Exploratory Data Analysis With Python and Pandas

Link to data source: https://www.kaggle.com/aungpyaeap/supermarket-sales

Here we will practice;

- Step-1: Initial Data Exploration
- Step-2: Univariate Analysis
- Step-3: Bivariate Analysis
- Step-4: Dealing With Duplicate Rows and Missing Values
- Step-5: Correlation Analysis

In [ ]:
```
# install library calmap: pip install calmap
# in your terminal, not within the Python interpreter
# If it still have an error "SyntaxError: invalid syntax", you can choose New te
```

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

Step-1: Initial Data Exploration

In [ ]:
```
df = pd.read_csv('supermarket_sales_sheet_1.csv')
```

In [ ]:
```
#See first 5 raws

df.head()
```

Out[ ]:

	Invoice ID	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%
0	750-67-8428	A	Yangon	Member	Female	Health and beauty	74.69	7	26.1415
1	226-31-3081	C	Naypyitaw	Normal	Female	Electronic accessories	15.28	5	3.8200
2	631-41-3108	A	Yangon	Normal	Male	Home and lifestyle	46.33	7	16.2155
3	123-19-1176	A	Yangon	Member	Male	Health and beauty	58.22	8	23.2880
4	373-73-7910	A	Yangon	Normal	Male	Sports and travel	86.31	7	30.2085

In [ ]:

```
#Last 8 raws

df.tail(8)
```

Out[ ]:

	Invoice ID	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5
992	745-74-0715	A	Yangon	Normal	Male	Electronic accessories	58.03	2	5.80
993	690-01-6631	B	Mandalay	Normal	Male	Fashion accessories	17.49	10	8.74
994	652-49-6720	C	Naypyitaw	Member	Female	Electronic accessories	60.95	1	3.04
995	233-67-5758	C	Naypyitaw	Normal	Male	Health and beauty	40.35	1	2.01
996	303-96-2227	B	Mandalay	Normal	Female	Home and lifestyle	97.38	10	48.69
997	727-02-1313	A	Yangon	Member	Male	Food and beverages	31.84	1	1.59
998	347-56-2442	A	Yangon	Normal	Male	Home and lifestyle	65.82	1	3.29
999	849-09-3807	A	Yangon	Member	Female	Fashion accessories	88.34	7	30.91

In [ ]:
```
#Display the all coulmn names in the dataset

df.columns
```

Out[ ]:
```
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')
```

In [ ]:
```
df.dtypes
```

Out[ ]:
```
Invoice ID                 object
Branch                     object
City                       object
Customer type              object
Gender                     object
Product line               object
Unit price                float64
Quantity                    int64
Tax 5%                    float64
Total                     float64
Date                       object
Time                       object
Payment                    object
cogs                      float64
gross margin percentage   float64
gross income              float64
Rating                    float64
dtype: object
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 17 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   Invoice ID              1000 non-null   object 
 1   Branch                  1000 non-null   object 
 2   City                    1000 non-null   object 
 3   Customer type           1000 non-null   object 
 4   Gender                  1000 non-null   object 
 5   Product line            1000 non-null   object 
 6   Unit price              1000 non-null   float64
 7   Quantity                1000 non-null   int64  
 8   Tax 5%                  1000 non-null   float64
 9   Total                   1000 non-null   float64
 10  Date                    1000 non-null   object 
 11  Time                    1000 non-null   object 
 12  Payment                 1000 non-null   object 
 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), object(9)
memory usage: 132.9+ KB
```

In [ ]:
```
df.shape
```

Out[ ]:
```
(1000, 17)
```

In [ ]:
```
df.describe() #to show statistics of numeric columns
```

Out[ ]:

	Unit price	Quantity	Tax 5%	Total	cogs	gross margin percentage	
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.00000	1000.000000	10
mean	55.672130	5.510000	15.379369	322.966749	307.58738	4.761905	
std	26.494628	2.923431	11.708825	245.885335	234.17651	0.000000	
min	10.080000	1.000000	0.508500	10.678500	10.17000	4.761905	
25%	32.875000	3.000000	5.924875	124.422375	118.49750	4.761905	
50%	55.230000	5.000000	12.088000	253.848000	241.76000	4.761905	
75%	77.935000	8.000000	22.445250	471.350250	448.90500	4.761905	
max	99.960000	10.000000	49.650000	1042.650000	993.00000	4.761905	

In [ ]:
```
df['Date']
```

Out[ ]:
```
0       1/5/2019
1       3/8/2019
2       3/3/2019
3      1/27/2019
4       2/8/2019
         ...
995    1/29/2019
996     3/2/2019
997     2/9/2019
998    2/22/2019
999    2/18/2019
Name: Date, Length: 1000, dtype: object
```

Note: Data type of dates are object, however, it should have been date time. We can convert it as;

In [ ]:
```
#Change data type from object to datetime

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

In [ ]:
```
df['Date']
```

Out[ ]:
```
0     2019-01-05
1     2019-03-08
2     2019-03-03
3     2019-01-27
4     2019-02-08
         ...
995   2019-01-29
996   2019-03-02
997   2019-02-09
998   2019-02-22
999   2019-02-18
Name: Date, Length: 1000, dtype: datetime64[ns]
```

- Setting the date column as the index for the data frame;

In [ ]:
```
#inplace=True is used in order to make permanent changes

df.set_index ('Date', inplace=True)
```

In [ ]:
```
df.head()
#The dates will be in the index column;
```

Out[ ]:

Date	Invoice ID	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax
2019-01-05	750-67-8428	A	Yangon	Member	Female	Health and beauty	74.69	7	26.
2019-03-08	226-31-3081	C	Naypyitaw	Normal	Female	Electronic accessories	15.28	5	3.
2019-03-03	631-41-3108	A	Yangon	Normal	Male	Home and lifestyle	46.33	7	16.
2019-01-27	123-19-1176	A	Yangon	Member	Male	Health and beauty	58.22	8	23.
2019-02-08	373-73-7910	A	Yangon	Normal	Male	Sports and travel	86.31	7	30.

Step-2: Univariate Analysis

(We will look at one variable at a time)

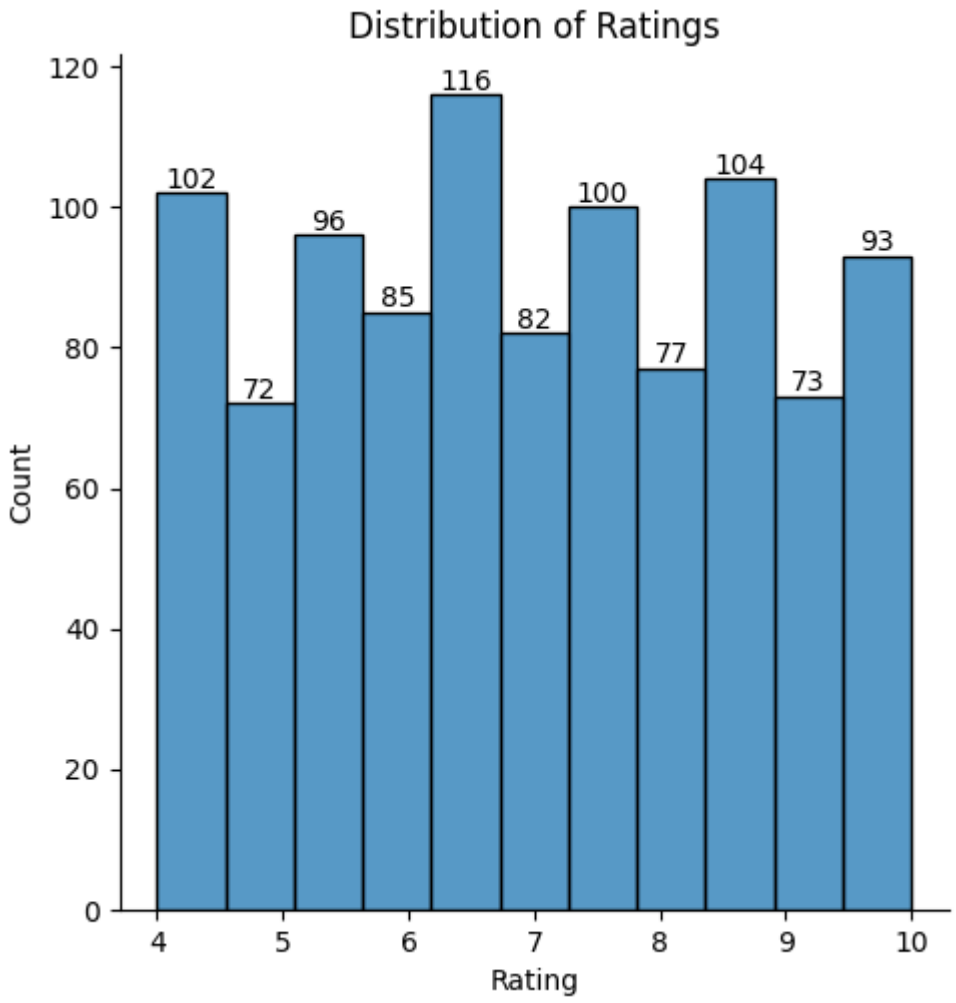
Question: What does the distribution of customer ratings look like?

In [ ]:
```
ratings=df['Rating']
ratings
```

Out[ ]:
```
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
Name: Rating, Length: 1000, dtype: float64
```

In [ ]:
```
# Plot the distribution of 'Rating'
sns.displot(df['Rating'])

