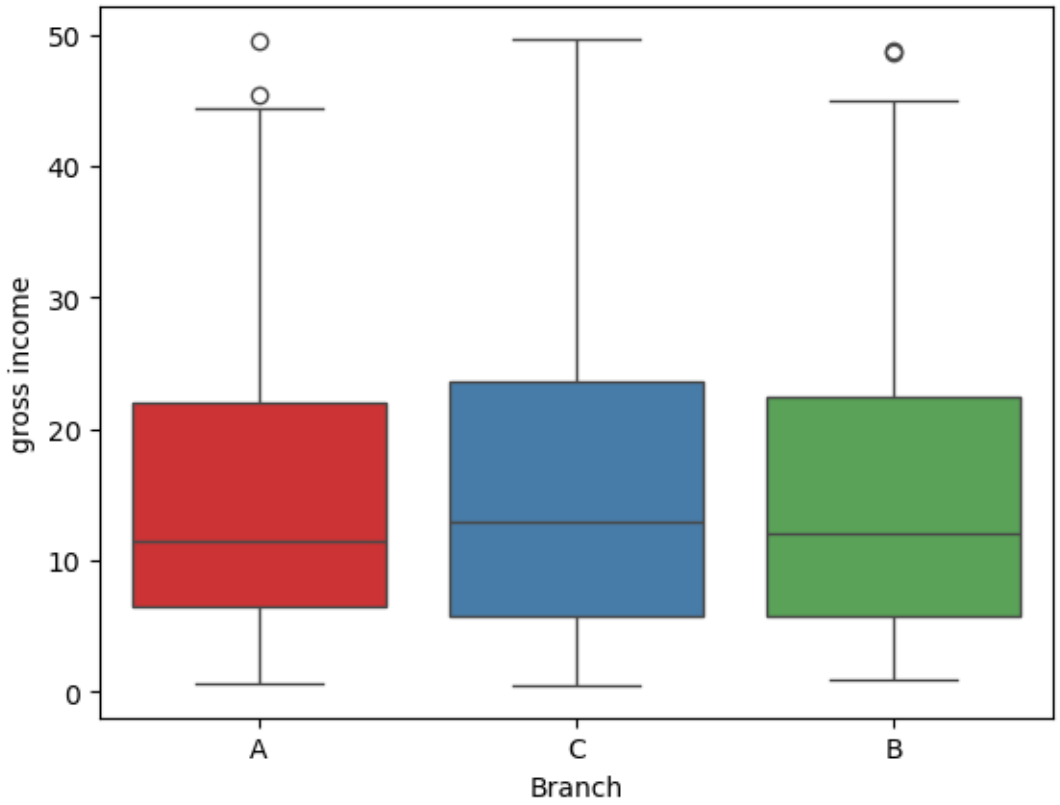


[28]:
```
# By above plot we can assume that there is no particular relation between␣
↪'Rating' and 'gross income'
```

[29]:
```
# boxplot between 'Branch' and 'gross income'
```

[30]:
```
sns.boxplot(x=df['Branch'],y=df['gross income'],palette='Set1')
```

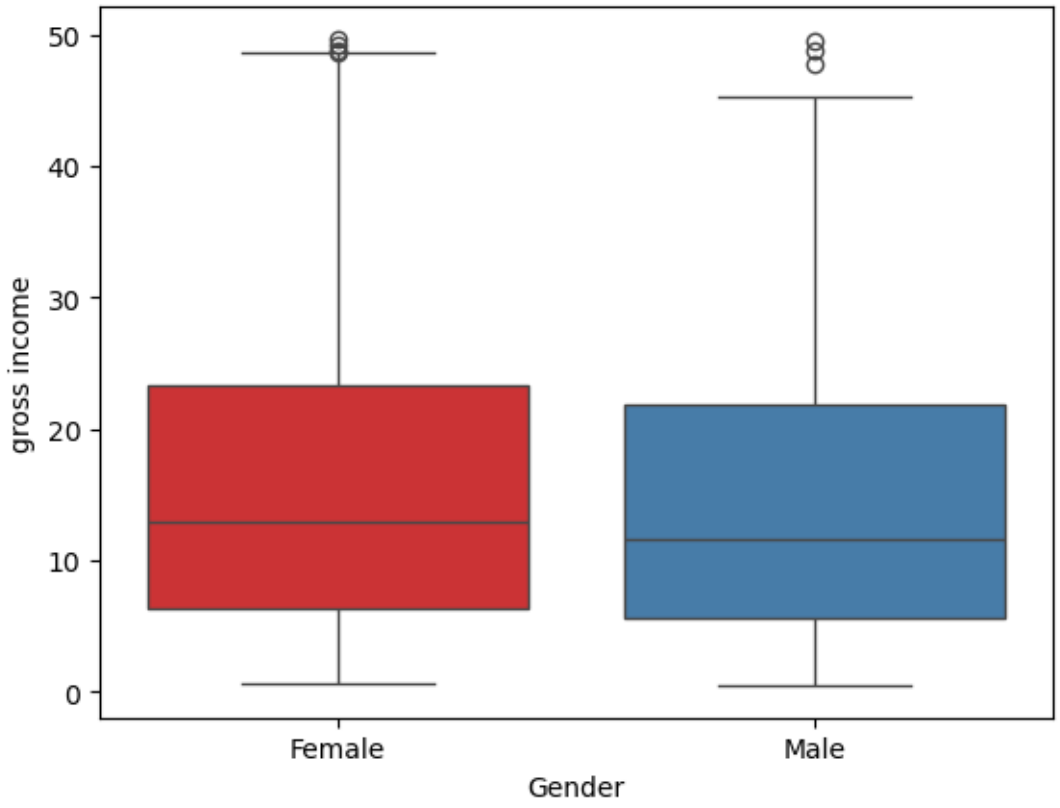
[30]:
```
<AxesSubplot: xlabel='Branch', ylabel='gross income'>
```



[31]:
```
#Relationship between 'Gender' and 'gross income'
```

[32]:
```
sns.boxplot(x=df['Gender'],y=df['gross income'],palette='Set1')
```

[32]:
```
<AxesSubplot: xlabel='Gender', ylabel='gross income'>
```



Question 4: is there a noticeable trend in gross income?