# Add a title to the plot
plt.title("Distribution of Ratings")

# Add value labels to the bars of the histogram
ax = plt.gca()
for p in ax.patches:
    ax.annotate(f'{p.get_height():.0f}', (p.get_x() + p.get_width() / 2, p.get_h
                ha='center', va='bottom', fontsize=10)

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

```
c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn
\axisgrid.py:118: UserWarning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```

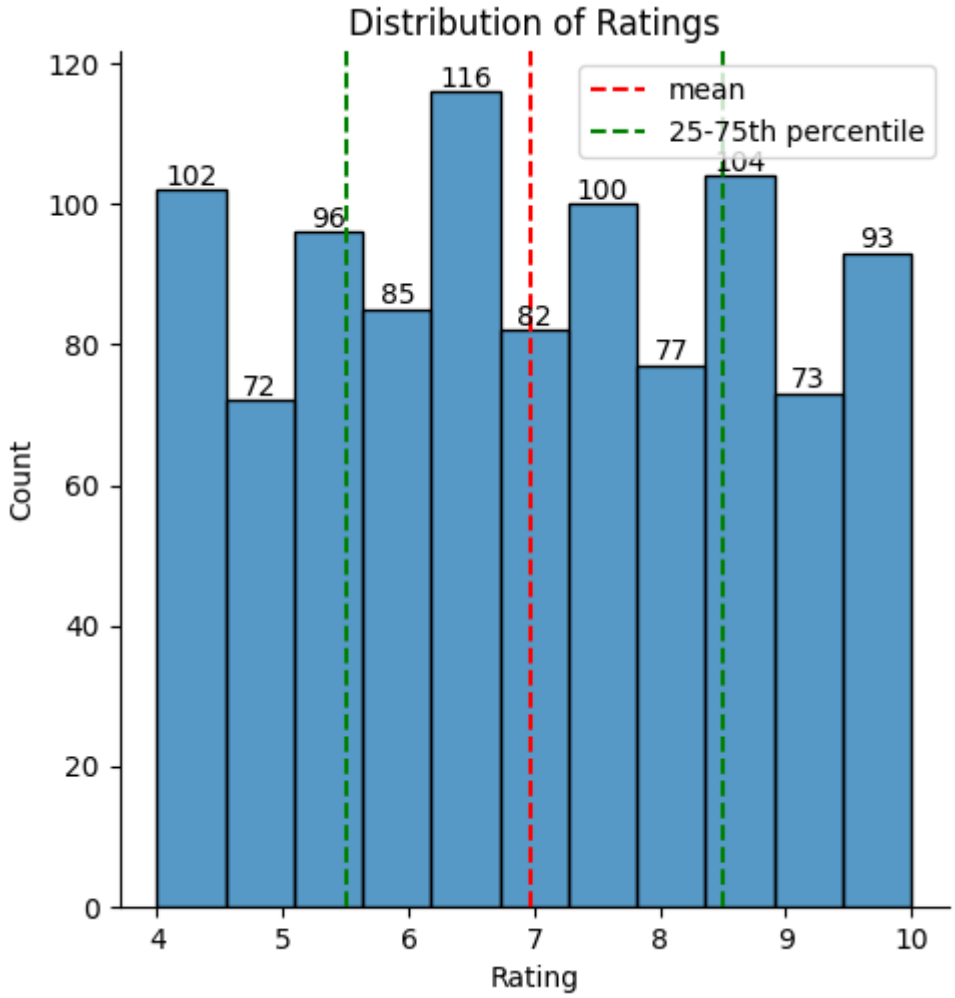


In [ ]:
```
# Plot the distribution of 'Rating'
sns.displot(df['Rating'])

# Add a title to the plot
plt.title("Distribution of Ratings")

# Add value labels to the bars of the histogram
ax = plt.gca()
for p in ax.patches:
    ax.annotate(f'{p.get_height():.0f}', (p.get_x() + p.get_width() / 2, p.get_h
```

```
                ha='center', va='bottom', fontsize=10)

#Display mean rating
#'vline' means vertical line; 'c' means color; 'ls' for line style
#we use numpy to calculate mean
#x is mean
plt.axvline(x = np.mean(df['Rating']), c= 'red', ls='--', label='mean')

#let's see 25th and 75th percentile
plt.axvline(x = np.percentile(df['Rating'],25), c= 'green', ls='--', label='25-7
plt.axvline(x = np.percentile(df['Rating'],75), c= 'green', ls='--')

#to see labels on the graph
plt.legend()
```

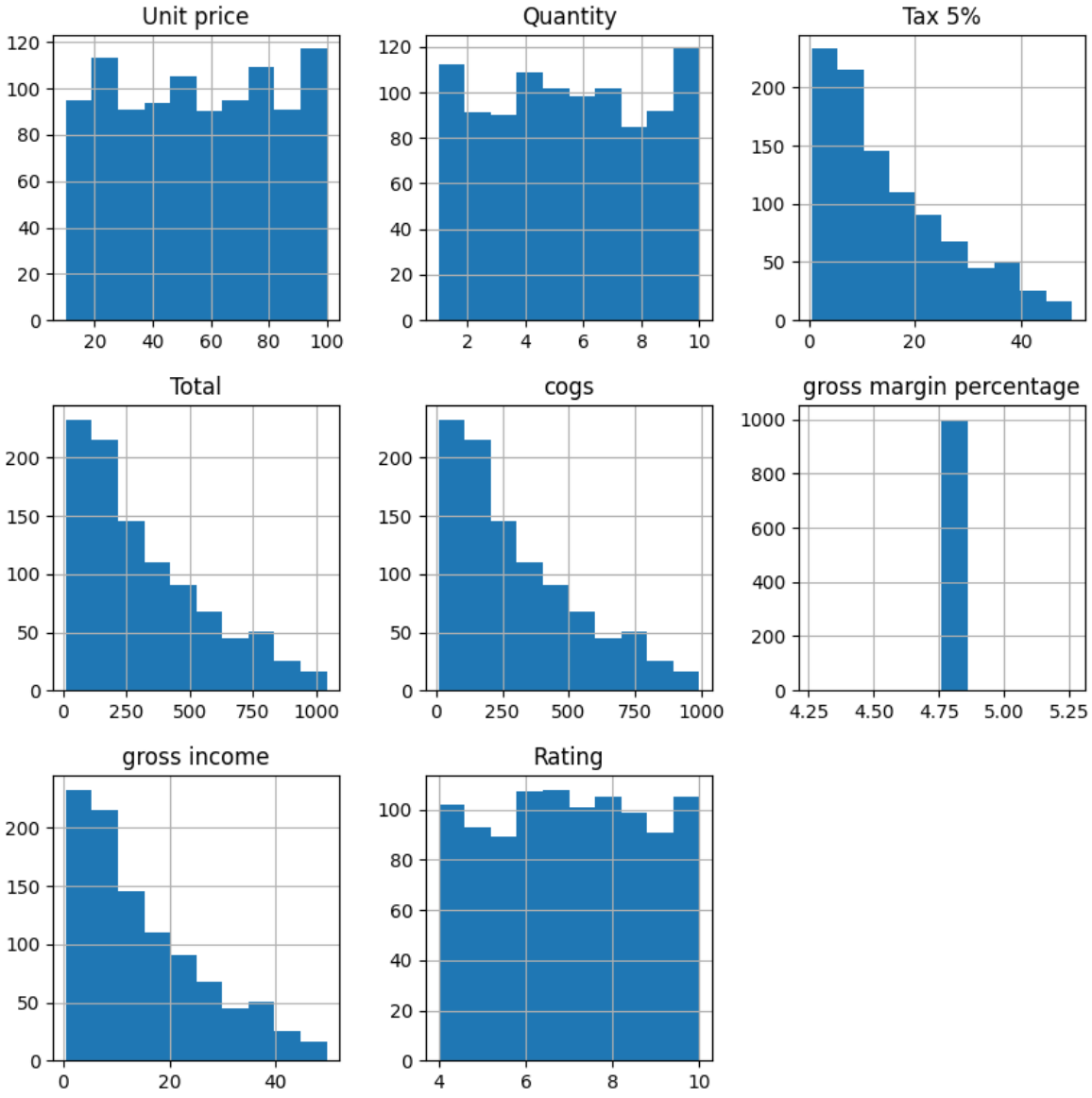
```
c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn
\axisgrid.py:118: UserWarning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```

Out[ ]: <matplotlib.legend.Legend at 0x1f4bf19dd80>



In [ ]:
```
#For seeing graphs of all numerical values

df.hist(figsize=(10,10))
```

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

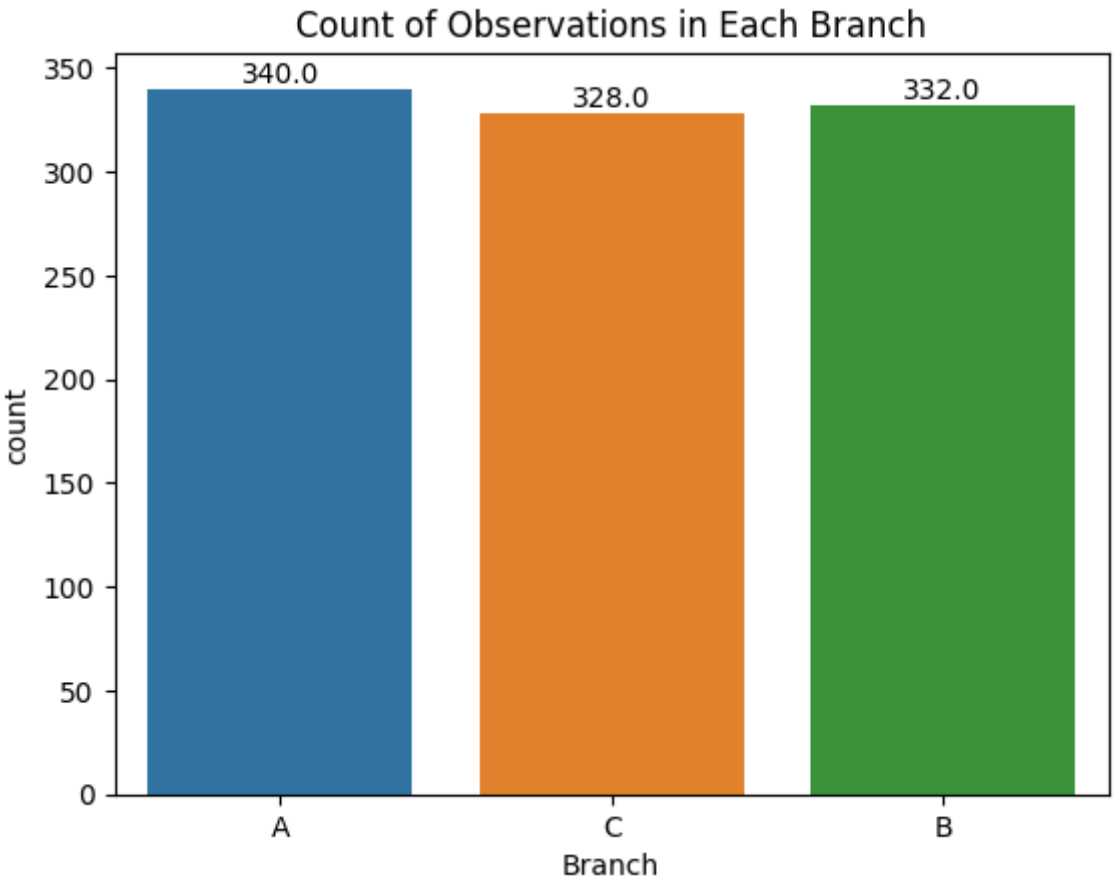


Question: Do aggregate sales numbers differ by much between branches?

In [ ]:
```
#df.info()
df['Branch']
```

Out[ ]:
```
Date
2019-01-05    A
2019-03-08    C
2019-03-03    A
2019-01-27    A
2019-02-08    A
              ..
2019-01-29    C
2019-03-02    B
2019-02-09    A
2019-02-22    A
2019-02-18    A
Name: Branch, Length: 1000, dtype: object
```

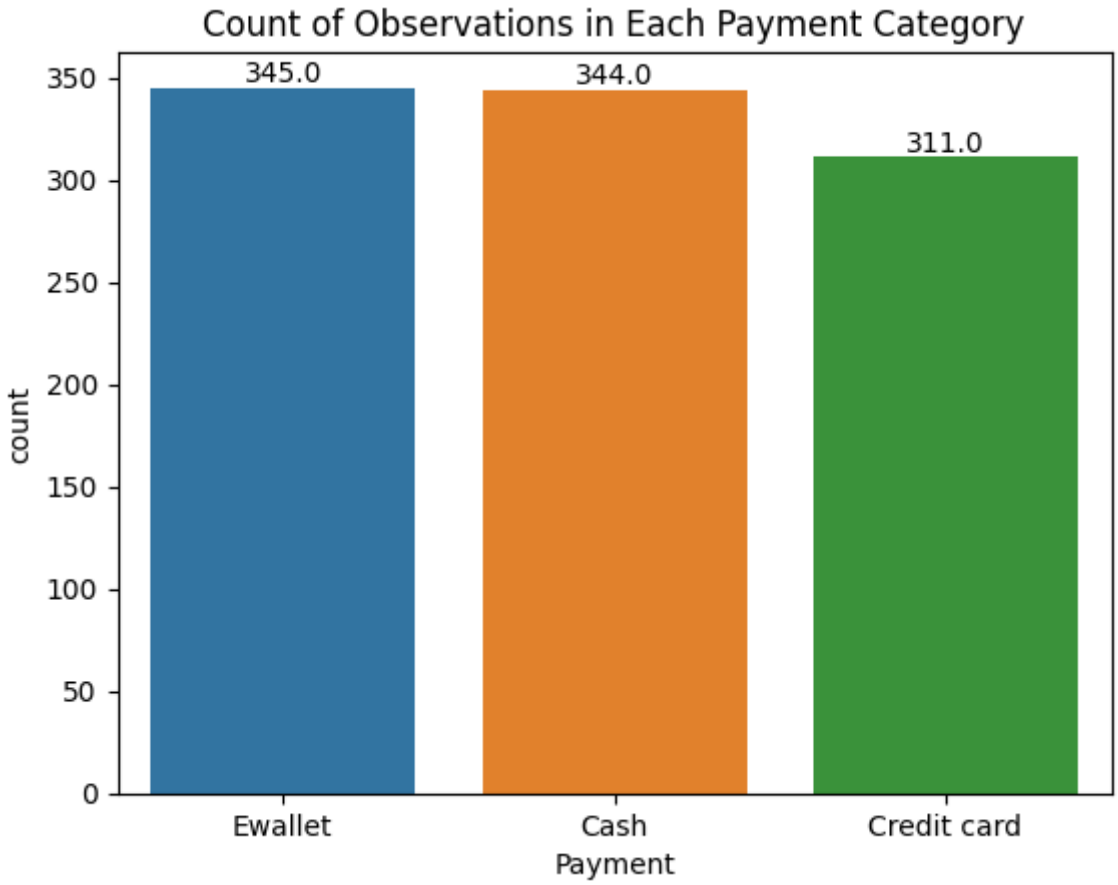
In [ ]:
```python
# Plot the count of each category in 'Branch'
ax = sns.countplot(x=df['Branch'])

# Add value labels to the bars
for p in ax.patches:
    height = p.get_height()
    ax.annotate(f'{height}', (p.get_x() + p.get_width() / 2, height),
                ha='center', va='bottom', fontsize=10)

# Add a title to the plot
plt.title("Count of Observations in Each Branch")

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



In [ ]:
```python
# Count the occurrences of each unique value in the 'Branch' column
branch_counts = df['Branch'].value_counts()
branch_counts
```

Out[ ]:
```
A    340
B    332
C    328
Name: Branch, dtype: int64
```

In [ ]:
```python
# Plot the count of each category in 'Payment'
ax = sns.countplot(x=df['Payment'])

# Add value labels to the bars
for p in ax.patches:
    height = p.get_height()
    ax.annotate(f'{height}', (p.get_x() + p.get_width() / 2, height),
                ha='center', va='bottom', fontsize=10)

# Add a title to the plot
plt.title("Count of Observations in Each Payment Category")

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



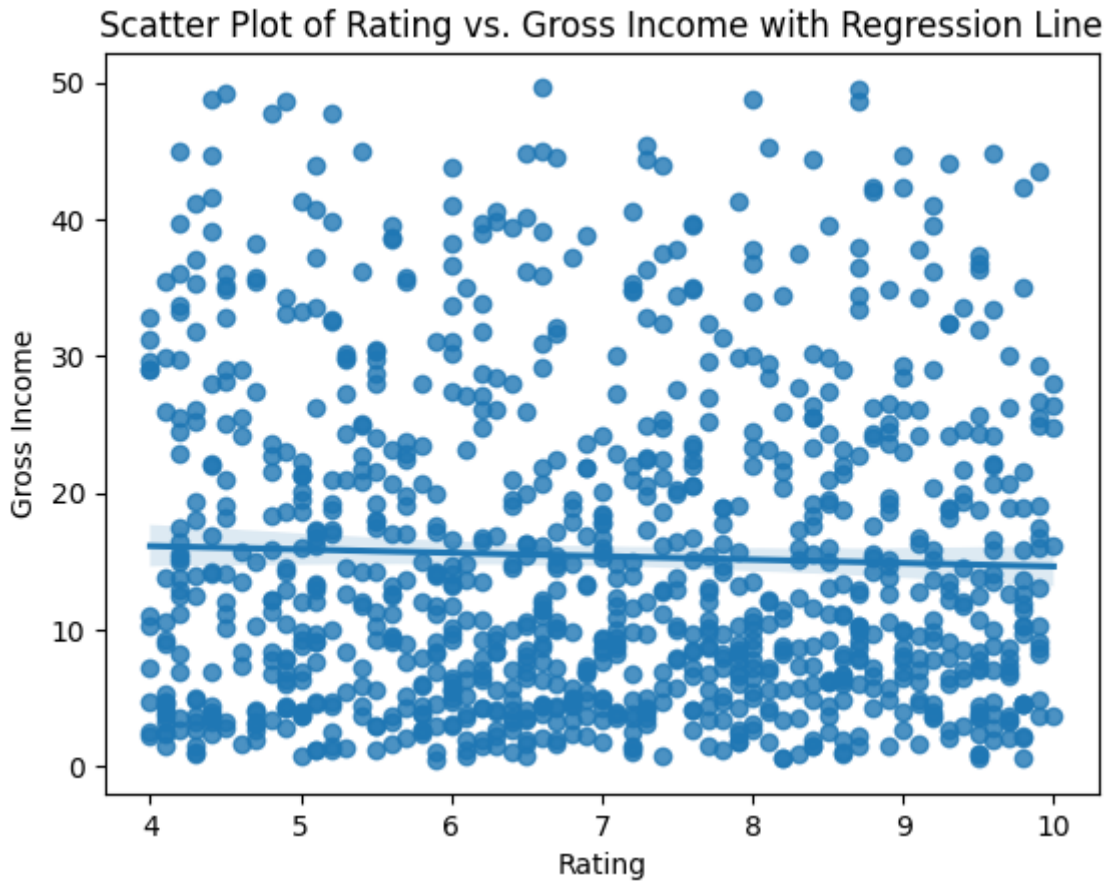
Step-3: Bivariate Analysis

Question: Is there a relationship between gross income and customer ratings?