[33]:
```
df.head()
```

[33]:

Date	Invoice ID	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%	Total	Time	Payment	cogs	gross margin percentage	gross income	\
2019-01-05	750-67-8428	A	Yangon	Member	Female	Health and beauty	74.69	7.0	26.1415	548.9715						
2019-03-08	226-31-3081	C	Naypyitaw	Normal	Female	Electronic accessories	15.28	5.0	3.8200	80.2200						
2019-03-03	631-41-3108	A	Yangon	Normal	Male	Home and lifestyle	46.33	7.0	16.2155	340.5255						
2019-01-27	123-19-1176	A	Yangon	Member	Male	Health and beauty	58.22	8.0	23.2880	489.0480						
2019-02-08	373-73-7910	A	Yangon	Normal	Male	Sports and travel	86.31	7.0	30.2085	634.3785						

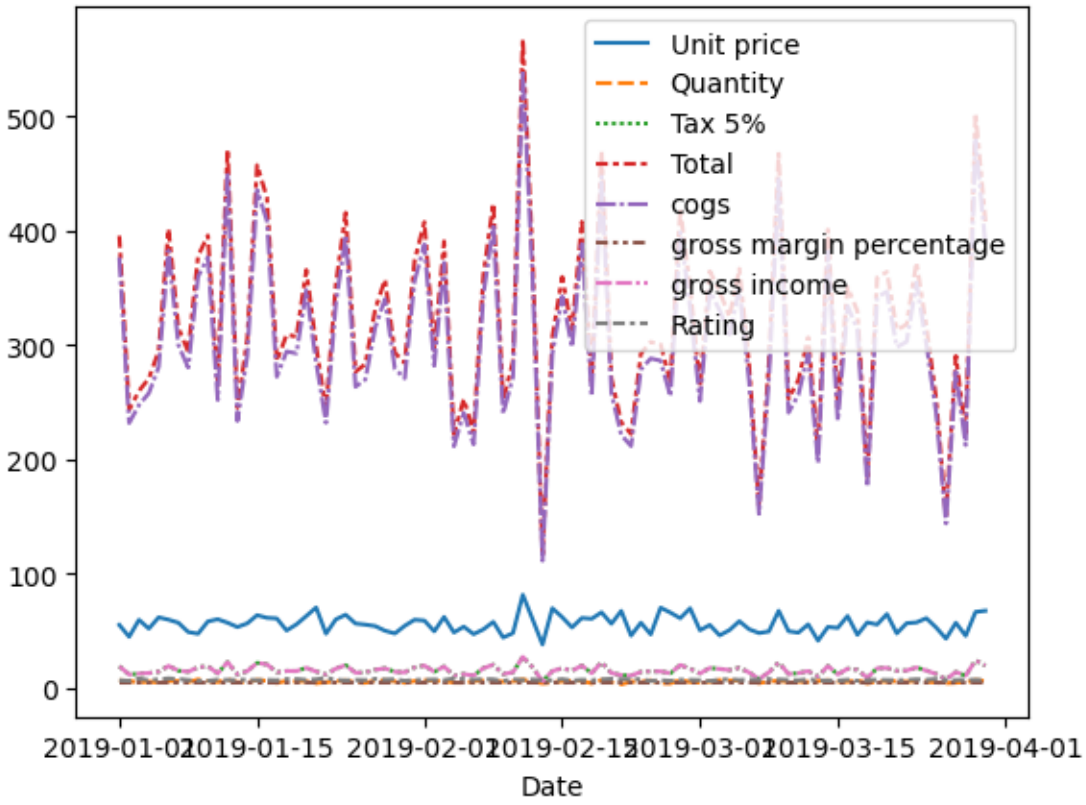
```
Date
2019-01-05  13:08      Ewallet  522.83                4.761905       26.1415
2019-03-08  10:29         Cash   76.40                4.761905        3.8200
2019-03-03  13:23  Credit card  324.31                4.761905       16.2155
2019-01-27  20:33      Ewallet  465.76                4.761905       23.2880
2019-02-08  10:37      Ewallet  604.17                4.761905       30.2085

            Rating
Date
2019-01-05     9.1
2019-03-08     9.6
2019-03-03     7.4
2019-01-27     8.4
2019-02-08     5.3
```

[34]:
```
# Select only numeric columns before applying the mean
df.groupby(df.index).mean(numeric_only=True)
```

[34]:
```
            Unit price  Quantity     Tax 5%       Total        cogs  \
Date
2019-01-01   54.995833  6.454545  18.830083  395.431750  376.601667
2019-01-02   44.635000  6.000000  11.580375  243.187875  231.607500
2019-01-03   59.457500  4.625000  12.369813  259.766062  247.396250
2019-01-04   51.743333  5.333333  12.886417  270.614750  257.728333
2019-01-05   61.636667  4.583333  14.034458  294.723625  280.689167
…                  …         …         …           …           …
2019-03-26   42.972308  4.000000   7.188692  150.962538  143.773846
2019-03-27   56.841000  4.500000  13.822950  290.281950  276.459000
2019-03-28   45.525000  4.800000  10.616200  222.940200  212.324000
2019-03-29   66.346250  6.750000  23.947875  502.905375  478.957500
2019-03-30   67.408182  5.888889  19.424500  407.914500  388.490000

            gross margin percentage  gross income    Rating
Date
2019-01-01                 4.761905     18.830083  6.583333
2019-01-02                 4.761905     11.580375  6.050000
2019-01-03                 4.761905     12.369813  8.112500
2019-01-04                 4.761905     12.886417  6.516667
2019-01-05                 4.761905     14.034458  7.433333
…                               …             …         …
2019-03-26                 4.761905      7.188692  6.623077
2019-03-27                 4.761905     13.822950  6.760000
2019-03-28                 4.761905     10.616200  7.050000
2019-03-29                 4.761905     23.947875  6.925000
2019-03-30                 4.761905     19.424500  6.800000

[89 rows x 8 columns]
```