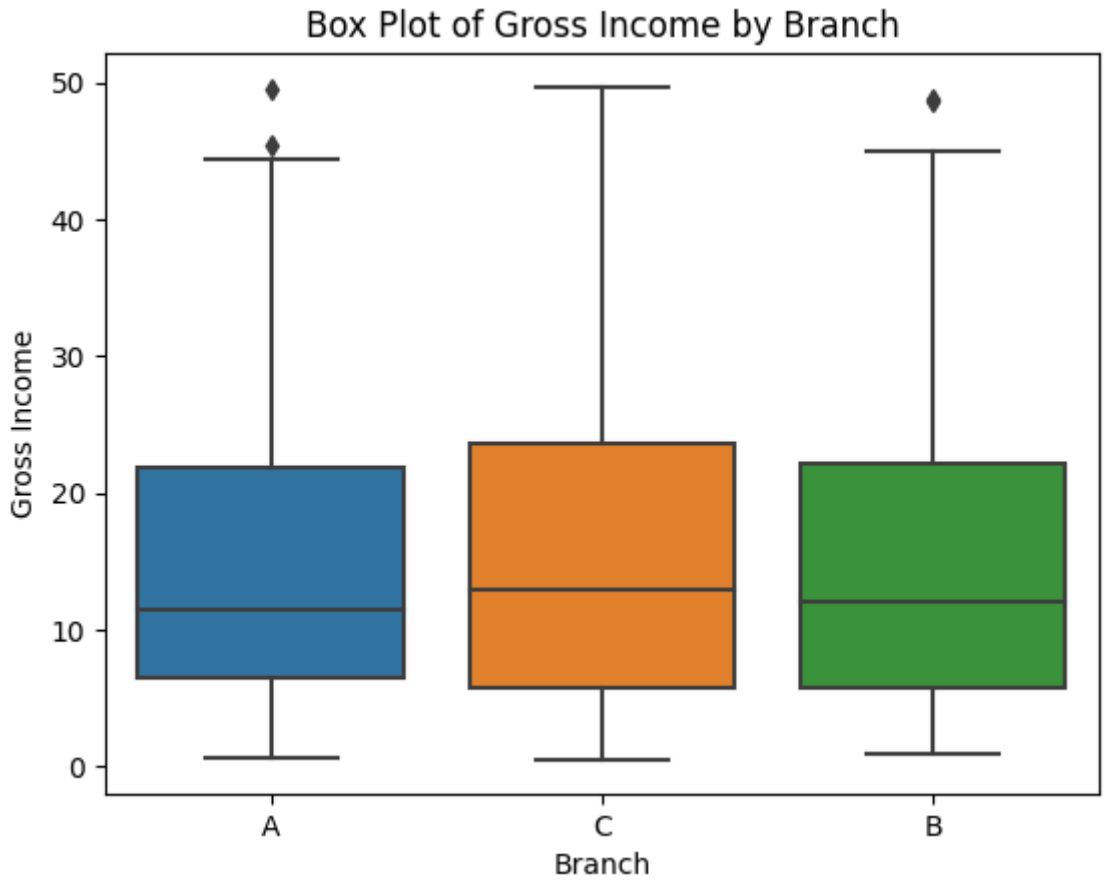
In [ ]:
```python
# Create a scatter plot between 'Rating' and 'gross income'
sns.scatterplot(x=df['Rating'], y=df['gross income'])

# Add labels to the axes
plt.xlabel('Rating')
plt.ylabel('Gross Income')
```

```
# Add a title to the plot
plt.title("Scatter Plot of Rating vs. Gross Income")

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

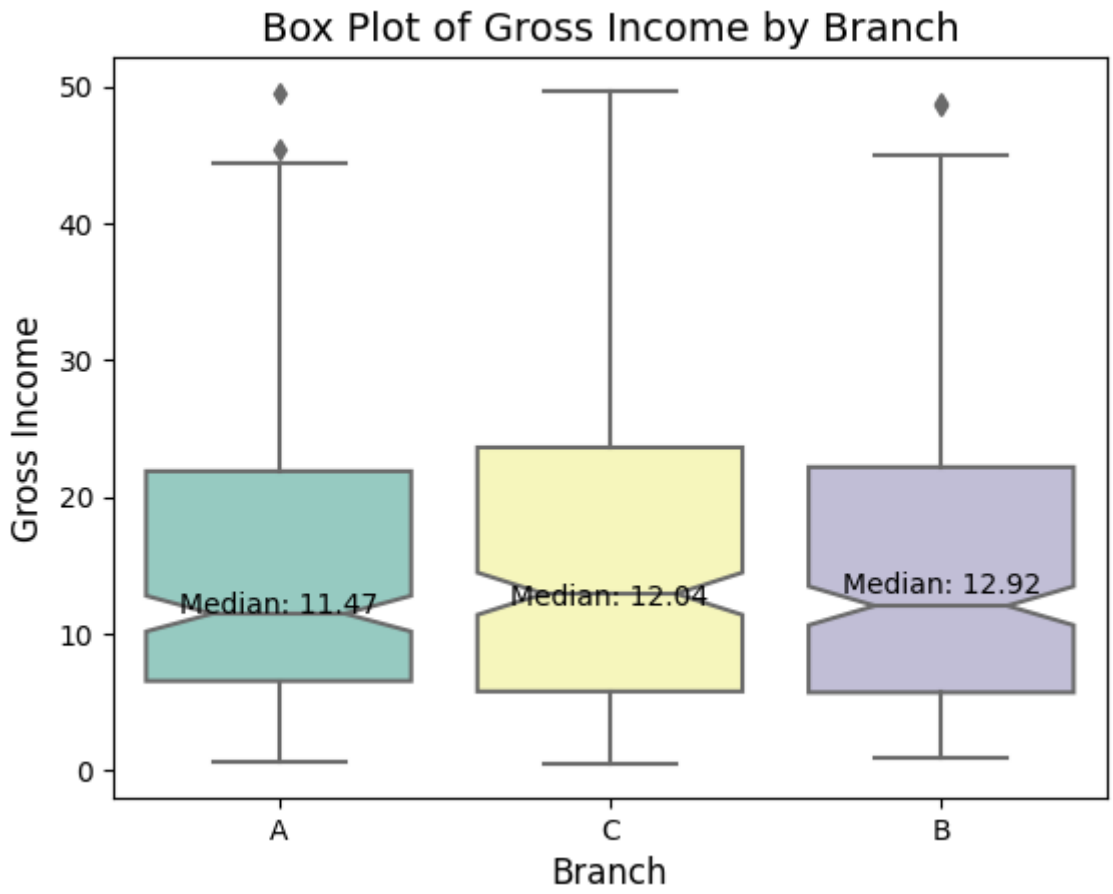
In [ ]:
```
# Create a scatter plot with a regression line between 'Rating' and 'gross incom
sns.regplot(x=df['Rating'], y=df['gross income'])