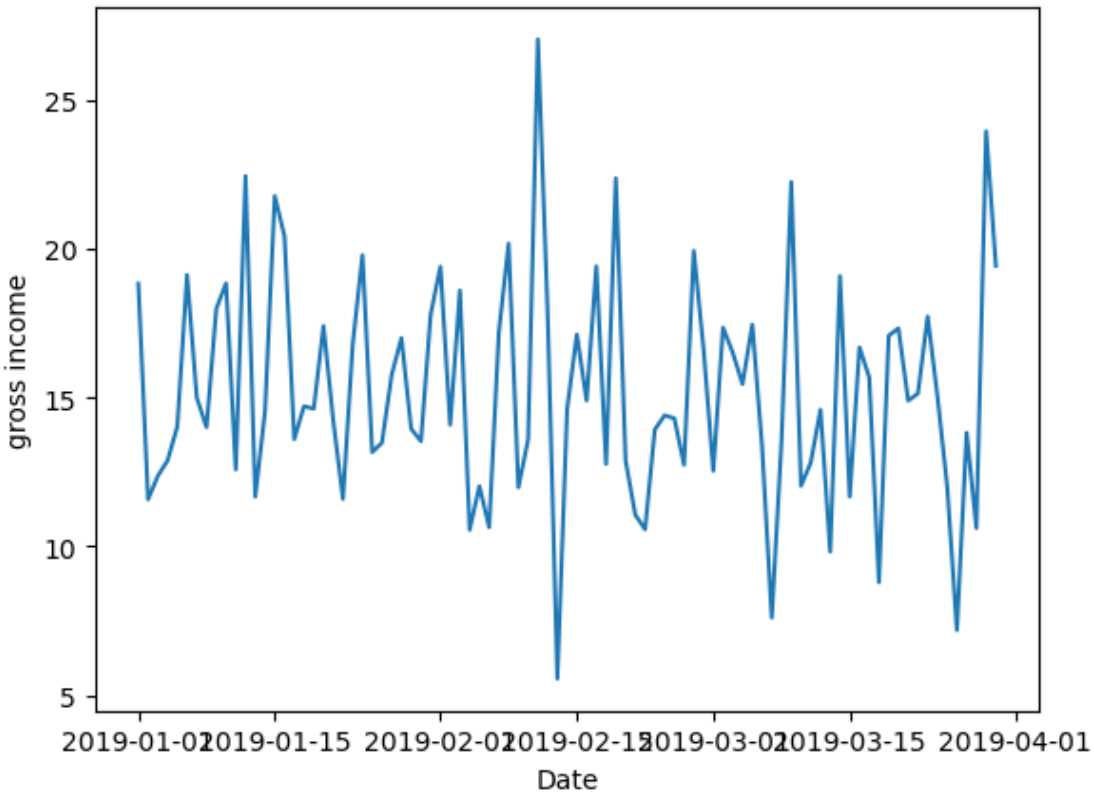
[35]:
```
sns.lineplot(df.groupby(df.index).mean(numeric_only=True))
```

[35]: <AxesSubplot: xlabel='Date'>



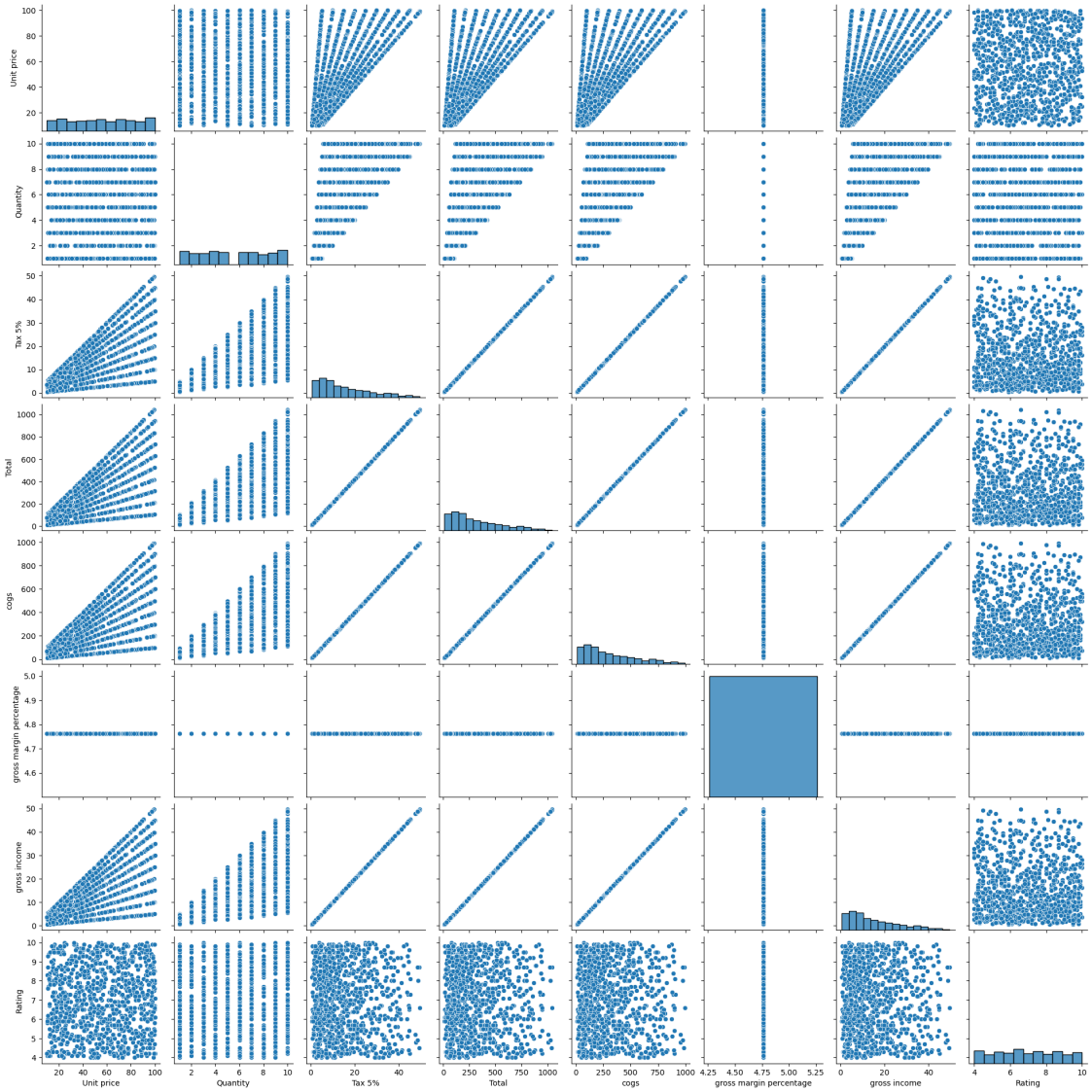
[36]:
```
# Group by index and calculate the mean of numeric columns
grouped_df = df.groupby(df.index).mean(numeric_only=True)

# Plot the line plot
sns.lineplot(x=grouped_df.index, y=grouped_df['gross income'])
```

[36]: <AxesSubplot: xlabel='Date', ylabel='gross income'>



[37]:
```
# Remove duplicate column labels (if any)
df = df.loc[:, ~df.columns.duplicated()]

# Select only numeric columns
numeric_df = df.select_dtypes(include='number')

# Drop rows with missing values (optional, if necessary)
cleaned_df = numeric_df.dropna()
# Create pairplot
sns.pairplot(cleaned_df)
```

[37]: <seaborn.axisgrid.PairGrid at 0x7aa419e63bb0>



Task 4: Dealing with Duplicate Rows and Missing Values

[38]:
```
df.duplicated()
```

[38]:
```
Date
2019-01-05    False
2019-03-08    False
2019-03-03    False
2019-01-27    False
2019-02-08    False
              ...
2019-02-22    False
2019-02-18    False
```

```
2019-02-18     True
2019-03-10     True
2019-01-26     True
Length: 1003, dtype: bool
```

[39]:
```
df.duplicated().sum()
```

[39]:
```
3
```

[40]:
```
df[df.duplicated()==True]
```

[40]:
```
             Invoice ID Branch      City Customer type  Gender  \
Date
2019-02-18  849-09-3807      A    Yangon        Member  Female
2019-03-10  745-74-0715      A    Yangon        Normal    Male
2019-01-26  452-04-8808      B  Mandalay        Normal    Male

                     Product line  Unit price  Quantity  Tax 5%    Total  \
Date
2019-02-18     Fashion accessories       88.34       7.0  30.919  649.299
2019-03-10  Electronic accessories         NaN       2.0   5.803  121.863
2019-01-26  Electronic accessories       87.08       NaN  30.478  640.038

             Time  Payment    cogs  gross margin percentage  gross income  \
Date
2019-02-18  13:28     Cash  618.38                 4.761905        30.919
2019-03-10  20:46  Ewallet  116.06                 4.761905         5.803
2019-01-26  15:17     Cash  609.56                 4.761905        30.478

            Rating
Date
2019-02-18     6.6
2019-03-10     8.8
2019-01-26     5.5
```

[41]:
```
df.drop_duplicates(inplace=True) #inplace is True because we are doing␣
↪permanenet change in Dataset
```

[42]:
```
df.duplicated().sum()
```

[42]:
```
0
```

[43]:
```
#Total no of missing Value per column
```

[44]:
```
df.isna().sum()
```

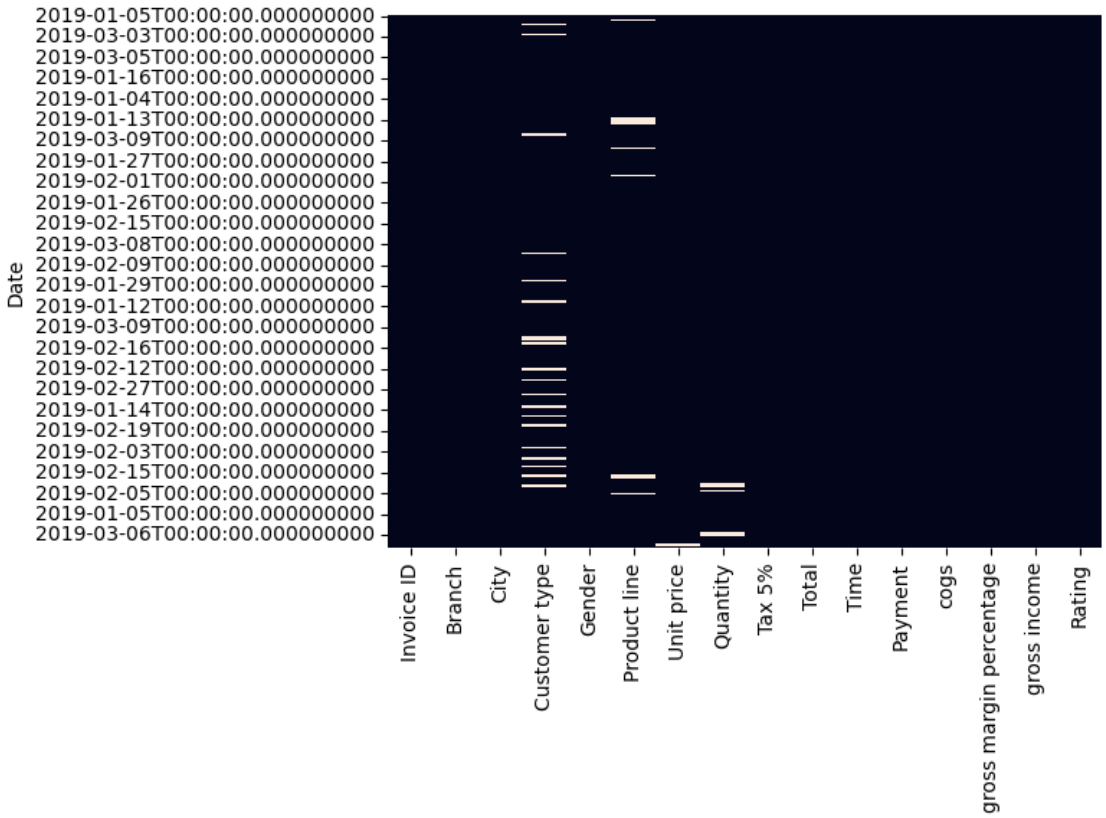
[44]:
```
Invoice ID                 0
Branch                     0
City                       0
Customer type             79
Gender                     0
Product line              43
Unit price                 6
Quantity                  19
Tax 5%                     0
Total                      0
Time                       0
Payment                    0
cogs                       0
gross margin percentage    0
gross income               0
Rating                     0
dtype: int64
```

[45]:
```
df.isna().sum()/len(df)
```

[45]:
```
Invoice ID                 0.000
Branch                     0.000
City                       0.000
Customer type              0.079
Gender                     0.000
Product line               0.043
Unit price                 0.006
Quantity                   0.019
Tax 5%                     0.000
Total                      0.000
Time                       0.000
Payment                    0.000
cogs                       0.000
gross margin percentage    0.000
gross income               0.000
Rating                     0.000
dtype: float64
```