# Add labels to the axes
plt.xlabel('Rating')
plt.ylabel('Gross Income')

# Add a title to the plot
plt.title("Scatter Plot of Rating vs. Gross Income with Regression Line")

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



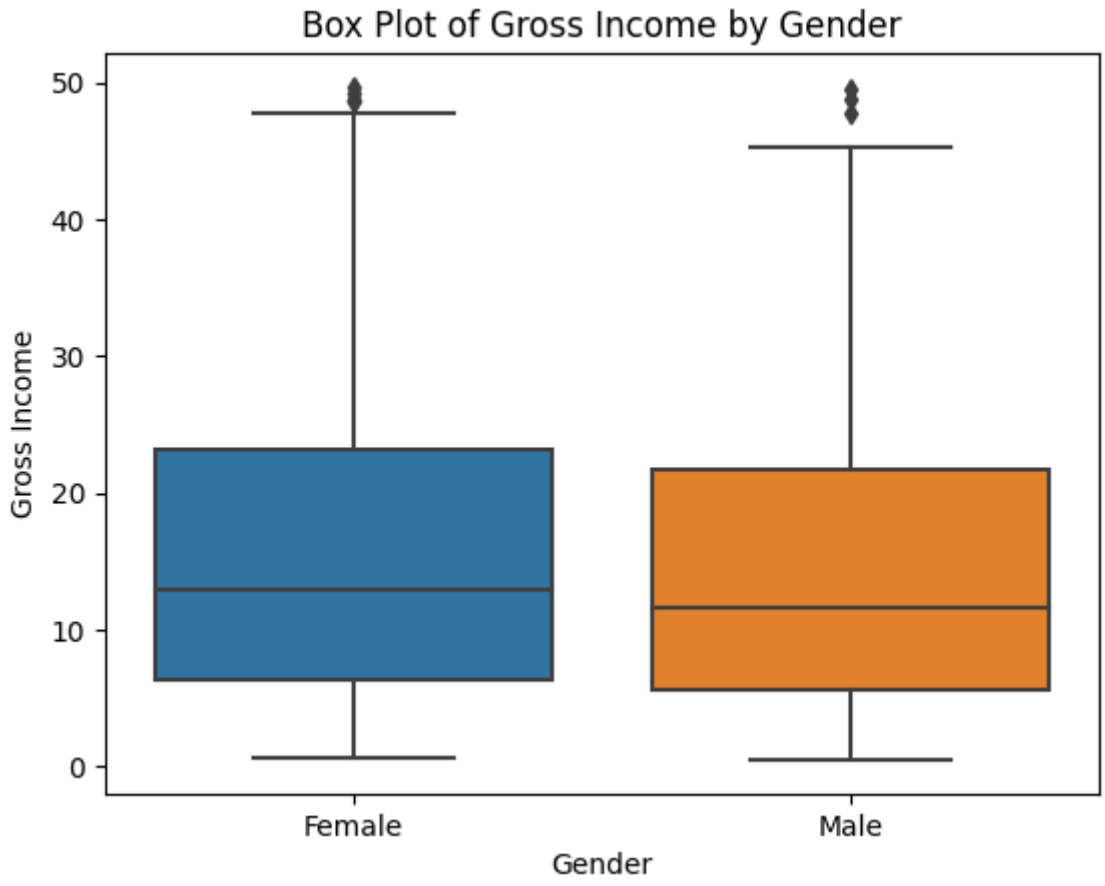
In [ ]:
```
# Create a box plot to visualize the relationship between 'Branch' and 'gross in
sns.boxplot(x=df['Branch'], y=df['gross income'])

# Add labels to the axes
plt.xlabel('Branch')
plt.ylabel('Gross Income')

# Add a title to the plot
plt.title("Box Plot of Gross Income by Branch")

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



In [ ]:
```
# Create a box plot to visualize the relationship between 'Branch' and 'gross in
sns.boxplot(x=df['Branch'], y=df['gross income'], palette='Set3', notch=True)

# Add labels to the axes
plt.xlabel('Branch', fontsize=12)
plt.ylabel('Gross Income', fontsize=12)

# Add a title to the plot
plt.title("Box Plot of Gross Income by Branch", fontsize=14)

# Customize the x-axis tick labels
plt.xticks(fontsize=10)

# Add annotations for median values
medians = df.groupby('Branch')['gross income'].median()
for i, median in enumerate(medians):
    plt.text(i, median, f'Median: {median:.2f}', horizontalalignment='center', f

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



In [ ]:
```
# Create a box plot to visualize the relationship between 'Gender' and 'gross in
sns.boxplot(x=df['Gender'], y=df['gross income'])

# Add labels to the axes
plt.xlabel('Gender')
plt.ylabel('Gross Income')

# Add a title to the plot
plt.title("Box Plot of Gross Income by Gender")

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



In [ ]:

```
# a better way
# Create a violin plot to visualize the relationship between 'Gender' and 'gross
sns.violinplot(x=df['Gender'], y=df['gross income'], palette='Set2', inner='quar

# Add labels to the axes
plt.xlabel('Gender')
plt.ylabel('Gross Income')

# Add a title to the plot
plt.title("Violin Plot of Gross Income by Gender")

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

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

Out[ ]:

Date	Invoice ID	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax
2019-01-05	750-67-8428	A	Yangon	Member	Female	Health and beauty	74.69	7	26.
2019-03-08	226-31-3081	C	Naypyitaw	Normal	Female	Electronic accessories	15.28	5	3.
2019-03-03	631-41-3108	A	Yangon	Normal	Male	Home and lifestyle	46.33	7	16.
2019-01-27	123-19-1176	A	Yangon	Member	Male	Health and beauty	58.22	8	23.
2019-02-08	373-73-7910	A	Yangon	Normal	Male	Sports and travel	86.31	7	30.

We need to aggragete the data somehow because the dates can be repeated (different customers may come same day). We can use "groupby" function in pandas for this purpose;

In [ ]:
```
#Grouping according to the dates, grouping by the mean
#Each date row will be unique, represents the average value for the particular a

# Calculate the mean for each column based on the index (row labels)
#mean_values = df.groupby(df.index).mean()
df.groupby(df.index).mean()
```

Out[ ]:

Date	Unit price	Quantity	Tax 5%	Total	cogs	gross margin percentage	gross income
2019-01-01	54.995833	6.750000	18.830083	395.431750	376.601667	4.761905	18.830083
2019-01-02	44.635000	6.000000	11.580375	243.187875	231.607500	4.761905	11.580375
2019-01-03	59.457500	4.625000	12.369813	259.766062	247.396250	4.761905	12.369813
2019-01-04	51.743333	5.333333	12.886417	270.614750	257.728333	4.761905	12.886417
2019-01-05	61.636667	4.583333	14.034458	294.723625	280.689167	4.761905	14.034458
...	...	...	...	...	...	...	...
2019-03-26	42.972308	4.000000	7.188692	150.962538	143.773846	4.761905	7.188692
2019-03-27	56.841000	4.500000	13.822950	290.281950	276.459000	4.761905	13.822950
2019-03-28	45.525000	4.800000	10.616200	222.940200	212.324000	4.761905	10.616200
2019-03-29	66.346250	6.750000	23.947875	502.905375	478.957500	4.761905	23.947875
2019-03-30	67.408182	6.090909	19.424500	407.914500	388.490000	4.761905	19.424500

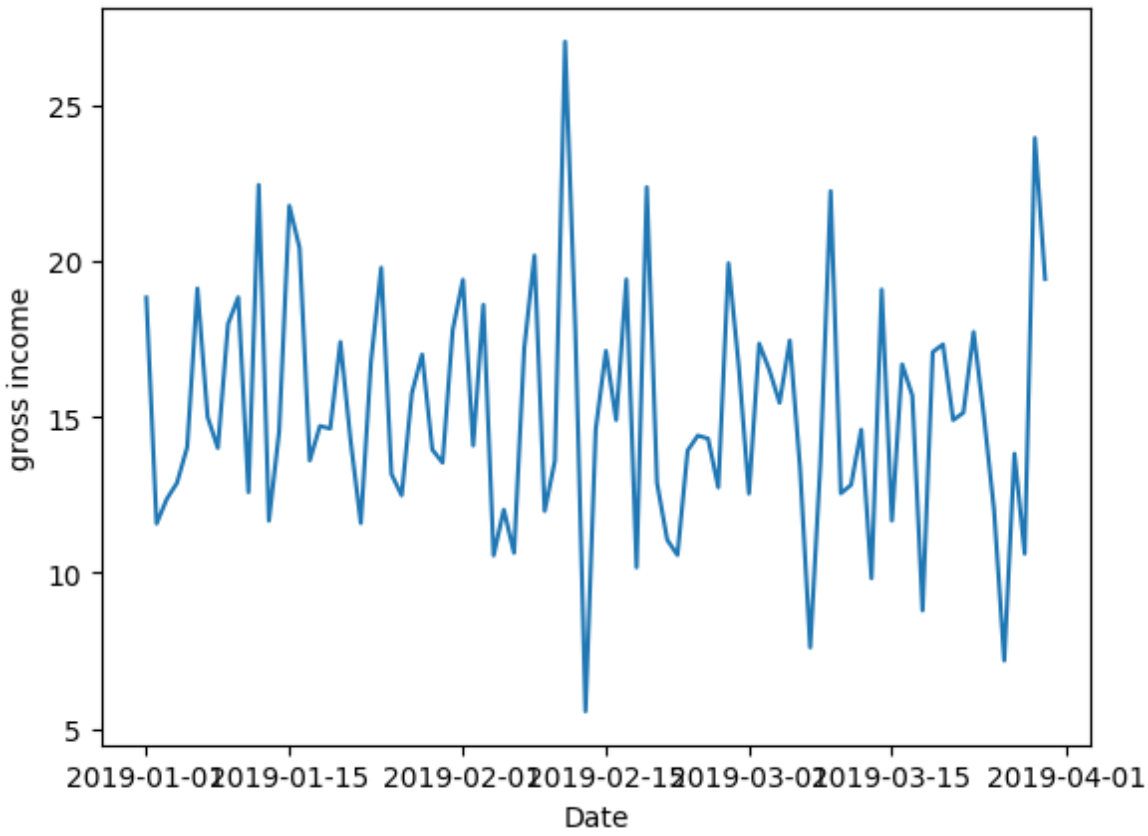
89 rows × 8 columns

In [ ]:
```
df.groupby(df.index).mean().index
```

Out[ ]:
```
DatetimeIndex(['2019-01-01', '2019-01-02', '2019-01-03', '2019-01-04',
               '2019-01-05', '2019-01-06', '2019-01-07', '2019-01-08',
               '2019-01-09', '2019-01-10', '2019-01-11', '2019-01-12',
               '2019-01-13', '2019-01-14', '2019-01-15', '2019-01-16',
               '2019-01-17', '2019-01-18', '2019-01-19', '2019-01-20',
               '2019-01-21', '2019-01-22', '2019-01-23', '2019-01-24',
               '2019-01-25', '2019-01-26', '2019-01-27', '2019-01-28',
               '2019-01-29', '2019-01-30', '2019-01-31', '2019-02-01',
               '2019-02-02', '2019-02-03', '2019-02-04', '2019-02-05',
               '2019-02-06', '2019-02-07', '2019-02-08', '2019-02-09',
               '2019-02-10', '2019-02-11', '2019-02-12', '2019-02-13',
               '2019-02-14', '2019-02-15', '2019-02-16', '2019-02-17',
               '2019-02-18', '2019-02-19', '2019-02-20', '2019-02-21',
               '2019-02-22', '2019-02-23', '2019-02-24', '2019-02-25',
               '2019-02-26', '2019-02-27', '2019-02-28', '2019-03-01',
               '2019-03-02', '2019-03-03', '2019-03-04', '2019-03-05',
               '2019-03-06', '2019-03-07', '2019-03-08', '2019-03-09',
               '2019-03-10', '2019-03-11', '2019-03-12', '2019-03-13',
               '2019-03-14', '2019-03-15', '2019-03-16', '2019-03-17',
               '2019-03-18', '2019-03-19', '2019-03-20', '2019-03-21',
               '2019-03-22', '2019-03-23', '2019-03-24', '2019-03-25',
               '2019-03-26', '2019-03-27', '2019-03-28', '2019-03-29',
               '2019-03-30'],
              dtype='datetime64[ns]', name='Date', freq=None)
```

In [ ]:
```
#x variables are the dates
#y variables are associated gross income with those dates

sns.lineplot(x = df.groupby(df.index).mean().index,
             y = df.groupby(df.index).mean()['gross income'])
```

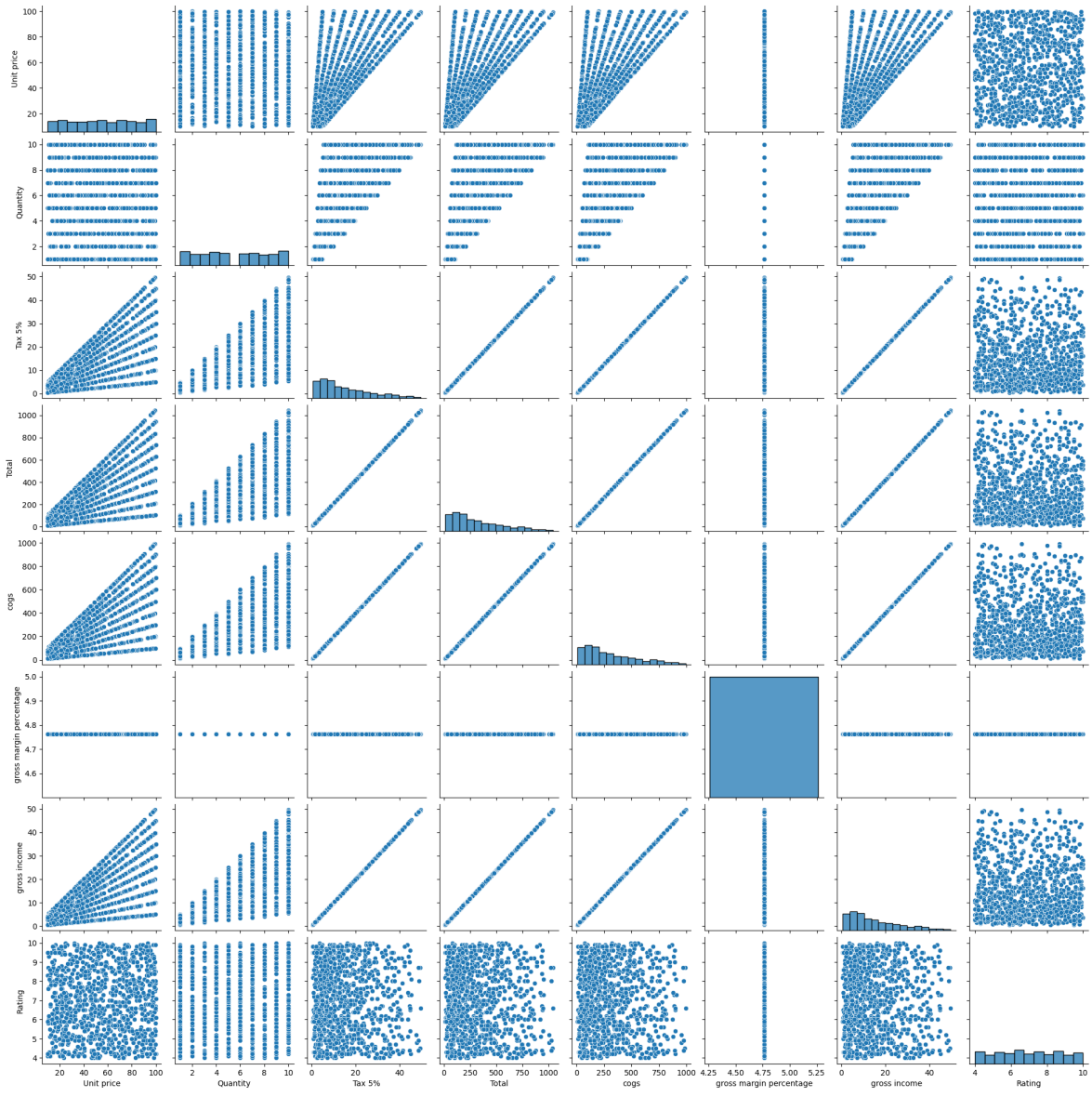
Out[ ]:
```
<Axes: xlabel='Date', ylabel='gross income'>
```



In [ ]:
```
#if we want to plot all the bivariate relationship possible, we can use seaborn
#Pairplot is NOT useful for large dataset
#You will see univeraite and pairwise distribution

sns.pairplot(df)
```

```
c:\Users\ADMIN\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn
\axisgrid.py:118: UserWarning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```

Out[ ]: <seaborn.axisgrid.PairGrid at 0x1f4bf86b940>



Step-4: Dealing with duplicate rows and missing values

In [ ]:
```
#Display the duplicated values

df.duplicated()
#If there is a duplicated value, we will see True; if the value is not duplicate
```

Out[ ]:
```
Date
2019-01-05    False
2019-03-08    False
2019-03-03    False
2019-01-27    False
2019-02-08    False
              ...
2019-01-29    False
2019-03-02    False
2019-02-09    False
2019-02-22    False
2019-02-18    False
Length: 1000, dtype: bool
```

In [ ]:
```
df.duplicated().sum()
#The output will tell us how many rows are duplicated. We have 3 in our dataset
```

Out[ ]:
```
0
```

In [ ]:
```
#to see the duplicated values

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

Out[ ]:

Invoice ID	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%	Total

Date

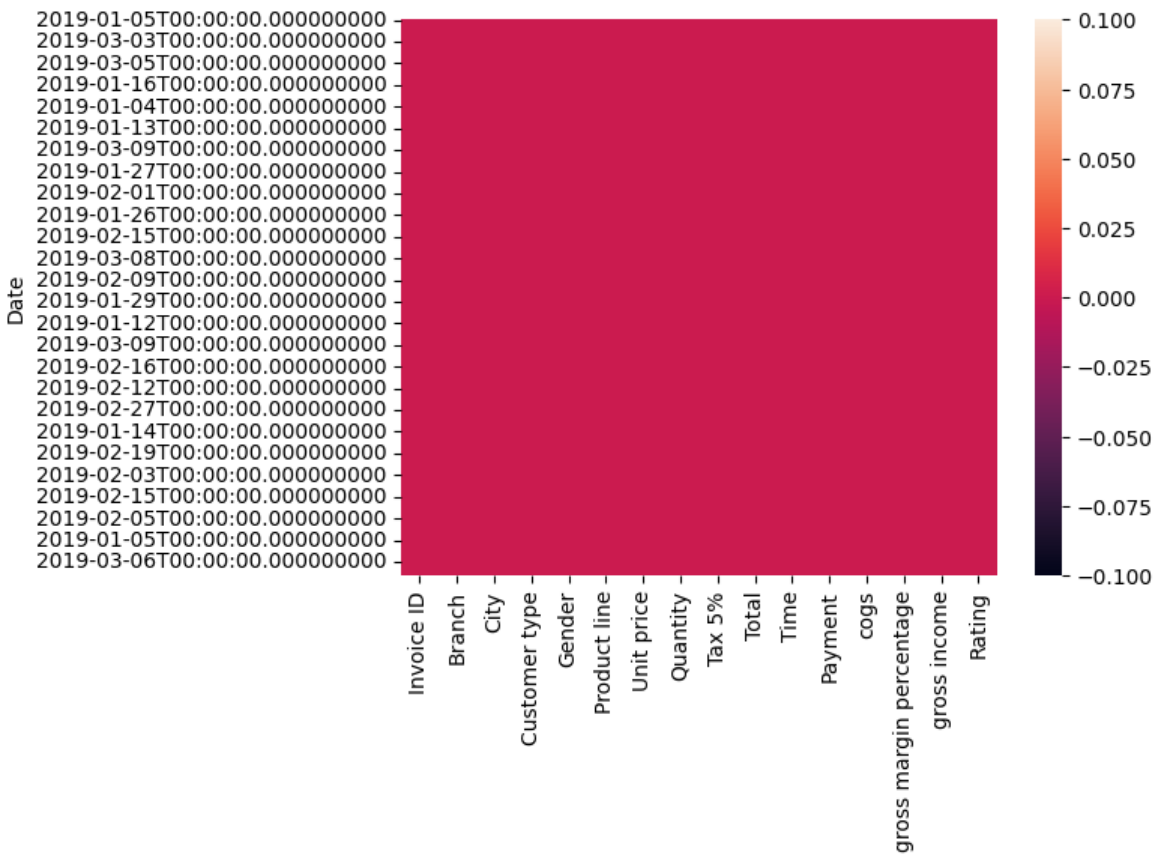
In [ ]:
```
#to remove duplicated rows,

df.drop_duplicates(inplace=True)

#And we made it permanent with inplace=True
```