[46]:
```
sns.heatmap(df.isnull(),cbar=False)
```

[46]:
```
<AxesSubplot: ylabel='Date'>
```



[47]:
```
df.fillna(df.mean(numeric_only=True), inplace=True)
# fill for numeric data only
```

[48]:
```
df
```

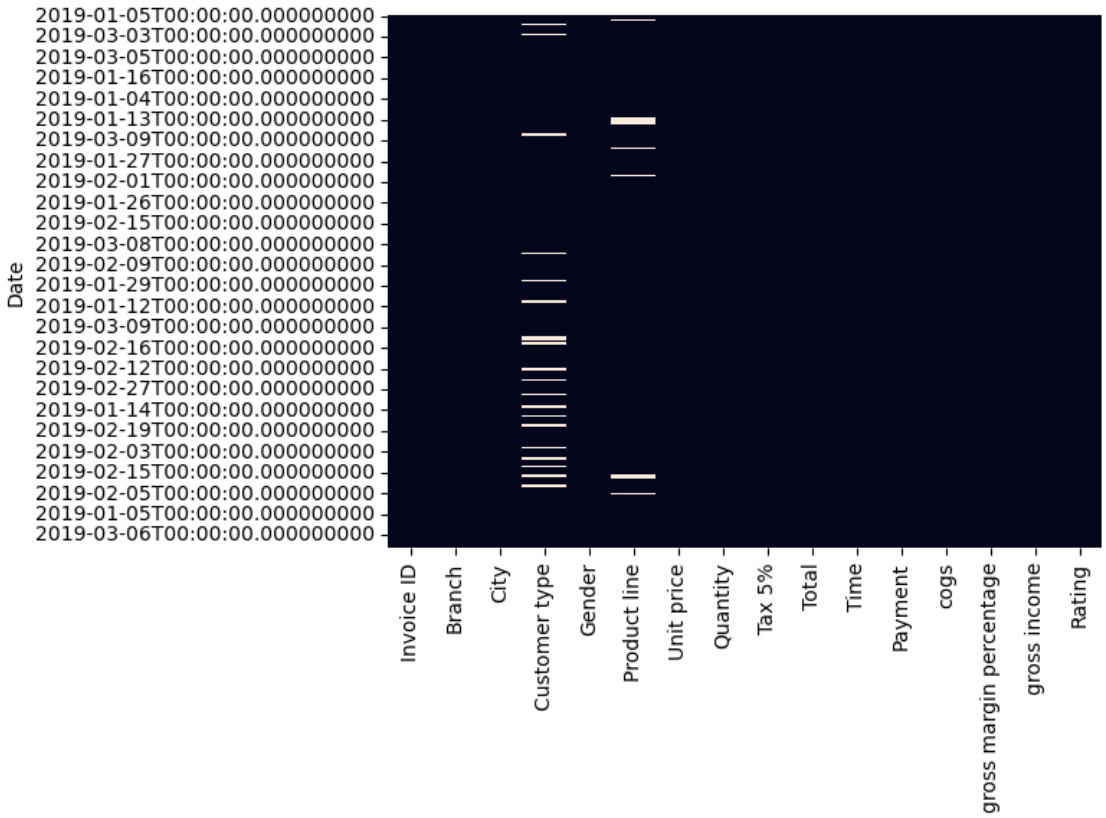
[48]:

Date	Invoice ID	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%	Total	\
2019-01-05	750-67-8428	A	Yangon	Member	Female						
2019-03-08	226-31-3081	C	Naypyitaw	Normal	Female						
2019-03-03	631-41-3108	A	Yangon	Normal	Male						
2019-01-27	123-19-1176	A	Yangon	Member	Male						
2019-02-08	373-73-7910	A	Yangon	Normal	Male						
…	…	…	…	…	…						
2019-01-29	233-67-5758	C	Naypyitaw	Normal	Male						
2019-03-02	303-96-2227	B	Mandalay	Normal	Female						
2019-02-09	727-02-1313	A	Yangon	Member	Male						
2019-02-22	347-56-2442	A	Yangon	Normal	Male						
2019-02-18	849-09-3807	A	Yangon	Member	Female						

```
2019-01-05       Health and beauty   74.690000       7.0  26.1415   548.9715
2019-03-08  Electronic accessories   15.280000       5.0   3.8200    80.2200
2019-03-03      Home and lifestyle   46.330000       7.0  16.2155   340.5255
2019-01-27       Health and beauty   58.220000       8.0  23.2880   489.0480
2019-02-08       Sports and travel   86.310000       7.0  30.2085   634.3785
…                              …         …       …      …       …
2019-01-29       Health and beauty   55.700292       1.0   2.0175    42.3675
2019-03-02      Home and lifestyle   55.700292      10.0  48.6900  1022.4900
2019-02-09      Food and beverages   55.700292       1.0   1.5920    33.4320
2019-02-22      Home and lifestyle   65.820000       1.0   3.2910    69.1110
2019-02-18     Fashion accessories   88.340000       7.0  30.9190   649.2990

             Time      Payment    cogs  gross margin percentage  gross income  \
Date
2019-01-05  13:08      Ewallet  522.83                 4.761905       26.1415
2019-03-08  10:29         Cash   76.40                 4.761905        3.8200
2019-03-03  13:23  Credit card  324.31                 4.761905       16.2155
2019-01-27  20:33      Ewallet  465.76                 4.761905       23.2880
2019-02-08  10:37      Ewallet  604.17                 4.761905       30.2085
…           …          …     …                      …           …
2019-01-29  13:46      Ewallet   40.35                 4.761905        2.0175
2019-03-02  17:16      Ewallet  973.80                 4.761905       48.6900
2019-02-09  13:22         Cash   31.84                 4.761905        1.5920
2019-02-22  15:33         Cash   65.82                 4.761905        3.2910
2019-02-18  13:28         Cash  618.38                 4.761905       30.9190

            Rating
Date
2019-01-05     9.1
2019-03-08     9.6
2019-03-03     7.4
2019-01-27     8.4
2019-02-08     5.3
…              …
2019-01-29     6.2
2019-03-02     4.4
2019-02-09     7.7
2019-02-22     4.1
2019-02-18     6.6

[1000 rows x 16 columns]
```