In [ ]:
```
#to see missing values;

missing_value = df.isna().sum()
missing_value
#For example, customer type has no missing value
```

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

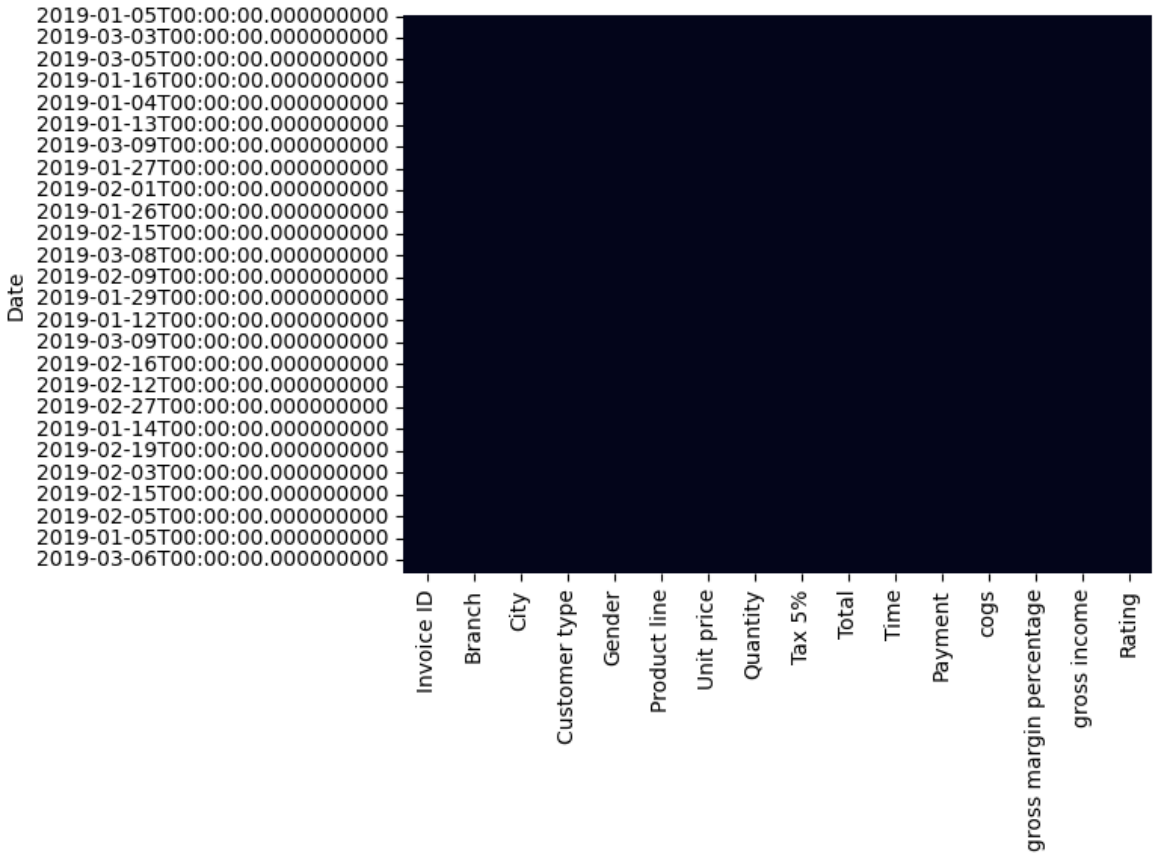
In [ ]:
```
#We can see it with seaborn heatmap, as well

sns.heatmap(df.isnull())
```

Out[ ]:
```
<Axes: ylabel='Date'>
```



In [ ]:
```
#White lines are missing values
#We can delete/not display cbar

sns.heatmap(df.isnull(),cbar=False)
```

Out[ ]:
```
<Axes: ylabel='Date'>
```



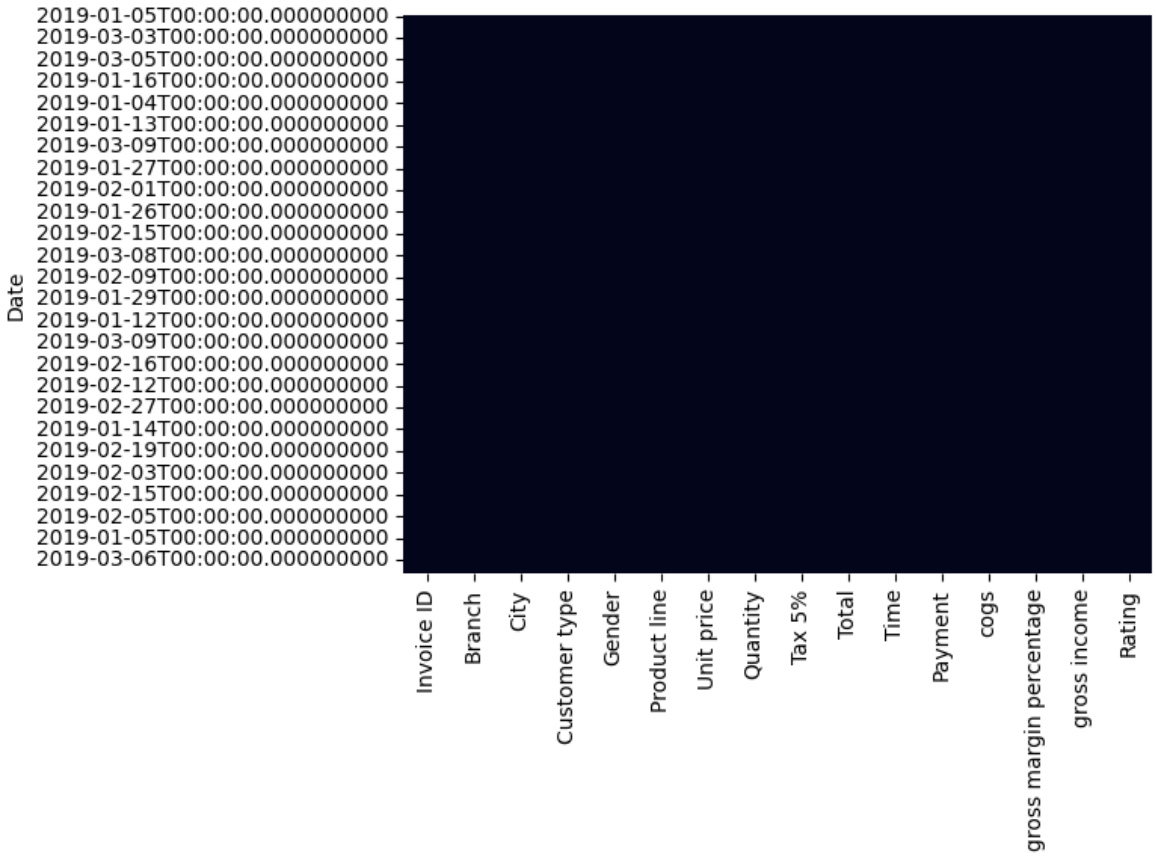
Here we will learn how to deal with missing values;

In [ ]:
```
#One way is filling the missing values with mean value;
df.fillna(df.mean(), inplace=True)

#df.fillna(df.mode().iloc[0], inplace=True) # iloc() function enables us to sele
#or we can fill them with 0  --> df.fillna(0)
```

```
C:\Users\ADMIN\AppData\Local\Temp\ipykernel_3884\3559958168.py:2: FutureWarning:
Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=None') i
s deprecated; in a future version this will raise TypeError.  Select only valid c
olumns before calling the reduction.
  df.fillna(df.mean(), inplace=True)
```

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

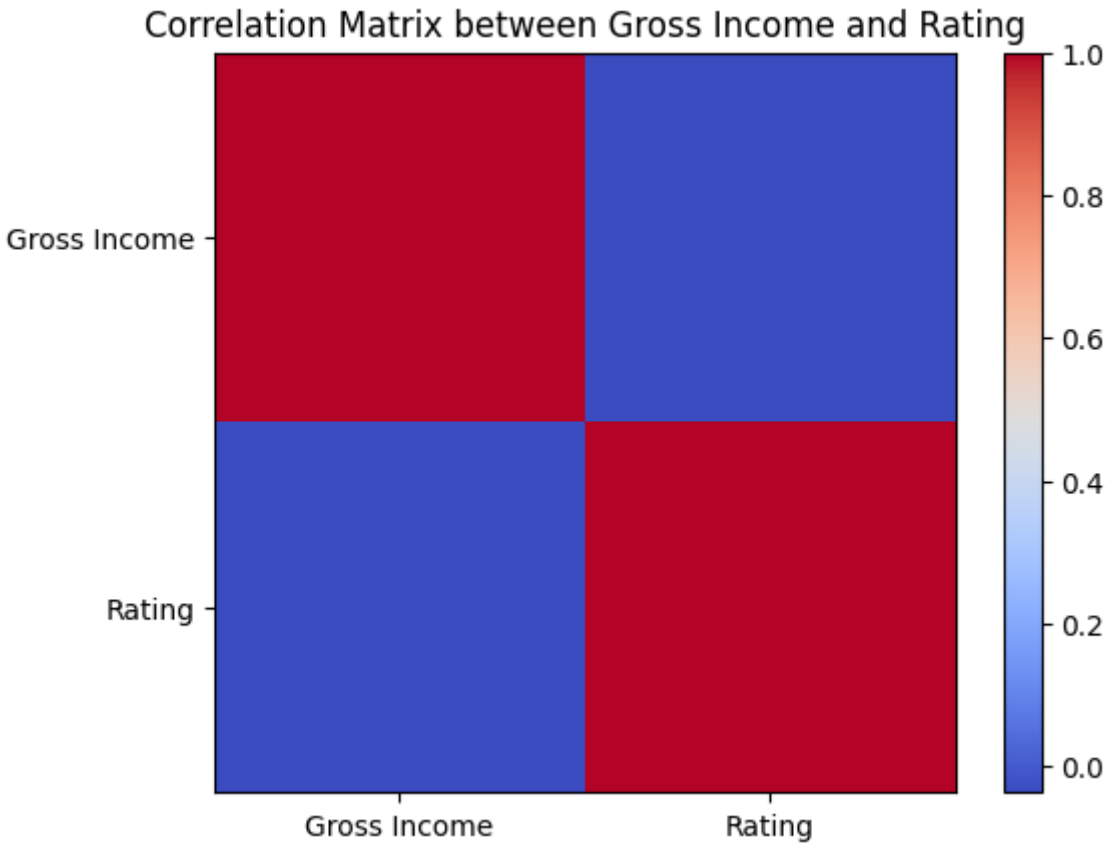
Out[ ]:
```
<Axes: ylabel='Date'>
```



In [ ]:
```
# dataset = pd.read_csv('Supermarket_sales.csv')
# prof = ProfileReport(dataset)
# prof
```

Step-5: Correlation Analysis

In [ ]:
```
#to see the correlation between 2 coulmns, np.corrcoef is used
#here we want to see the correlation between gross income and rating

correlation_matrix = np.corrcoef(df['gross income'], df['Rating'])

# Plot the correlation matrix as a heatmap
plt.imshow(correlation_matrix, cmap='coolwarm', interpolation='nearest')
plt.colorbar()
plt.title('Correlation Matrix between Gross Income and Rating')
plt.xticks([0, 1], ['Gross Income', 'Rating'])
plt.yticks([0, 1], ['Gross Income', 'Rating'])
plt.show()
```



In [ ]:
```
#We had a negative value from the previous cell (-0.036)
#If we want to get that specific number, we need to subset it

correlation_coefficient = np.corrcoef(df['gross income'], df['Rating'])[1][0]
rounded_correlation = round(correlation_coefficient, 2)
rounded_correlation
```

Out[ ]: -0.04

In [ ]:
```
#to see all correlation matrix; it means we find the correlation of each column

correlation_matrix = df.corr()

correlation_matrix
```

Out[ ]:

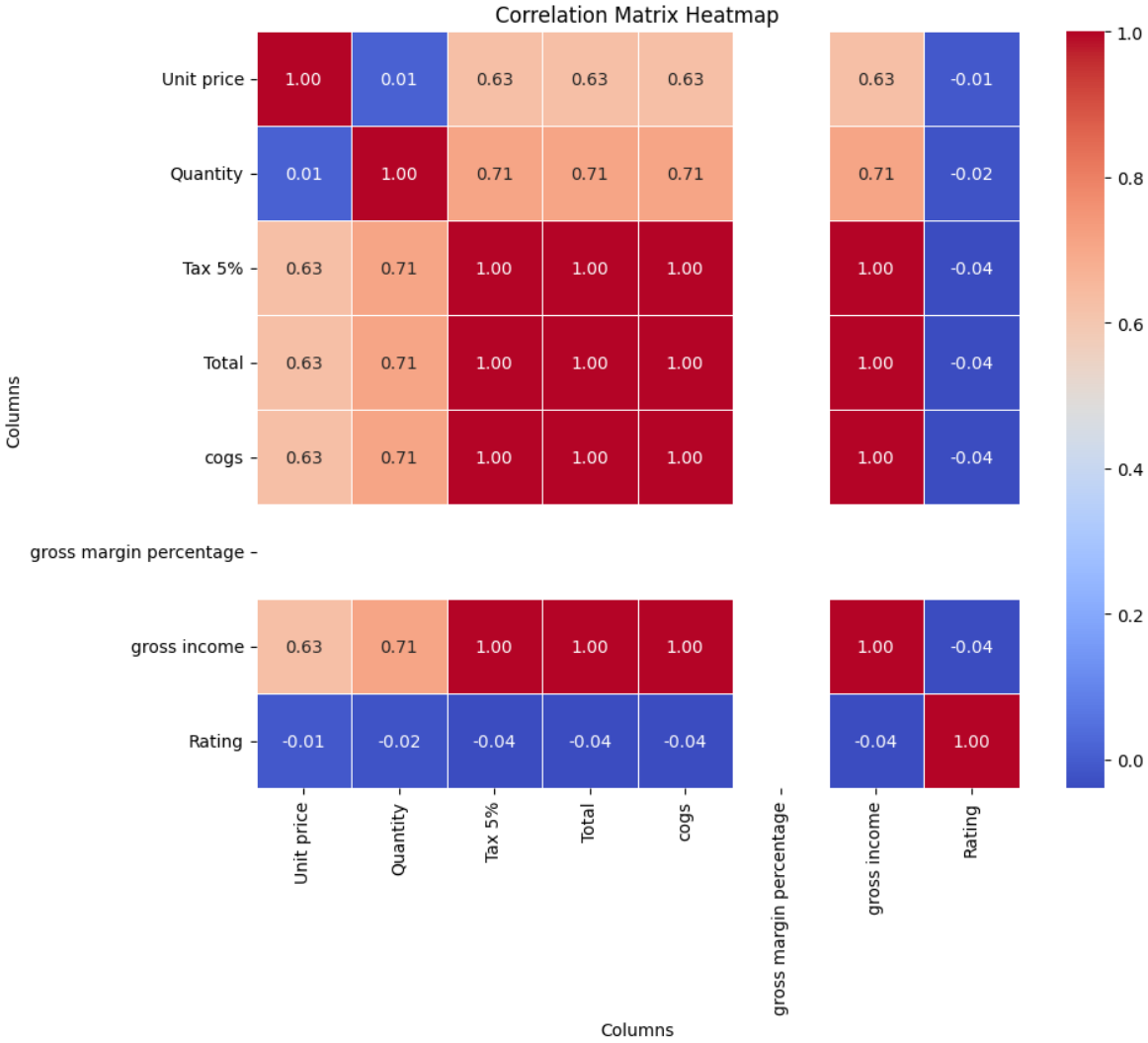
	Unit price	Quantity	Tax 5%	Total	cogs	gross margin percentage	gross income
Unit price	1.000000	0.010778	0.633962	0.633962	0.633962	NaN	0.633962
Quantity	0.010778	1.000000	0.705510	0.705510	0.705510	NaN	0.705510
Tax 5%	0.633962	0.705510	1.000000	1.000000	1.000000	NaN	1.000000
Total	0.633962	0.705510	1.000000	1.000000	1.000000	NaN	1.000000
cogs	0.633962	0.705510	1.000000	1.000000	1.000000	NaN	1.000000
gross margin percentage	NaN	NaN	NaN	NaN	NaN	NaN	NaN
gross income	0.633962	0.705510	1.000000	1.000000	1.000000	NaN	1.000000
Rating	-0.008778	-0.015815	-0.036442	-0.036442	-0.036442	NaN	-0.036442

In [ ]:
```
#We can round the numbers to 2 decimals

np.round(df.corr(),2)
```

Out[ ]:

	Unit price	Quantity	Tax 5%	Total	cogs	gross margin percentage	gross income	Rating
Unit price	1.00	0.01	0.63	0.63	0.63	NaN	0.63	-0.01
Quantity	0.01	1.00	0.71	0.71	0.71	NaN	0.71	-0.02
Tax 5%	0.63	0.71	1.00	1.00	1.00	NaN	1.00	-0.04
Total	0.63	0.71	1.00	1.00	1.00	NaN	1.00	-0.04
cogs	0.63	0.71	1.00	1.00	1.00	NaN	1.00	-0.04
gross margin percentage	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
gross income	0.63	0.71	1.00	1.00	1.00	NaN	1.00	-0.04
Rating	-0.01	-0.02	-0.04	-0.04	-0.04	NaN	-0.04	1.00

In [ ]:
```
#in order to see the previous table visually, we can use heatmap

correlation_matrix = df.corr()

# Create a heatmap using Seaborn
plt.figure(figsize=(10, 8))
sns.heatmap(np.round(correlation_matrix, 2), annot=True, cmap='coolwarm', fmt='.
plt.title('Correlation Matrix Heatmap')
plt.xlabel('Columns')
plt.ylabel('Columns')
plt.show()
```



Many thanks to Sevdasanver and Instructor (Bassim Eledath)