[49]:
```
sns.heatmap(df.isnull(),cbar=False)
```

[49]:
```
<AxesSubplot: ylabel='Date'>
```



[50]:
```
#in above heatmap only column which have only string remain bcz in string there is no mean
```

#for string missing value

[51]:
```
df.mode().iloc[0]
```

[51]:
```
Invoice ID                    101-17-6199
Branch                                  A
City                               Yangon
Customer type                      Normal
Gender                             Female
Product line          Fashion accessories
Unit price                      55.700292
Quantity                             10.0
Tax 5%                              4.154
Total                              87.234
Time                                14:42
Payment                          Ewallet
cogs                               83.08
gross margin percentage         4.761905
```

```
gross income                            4.154
Rating                                    6.0
Name: 0, dtype: object
```

#The code `df.mode().iloc[0]` performs the following steps:

1. `df.mode()`:
   - The `.mode()` function in Pandas is used to calculate the mode (the most frequent value) for each column in the DataFrame `df`.
   - It returns a DataFrame where each column contains one or more rows representing the mode for that column. If there are multiple modes (i.e., several values appear with the same highest frequency), the result will have multiple rows.
2. `.iloc[0]`:
   - `.iloc[]` is used for positional indexing. Here, `.iloc[0]` is selecting the first row of the DataFrame returned by `df.mode()`.
   - Since `.mode()` can return multiple rows when there are multiple modes, `.iloc[0]` ensures that you only get the first mode for each column.

0.0.1 Example:

Suppose you have the following DataFrame `df`:

```
   A  B  C
0  1  2  3
1  2  2  4
2  2  3  3
3  3  3  3
```

If you run `df.mode()`, you might get:

```
     A    B    C
0  2.0  2.0  3.0
1  NaN  3.0  NaN
```

Here: - For column `A`, the mode is `2`. - For column `B`, there are two modes: `2` and `3`. - For column `C`, the mode is `3`.

By using `.iloc[0]`, you select the first row:

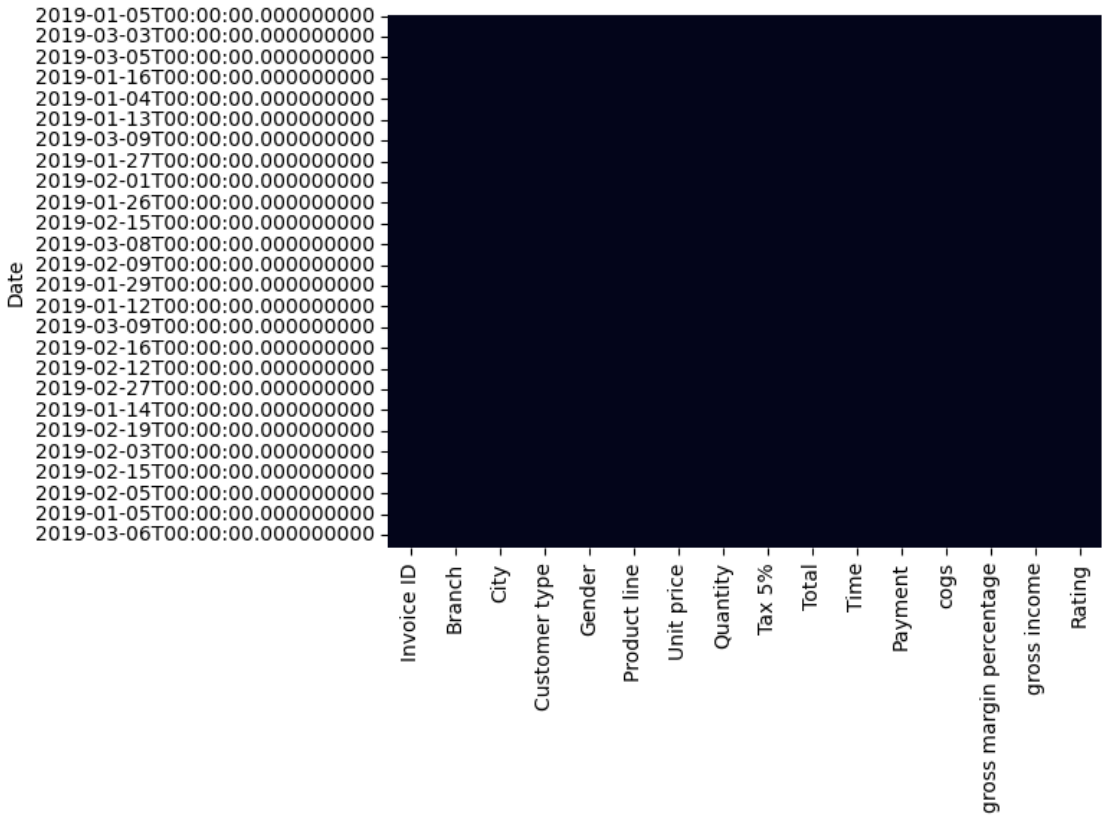
```
A    2.0
B    2.0
C    3.0
```

Thus, `df.mode().iloc[0]` returns the first mode for each column in the DataFrame.

[52]:
```
df.fillna(df.mode().iloc[0], inplace=True)
```

[53]:
```
sns.heatmap(df.isnull(),cbar=False)
```

[53]:
```
<AxesSubplot: ylabel='Date'>
```



[54]:
```
#above heatmap give no missing Value
```

[55]:
```
####### summing up all project
```

[56]:
```
dataset =pd.read_csv('supermarket_sales.csv')
prof = ProfileReport(dataset)
prof
```

```
Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/s]

Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]

Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]

<IPython.core.display.HTML object>
```

[56]:

Task 5:Correlation Analysis

[57]:
```
#when we want to find the correlation between  any two column is we can use␣
↪numpy
```

[58]:
```
np.corrcoef(df['gross income'],df['Rating'])
```

[58]:
```
array([[ 1.       , -0.0364417],
       [-0.0364417,  1.       ]])
```

[59]:
```
# to get the specific no from above result we can sunbset it
```

[60]:
```
np.corrcoef(df['gross income'],df['Rating'])[0][1]
```

[60]:
```
-0.03644170499701835
```

[61]:
```
# rounding up
```

[62]:
```
round(np.corrcoef(df['gross income'],df['Rating'])[0][1],2)
```

[62]:
```
-0.04
```

[63]:
```
#correlation matrix
```

[65]:
```
df.corr(numeric_only=True)
```

[65]:

	Unit price	Quantity	Tax 5%	Total	cogs
Unit price	1.000000	0.014786	0.629034	0.629034	0.629034
Quantity	0.014786	1.000000	0.704067	0.704067	0.704067
Tax 5%	0.629034	0.704067	1.000000	1.000000	1.000000
Total	0.629034	0.704067	1.000000	1.000000	1.000000
cogs	0.629034	0.704067	1.000000	1.000000	1.000000
gross margin percentage	NaN	NaN	NaN	NaN	NaN
gross income	0.629034	0.704067	1.000000	1.000000	1.000000
Rating	-0.006601	-0.021225	-0.036442	-0.036442	-0.036442

	gross margin percentage	gross income	Rating
Unit price	NaN	0.629034	-0.006601
Quantity	NaN	0.704067	-0.021225
Tax 5%	NaN	1.000000	-0.036442
Total	NaN	1.000000	-0.036442
cogs	NaN	1.000000	-0.036442
gross margin percentage	NaN	NaN	NaN
gross income	NaN	1.000000	-0.036442
Rating	NaN	-0.036442	1.000000

[66]:
```
np.round(df.corr(numeric_only=True),2)
```

[66]:

	Unit price	Quantity	Tax 5%	Total	cogs
Unit price	1.00	0.01	0.63	0.63	0.63
Quantity	0.01	1.00	0.70	0.70	0.70
Tax 5%	0.63	0.70	1.00	1.00	1.00

```
Total                          0.63      0.70    1.00   1.00  1.00
cogs                           0.63      0.70    1.00   1.00  1.00
gross margin percentage         NaN       NaN     NaN    NaN   NaN
gross income                   0.63      0.70    1.00   1.00  1.00
Rating                        -0.01     -0.02   -0.04  -0.04 -0.04
```

```
                        gross margin percentage  gross income  Rating
Unit price                                  NaN          0.63   -0.01
Quantity                                    NaN          0.70   -0.02
Tax 5%                                      NaN          1.00   -0.04
Total                                       NaN          1.00   -0.04
cogs                                        NaN          1.00   -0.04
gross margin percentage                     NaN           NaN     NaN
gross income                                NaN          1.00   -0.04
Rating                                      NaN         -0.04    1.00
```

To remove `NaN` values from the correlation matrix, you can use the `dropna()` function after computing the correlation matrix. Here's how you can do it:

0.0.2 Steps:

1. Compute the correlation matrix.
2. Remove rows and columns that contain `NaN` values using `dropna()`.

Here's the corrected code:

```
corr_matrix = df.corr(numeric_only=True)  # Step 1: Compute correlation matrix
clean_corr_matrix = corr_matrix.dropna(how='any')  # Step 2: Remove rows/columns with NaN valu
```

0.0.3 Explanation:

- `df.corr(numeric_only=True)` computes the correlation matrix, considering only numeric columns.
- `dropna(how='any')` removes rows (and corresponding columns) that have `NaN` values in the matrix.
  - `how='any'` means it will drop rows/columns if any `NaN` is present.
  - You can also use `how='all'` to drop rows/columns where **all** values are `NaN`.

If you want to keep rows and columns but just fill in `NaN` values, you can use `fillna()` instead:

```
clean_corr_matrix = corr_matrix.fillna(0)  # Replaces NaN values with 0
```

This way, `NaN` values in the matrix will be replaced with zeros, which means no correlation between those variables.

[69]:
```
corr_matrix = df.corr(numeric_only=True)  # Step 1: Compute correlation matrix
clean_corr_matrix = corr_matrix.dropna(how='any')  # Step 2: Remove rows/
 ↪columns with NaN values
clean_corr_matrix = corr_matrix.fillna(0)  # Replaces NaN values with 0
```

[71]:
```
np.round(clean_corr_matrix,2)
```

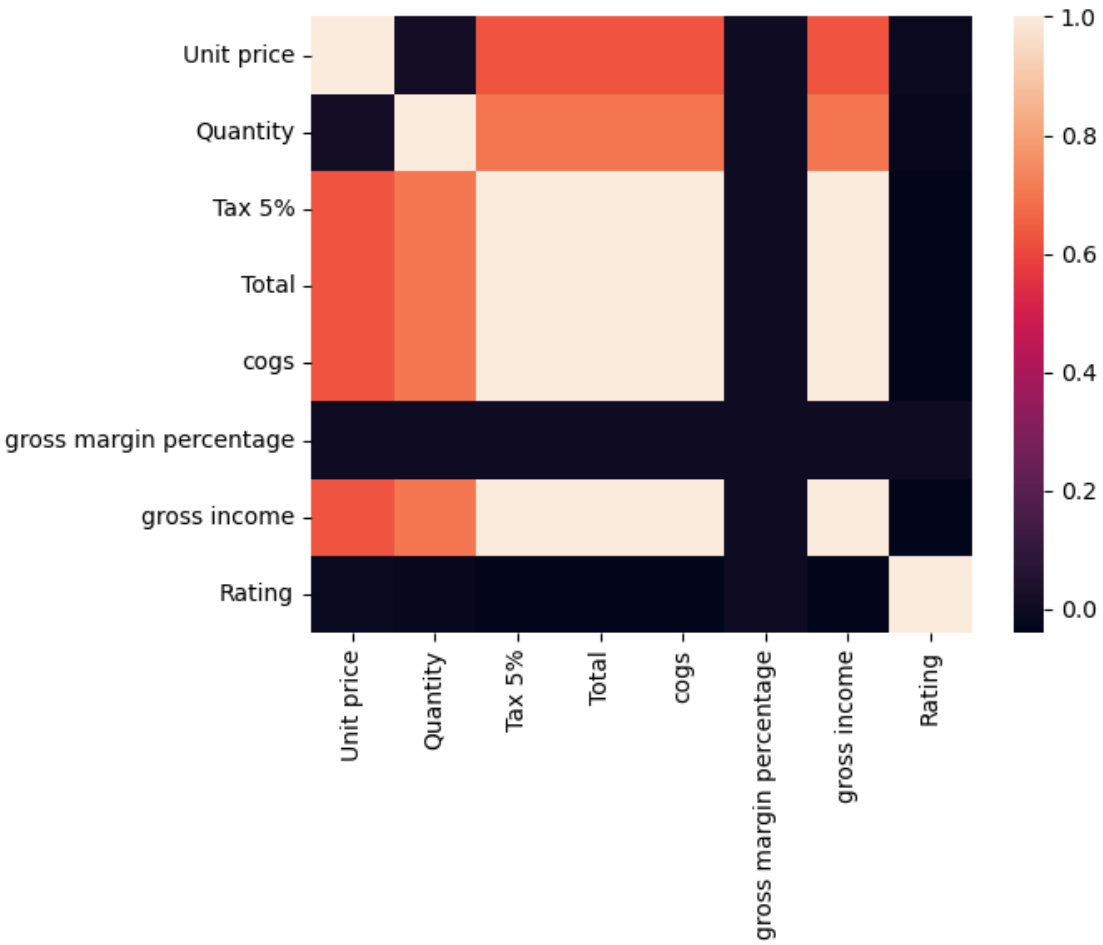
[71]:

	Unit price	Quantity	Tax 5%	Total	cogs	\
Unit price	1.00	0.01	0.63	0.63	0.63	
Quantity	0.01	1.00	0.70	0.70	0.70	
Tax 5%	0.63	0.70	1.00	1.00	1.00	
Total	0.63	0.70	1.00	1.00	1.00	
cogs	0.63	0.70	1.00	1.00	1.00	
gross margin percentage	0.00	0.00	0.00	0.00	0.00	
gross income	0.63	0.70	1.00	1.00	1.00	
Rating	-0.01	-0.02	-0.04	-0.04	-0.04	

	gross margin percentage	gross income	Rating
Unit price	0.0	0.63	-0.01
Quantity	0.0	0.70	-0.02
Tax 5%	0.0	1.00	-0.04
Total	0.0	1.00	-0.04
cogs	0.0	1.00	-0.04
gross margin percentage	0.0	0.00	0.00
gross income	0.0	1.00	-0.04
Rating	0.0	-0.04	1.00

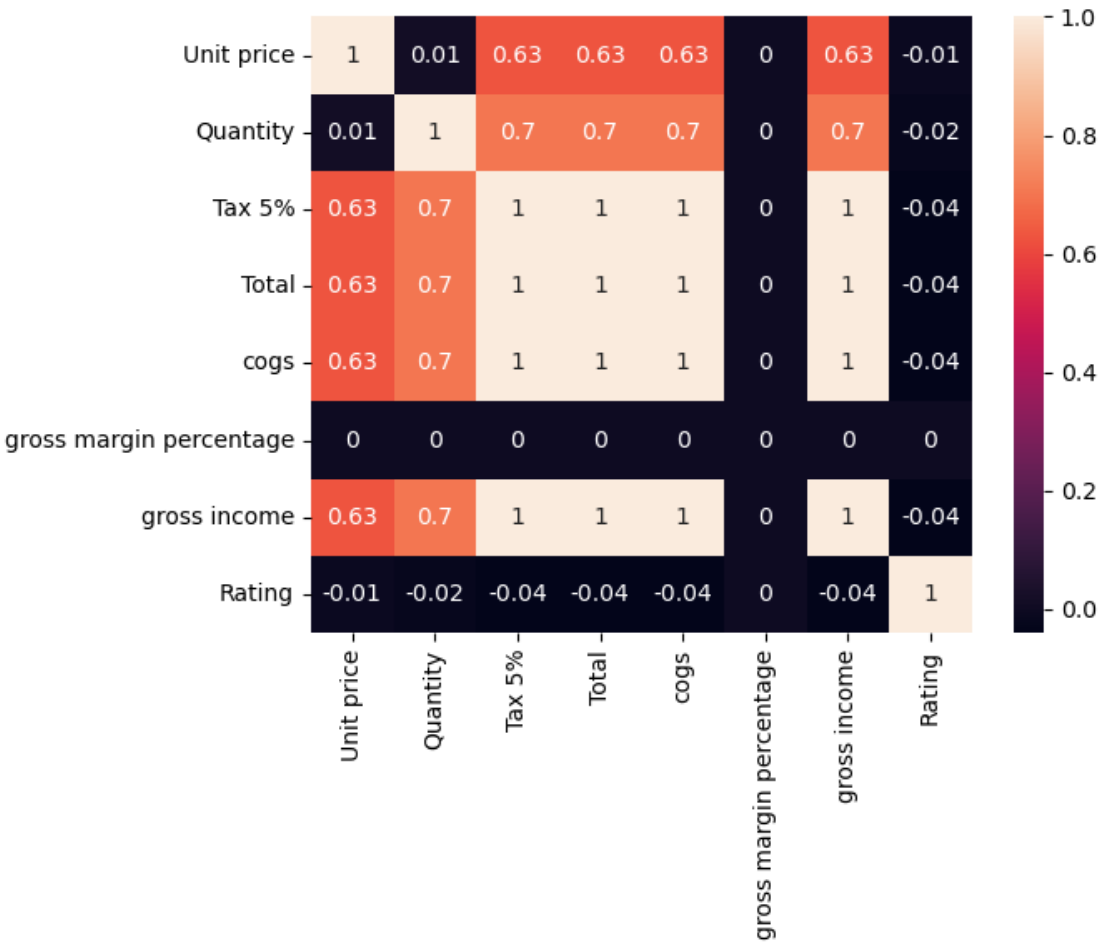
[72]:
```
sns.heatmap(np.round(clean_corr_matrix,2))
```

[72]: <AxesSubplot: >



[75]:
```
sns.heatmap(np.round(clean_corr_matrix,2),annot=True)
```

[75]: <AxesSubplot: >


Unit price
Quantity
Tax 5%
Total
cogs
gross margin percentage
gross income
Rating
1 0.01 0.63 0.63 0.63 0 0.63 -0.01
0.01 1 0.7 0.7 0.7 0 0.7 -0.02
0.63 0.7 1 1 1 0 1 -0.04
0.63 0.7 1 1 1 0 1 -0.04
0.63 0.7 1 1 1 0 1 -0.04
0 0 0 0 0 0 0 0
0.63 0.7 1 1 1 0 1 -0.04
-0.01 -0.02 -0.04 -0.04 -0.04 0 -0.04 1
1.0
0.8
0.6
0.4
0.2
0.0

[ ]: