Super Market Exploratory Data Analysis

Goal of Project :

Conduct comprehensive exploratory data analysis (EDA) on supermarket sales data to gain valuable insights into customer purchasing behavior, product performance, and market trends. Through thorough examination and visualization of the dataset, aim to identify patterns, correlations, and anomalies that can inform strategic decision-making processes, optimize inventory management, enhance marketing strategies, and ultimately drive revenue growth for the supermarket.

Source Of Data : Kaggle

Work Flow :

- Step 1 : Importing the required libraries
- Step 2 : Loading the Dataset

- Step 3 : Basic Understanding of Data
- Step 4 : Preprocessing
- Step 5 : Analysis and Insights

Step 1 : Importing the required libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Step 2 : Loading the Dataset

In [2]:
```
df=pd.read_csv("C:\\Users\\Rahul\\Desktop\\data learing folder\\python\\Pandas_fo
```

Step 3 : Basic Understanding of Data

Preview of Data

In [3]:
```
df.sample(3)
```

Out[3]:

	Invoice ID	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%	Total
411	569-71-4390	B	Mandalay	Normal	Male	Sports and travel	21.87	2	2.187	45.927
115	225-98-1496	C	Naypyitaw	Normal	Female	Fashion accessories	27.02	3	4.053	85.113
468	746-19-0921	C	Naypyitaw	Normal	Male	Food and beverages	21.58	1	1.079	22.659

Previewing the data allows us to understand the structure of the dataset, including the number of columns, their names, and the type of data stored in each column.

This understanding helps you plan your analysis effectively and choose appropriate techniques and tools.

How big is the data

In [4]:
```
#checking the shape of the data
df.shape
```

Out[4]:
```
(1000, 17)
```

Observation : This data has 1000 rows and 17 columns

Fetching the column names

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

Out[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')
```

Column Description

- Invoice id: Computer generated sales slip invoice identification number
- Branch: Branch of supercenter (3 branches are available identified by A, B and C).
- City: Location of supercenters
- Customer type: Type of customers, recorded by Members for customers using member card and Normal for without member card.
- Gender: Gender type of customer
- Product line: General item categorization groups - Electronic accessories, Fashion accessories, Food and beverages, Health and beauty, Home and lifestyle, Sports and travel
- Unit price: Price of each product in $
- Quantity: Number of products purchased by customer
- Tax: 5% tax fee for customer buying
- Total: Total price including tax
- Date: Date of purchase (Record available from January 2019 to March 2019)
- Time: Purchase time (10am to 9pm)
- Payment: Payment used by customer for purchase (3 methods are available – Cash, Credit card and Ewallet)

- COGS: Cost of goods sold
- Gross margin percentage: Gross margin percentage
- Gross income: Gross income
- Rating: Customer stratification rating on their overall shopping experience (On a scale of 1 to 10)

Basic information of Data

In [6]:
```
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
```

Observation : From above output we can see all columns are in appropriate dtype except date and time so we need to convert into datetime for better analysis and invoice Id column seems irrelevant w.r.t EDA so we will drop it later.

Step 4 : Preprocessing

Detecting the missing values

In [7]:
```
df.isnull().sum().to_frame().rename(columns={0:"Missing values count "}).T
```

Out[7]:

	Invoice ID	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%	Total	Date
Missing values count	0	0	0	0	0	0	0	0	0	0	0

Observation : Their is no missing values in the data

Detecting the inconsistency of the data

checking inconsistency in object columns

In [8]:
```
df_obj=df.select_dtypes(include=object)
for i in df_obj.columns[1:]:
    print(i,"------------",df[i].unique(),"has",df[i].nunique(),"unique value\n\n
```

```
Branch ------------ ['A' 'C' 'B'] has 3 unique value


City ------------ ['Yangon' 'Naypyitaw' 'Mandalay'] has 3 unique value


Customer type ------------ ['Member' 'Normal'] has 2 unique value


Gender ------------ ['Female' 'Male'] has 2 unique value


Product line ------------ ['Health and beauty' 'Electronic accessories' 'Home and
lifestyle'
 'Sports and travel' 'Food and beverages' 'Fashion accessories'] has 6 unique val
ue


Date ------------ ['1/5/2019' '3/8/2019' '3/3/2019' '1/27/2019' '2/8/2019' '3/25/
2019'
 '2/25/2019' '2/24/2019' '1/10/2019' '2/20/2019' '2/6/2019' '3/9/2019'
 '2/12/2019' '2/7/2019' '3/29/2019' '1/15/2019' '3/11/2019' '1/1/2019'
 '1/21/2019' '3/5/2019' '3/15/2019' '2/17/2019' '3/2/2019' '3/22/2019'
 '3/10/2019' '1/25/2019' '1/28/2019' '1/7/2019' '3/23/2019' '1/17/2019'
 '2/2/2019' '3/4/2019' '3/16/2019' '2/27/2019' '2/10/2019' '3/19/2019'
 '2/3/2019' '3/7/2019' '2/28/2019' '3/27/2019' '1/20/2019' '3/12/2019'
 '2/15/2019' '3/6/2019' '2/14/2019' '3/13/2019' '1/24/2019' '1/6/2019'
 '2/11/2019' '1/22/2019' '1/13/2019' '1/9/2019' '1/12/2019' '1/26/2019'
 '1/23/2019' '2/23/2019' '1/2/2019' '2/9/2019' '3/26/2019' '3/1/2019'
 '2/1/2019' '3/28/2019' '3/24/2019' '2/5/2019' '1/19/2019' '1/16/2019'
 '1/8/2019' '2/18/2019' '1/18/2019' '2/16/2019' '2/22/2019' '1/29/2019'
 '1/4/2019' '3/30/2019' '1/30/2019' '1/3/2019' '3/21/2019' '2/13/2019'
 '1/14/2019' '3/18/2019' '3/20/2019' '2/21/2019' '1/31/2019' '1/11/2019'
 '2/26/2019' '3/17/2019' '3/14/2019' '2/4/2019' '2/19/2019'] has 89 unique value


Time ------------ ['13:08' '10:29' '13:23' '20:33' '10:37' '18:30' '14:36' '11:3
8' '17:15'
 '13:27' '18:07' '17:03' '10:25' '16:48' '19:21' '16:19' '11:03' '10:39'
 '18:00' '15:30' '11:24' '10:40' '12:20' '11:15' '17:36' '19:20' '15:31'
 '12:17' '19:48' '15:36' '19:39' '12:43' '14:49' '10:12' '10:42' '12:28'
 '19:15' '17:17' '13:24' '13:01' '18:45' '10:11' '13:03' '20:39' '19:47'
 '17:24' '15:47' '12:45' '17:08' '10:19' '15:10' '14:42' '15:46' '11:49'
 '19:01' '11:26' '11:28' '15:55' '20:36' '17:47' '10:55' '13:40' '12:27'
 '14:35' '16:40' '15:43' '15:01' '10:04' '18:50' '12:46' '18:17' '18:21'
 '17:04' '14:20' '15:48' '16:24' '18:56' '19:56' '18:37' '10:17' '14:31'
 '10:23' '20:35' '16:57' '17:55' '19:54' '16:42' '12:09' '20:05' '20:38'
 '13:11' '10:16' '18:14' '13:22' '11:27' '16:44' '18:19' '14:50' '20:54'
 '20:19' '10:43' '14:30' '11:32' '10:41' '12:44' '20:07' '20:31' '12:29'
 '15:26' '20:48' '12:02' '17:26' '19:52' '14:57' '18:44' '13:26' '16:17'
 '15:57' '13:18' '20:34' '18:36' '14:40' '16:43' '20:59' '15:39' '12:21'
 '19:25' '13:00' '13:48' '19:57' '10:36' '16:37' '17:11' '15:07' '16:07'
 '11:56' '18:23' '13:05' '19:40' '13:58' '14:43' '19:18' '16:21' '19:44'
 '19:42' '15:24' '14:12' '13:32' '16:20' '16:31' '11:36' '19:17' '17:34'
 '12:04' '17:01' '10:50' '19:16' '16:47' '10:00' '11:51' '15:00' '11:19'
 '19:46' '19:00' '10:53' '12:50' '20:50' '13:41' '19:08' '20:23' '11:30'
 '19:30' '18:03' '10:13' '19:58' '10:01' '11:57' '10:02' '14:51' '12:42'
 '17:38' '20:24' '18:08' '15:53' '15:05' '18:27' '16:55' '12:58' '18:59'
 '13:44' '13:46' '18:06' '12:38' '15:56' '14:29' '19:14' '10:52' '12:55'
 '19:28' '13:52' '10:54' '18:31' '18:24' '18:09' '15:16' '17:07' '19:26'
 '11:20' '16:49' '12:01' '11:25' '18:42' '14:47' '19:43' '14:04' '16:11'
 '19:06' '15:34' '11:22' '11:23' '10:46' '13:25' '14:53' '19:22' '11:00'
 '19:24' '17:22' '20:55' '16:05' '13:34' '18:13' '11:44' '15:51' '16:52'
 '20:52' '16:28' '13:29' '11:09' '15:02' '14:21' '18:01' '13:30' '14:38'
```

```
'17:37' '17:20' '20:29' '11:46' '13:42' '14:44' '14:16' '15:54' '10:21'
'16:46' '20:14' '17:09' '17:43' '19:05' '10:08' '13:12' '20:51' '17:29'
'11:34' '18:58' '20:26' '15:08' '13:21' '12:48' '19:53' '19:09' '16:30'
'13:07' '18:48' '17:27' '15:59' '11:21' '15:49' '13:02' '20:21' '15:04'
'16:10' '12:14' '11:06' '18:22' '19:02' '15:44' '20:01' '13:45' '15:40'
'16:58' '11:12' '15:12' '20:37' '17:44' '16:23' '12:12' '19:33' '14:28'
'17:54' '12:25' '12:52' '19:50' '15:32' '13:19' '13:37' '14:55' '12:31'
'10:26' '20:18' '20:04' '13:38' '17:30' '15:28' '19:07' '18:55' '19:36'
'10:57' '17:13' '13:57' '13:53' '16:53' '16:51' '15:37' '20:15' '19:35'
'15:42' '14:11' '17:58' '11:02' '15:09' '13:47' '16:59' '14:15' '15:19'
'18:33' '12:10' '11:40' '16:54' '15:25' '20:47' '18:20' '11:48' '14:14'
'11:17' '12:40' '17:53' '16:36' '10:48' '18:05' '12:07' '19:49' '15:52'
'20:46' '10:34' '13:55' '11:43' '16:03' '20:03' '19:41' '18:04' '10:31'
'13:28' '17:16' '18:43' '10:30' '20:40' '12:08' '17:45' '10:28' '10:49'
'12:34' '18:51' '19:38' '12:32' '10:33' '19:55' '14:33' '13:54' '12:15'
'12:37' '15:06' '15:58' '14:03' '16:38' '11:07' '12:23' '14:13' '19:11'
'18:53' '14:22' '10:06' '20:08' '12:56' '10:18' '11:45' '16:08' '12:24'
'19:51' '18:10' '15:27' '16:04' '14:41' '14:19' '14:08' '11:29' '12:16'
'20:00' '15:29' '14:58' '11:52' '17:46' '14:45' '11:39' '13:06' '20:43'
'16:34' '13:10' '17:10' '10:22' '19:29' '14:27' '12:22' '11:59' '17:59'
'12:51' '13:56' '19:45' '16:18' '18:57' '11:18' '14:06' '20:13' '15:14'
'16:06' '12:47' '20:42' '20:10' '14:24' '11:42' '17:49' '15:33' '10:38'
'12:39' '14:26' '12:41' '15:20' '16:33' '20:44' '11:16' '12:30' '17:48'
'20:30' '13:59' '11:58' '16:50' '18:02' '17:52' '20:32' '16:09' '11:33'
'15:15' '20:06' '16:26' '18:38' '16:45' '18:41' '17:12' '14:00' '16:32'
'10:10' '10:05' '18:15' '11:01' '15:21' '16:16' '11:05' '19:31' '18:35'
'13:51' '12:35' '11:55' '15:11' '14:48' '12:36' '13:35' '15:45' '14:25'
'15:18' '10:03' '13:14' '16:35' '20:57' '13:50' '17:35' '17:56' '10:44'
'10:09' '10:58' '13:49' '11:10' '13:33' '14:05' '16:27' '15:23' '18:18'
'15:17' '19:12'] has 506 unique value


Payment ------------ ['Ewallet' 'Cash' 'Credit card'] has 3 unique value
```

checking inconsistency in numeric columns

In [9]:
```
df_numeric=df.select_dtypes(exclude=object)
for i in df_numeric.columns:
        print(i,"------------",df[i].unique(),"has",df[i].nunique(),"unique value
```

```
Unit price ------------ [74.69 15.28 46.33 58.22 86.31 85.39 68.84 73.56 36.26 5
4.84 14.48 25.51
 46.95 43.19 71.38 93.72 68.93 72.61 54.67 40.3  86.04 87.98 33.2  34.56
 88.63 52.59 33.52 87.67 88.36 24.89 94.13 78.07 83.78 96.58 99.42 68.12
 62.62 60.88 54.92 30.12 86.72 56.11 69.12 98.7  15.37 93.96 56.69 20.01
 18.93 82.63 91.4  44.59 17.87 15.43 16.16 85.98 44.34 89.6  72.35 30.61
 24.74 55.73 55.07 15.81 75.74 15.87 33.47 97.61 78.77 18.33 89.48 62.12
 48.52 75.91 74.67 41.65 49.04 78.31 20.38 99.19 96.68 19.25 80.36 48.91
 83.06 76.52 49.38 42.47 76.99 47.38 44.86 21.98 64.36 89.75 97.16 87.87
 12.45 52.75 82.7  48.71 78.55 23.07 58.26 30.35 88.67 27.38 62.13 33.98
 81.97 16.49 98.21 72.84 58.07 80.79 27.02 21.94 51.36 10.96 53.44 99.56
 57.12 99.96 63.91 56.47 93.69 32.25 31.73 68.54 90.28 39.62 92.13 34.84
 87.45 81.3  90.22 26.31 34.42 51.91 72.5  89.8  90.5  68.6  30.41 77.95
 46.26 30.14 66.14 71.86 32.46 91.54 83.24 16.48 80.97 92.29 72.17 50.28
 97.22 93.39 43.18 63.69 45.79 76.4  39.9  42.57 95.58 98.98 51.28 69.52
 70.01 80.05 20.85 52.89 19.79 33.84 22.17 22.51 73.88 86.8  64.26 38.47
 15.5  34.31 12.34 18.08 94.49 46.47 74.07 69.81 77.04 73.52 87.8  25.55
 32.71 74.29 43.7  25.29 41.5  71.39 19.15 57.49 61.41 25.9  17.77 23.03
 66.65 28.53 30.37 99.73 26.23 93.26 92.36 46.42 29.61 18.28 24.77 94.64
 94.87 57.34 45.35 62.08 11.81 12.54 43.25 87.16 69.37 37.06 90.7  63.42
 81.37 10.59 84.09 73.82 51.94 93.14 17.41 44.22 13.22 89.69 24.94 59.77
 93.2  62.65 93.87 47.59 81.4  17.94 77.72 73.06 46.55 35.19 14.39 23.75
 58.9  32.62 66.35 25.91 65.94 75.06 16.45 38.3  22.24 54.45 98.4  35.47
 74.6  70.74 35.54 67.43 21.12 21.54 12.03 99.71 47.97 21.82 95.42 70.99
 44.02 69.96 37.   15.34 99.83 47.67 66.68 74.86 48.51 94.88 27.85 62.48
 36.36 18.11 51.92 28.84 78.38 60.01 88.61 99.82 39.01 48.61 51.19 14.96
 72.2  40.23 88.79 26.48 81.91 79.93 69.33 14.23 15.55 78.13 99.37 21.08
 74.79 29.67 44.07 22.93 39.42 15.26 61.77 21.52 97.74 99.78 94.26 51.13
 22.02 32.9  77.02 23.48 14.7  28.45 57.95 47.65 42.82 48.09 55.97 76.9
 97.03 44.65 77.93 71.95 89.25 26.02 13.5  99.3  51.69 54.73 27.   30.24
 89.14 37.55 95.44 27.5  74.97 80.96 94.47 99.79 73.22 41.24 81.68 51.32
 14.36 21.5  26.26 60.96 70.11 42.08 67.09 96.7  35.38 95.49 96.98 23.65
 82.33 26.61 99.69 74.89 40.94 75.82 46.77 32.32 54.07 18.22 80.48 37.95
 76.82 52.26 79.74 77.5  54.27 13.59 41.06 19.24 39.43 46.22 13.98 39.75
 97.79 67.26 13.79 68.71 56.53 23.82 34.21 21.87 20.97 25.84 50.93 96.11
 45.38 81.51 57.22 25.22 38.6  84.05 97.21 25.42 16.28 40.61 53.17 20.87
 67.27 90.65 69.08 43.27 23.46 95.54 47.44 99.24 82.93 33.99 17.04 40.86
 17.44 88.43 89.21 12.78 19.1  27.66 45.74 27.07 39.12 74.71 22.01 63.61
 25.   20.77 29.56 77.4  79.39 46.57 35.89 40.52 73.05 73.95 22.62 51.34
 54.55 37.15 37.02 21.58 98.84 83.77 40.05 43.13 72.57 64.44 65.18 33.26
 84.07 34.37 65.97 32.8  37.14 60.38 36.98 49.49 41.09 22.96 77.68 34.7
 19.66 25.32 12.12 99.89 75.92 63.22 90.24 98.13 51.52 73.97 31.9  69.4
 93.31 88.45 24.18 48.5  61.29 15.95 90.74 42.91 54.28 99.55 58.39 51.47
 54.86 39.39 34.73 71.92 45.71 83.17 37.44 62.87 81.71 91.41 39.21 59.86
 54.36 98.09 25.43 86.68 22.95 16.31 28.32 16.67 73.96 97.94 87.48 30.68
 75.88 20.18 18.77 71.2  38.81 29.42 60.95 51.54 66.06 57.27 54.31 58.24
 22.21 19.32 37.48 72.04 98.52 41.66 72.42 89.2  42.42 74.51 99.25 81.21
 49.33 65.74 79.86 73.98 82.04 26.67 10.13 72.39 85.91 81.31 60.3  31.77
 64.27 69.51 27.22 92.98 63.06 51.71 52.34 43.06 59.61 14.62 46.53 24.24
 45.58 75.2  96.8  14.82 52.2  46.66 36.85 70.32 83.08 64.99 77.56 54.51
 51.89 31.75 53.65 49.79 57.89 28.96 98.97 93.22 80.93 67.45 38.72 72.6
 87.91 98.53 43.46 71.68 91.61 94.59 83.25 91.35 78.88 60.87 82.58 53.3
 12.09 64.19 99.7  79.91 66.47 28.95 46.2  17.63 52.42 98.79 88.55 55.67
 72.52 12.05 19.36 70.21 33.63 15.49 75.66 55.81 72.78 37.32 60.18 15.69
 88.15 27.93 55.45 42.97 17.14 58.75 87.1  98.8  48.63 57.74 17.97 47.71
 40.62 56.04 93.4  73.41 33.64 45.48 64.08 73.47 58.95 39.48 34.81 49.32
 21.48 23.08 49.1  64.83 63.56 72.88 67.1  70.19 55.04 73.38 52.6  87.37
 27.04 62.19 69.58 97.5  60.41 19.77 80.47 88.39 71.77 43.   68.98 15.62
 25.7  80.62 75.53 77.63 13.85 35.68 71.46 11.94 17.48 25.56 90.63 44.12
 36.77 23.34 28.5  55.57 69.74 97.26 52.18 22.32 56.   19.7  53.72 81.95
 81.2  58.76 91.56 55.61 84.83 71.63 37.69 31.67 38.42 65.23 10.53 12.29
 81.23 27.28 17.42 73.28 84.87 97.29 35.74 96.52 18.85 55.39 77.2  72.13
 63.88 10.69 55.5  95.46 76.06 13.69 95.64 11.43 85.87 67.99 65.65 28.86
 65.31 93.38 25.25 21.8  94.76 30.62 44.01 10.16 74.58 71.89 10.99 60.47
```

```
58.91 46.41 68.55 97.37 92.6  46.61 27.18 24.49 92.78 86.69 23.01 30.2
67.39 48.96 75.59 77.47 93.18 50.23 17.75 62.18 10.75 40.26 64.97 95.15
48.62 53.21 45.44 33.88 96.16 47.16 47.68 10.17 60.08 72.11 41.28 64.95
74.22 10.56 62.57 11.85 91.3  40.73 52.38 38.54 44.63 55.87 29.22 39.47
14.87 21.32 93.78 73.26 22.38 99.1  74.1  98.48 53.19 52.79 95.95 36.51
28.31 57.59 47.63 86.27 12.76 11.28 51.07 79.59 33.81 90.53 62.82 24.31
64.59 24.82 56.5  21.43 89.06 23.29 65.26 52.35 90.02 12.1  33.21 10.18
31.99 83.34 87.9  12.19 76.92 83.66 57.91 92.49 28.38 50.45 99.16 60.74
47.27 85.6  35.04 44.84 45.97 27.73 11.53 58.32 84.61 82.88 79.54 49.01
29.15 56.13 93.12 99.6  35.49 42.85 94.67 68.97 35.79 16.37 12.73 83.14
35.22 13.78 88.31 88.25 25.31 99.92 83.35 74.44 63.15 85.72 78.89 92.09
57.29 66.52 45.68 50.79 10.08 93.88 84.25 53.78 35.81 26.43 39.91 21.9
62.85 21.04 65.91 50.49 46.02 15.8  98.66 91.98 20.89 96.82 33.33 38.27
33.3  81.01 34.49 84.63 36.91 87.08 80.08 86.13 49.92 74.66 26.6  25.45
67.77 59.59 58.15 97.48 96.37 63.71 14.76 62.   82.34 75.37 56.56 76.6
58.03 17.49 40.35 97.38 31.84 65.82 88.34] has 943 unique value


Quantity ------------ [ 7  5  8  6 10  2  3  4  1  9] has 10 unique value


Tax 5% ------------ [26.1415  3.82   16.2155 23.288  30.2085 29.8865 20.652  36.7
8    3.626
  8.226   2.896   5.102  11.7375 21.595  35.69   28.116  24.1255 21.783
  8.2005  4.03   21.51   13.197   3.32    8.64   13.2945 21.036   1.676
  8.767  22.09   11.2005 23.5325 35.1315 33.512   9.658  19.884   3.406
 15.655  27.396  21.968  12.048   4.336   5.611  20.736  39.48    1.537
 18.792  25.5105  9.0045  5.679  41.315  31.99   11.1475  3.574   0.7715
  1.616  34.392   4.434  35.84   36.175   9.183   3.711  16.719  24.7815
  7.905  15.148   7.935   3.347  29.283  39.385   0.9165 44.74   31.06
  7.278  22.773  33.6015 20.825  22.068  39.155   5.095  29.757  14.502
  7.7    16.072  12.2275 29.071  19.13   17.283   2.1235 23.097   9.476
 22.43    7.693  28.962   4.4875  4.858  43.935   3.735   7.9125 24.81
  2.4355 35.3475 10.3815 17.478  10.6225 44.335   8.214  18.639  15.291
 40.985   1.649  14.7315 25.494  26.1315 36.3555  4.053   5.485   2.568
  5.48    5.344  39.824  19.992  44.982  25.564  22.588  32.7915  8.0625
 14.2785 27.416  40.626  13.867  27.639   6.968  26.235  24.39   13.533
  6.5775 10.326  25.955  29.     44.9    45.25   34.3     1.5205 23.385
 13.878  15.07   13.228  28.744  12.984  18.308  12.096  37.458   4.944
 32.388  23.0725  3.6085 12.57   43.749  28.017  17.272   3.1845 16.0265
  7.64   19.95   17.028  47.79   49.49   15.384  24.332  17.5025 20.0125
  8.34   15.867   7.916  15.228   8.868   7.8785 22.164  13.02   22.491
 15.388   7.75   13.724   4.319   2.712  37.796   9.294   3.7035 13.962
 11.556   7.352  39.51    5.11    8.1775  3.7145  4.37    1.2645  8.3
 17.8475  5.745  11.498  21.4935 12.95    4.4425 10.3635 29.9925 14.265
  4.5555 44.8785 11.8035 41.967  23.09    6.963   0.914   6.1925 14.196
 37.948   8.601  13.605  21.728   2.9525  0.627   4.325   8.716  31.2165
  7.412  27.21   25.368   8.137   1.5885 37.8405 14.764  25.97    9.314
  4.3525 11.055   3.305   4.4845 11.223   5.977   9.32   12.53   37.548
 19.036  12.21    4.485  15.544  25.571  20.9475 17.595   1.439   4.75
 23.56    6.524   3.3175  7.773   6.45   13.188  33.777   3.29    7.66
 11.12    2.7225 34.44    7.094  37.3    14.148  17.77   16.8575  2.112
  9.693   1.203  29.913  16.7895 10.91   19.084  35.495  22.01   27.984
  1.85    0.767  29.949   9.534  16.67    3.743  10.6875 16.9785 33.208
 20.15    9.7475  3.124   3.636   9.055  12.98    5.768  23.514  12.002
  4.4305  9.982   1.9505  2.4305 10.238   5.984  25.27   14.0805 35.516
  3.972   8.191  23.979   6.933   3.5575  6.9975 39.065   9.937   3.162
 18.6975 10.3845  8.814  10.3185  1.971   4.578  15.4425  6.456  19.548
 24.945  18.852  10.226   7.272   9.909   4.935  19.255   2.348   3.675
  7.1125 34.38   17.385   7.1475 19.269   7.2135 19.5895 26.915  24.2575
  6.6975 35.0685  3.5975 35.7     9.107   6.75   49.65   18.0915 19.1555
 12.15    1.512  17.828  18.775  47.72    4.125   3.7485 32.384  37.788
  9.979  21.966   8.248  16.336  23.094   7.18    9.675   9.191   6.096
 21.033  12.624  16.7725 24.175  15.921  33.4215 19.396   4.73   16.466
```

```
 2.661  24.9225 14.978  10.235   3.791  14.031  16.16   24.3315  6.377
12.072  18.975   3.841  26.13    3.987  19.375  13.5675  6.1155 12.318
 8.658  11.829   9.244   0.699   9.9375 34.2265 13.452   3.4475 13.742
11.306   5.955  17.105   2.187   5.2425  3.876  20.372   4.8055  9.076
 4.0755  5.722   8.827   5.79   12.6075 48.605  10.168   0.814  18.2745
18.6095  3.1305 16.8175 45.325   6.908   4.327   7.038  33.439   2.372
44.658  16.586  10.197   3.408  16.344   4.36   35.372  40.1445  0.639
 6.685   0.9575 13.83    6.861   1.3535  1.956  22.413   6.603  15.9025
 1.25    4.154   7.39   34.83   39.695  23.285   1.7945 10.13   36.525
14.79    1.131  12.835  27.275  13.0025 11.106   1.079   4.942  25.131
 8.01   21.565  29.028  16.11    9.777   8.315  16.814  17.185   1.93
26.388  16.4     9.285  30.19   18.49    9.898  20.545   7.43    1.148
34.956   3.47    9.83   10.128   6.06    9.989  30.368   6.322  27.072
 4.9065 20.608   3.6985  1.595   6.94    9.331   4.4225  9.672   7.275
25.215  15.3225  4.785  31.759  10.7275 18.998  34.8425 20.4365  2.5735
13.715   9.8475  3.473  17.98    6.8565 24.951  11.232   6.287  24.513
22.8525  7.842   5.986  27.18   44.1405  7.629  34.672  11.475   7.3395
 7.08    5.8345  3.698   4.897  14.61   26.244   4.602   3.794   4.036
 5.631   3.56    7.762  14.71   27.4275 12.885  19.818   8.5905 24.4395
26.208   6.663   6.762   5.622   7.204  49.26   12.498  10.863   9.711
44.6    16.968  22.353   9.925  40.605  24.665  29.583  27.951  25.893
20.51   13.335   3.5455  7.239  21.4775 28.4585 12.06    6.354  12.854
 6.951   4.083  15.536   9.298   3.616   9.459  10.342   7.851  10.765
29.805   3.655  13.959   8.484   2.279  11.28   14.52    2.223   7.83
20.997   9.2125  7.032   3.2495 38.78   16.353  18.1615  6.35   18.7775
 9.958   1.5305  5.789   1.448  44.5365 13.983   4.0465 33.725  17.424
21.78   21.9775 29.559  13.038  10.752   4.5805 33.1065 41.625   4.5675
 7.888   6.087  41.29    7.995   0.6045 32.095  11.7465  8.377  14.955
11.9865 33.235  10.1325  2.31    4.4075  7.863  14.8185 35.42    5.567
29.008   3.0125  8.712  21.063   1.6815  1.549  12.37   18.915  16.743
36.39   16.794  12.036   2.3535  4.9845 13.2225  6.9825  2.7725  6.4455
 5.999  17.625  43.55    9.88    9.726   8.661   3.594  14.313   4.062
28.02    9.34   11.0115 13.456  22.74   22.428  14.694  29.475  14.55
 1.974   1.7405 14.796   2.148   6.924   4.91    6.483  31.78    7.288
10.065  31.5855 19.264  24.315  25.683  23.67   21.8425  5.408  12.438
31.311  48.75   24.164   4.848   9.885  36.2115 39.7755 25.1195  8.6
 3.449   6.248   3.855  24.186  15.106  34.9335  6.2325  8.92   25.011
 1.791   6.807   5.244   8.946  40.7835  6.618  12.8695  4.668  11.4
 8.3355 34.87   19.452  18.263   4.464   8.4     0.985  26.558   2.686
40.975  28.42   29.38   36.624  42.282  19.4635  4.2415  7.163   3.769
12.668   1.921  32.615   2.6325  5.5305 28.4305  6.82    8.71   18.32
12.7305 38.916  14.296  28.956   9.425  11.078  38.6    36.065  25.552
 2.6725 11.1    38.184  11.409   4.107  19.128   3.429  19.108  30.0545
23.7965  2.621   6.565   7.215  22.8585  4.669   6.3125 39.5415  8.72
18.952   1.531  17.604   2.54   26.103  28.756   2.7475  9.0705 20.6185
 2.3205 13.71   48.685  32.41    4.661   2.718   3.0435 12.245   4.639
21.6725  6.903  12.08   23.5865 22.032  34.0155 15.494   9.318  10.046
 0.8875 31.09    4.3    20.13   16.2425  4.7575 19.448  21.284  15.904
13.552  19.232  11.79   10.578   4.768   0.5085 10.3065 21.028   4.402
32.4495  6.192  32.475  37.11    4.224  12.514   4.74    4.565  14.2555
 2.619   9.635  13.389  27.935   8.766   7.791   3.015   3.947   1.487
 1.066  14.067   3.663   1.119  32.796  29.73    3.705   9.848  18.6165
26.395  23.9875 16.4295  8.448   5.662  17.277  21.4335  4.3135  1.276
 5.076  17.8745 11.9385  5.0715 36.212   6.282   3.6465 12.918   8.687
 2.825  10.715  26.718   4.658  26.104   2.6175  1.9875 36.008   4.84
16.605   4.072  15.995   8.334  15.953   4.395  36.735   4.876  38.46
20.915  23.164  23.1225  7.095  15.135  39.664  21.259  14.181  29.96
15.768  20.178   9.194   6.9325  4.0355  5.832  15.676  42.305  20.72
 7.954  24.505   4.3725 11.226  37.248  20.536  14.94   10.647   2.1425
18.934  10.3455  3.939  16.1055  4.911   1.273  29.099  10.566   2.756
 4.4155 17.829  39.7125  2.531  29.976   8.335  37.22   18.945  12.858
27.6115 22.37   13.8135 17.187  13.304  44.919  22.84   12.6975  3.528
32.858   8.425   2.689   8.9525 10.572   5.9865  3.285   4.208  19.773
14.8995 22.7205 13.806   7.9    44.397   4.599   2.089   0.775  14.523
```

```
  3.333   3.827  14.985  12.1515  2.37    8.6225 42.315  12.9185 30.478
 12.012   8.613   4.992  14.932   7.98    1.2725  3.3885 11.918  11.63
 43.866  34.986  33.7295 15.9275  1.476  24.8    41.17   30.148  14.14
 38.3     5.803   8.745   3.0475  2.0175 48.69    1.592   3.291  30.919 ] has 990
unique value


Total ------------ [ 548.9715   80.22    340.5255  489.048   634.3785  627.6165
433.692
  772.38    76.146   172.746    60.816   107.142   246.4875  453.495
  749.49   590.436   506.6355  457.443   172.2105   84.63    451.71
  277.137   69.72    181.44    279.1845  441.756    35.196   184.107
  463.89   235.2105  494.1825  737.7615  703.752   202.818   417.564
   71.526  328.755   575.316   461.328   253.008    91.056   117.831
  435.456  829.08     32.277   394.632   535.7205  189.0945  119.259
  867.615  671.79    234.0975   75.054    16.2015   33.936   722.232
   93.114  752.64    759.675   192.843    77.931   351.099   520.4115
  166.005  318.108   166.635    70.287   614.943   827.085    19.2465
  939.54   652.26    152.838   478.233   705.6315  437.325   463.428
  822.255  106.995   624.897   304.542   161.7     337.512   256.7775
  610.491  401.73    362.943    44.5935  485.037   198.996   471.03
  161.553  608.202    94.2375  102.018   922.635    78.435   166.1625
  521.01    51.1455  742.2975  218.0115  367.038   223.0725  931.035
  172.494  391.419   321.111   860.685    34.629   309.3615  535.374
  548.7615 763.4655   85.113   115.185    53.928   115.08    112.224
  836.304  419.832   944.622   536.844   474.348   688.6215  169.3125
  299.8485 575.736   853.146   291.207   580.419   146.328   550.935
  512.19   284.193   138.1275  216.846   545.055   609.      942.9
  950.25   720.3      31.9305  491.085   291.438   316.47    277.788
  603.624  272.664   384.468   254.016   786.618   103.824   680.148
  484.5225  75.7785  263.97    918.729   588.357   362.712    66.8745
  336.5565 160.44    418.95    357.588  1003.59   1039.29    323.064
  510.972  367.5525  420.2625  175.14    333.207   166.236   319.788
  186.228  165.4485  465.444   273.42    472.311   323.148   162.75
  288.204   90.699    56.952   793.716   195.174    77.7735  293.202
  242.676  154.392   829.71    107.31    171.7275   78.0045   91.77
   26.5545 174.3     374.7975  120.645   241.458   451.3635  271.95
   93.2925 217.6335  629.8425  299.565    95.6655  942.4485  247.8735
  881.307  484.89    146.223    19.194   130.0425  298.116   796.908
  180.621  285.705   456.288    62.0025   13.167    90.825   183.036
  655.5465 155.652   571.41    532.728   170.877    33.3585  794.6505
  310.044  545.37    195.594    91.4025  232.155    69.405    94.1745
  235.683  125.517   195.72    263.13    788.508   399.756   256.41
   94.185  326.424   536.991   439.8975  369.495    30.219    99.75
  494.76   137.004    69.6675  163.233   135.45    276.948   709.317
   69.09   160.86    233.52     57.1725  723.24    148.974   783.3
  297.108  373.17    354.0075   44.352   203.553    25.263   628.173
  352.5795 229.11    400.764   745.395   462.21    587.664    38.85
   16.107  628.929   200.214   350.07     78.603   224.4375  356.5485
  697.368  423.15    204.6975   65.604    76.356   190.155   272.58
  121.128  493.794   252.042    93.0405  209.622    40.9605   51.0405
  214.998  125.664   530.67    295.6905  745.836    83.412   172.011
  503.559  145.593    74.7075  146.9475  820.365   208.677    66.402
  392.6475 218.0745  185.094   216.6885   41.391    96.138   324.2925
  135.576  410.508   523.845   395.892   214.746   152.712   208.089
  103.635  404.355    49.308    77.175   149.3625  721.98    365.085
  150.0975 404.649   151.4835  411.3795  565.215   509.4075  140.6475
  736.4385  75.5475  749.7     191.247   141.75   1042.65    379.9215
  402.2655 255.15     31.752   374.388   394.275  1002.12     86.625
   78.7185 680.064   793.548   209.559   461.286   173.208   343.056
  484.974  150.78    203.175   193.011   128.016   441.693   265.104
  352.2225 507.675   334.341   701.8515  407.316    99.33    345.786
   55.881  523.3725  314.538   214.935    79.611   294.651   339.36
  510.9615 133.917   253.512   398.475    80.661   548.73     83.727
```

```
406.875   284.9175  128.4255  258.678   181.818   248.409   194.124
 14.679   208.6875  718.7565  282.492    72.3975  288.582   237.426
125.055   359.205    45.927   110.0925   81.396   427.812   100.9155
190.596    85.5855  120.162   185.367   121.59    264.7575 1020.705
213.528    17.094   383.7645  390.7995   65.7405  353.1675  951.825
145.068    90.867   147.798   702.219    49.812   937.818   348.306
214.137    71.568   343.224    91.56    742.812   843.0345   13.419
140.385    20.1075  290.43    144.081    28.4235   41.076   470.673
138.663   333.9525   26.25     87.234   155.19    731.43    833.595
488.985    37.6845  212.73    767.025   310.59     23.751   269.535
572.775   273.0525  233.226    22.659   103.782   527.751   168.21
452.865   609.588   338.31    205.317   174.615   353.094   360.885
 40.53    554.148   344.4     194.985   633.99    388.29    207.858
431.445   156.03     24.108   734.076    72.87    206.43    212.688
127.26    209.769   637.728   132.762   568.512   103.0365  432.768
 77.6685   33.495   145.74    195.951    92.8725  203.112   152.775
529.515   321.7725  100.485   666.939   225.2775  398.958   731.6925
429.1665   54.0435  288.015   206.7975   72.933   377.58    143.9865
523.971   235.872   132.027   514.773   479.9025  164.682   125.706
570.78    926.9505  160.209   728.112   240.975   154.1295  148.68
122.5245   77.658   102.837   306.81    551.124    96.642    79.674
 84.756   118.251    74.76    163.002   308.91    575.9775  270.585
416.178   180.4005  513.2295  550.368   139.923   142.002   118.062
151.284  1034.46    262.458   228.123   203.931   936.6     356.328
469.413   208.425   852.705   517.965   621.243   586.971   543.753
430.71    280.035    74.4555  152.019   451.0275  597.6285  253.26
133.434   269.934   145.971    85.743   326.256   195.258    75.936
198.639   217.182   164.871   226.065   625.905    76.755   293.139
178.164    47.859   236.88    304.92     46.683   164.43    440.937
193.4625  147.672    68.2395  814.38    343.413   381.3915  133.35
394.3275  209.118    32.1405  121.569    30.408   935.2665  293.643
 84.9765  708.225   365.904   457.38    461.5275  620.739   273.798
225.792    96.1905  695.2365  874.125    95.9175  165.648   127.827
867.09    167.895    12.6945  673.995   246.6765  175.917   314.055
251.7165  697.935   212.7825   48.51     92.5575  165.123   311.1885
743.82    116.907   609.168    63.2625  182.952   442.323    35.3115
 32.529   259.77    397.215   351.603   764.19    352.674   252.756
 49.4235  104.6745  277.6725  146.6325   58.2225  135.3555  125.979
370.125   914.55    207.48    204.246   181.881    75.474   300.573
 85.302   588.42    196.14    231.2415  282.576   477.54    470.988
308.574   618.975   305.55     41.454    36.5505  310.716    45.108
145.404   103.11    136.143   667.38    153.048   211.365   663.2955
404.544   510.615   539.343   497.07    458.6925  113.568   261.198
657.531  1023.75    507.444   101.808   207.585   760.4415  835.2855
527.5095  180.6      72.429   131.208    80.955   507.906   317.226
733.6035  130.8825  187.32    525.231    37.611   142.947   110.124
187.866   856.4535  138.978   270.2595   98.028   239.4     175.0455
732.27    408.492   383.523    93.744   176.4      20.685   557.718
 56.406   860.475   596.82    616.98    769.104   887.922   408.7335
 89.0715  150.423    79.149   266.028    40.341   684.915    55.2825
116.1405  597.0405  143.22    182.91    384.72    267.3405  817.236
300.216   608.076   197.925   232.638   810.6     757.365   536.592
 56.1225  233.1     801.864   239.589    86.247   401.688    72.009
401.268   631.1445  499.7265   55.041   137.865   151.515   480.0285
 98.049   132.5625  830.3715  183.12    397.992    32.151   369.684
 53.34    548.163   603.876    57.6975  190.4805  432.9885   48.7305
287.91   1022.385   680.61     97.881    57.078    63.9135  257.145
 97.419   455.1225  144.963   253.68    495.3165  462.672   714.3255
325.374   195.678   210.966    18.6375  652.89     90.3     422.73
341.0925   99.9075  408.408   446.964   333.984   284.592   403.872
247.59    222.138   100.128    10.6785  216.4365  441.588    92.442
681.4395  130.032   681.975   779.31     88.704   262.794    99.54
 95.865   299.3655   54.999   202.335   281.169   586.635   184.086
163.611    63.315    82.887    31.227    22.386   295.407    76.923
```

```
  23.499   688.716   624.33    77.805   206.808   390.9465  554.295
 503.7375  345.0195  177.408  118.902   362.817   450.1035   90.5835
  26.796   106.596   375.3645 250.7085  106.5015  760.452   131.922
  76.5765  271.278   182.427   59.325   225.015   561.078    97.818
 548.184    54.9675   41.7375 756.168   101.64    348.705    85.512
 335.895   175.014   335.013   92.295   771.435   102.396   807.66
 439.215   486.444   485.5725 148.995   317.835   832.944   446.439
 297.801   629.16    331.128  423.738   193.074   145.5825   84.7455
 122.472   329.196   888.405  435.12    167.034   514.605    91.8225
 235.746   782.208   431.256  313.74    223.587    44.9925  397.614
 217.2555   82.719   338.2155 103.131    26.733   611.079   221.886
  57.876    92.7255  374.409  833.9625   53.151   629.496   175.035
 781.62    397.845   270.018  579.8415  469.77    290.0835  360.927
 279.384   943.299   479.64   266.6475   74.088   690.018   176.925
  56.469   188.0025  222.012  125.7165   68.985    88.368   415.233
 312.8895  477.1305  289.926  165.9     932.337    96.579    43.869
  16.275   304.983    69.993   80.367   314.685   255.1815   49.77
 181.0725  888.615   271.2885 640.038   252.252   180.873   104.832
 313.572   167.58     26.7225  71.1585  250.278   244.23    921.186
 734.706   708.3195  334.4775  30.996   520.8     864.57    633.108
 296.94    804.3     121.863  183.645    63.9975   42.3675 1022.49
  33.432    69.111   649.299 ] has 990 unique value


cogs ------------ [522.83  76.4  324.31 465.76 604.17 597.73 413.04 735.6   72.52
164.52
  57.92 102.04 234.75 431.9  713.8  562.32 482.51 435.66 164.01  80.6
 430.2  263.94  66.4  172.8  265.89 420.72  33.52 175.34 441.8  224.01
 470.65 702.63 670.24 193.16 397.68  68.12 313.1  547.92 439.36 240.96
  86.72 112.22 414.72 789.6   30.74 375.84 510.21 180.09 113.58 826.3
 639.8  222.95  71.48  15.43  32.32 687.84  88.68 716.8  723.5  183.66
  74.22 334.38 495.63 158.1  302.96 158.7   66.94 585.66 787.7   18.33
 894.8  621.2  145.56 455.46 672.03 416.5  441.36 783.1  101.9  595.14
 290.04 154.   321.44 244.55 581.42 382.6  345.66  42.47 461.94 189.52
 448.6  153.86 579.24  89.75  97.16 878.7   74.7  158.25 496.2   48.71
 706.95 207.63 349.56 212.45 886.7  164.28 372.78 305.82 819.7   32.98
 294.63 509.88 522.63 727.11  81.06 109.7   51.36 109.6  106.88 796.48
 399.84 899.64 511.28 451.76 655.83 161.25 285.57 548.32 812.52 277.34
 552.78 139.36 524.7  487.8  270.66 131.55 206.52 519.1  580.   898.
 905.   686.    30.41 467.7  277.56 301.4  264.56 574.88 259.68 366.16
 241.92 749.16  98.88 647.76 461.45  72.17 251.4  874.98 560.34 345.44
  63.69 320.53 152.8  399.   340.56 955.8  989.8  307.68 486.64 350.05
 400.25 166.8  317.34 158.32 304.56 177.36 157.57 443.28 260.4  449.82
 307.76 155.   274.48  86.38  54.24 755.92 185.88  74.07 279.24 231.12
 147.04 790.2  102.2  163.55  74.29  87.4   25.29 166.   356.95 114.9
 229.96 429.87 259.    88.85 207.27 599.85 285.3   91.11 897.57 236.07
 839.34 461.8  139.26  18.28 123.85 283.92 758.96 172.02 272.1  434.56
  59.05  12.54  86.5  174.32 624.33 148.24 544.2  507.36 162.74  31.77
 756.81 295.28 519.4  186.28  87.05 221.1   66.1   89.69 224.46 119.54
 186.4  250.6  750.96 380.72 244.2   89.7  310.88 511.42 418.95 351.9
  28.78  95.   471.2  130.48  66.35 155.46 129.   263.76 675.54  65.8
 153.2  222.4   54.45 688.8  141.88 746.   282.96 355.4  337.15  42.24
 193.86  24.06 598.26 335.79 218.2  381.68 709.9  440.2  559.68  37.
  15.34 598.98 190.68 333.4   74.86 213.75 339.57 664.16 403.   194.95
  62.48  72.72 181.1  259.6  115.36 470.28 240.04  88.61 199.64  39.01
  48.61 204.76 119.68 505.4  281.61 710.32  79.44 163.82 479.58 138.66
  71.15 139.95 781.3  198.74  63.24 373.95 207.69 176.28 206.37  39.42
  91.56 308.85 129.12 390.96 498.9  377.04 204.52 145.44 198.18  98.7
 385.1   46.96  73.5  142.25 687.6  347.7  142.95 385.38 144.27 391.79
 538.3  485.15 133.95 701.37  71.95 714.   182.14 135.   993.   361.83
 383.11 243.    30.24 356.56 375.5  954.4   82.5   74.97 647.68 755.76
 199.58 439.32 164.96 326.72 461.88 143.6  193.5  183.82 121.92 420.66
 252.48 335.45 483.5  318.42 668.43 387.92  94.6  329.32  53.22 498.45
 299.56 204.7   75.82 280.62 323.2  486.63 127.54 241.44 379.5   76.82
```

```
 522.6   79.74 387.5  271.35 122.31 246.36 173.16 236.58 184.88  13.98
198.75 684.53 269.04  68.95 274.84 226.12 119.1  342.1   43.74 104.85
 77.52 407.44  96.11 181.52  81.51 114.44 176.54 115.8  252.15 972.1
203.36  16.28 365.49 372.19  62.61 336.35 906.5  138.16  86.54 140.76
668.78  47.44 893.16 331.72 203.94  68.16 326.88  87.2  707.44 802.89
 12.78 133.7   19.15 276.6  137.22  27.07  39.12 448.26 132.06 318.05
 25.    83.08 147.8  696.6  793.9  465.7   35.89 202.6  730.5  295.8
 22.62 256.7  545.5  260.05 222.12  21.58  98.84 502.62 160.2  431.3
580.56 322.2  195.54 166.3  336.28 343.7   38.6  527.76 328.   185.7
603.8  369.8  197.96 410.9  148.6   22.96 699.12  69.4  196.6  202.56
121.2  199.78 607.36 126.44 541.44  98.13 412.16  73.97  31.9  138.8
186.62  88.45 193.44 145.5  504.3  306.45  95.7  635.18 214.55 379.96
696.85 408.73  51.47 274.3  196.95  69.46 359.6  137.13 499.02 224.64
125.74 490.26 457.05 156.84 119.72 543.6  882.81 152.58 693.44 229.5
146.79 141.6  116.69  73.96  97.94 292.2  524.88  92.04  75.88  80.72
112.62  71.2  155.24 294.2  548.55 257.7  396.36 171.81 488.79 524.16
133.26 135.24 112.44 144.08 985.2  249.96 217.26 194.22 892.   339.36
447.06 198.5  812.1  493.3  591.66 559.02 517.86 410.2  266.7   70.91
144.78 429.55 569.17 241.2  127.08 257.08 139.02  81.66 310.72 185.96
 72.32 189.18 206.84 157.02 215.3  596.1   73.1  279.18 169.68  45.58
225.6  290.4   44.46 156.6  419.94 184.25 140.64  64.99 775.6  327.06
363.23 127.   375.55 199.16  30.61 115.78  28.96 890.73 279.66  80.93
674.5  348.48 435.6  439.55 591.18 260.76 215.04  91.61 662.13 832.5
 91.35 157.76 121.74 825.8  159.9   12.09 641.9  234.93 167.54 299.1
239.73 664.7  202.65  46.2   88.15 157.26 296.37 708.4  111.34 580.16
 60.25 174.24 421.26  33.63  30.98 247.4  378.3  334.86 727.8  335.88
240.72  47.07  99.69 264.45 139.65  55.45 128.91 119.98 352.5  871.
197.6  194.52 173.22  71.88 286.26  81.24 560.4  186.8  220.23 269.12
454.8  448.56 293.88 589.5  291.    39.48  34.81 295.92  42.96 138.48
 98.2  129.66 635.6  145.76 201.3  631.71 385.28 486.3  513.66 473.4
436.85 108.16 248.76 626.22 975.   483.28  96.96 197.7  724.23 795.51
502.39 172.    68.98 124.96  77.1  483.72 302.12 698.67 124.65 178.4
500.22  35.82 136.14 104.88 178.92 815.67 132.36 257.39  93.36 228.
166.71 697.4  389.04 365.26  89.28 168.    19.7  531.16  53.72 819.5
568.4  587.6  732.48 845.64 389.27  84.83 143.26  75.38 253.36  38.42
652.3   52.65 110.61 568.61 136.4  174.2  366.4  254.61 778.32 285.92
579.12 188.5  221.56 772.   721.3  511.04  53.45 222.   763.68 228.18
 82.14 382.56  68.58 382.16 601.09 475.93  52.42 131.3  144.3  457.17
 93.38 126.25 790.83 174.4  379.04  30.62 352.08  50.8  522.06 575.12
 54.95 181.41 412.37  46.41 274.2  973.7  648.2   93.22  54.36  60.87
244.9   92.78 433.45 138.06 241.6  471.73 440.64 680.31 309.88 186.36
200.92  17.75 621.8   86.   402.6  324.85  95.15 388.96 425.68 318.08
271.04 384.64 235.8  211.56  95.36  10.17 206.13 420.56  88.04 648.99
123.84 649.5  742.2   84.48 250.28  94.8   91.3  285.11  52.38 192.7
267.78 558.7  175.32 155.82  60.3   78.94  29.74  21.32 281.34  73.26
 22.38 655.92 594.6   74.1  196.96 372.33 527.9  479.75 328.59 168.96
113.24 345.54 428.67  86.27  25.52 101.52 357.49 238.77 101.43 724.24
125.64  72.93 258.36 173.74  56.5  214.3  534.36  93.16 522.08  52.35
 39.75 720.16  96.8  332.1   81.44 319.9  166.68 319.06  87.9  734.7
 97.52 769.2  418.3  463.28 462.45 141.9  302.7  793.28 425.18 283.62
599.2  315.36 403.56 183.88 138.65  80.71 116.64 313.52 846.1  414.4
159.08 490.1   87.45 224.52 744.96 410.72 298.8  212.94  42.85 378.68
206.91  78.78 322.11  98.22  25.46 581.98 211.32  55.12  88.31 356.58
794.25  50.62 599.52 166.7  744.4  378.9  257.16 552.23 447.4  276.27
343.74 266.08 898.38 456.8  253.95  70.56 657.16 168.5   53.78 179.05
211.44 119.73  65.7   84.16 395.46 297.99 454.41 276.12 158.   887.94
 91.98  41.78  15.5  290.46  66.66  76.54 299.7  243.03  47.4  172.45
846.3  258.37 609.56 240.24 172.26  99.84 298.64 159.6   25.45  67.77
238.36 232.6  877.32 699.72 674.59 318.55  29.52 496.   823.4  602.96
 282.8  766.   116.06 174.9   60.95  40.35 973.8   31.84  65.82 618.38] has 990 u
nique value


gross margin percentage ------------ [4.76190476] has 1 unique value
```

```
gross income ------------ [26.1415  3.82   16.2155 23.288  30.2085 29.8865 20.652
36.78    3.626
  8.226   2.896   5.102  11.7375 21.595  35.69   28.116  24.1255 21.783
  8.2005  4.03   21.51   13.197   3.32    8.64   13.2945 21.036   1.676
  8.767  22.09   11.2005 23.5325 35.1315 33.512   9.658  19.884   3.406
 15.655  27.396  21.968  12.048   4.336   5.611  20.736  39.48    1.537
 18.792  25.5105  9.0045  5.679  41.315  31.99   11.1475  3.574   0.7715
  1.616  34.392   4.434  35.84   36.175   9.183   3.711  16.719  24.7815
  7.905  15.148   7.935   3.347  29.283  39.385   0.9165 44.74   31.06
  7.278  22.773  33.6015 20.825  22.068  39.155   5.095  29.757  14.502
  7.7    16.072  12.2275 29.071  19.13   17.283   2.1235 23.097   9.476
 22.43    7.693  28.962   4.4875  4.858  43.935   3.735   7.9125 24.81
  2.4355 35.3475 10.3815 17.478  10.6225 44.335   8.214  18.639  15.291
 40.985   1.649  14.7315 25.494  26.1315 36.3555  4.053   5.485   2.568
  5.48    5.344  39.824  19.992  44.982  25.564  22.588  32.7915  8.0625
 14.2785 27.416  40.626  13.867  27.639   6.968  26.235  24.39   13.533
  6.5775 10.326  25.955  29.     44.9    45.25   34.3     1.5205 23.385
 13.878  15.07   13.228  28.744  12.984  18.308  12.096  37.458   4.944
 32.388  23.0725  3.6085 12.57   43.749  28.017  17.272   3.1845 16.0265
  7.64   19.95   17.028  47.79   49.49   15.384  24.332  17.5025 20.0125
  8.34   15.867   7.916  15.228   8.868   7.8785 22.164  13.02   22.491
 15.388   7.75   13.724   4.319   2.712  37.796   9.294   3.7035 13.962
 11.556   7.352  39.51    5.11    8.1775  3.7145  4.37    1.2645  8.3
 17.8475  5.745  11.498  21.4935 12.95    4.4425 10.3635 29.9925 14.265
  4.5555 44.8785 11.8035 41.967  23.09    6.963   0.914   6.1925 14.196
 37.948   8.601  13.605  21.728   2.9525  0.627   4.325   8.716  31.2165
  7.412  27.21   25.368   8.137   1.5885 37.8405 14.764  25.97    9.314
  4.3525 11.055   3.305   4.4845 11.223   5.977   9.32   12.53   37.548
 19.036  12.21    4.485  15.544  25.571  20.9475 17.595   1.439   4.75
 23.56    6.524   3.3175  7.773   6.45   13.188  33.777   3.29    7.66
 11.12    2.7225 34.44    7.094  37.3    14.148  17.77   16.8575  2.112
  9.693   1.203  29.913  16.7895 10.91   19.084  35.495  22.01   27.984
  1.85    0.767  29.949   9.534  16.67    3.743  10.6875 16.9785 33.208
 20.15    9.7475  3.124   3.636   9.055  12.98    5.768  23.514  12.002
  4.4305  9.982   1.9505  2.4305 10.238   5.984  25.27   14.0805 35.516
  3.972   8.191  23.979   6.933   3.5575  6.9975 39.065   9.937   3.162
 18.6975 10.3845  8.814  10.3185  1.971   4.578  15.4425  6.456  19.548
 24.945  18.852  10.226   7.272   9.909   4.935  19.255   2.348   3.675
  7.1125 34.38   17.385   7.1475 19.269   7.2135 19.5895 26.915  24.2575
  6.6975 35.0685  3.5975 35.7     9.107   6.75   49.65   18.0915 19.1555
 12.15    1.512  17.828  18.775  47.72    4.125   3.7485 32.384  37.788
  9.979  21.966   8.248  16.336  23.094   7.18    9.675   9.191   6.096
 21.033  12.624  16.7725 24.175  15.921  33.4215 19.396   4.73   16.466
  2.661  24.9225 14.978  10.235   3.791  14.031  16.16   24.3315  6.377
 12.072  18.975   3.841  26.13    3.987  19.375  13.5675  6.1155 12.318
  8.658  11.829   9.244   0.699   9.9375 34.2265 13.452   3.4475 13.742
 11.306   5.955  17.105   2.187   5.2425  3.876  20.372   4.8055  9.076
  4.0755  5.722   8.827   5.79   12.6075 48.605  10.168   0.814  18.2745
 18.6095  3.1305 16.8175 45.325   6.908   4.327   7.038  33.439   2.372
 44.658  16.586  10.197   3.408  16.344   4.36   35.372  40.1445  0.639
  6.685   0.9575 13.83    6.861   1.3535  1.956  22.413   6.603  15.9025
  1.25    4.154   7.39   34.83   39.695  23.285   1.7945 10.13   36.525
 14.79    1.131  12.835  27.275  13.0025 11.106   1.079   4.942  25.131
  8.01   21.565  29.028  16.11    9.777   8.315  16.814  17.185   1.93
 26.388  16.4     9.285  30.19   18.49    9.898  20.545   7.43    1.148
 34.956   3.47    9.83   10.128   6.06    9.989  30.368   6.322  27.072
  4.9065 20.608   3.6985  1.595   6.94    9.331   4.4225  9.672   7.275
 25.215  15.3225  4.785  31.759  10.7275 18.998  34.8425 20.4365  2.5735
 13.715   9.8475  3.473  17.98    6.8565 24.951  11.232   6.287  24.513
 22.8525  7.842   5.986  27.18   44.1405  7.629  34.672  11.475   7.3395
  7.08    5.8345  3.698   4.897  14.61   26.244   4.602   3.794   4.036
  5.631   3.56    7.762  14.71   27.4275 12.885  19.818   8.5905 24.4395
```

```
 26.208   6.663   6.762   5.622   7.204  49.26   12.498  10.863   9.711
 44.6    16.968  22.353   9.925  40.605  24.665  29.583  27.951  25.893
 20.51   13.335   3.5455  7.239  21.4775 28.4585 12.06    6.354  12.854
  6.951   4.083  15.536   9.298   3.616   9.459  10.342   7.851  10.765
 29.805   3.655  13.959   8.484   2.279  11.28   14.52    2.223   7.83
 20.997   9.2125  7.032   3.2495 38.78   16.353  18.1615  6.35   18.7775
  9.958   1.5305  5.789   1.448  44.5365 13.983   4.0465 33.725  17.424
 21.78   21.9775 29.559  13.038  10.752   4.5805 33.1065 41.625   4.5675
  7.888   6.087  41.29    7.995   0.6045 32.095  11.7465  8.377  14.955
 11.9865 33.235  10.1325  2.31    4.4075  7.863  14.8185 35.42    5.567
 29.008   3.0125  8.712  21.063   1.6815  1.549  12.37   18.915  16.743
 36.39   16.794  12.036   2.3535  4.9845 13.2225  6.9825  2.7725  6.4455
  5.999  17.625  43.55    9.88    9.726   8.661   3.594  14.313   4.062
 28.02    9.34   11.0115 13.456  22.74   22.428  14.694  29.475  14.55
  1.974   1.7405 14.796   2.148   6.924   4.91    6.483  31.78    7.288
 10.065  31.5855 19.264  24.315  25.683  23.67   21.8425  5.408  12.438
 31.311  48.75   24.164   4.848   9.885  36.2115 39.7755 25.1195  8.6
  3.449   6.248   3.855  24.186  15.106  34.9335  6.2325  8.92   25.011
  1.791   6.807   5.244   8.946  40.7835  6.618  12.8695  4.668  11.4
  8.3355 34.87   19.452  18.263   4.464   8.4     0.985  26.558   2.686
 40.975  28.42   29.38   36.624  42.282  19.4635  4.2415  7.163   3.769
 12.668   1.921  32.615   2.6325  5.5305 28.4305  6.82    8.71   18.32
 12.7305 38.916  14.296  28.956   9.425  11.078  38.6    36.065  25.552
  2.6725 11.1    38.184  11.409   4.107  19.128   3.429  19.108  30.0545
 23.7965  2.621   6.565   7.215  22.8585  4.669   6.3125 39.5415  8.72
 18.952   1.531  17.604   2.54   26.103  28.756   2.7475  9.0705 20.6185
  2.3205 13.71   48.685  32.41    4.661   2.718   3.0435 12.245   4.639
 21.6725  6.903  12.08   23.5865 22.032  34.0155 15.494   9.318  10.046
  0.8875 31.09    4.3    20.13   16.2425  4.7575 19.448  21.284  15.904
 13.552  19.232  11.79   10.578   4.768   0.5085 10.3065 21.028   4.402
 32.4495  6.192  32.475  37.11    4.224  12.514   4.74    4.565  14.2555
  2.619   9.635  13.389  27.935   8.766   7.791   3.015   3.947   1.487
  1.066  14.067   3.663   1.119  32.796  29.73    3.705   9.848  18.6165
 26.395  23.9875 16.4295  8.448   5.662  17.277  21.4335  4.3135  1.276
  5.076  17.8745 11.9385  5.0715 36.212   6.282   3.6465 12.918   8.687
  2.825  10.715  26.718   4.658  26.104   2.6175  1.9875 36.008   4.84
 16.605   4.072  15.995   8.334  15.953   4.395  36.735   4.876  38.46
 20.915  23.164  23.1225  7.095  15.135  39.664  21.259  14.181  29.96
 15.768  20.178   9.194   6.9325  4.0355  5.832  15.676  42.305  20.72
  7.954  24.505   4.3725 11.226  37.248  20.536  14.94   10.647   2.1425
 18.934  10.3455  3.939  16.1055  4.911   1.273  29.099  10.566   2.756
  4.4155 17.829  39.7125  2.531  29.976   8.335  37.22   18.945  12.858
 27.6115 22.37   13.8135 17.187  13.304  44.919  22.84   12.6975  3.528
 32.858   8.425   2.689   8.9525 10.572   5.9865  3.285   4.208  19.773
 14.8995 22.7205 13.806   7.9    44.397   4.599   2.089   0.775  14.523
  3.333   3.827  14.985  12.1515  2.37    8.6225 42.315  12.9185 30.478
 12.012   8.613   4.992  14.932   7.98    1.2725  3.3885 11.918  11.63
 43.866  34.986  33.7295 15.9275  1.476  24.8    41.17   30.148  14.14
 38.3     5.803   8.745   3.0475  2.0175 48.69    1.592   3.291  30.919 ] has 990
unique value


Rating ------------ [ 9.1  9.6  7.4  8.4  5.3  4.1  5.8  8.   7.2  5.9  4.5  6.8
7.1  8.2
  5.7  4.6  6.9  8.6  4.4  4.8  5.1  9.9  6.   8.5  6.7  7.7  7.5  7.
  4.7  7.6  7.9  6.3  5.6  9.5  8.1  6.5  6.1  6.6  5.4  9.3 10.   6.4
  4.3  4.   8.7  9.4  5.5  8.3  7.3  4.9  4.2  9.2  7.8  5.2  9.   8.8
  6.2  9.8  9.7  5.   8.9] has 61 unique value
```

In [10]:

```
for i in df_numeric:
    for j in df[i]:
```

```
        if j<=0:
            print("invalid entry is",j,"in",df[i])
```

Observation : Their is not any inconsistency in both object and numeric columns of data

Checking the duplicacy

In [11]:
```
df.duplicated().sum()
```

Out[11]:
```
0
```

Observation : Their is no duplicates values in the data

Feature engineering column extraction

In [12]:
```
df.columns
```

Out[12]:
```
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 [13]:
```
df["Date"]=pd.to_datetime(df["Date"])
df["Time"]=pd.to_datetime(df["Time"])
```

```
C:\Users\Rahul\AppData\Local\Temp\ipykernel_15336\2384983477.py:2: UserWarning: C
ould not infer format, so each element will be parsed individually, falling back
to `dateutil`. To ensure parsing is consistent and as-expected, please specify a
format.
  df["Time"]=pd.to_datetime(df["Time"])
```

In [14]:
```
df.dtypes
```

Out[14]:
```
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                     datetime64[ns]
Time                     datetime64[ns]
Payment                          object
cogs                            float64
gross margin percentage         float64
gross income                    float64
Rating                          float64
dtype: object
```

In [15]:
```
from datetime import datetime as dt
```

In [16]:
```
df["month"]=df.Date.dt.month_name()
df["day_name"]=df.Date.dt.day_name()
df["day"]=df.Date.dt.day
```

```
df["year"]=df.Date.dt.year
df["Hour"]=df.Time.dt.hour
```

In [17]:
```
df.sample(2)
```

Out[17]:

	Invoice ID	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%	Total
246	866-05-7563	B	Mandalay	Member	Female	Electronic accessories	81.40	3	12.210	256.410
371	261-12-8671	B	Mandalay	Normal	Female	Fashion accessories	60.96	2	6.096	128.016

2 rows × 22 columns

In [18]:
```
df.groupby(["City"])["Branch"].unique()
```

Out[18]:
```
City
Mandalay     [B]
Naypyitaw    [C]
Yangon       [A]
Name: Branch, dtype: object
```

Since we are deleting the invoice ID as it seems irrelevant and also we are deleting branch each city maps single branch

City --- Branch

Mandalay --- [B]

Naypyitaw --- [C]

Yangon --- [A]

In [19]:
```
df.drop(columns=["Invoice ID","Branch"],inplace=True)
```

In [20]:
```
df
```

Out[20]:

	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%	Total	Date	Ti
0	Yangon	Member	Female	Health and beauty	74.69	7	26.1415	548.9715	2019-01-05	20 05 13:08
1	Naypyitaw	Normal	Female	Electronic accessories	15.28	5	3.8200	80.2200	2019-03-08	20 05 10:29
2	Yangon	Normal	Male	Home and lifestyle	46.33	7	16.2155	340.5255	2019-03-03	20 05 13:23
3	Yangon	Member	Male	Health and beauty	58.22	8	23.2880	489.0480	2019-01-27	20 05 20:33
4	Yangon	Normal	Male	Sports and travel	86.31	7	30.2085	634.3785	2019-02-08	20 05 10:37
...	...	...	...	...	...	...	...	...	...	
995	Naypyitaw	Normal	Male	Health and beauty	40.35	1	2.0175	42.3675	2019-01-29	20 05 13:46
996	Mandalay	Normal	Female	Home and lifestyle	97.38	10	48.6900	1022.4900	2019-03-02	20 05 17:16
997	Yangon	Member	Male	Food and beverages	31.84	1	1.5920	33.4320	2019-02-09	20 05 13:22
998	Yangon	Normal	Male	Home and lifestyle	65.82	1	3.2910	69.1110	2019-02-22	20 05 15:33
999	Yangon	Member	Female	Fashion accessories	88.34	7	30.9190	649.2990	2019-02-18	20 05 13:28

1000 rows × 20 columns

Step 5: Analysis and Insights

Statistical Summary

In [21]:
```
df.describe().T
```

Out[21]:

	count	mean	min	25%	50%	75%	max	
Unit price	1000.0	55.67213	10.08	32.875	55.23	77.935	99.96	26.494
Quantity	1000.0	5.51	1.0	3.0	5.0	8.0	10.0	2.923
Tax 5%	1000.0	15.379369	0.5085	5.924875	12.088	22.44525	49.65	11.708
Total	1000.0	322.966749	10.6785	124.422375	253.848	471.35025	1042.65	245.885
Date	1000	2019-02-14 00:05:45.600000	2019-01-01 00:00:00	2019-01-24 00:00:00	2019-02-13 00:00:00	2019-03-08 00:00:00	2019-03-30 00:00:00	N
Time	1000	2024-05-27 15:24:41.880000	2024-05-27 10:00:00	2024-05-27 12:43:00	2024-05-27 15:19:00	2024-05-27 18:15:00	2024-05-27 20:59:00	N
cogs	1000.0	307.58738	10.17	118.4975	241.76	448.905	993.0	234.17
gross margin percentage	1000.0	4.761905	4.761905	4.761905	4.761905	4.761905	4.761905	
gross income	1000.0	15.379369	0.5085	5.924875	12.088	22.44525	49.65	11.708
Rating	1000.0	6.9727	4.0	5.5	7.0	8.5	10.0	1.71
day	1000.0	15.256	1.0	8.0	15.0	23.0	31.0	8.693
year	1000.0	2019.0	2019.0	2019.0	2019.0	2019.0	2019.0	
Hour	1000.0	14.91	10.0	12.0	15.0	18.0	20.0	3.186

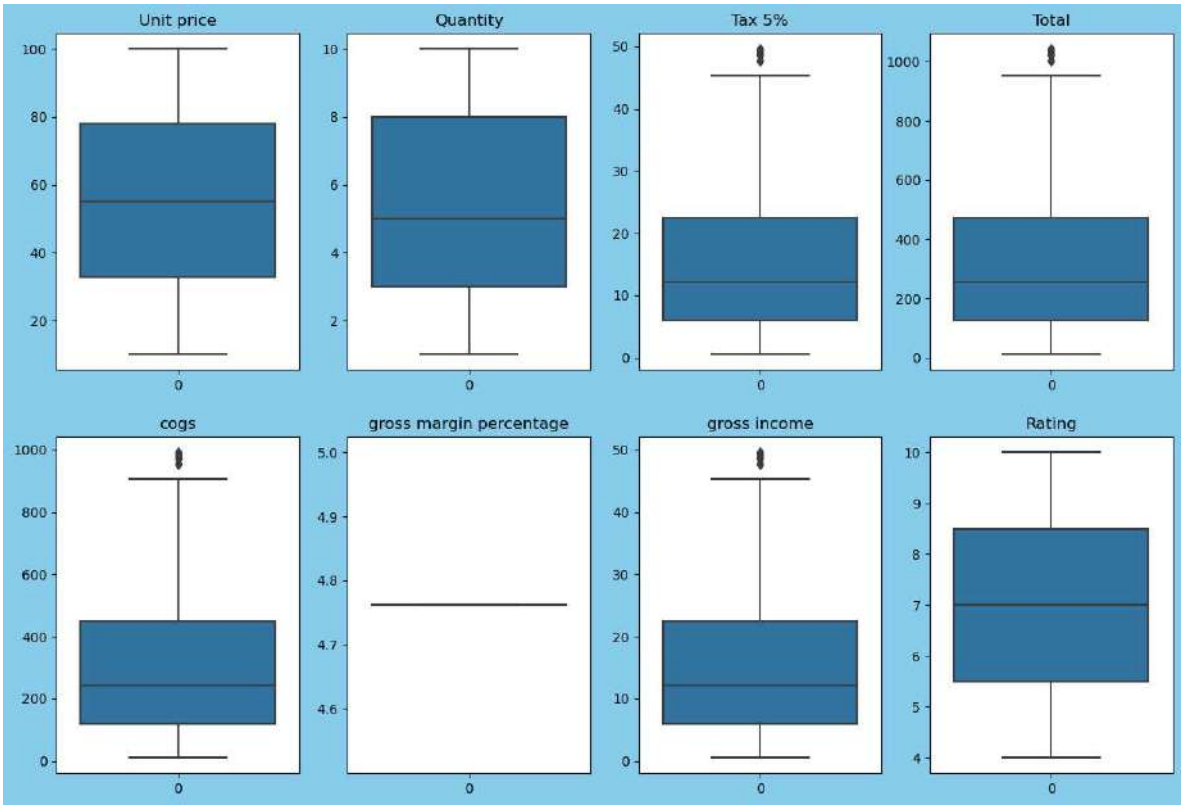
Observation : This statistical summary give me the hint their is a outliers in some numeric columns of the data

Detecting the outliers from boxplot

In [22]:
```
col=df.describe().T.index[[0,1,2,3,6,7,8,9]]
col
```

Out[22]:
```
Index(['Unit price', 'Quantity', 'Tax 5%', 'Total', 'cogs',
       'gross margin percentage', 'gross income', 'Rating'],
      dtype='object')
```

In [23]:
```
pos=1
plt.figure(figsize=(15,10),facecolor="skyblue")
for i in col:
    plt.subplot(2,4,pos)
    sns.boxplot(df[i])
    plt.title(i)
    pos+=1
```



Observation: in above boxplots, we can see boxplot of cogs,gross income,tax 5% and total is quite similar. so we have to check the relation between them how the affect each others for that we have to calculate the correlation between them

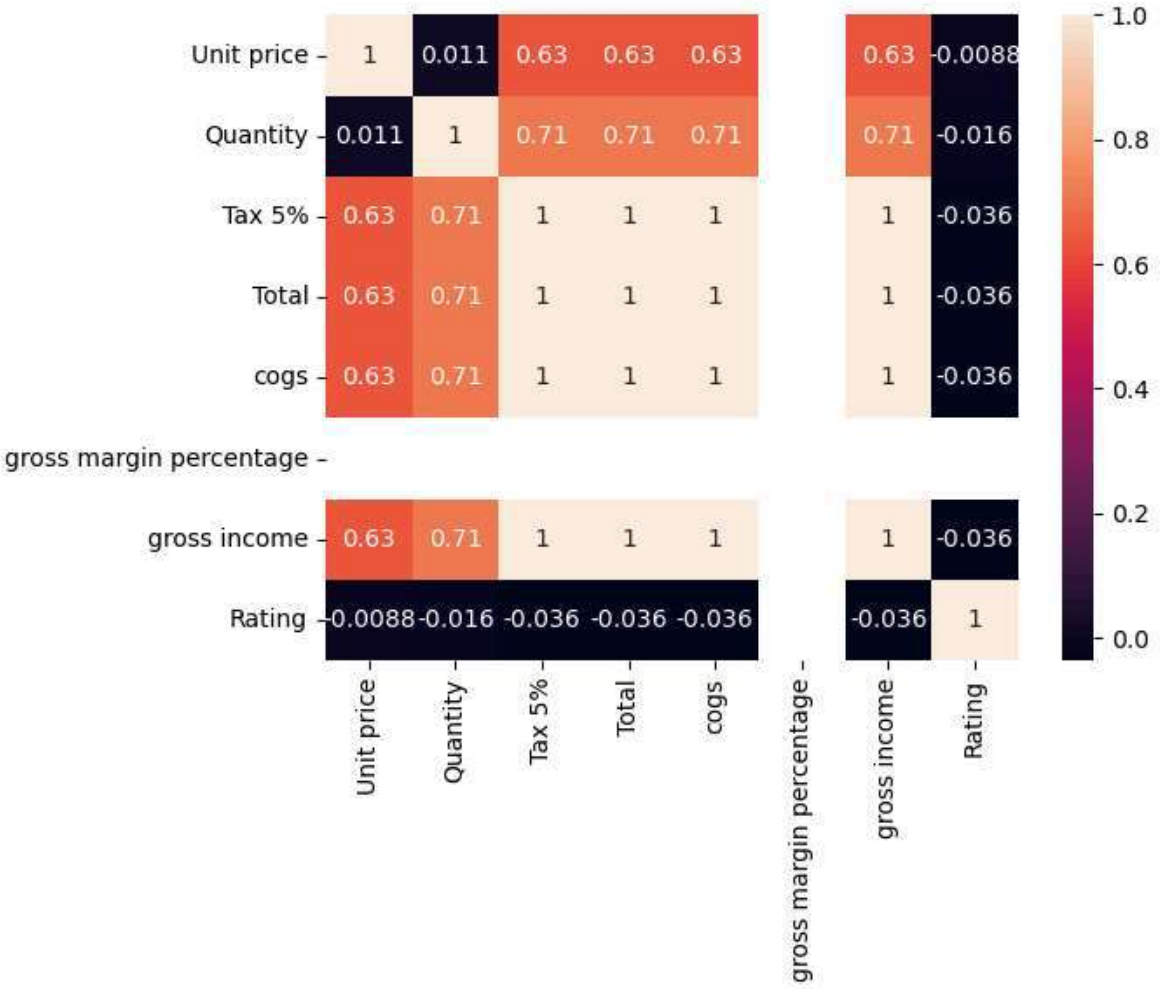
Correlation

In [24]:
```
df_numeric.corr()
```

Out[24]:

	Unit price	Quantity	Tax 5%	Total	cogs	gross margin percentage	gross income	Ratir
Unit price	1.000000	0.010778	0.633962	0.633962	0.633962	NaN	0.633962	-0.0087
Quantity	0.010778	1.000000	0.705510	0.705510	0.705510	NaN	0.705510	-0.0158
Tax 5%	0.633962	0.705510	1.000000	1.000000	1.000000	NaN	1.000000	-0.0364
Total	0.633962	0.705510	1.000000	1.000000	1.000000	NaN	1.000000	-0.0364
cogs	0.633962	0.705510	1.000000	1.000000	1.000000	NaN	1.000000	-0.0364
gross margin percentage	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
gross income	0.633962	0.705510	1.000000	1.000000	1.000000	NaN	1.000000	-0.0364
Rating	-0.008778	-0.015815	-0.036442	-0.036442	-0.036442	NaN	-0.036442	1.0000

In [25]:
```
sns.heatmap(df_numeric.corr(),annot=True);
```



Observation :

- There is a perfect positive relation found between Total,Gross income,Cogs,Tax 5%. we will keep total columns only , delete the rest all column.
- There is sort of positive relation between Total , quantity.
- There is no relation between rating and quantity.

In [26]:
```
df.drop(columns=["Tax 5%","cogs","gross income","gross margin percentage"],inplac
```

In [27]:
```
df.head(2)
```

Out[27]:

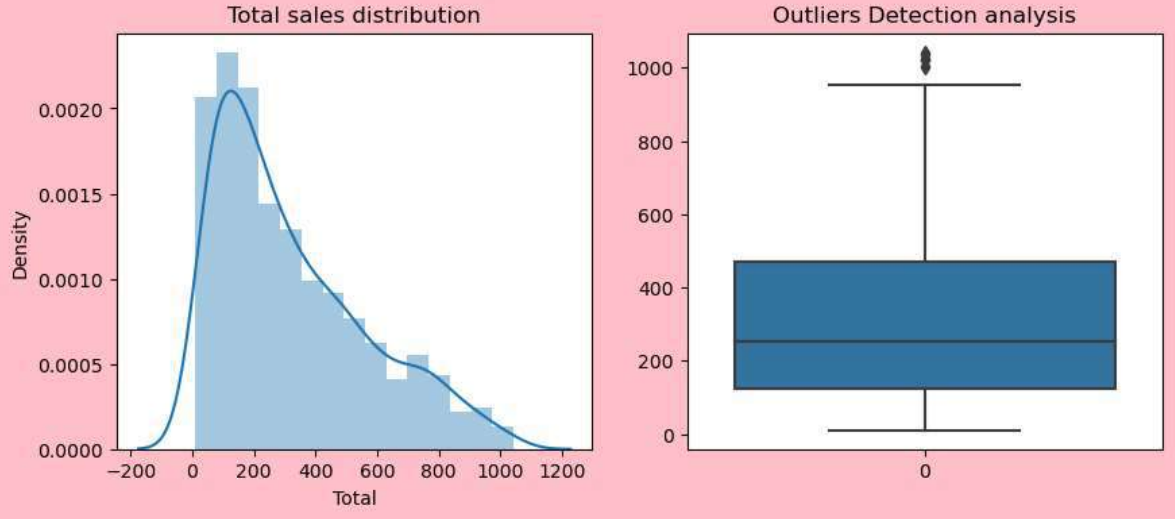
	City	Customer type	Gender	Product line	Unit price	Quantity	Total	Date	Time	Paymen
0	Yangon	Member	Female	Health and beauty	74.69	7	548.9715	2019-01-05	2024-05-27 13:08:00	Ewalle
1	Naypyitaw	Normal	Female	Electronic accessories	15.28	5	80.2200	2019-03-08	2024-05-27 10:29:00	Cas

Observation: Since the main columns in sales data is **Total** from business point of view. we need to do univeriate analysis of total

In [28]:
```
plt.figure(figsize=(10,4),facecolor="pink")
plt.subplot(1,2,1)
```

```
sns.distplot(df["Total"])
plt.title("Total sales distribution",fontsize=12)
plt.subplot(1,2,2)
sns.boxplot(df["Total"])
plt.title("Outliers Detection analysis",fontsize=12);
```

```
C:\Users\Rahul\AppData\Local\Temp\ipykernel_15336\3138082297.py:3: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df["Total"])
```



In [29]:
```
np.percentile(df["Total"],[25,75])
```

Out[29]:
```
array([124.422375, 471.35025 ])
```

Observation:

- Mostly total bill are under 125 to 470 dollars.
- some of the total bill value are quite expensive because their is a outliers

Total and average sale of the company

In [30]:
```
df["Total"].agg(["sum","mean"])
```

Out[30]:
```
sum     322966.749000
mean       322.966749
Name: Total, dtype: float64
```

Since, the some product are expensive means they are considered as outliers so we find median since the above average sales does give the genuine insight.

In [31]:
```
df["Total"].agg(["sum","median"])
```

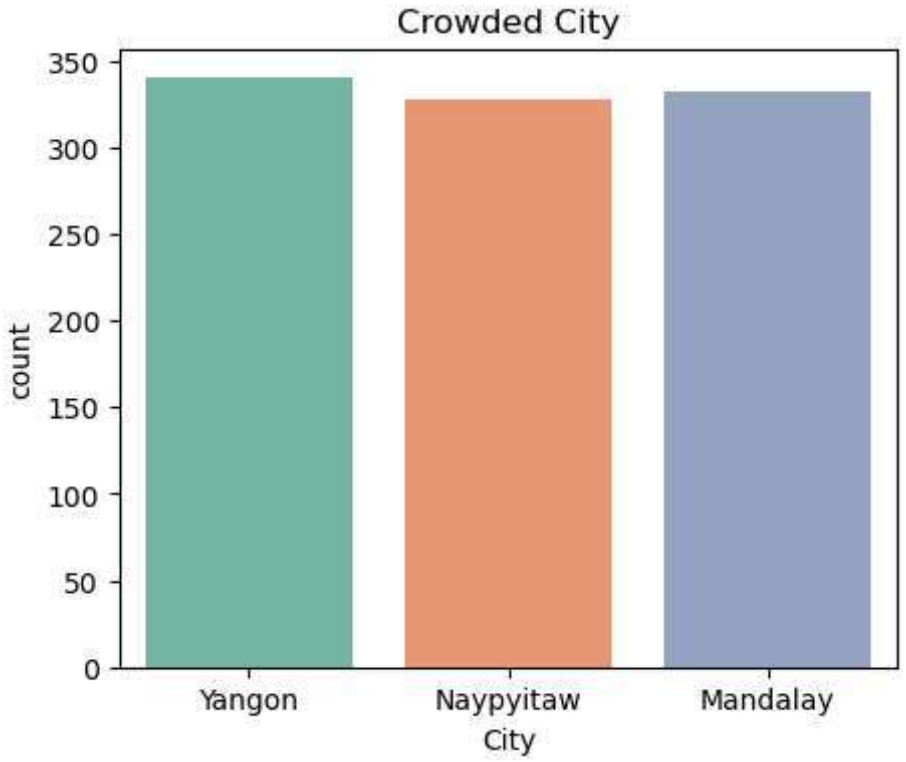
Out[31]:
```
sum       322966.749
median       253.848
Name: Total, dtype: float64
```

Observation:

- Total sale of the company is approx 322966.74 dollars
- the Average sale of the company is approx 253.84 dollars

Which city is found to be more crowded ?

In [32]:
```
plt.figure(figsize=(5,4))
sns.countplot(x="City",data=df,palette="Set2")
plt.title("Crowded City");
```



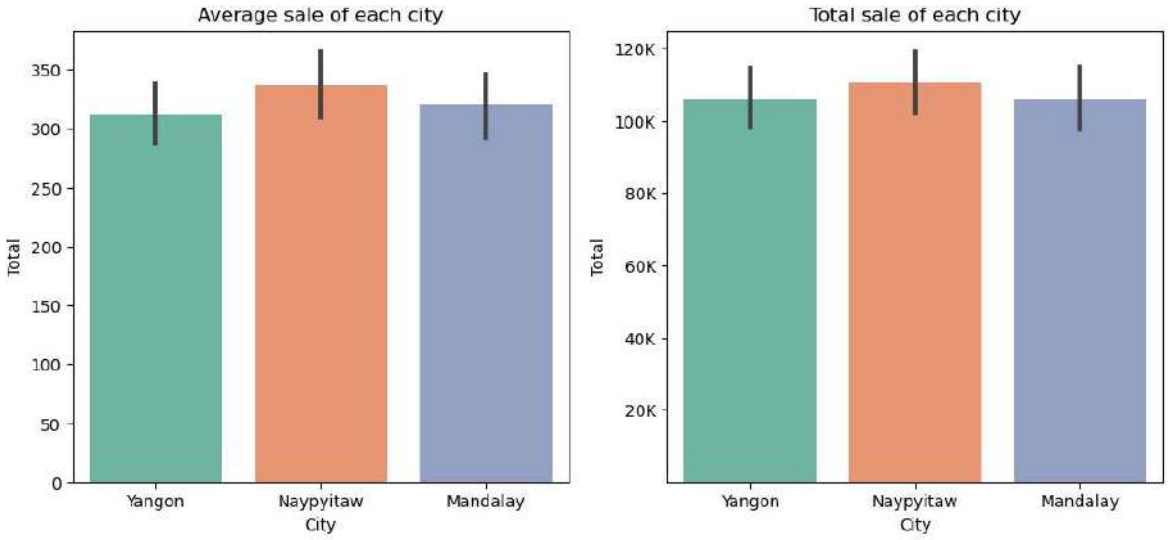
Observation:

- All three cities have approximately the same level of crowding, with very little difference between them.
- Yangon city is most crowded.

Total sale and average sale of each city/branch

In [33]:
```
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
sns.barplot(x="City",y="Total",data=df,palette="Set2")
plt.title(" Average sale of each city ")
plt.subplot(1,2,2)
sns.barplot(x="City",y="Total",data=df,estimator=sum,palette="Set2")
plt.yticks([20000,40000,60000,80000,100000,120000],["20K","40K","60K","80K","100K
plt.title(" Total sale of each city ")
```

Out[33]:
```
Text(0.5, 1.0, ' Total sale of each city ')
```



Observation : Though the crowded city is Yangon bot hot selling city is found to be Naypyitaw . There is not much significant difference between three supermarkets in each city name Naypyitaw city if found to be hot selling city.

Find the highest sale in Naypytiaw in any day

In [34]:
```
df_n=df[df["City"]=="Naypyitaw"]
df_n[df_n["Total"]==df_n["Total"].max()]
```

Out[34]:

	City	Customer type	Gender	Product line	Unit price	Quantity	Total	Date	Time	Payme
350	Naypyitaw	Member	Female	Fashion accessories	99.3	10	1042.65	2019-02-15	2024-05-27 14:53:00	Cre ca

Find the lowest sale in Naypytiaw in any day

In [35]:
```
df_n=df[df["City"]=="Naypyitaw"]
df_n[df_n["Total"]==df_n["Total"].min()]
```

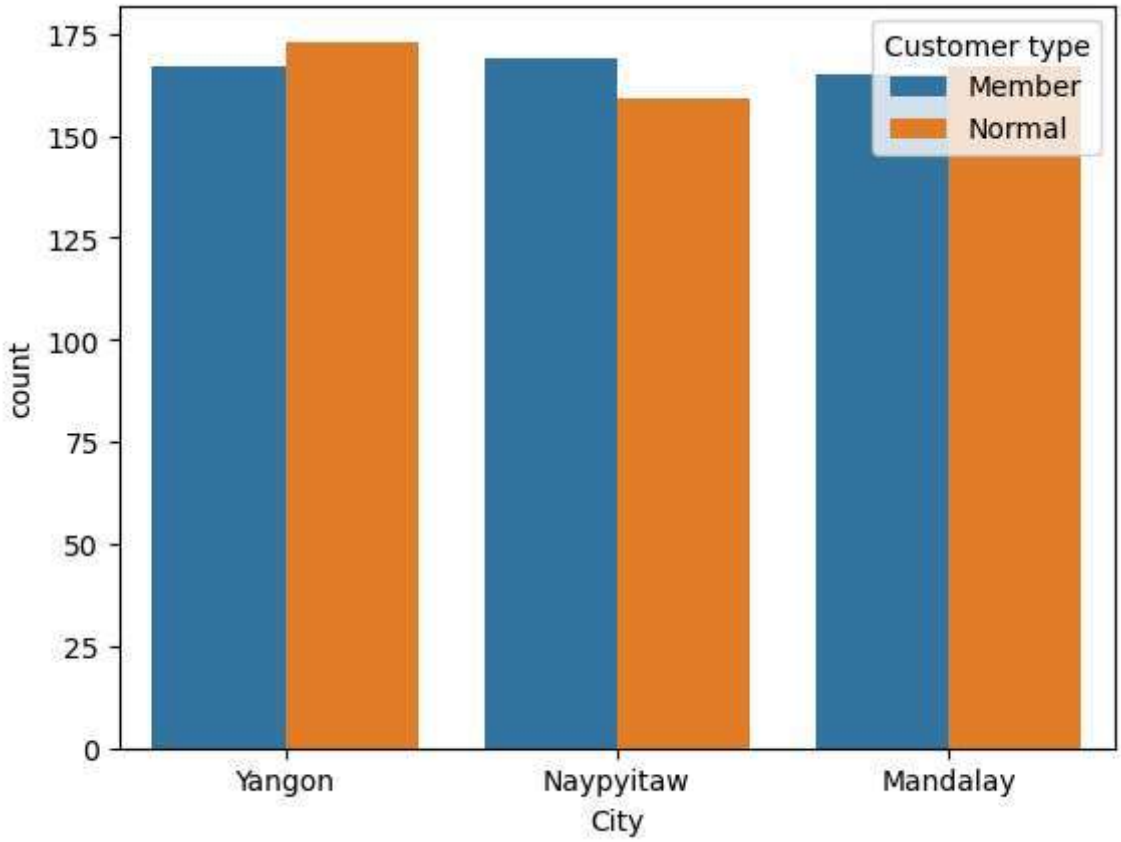
Out[35]:

	City	Customer type	Gender	Product line	Unit price	Quantity	Total	Date	Time	Payment
822	Naypyitaw	Member	Male	Sports and travel	10.17	1	10.6785	2019-02-07	2024-05-27 14:15:00	Cash

Root cause analysis of hot selling cities

1. Customer Type

In [36]:
```
#df.groupby(["City"])["Customer type"].value_counts()
sns.countplot(x="City",data=df,hue="Customer type");
```



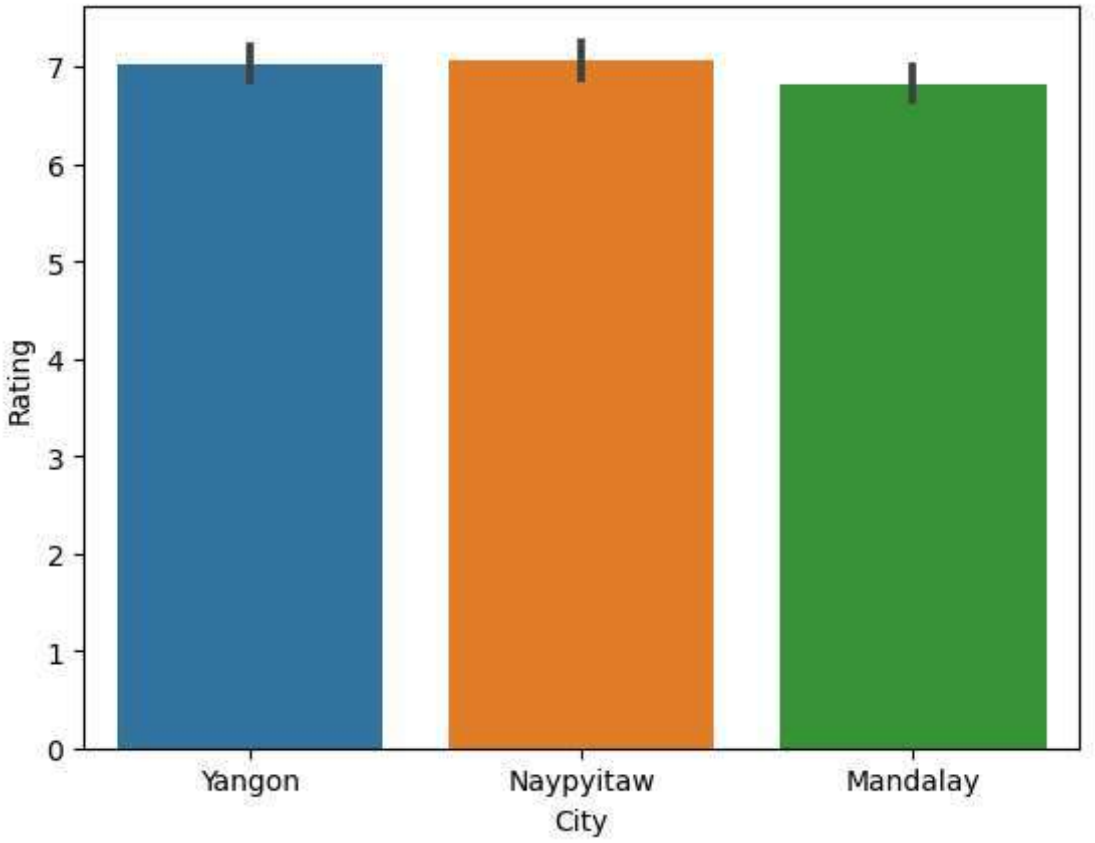
Observation : there is not much difference in customer type

2. Average rating of each cities

In [37]:
```
df.groupby(["City"])["Rating"].mean()
```

Out[37]:
```
City
Mandalay      6.818072
Naypyitaw     7.072866
Yangon        7.027059
Name: Rating, dtype: float64
```

In [38]:
```
sns.barplot(x="City",y="Rating",data=df);
# by default estimator gives mean
```



3. Product line

In [39]:
```
df[df["Unit price"]==df["Unit price"].max()]
```

Out[39]:

	City	Customer type	Gender	Product line	Unit price	Quantity	Total	Date	Time	Payment
122	Mandalay	Member	Male	Sports and travel	99.96	9	944.622	2019-03-09	2024-05-27 17:26:00	Credit card
983	Naypyitaw	Normal	Male	Health and beauty	99.96	7	734.706	2019-01-23	2024-05-27 10:33:00	Cash

In [40]:
```
df.groupby(["City"])["Product line"].agg(["value_counts"]).unstack()
```

Out[40]:

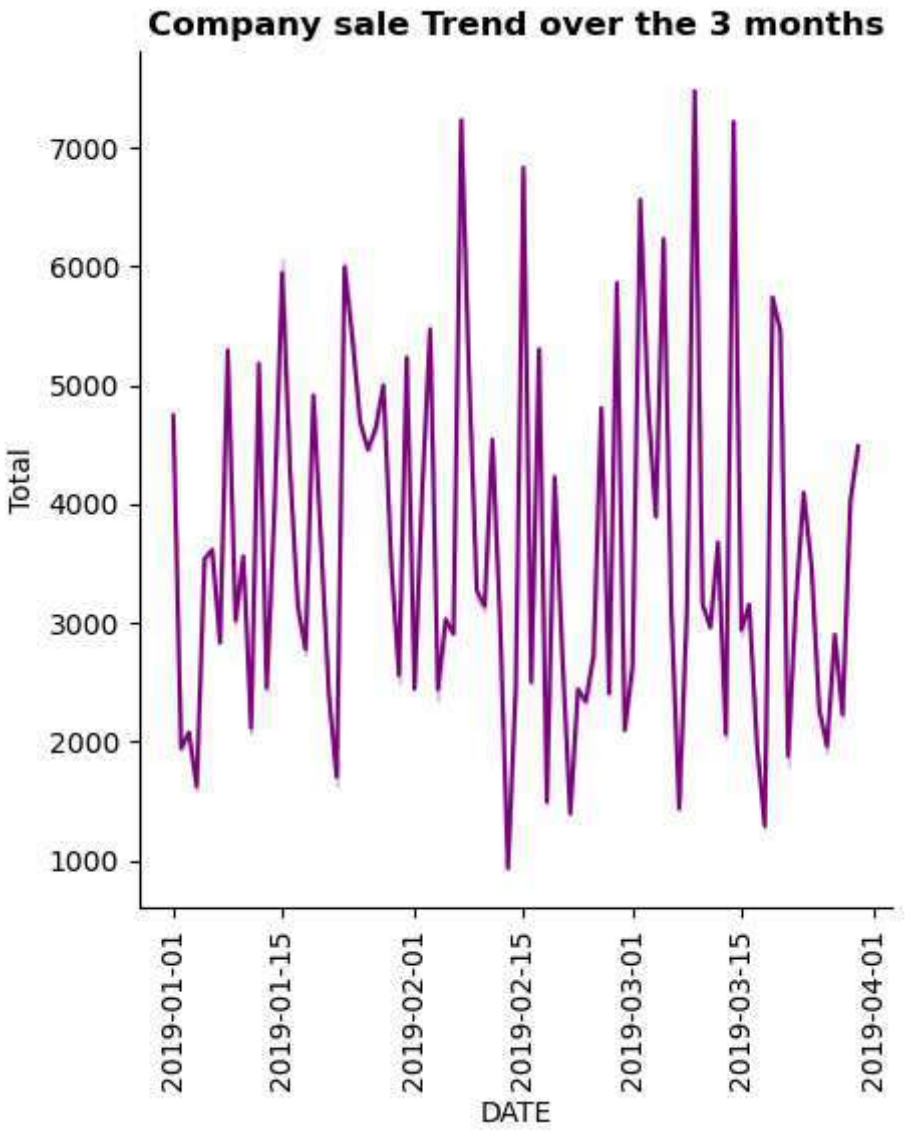
	value_counts					
Product line	**Electronic accessories**	**Fashion accessories**	**Food and beverages**	**Health and beauty**	**Home and lifestyle**	**Sports and travel**
City						
Mandalay	55	62	50	53	50	62
Naypyitaw	55	65	66	52	45	45
Yangon	60	51	58	47	65	59

Observation : There is not a specific cause behind the hot selling branch /city this could be due to natural factor

Sale trend of company over the three months

In [41]:
```
plt.figure(figsize=(3,2))
sns.relplot(x="Date",y="Total",data=df,kind="line",ci=False,color="purple",estima
plt.xlabel("DATE")
plt.xticks(rotation=90)
plt.title("Company sale Trend over the 3 months",fontweight="bold");
```

```
C:\Users\Rahul\anaconda3\Lib\site-packages\seaborn\axisgrid.py:848: FutureWarnin
g:

The `ci` parameter is deprecated. Use `errorbar=('ci', False)` for the same effec
t.

  func(*plot_args, **plot_kwargs)
C:\Users\Rahul\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning:
The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
<Figure size 300x200 with 0 Axes>
```

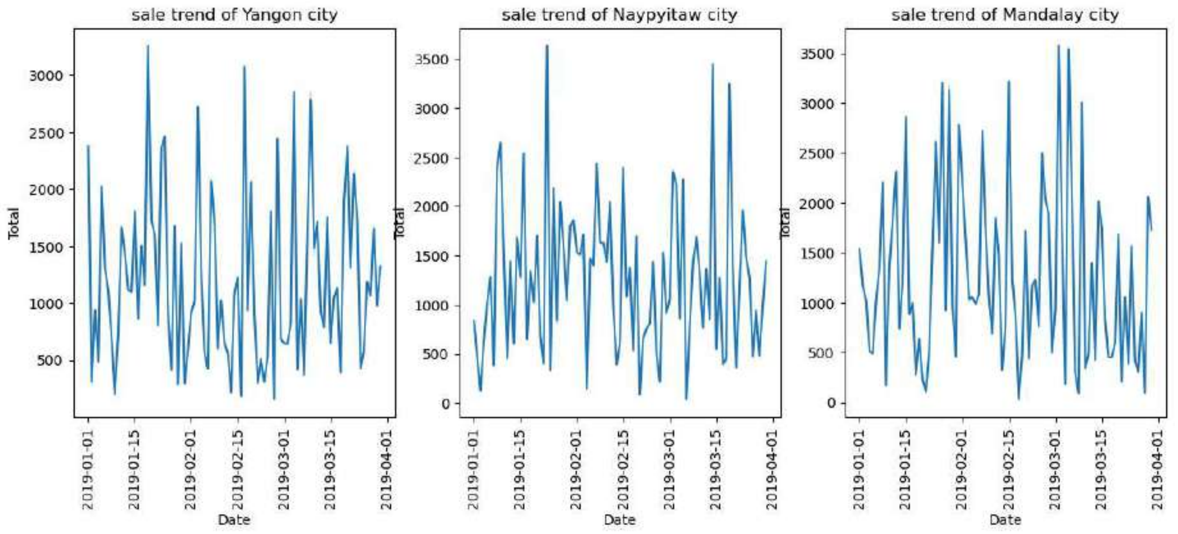


Sale Trend of each city

In [42]:
```
df_n=df[df["City"]=="Naypyitaw"]
df_y=df[df["City"]=="Yangon"]
df_m=df[df["City"]=="Mandalay"]
```

In [43]:
```
city_name=['Yangon', 'Naypyitaw', 'Mandalay']
cities=[df_y,df_n,df_m]
pos=0
plt.figure(figsize=(14,5))
for i in cities:
    plt.subplot(1,3,pos+1)
    sns.lineplot(x="Date",y="Total",data=i,estimator=sum,ci=False)
    plt.title(f"sale trend of {city_name[pos]} city" )
    plt.xticks(rotation=90)
    pos+=1
```

```
C:\Users\Rahul\AppData\Local\Temp\ipykernel_15336\1653840656.py:7: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=('ci', False)` for the same effec
t.

  sns.lineplot(x="Date",y="Total",data=i,estimator=sum,ci=False)
C:\Users\Rahul\AppData\Local\Temp\ipykernel_15336\1653840656.py:7: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=('ci', False)` for the same effec
t.

  sns.lineplot(x="Date",y="Total",data=i,estimator=sum,ci=False)
C:\Users\Rahul\AppData\Local\Temp\ipykernel_15336\1653840656.py:7: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=('ci', False)` for the same effec
t.

  sns.lineplot(x="Date",y="Total",data=i,estimator=sum,ci=False)
```

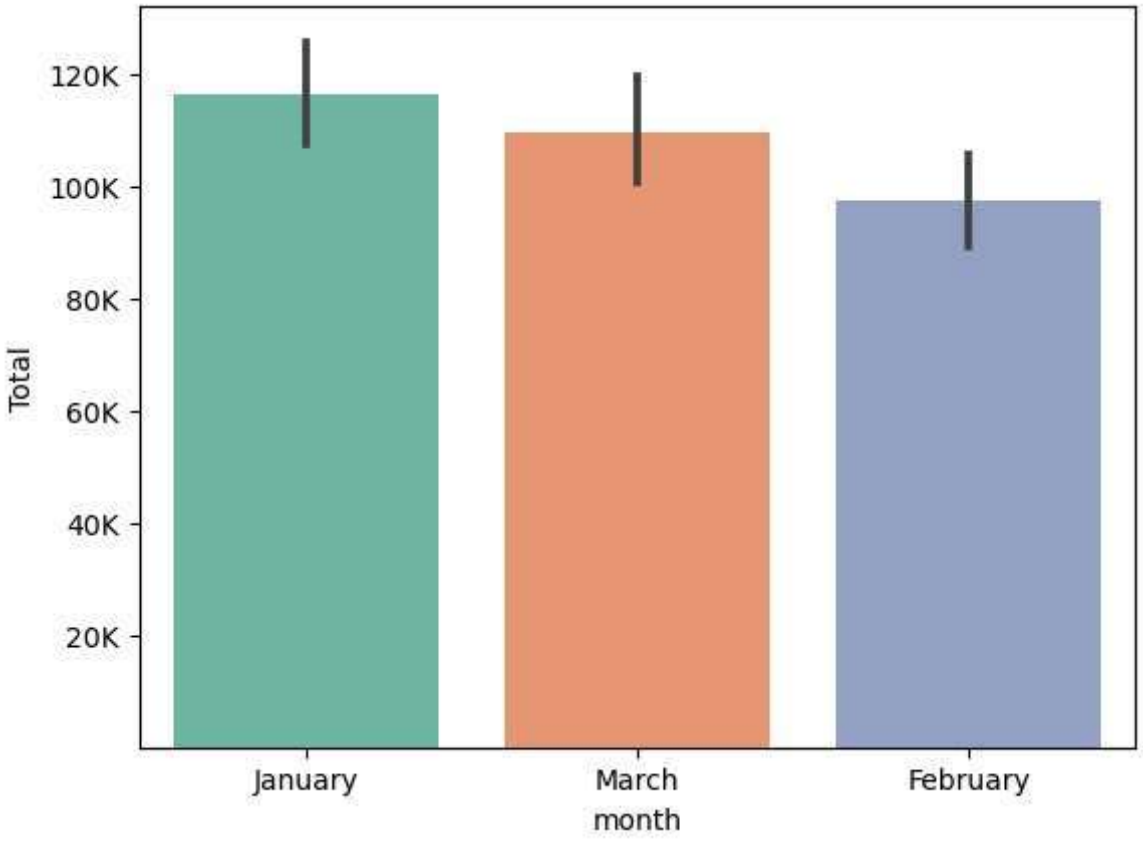


Find the highest business revenue month of the company

In [44]:
```
df.columns
```

Out[44]:
```
Index(['City', 'Customer type', 'Gender', 'Product line', 'Unit price',
       'Quantity', 'Total', 'Date', 'Time', 'Payment', 'Rating', 'month',
       'day_name', 'day', 'year', 'Hour'],
      dtype='object')
```

In [45]:
```
sns.barplot(x="month",y="Total",data=df,estimator=sum,palette="Set2")
plt.yticks([20000,40000,60000,80000,100000,120000],["20K","40K","60K","80K","100K
```



Percent contribution business revenue of each month

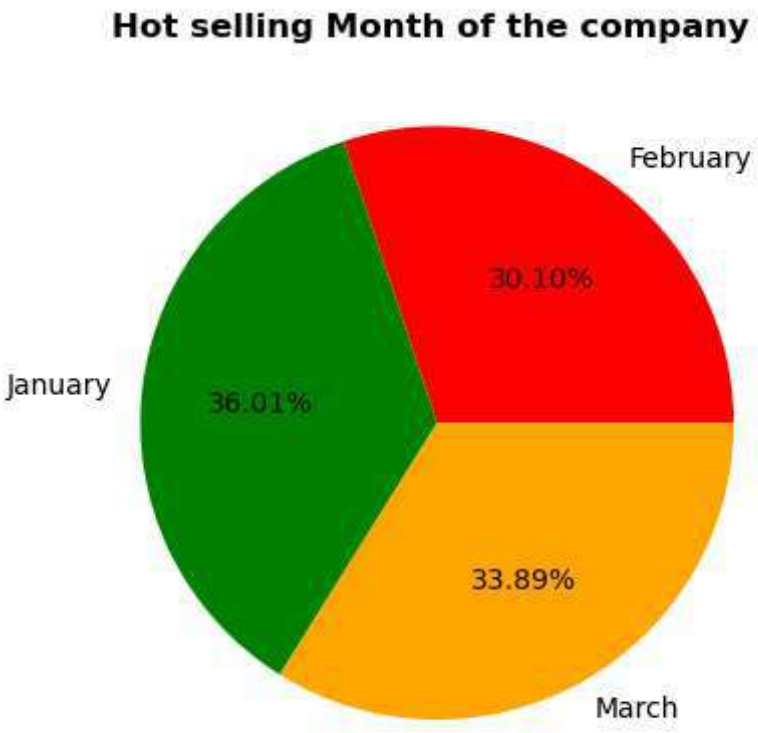
In [46]:
```
df.groupby(["month"])["Total"].agg(["sum"])
```

Out[46]:

month	sum
February	97219.374
January	116291.868
March	109455.507

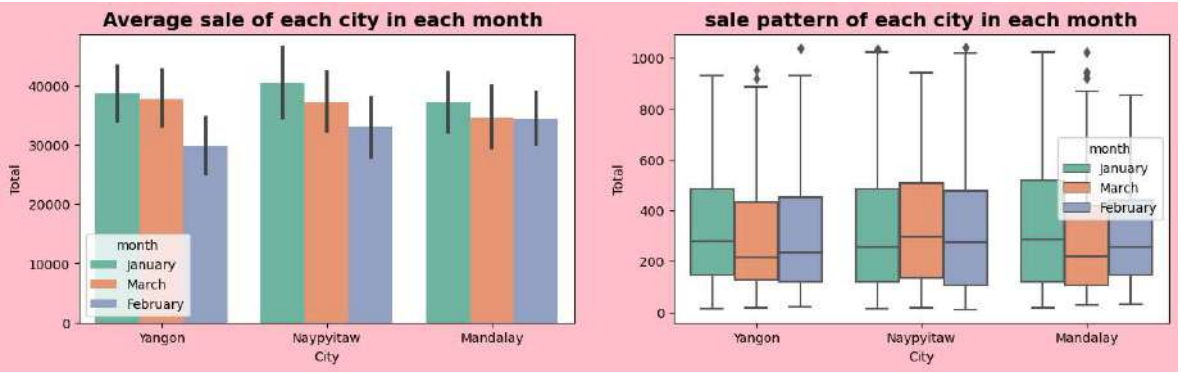
In [47]:
```
plt.pie(df.groupby(["month"])["Total"].agg(["sum"])["sum"],labels=df.groupby(["mo
plt.title("Hot selling Month of the company ",fontsize=12,color="black",fontweigh
```

Out[47]:
```
Text(0.5, 1.0, 'Hot selling Month of the company ')
```



Find the highest business revenue month of each city alse sales pattern

In [48]:
```
plt.figure(figsize=(15,4),facecolor="pink")
plt.subplot(1,2,1)
sns.barplot(x="City",y="Total",data=df,estimator=sum,palette="Set2",hue="month");
plt.title("Average sale of each city in each month ",fontsize=15,color="black",fo
plt.subplot(1,2,2)
sns.boxplot(x="City",y="Total",data=df,palette="Set2",hue="month");
plt.title("sale pattern of each city in each month",fontsize=15,color="black",fon
```

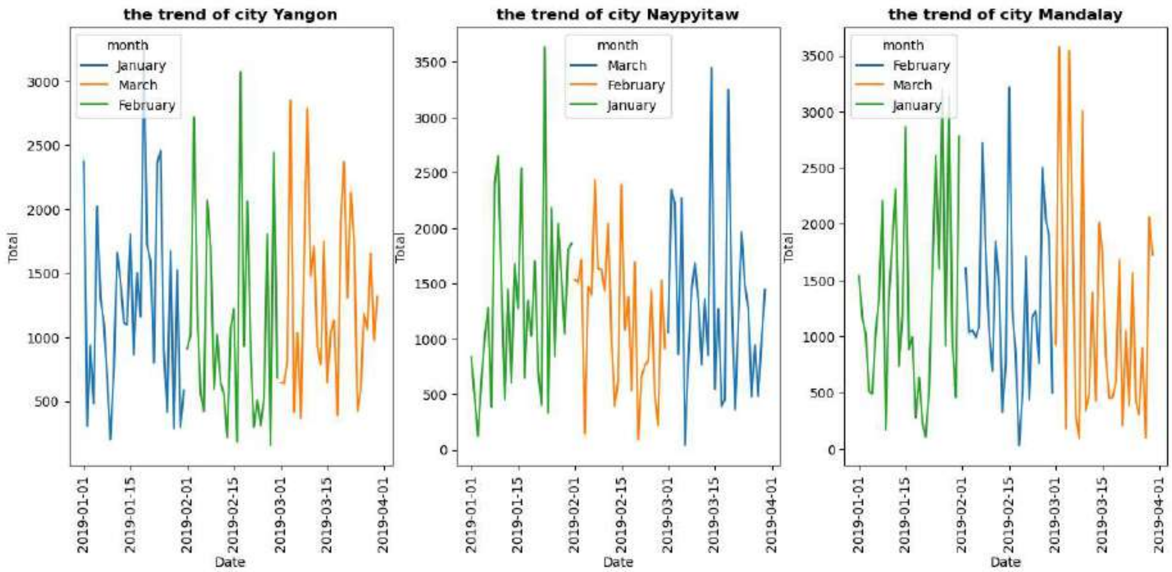


Customer visiting trend of each city/month at each month

In [49]:
```
city_name=["Yangon", 'Naypyitaw', 'Mandalay']
pos=1
plt.figure(figsize=(15,6))
for city in cities:
    #print(city)
    plt.subplot(1,3,pos)
    sns.lineplot(x="Date",y="Total",data=city,estimator=sum,ci=False,hue="month")
```

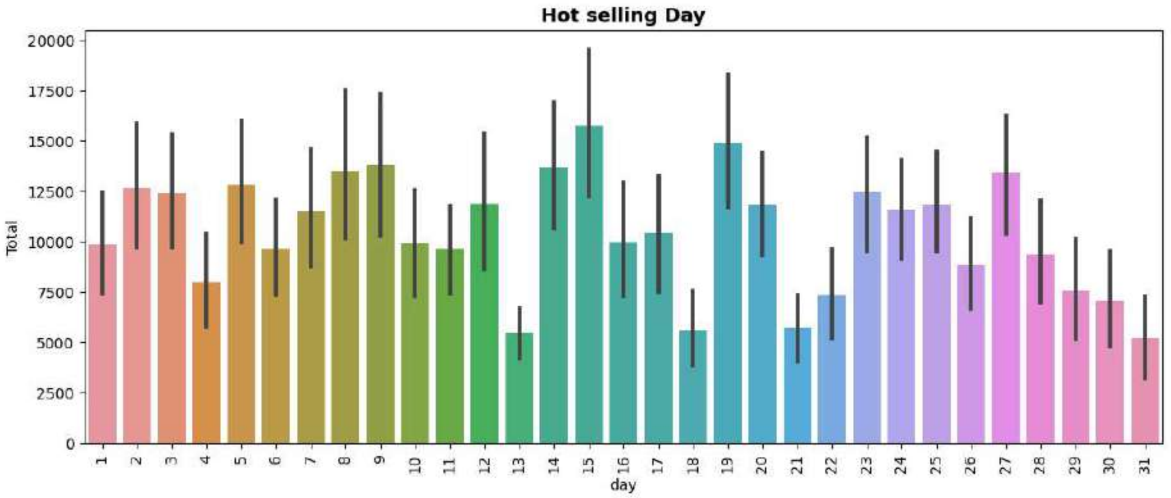
```
    plt.title(f"the trend of city {city_name[pos-1]}",fontweight="bold")
    plt.xticks(rotation=90)
    pos=pos+1
```

```
C:\Users\Rahul\AppData\Local\Temp\ipykernel_15336\3859639863.py:7: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=('ci', False)` for the same effec
t.

  sns.lineplot(x="Date",y="Total",data=city,estimator=sum,ci=False,hue="month")
C:\Users\Rahul\AppData\Local\Temp\ipykernel_15336\3859639863.py:7: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=('ci', False)` for the same effec
t.

  sns.lineplot(x="Date",y="Total",data=city,estimator=sum,ci=False,hue="month")
C:\Users\Rahul\AppData\Local\Temp\ipykernel_15336\3859639863.py:7: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=('ci', False)` for the same effec
t.

  sns.lineplot(x="Date",y="Total",data=city,estimator=sum,ci=False,hue="month")
```



Hot selling day

In [50]:
```
plt.figure(figsize=(13,5))
sns.barplot(x="day",y="Total",data=df,estimator=sum)
plt.xticks(rotation=90)
plt.title("Hot selling Day",fontweight="bold",fontsize=13);
```



In [51]:
```
df.groupby(["day"])["Total"].agg(["sum"])[df.groupby(["day"])["Total"].agg(["sum"
```

Out[51]:

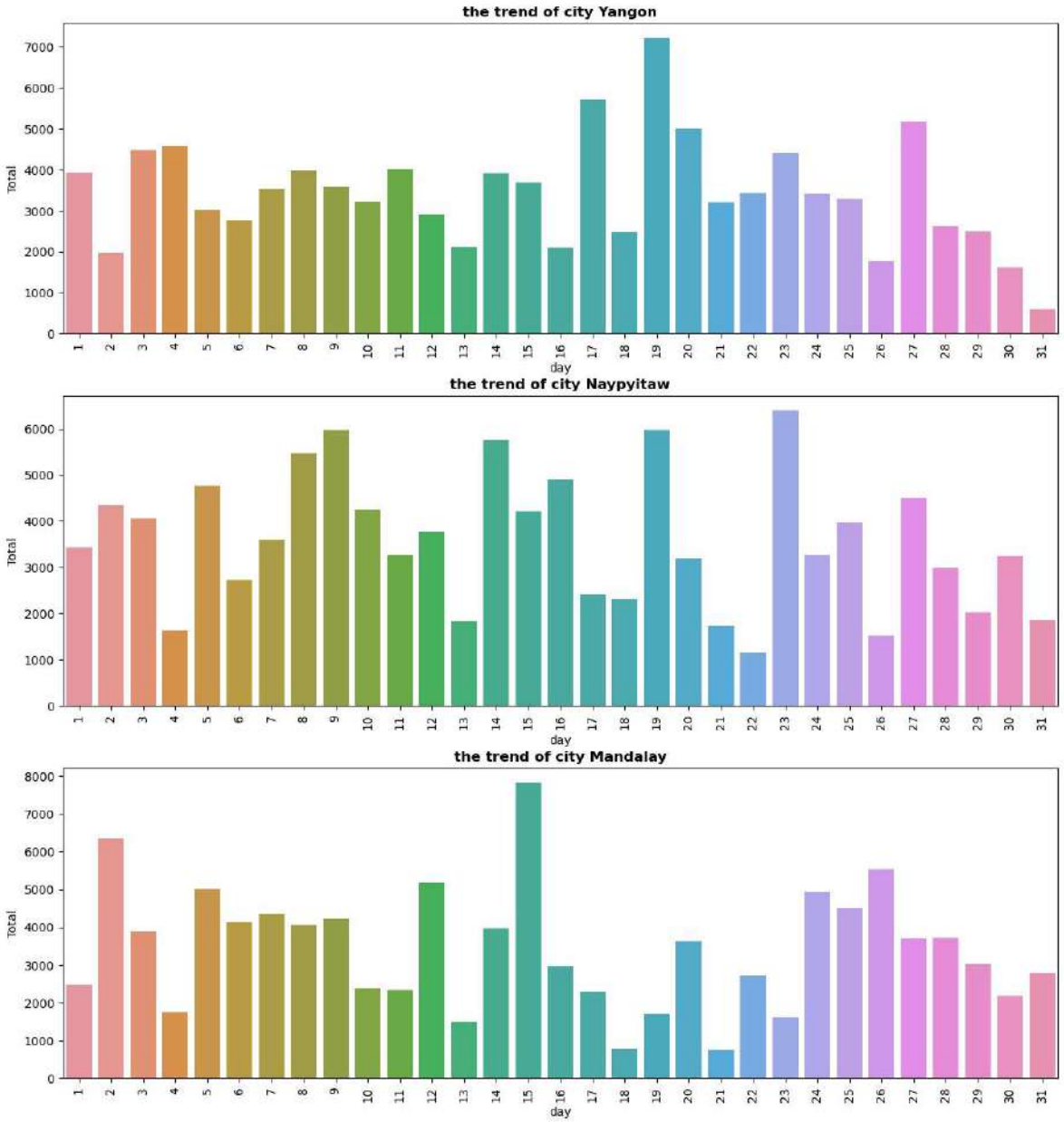
	sum
day	
15	15717.4605

Observation : The hottest selling day is the 15th. On that day, we achieved our maximum sales, approximately $15,717.46 .

Hot selling day of each city/branch

In [52]:
```
city_name=["Yangon", 'Naypyitaw', 'Mandalay']
pos=1
plt.figure(figsize=(15,16))
for city in cities:
    #print(city)
    plt.subplot(3,1,pos)
    sns.barplot(x="day",y="Total",data=city,estimator=sum,ci=False)
    plt.title(f"the trend of city {city_name[pos-1]}",fontweight="bold")
    plt.xticks(rotation=90)
    pos=pos+1
```

```
C:\Users\Rahul\AppData\Local\Temp\ipykernel_15336\817640088.py:7: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=('ci', False)` for the same effec
t.

  sns.barplot(x="day",y="Total",data=city,estimator=sum,ci=False)
C:\Users\Rahul\AppData\Local\Temp\ipykernel_15336\817640088.py:7: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=('ci', False)` for the same effec
t.

  sns.barplot(x="day",y="Total",data=city,estimator=sum,ci=False)
C:\Users\Rahul\AppData\Local\Temp\ipykernel_15336\817640088.py:7: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=('ci', False)` for the same effec
t.

  sns.barplot(x="day",y="Total",data=city,estimator=sum,ci=False)
```



Observation : The peak selling days vary across branches: Mandalay's hot selling day falls on the 15th, Naypyitaw's on the 23rd, and Yangon's on the 19th. Leveraging these insights on each branch's high-traffic days can inform the development of valuable strategies.

find the total weekday and weekend sale of the company

In [53]:
```
df["weekday/weekend"]=df["day_name"].apply(lambda x:"weekend" if (x=="Saturday" o
```

In [54]:
```
sns.barplot(x="weekday/weekend",y="Total",data=df,estimator=sum,palette="Set2")
plt.yticks([50000,100000,150000,200000],["50K","100K","150K","200K"])
plt.title("Sales genrated in weekday/weekend",fontweight="bold");
```

In [55]:
```
df.groupby(["weekday/weekend"])["Total"].agg(["sum"])
```

Out[55]:

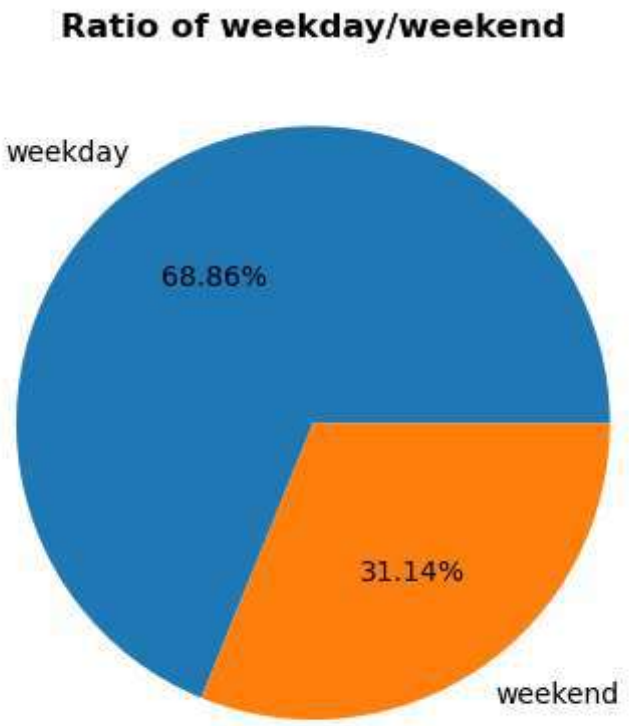
weekday/weekend	sum
weekday	222388.047
weekend	100578.702

Observation: Sales figures demonstrate a clear disparity between weekends and weekdays, with weekend sales totaling $100,578.702 and weekday sales amounting to 222,388.047. Understanding this contrast can guide strategic decisions tailored to capitalize on both weekday and weekend consumer behaviors.

find the ratio of total weekday and weekend sale of the company

In [56]:
```
plt.pie(x=df.groupby(["weekday/weekend"])["Total"].agg(["sum"])["sum"],labels=df.
plt.title("Ratio of weekday/weekend",fontweight="bold")
```

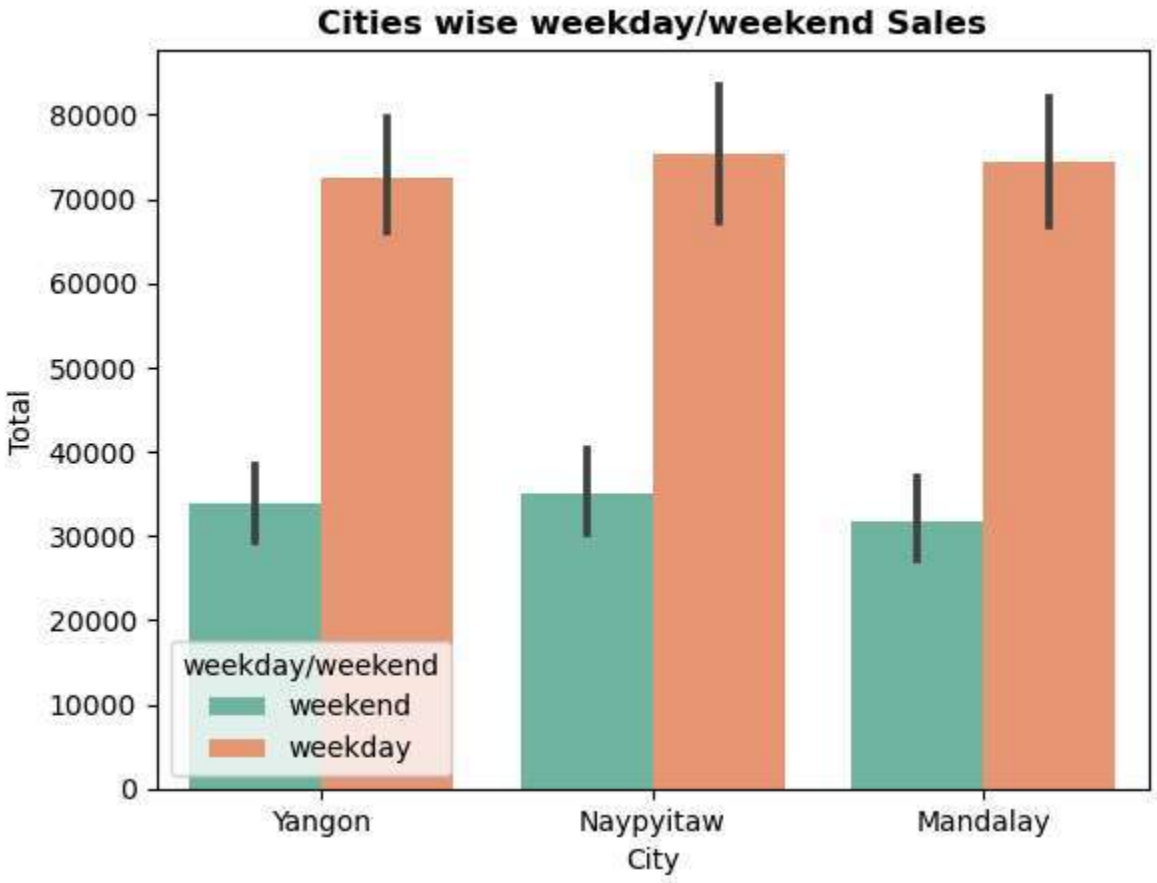
Out[56]:
```
Text(0.5, 1.0, 'Ratio of weekday/weekend')
```



Obsrevation : Weekdays account for the majority of sales at 68.8%, while weekends represent 31.2% of the total sales volume.

find the total weekday and weekend sale of the company of each city

In [57]:
```
sns.barplot(x="City",y="Total",data=df,estimator=sum,hue="weekday/weekend",palett
plt.title("Cities wise weekday/weekend Sales",fontweight="bold");
```



In [58]:
```
df.groupby(["City","weekday/weekend"])["Total"].agg(["sum"]).unstack()
```

Out[58]:

	sum	
weekday/weekend	**weekday**	**weekend**
City		
Mandalay	74497.920	31699.7520
Naypyitaw	75461.988	35106.7185
Yangon	72428.139	33772.2315

Observation: Across the three cities (Mandalay, Naypyitaw, and Yangon), the total sales during weekdays exceed those on weekends. Naypyitaw leads in weekday sales with 75,461.988, followed closely by Mandalay at 74,497.92, and Yangon at 72,428.139. Conversely, weekend sales are notably lower, with Mandalay recording 31,699.752, Naypyitaw at 35,106.7185, and Yangon at 33,772.2315. This pattern underscores the importance of weekday sales in driving overall revenue across the cities.

Find the total weekday and weekend sale of the company of each city in each month

In [59]:
```
df.groupby(["City","month","weekday/weekend"])["Total"].agg(["sum"]).unstack()
```

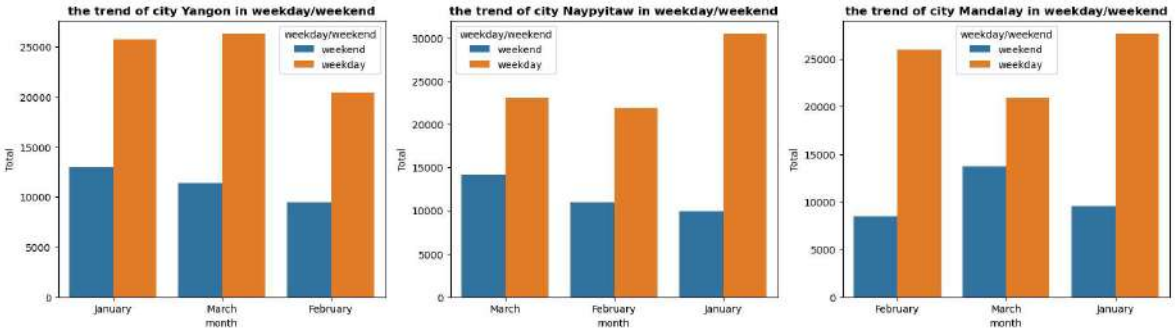
Out[59]:

		sum	
	weekday/weekend	weekday	weekend
City	month		
Mandalay	February	25964.4525	8459.8185
	January	27634.8870	9541.1715
	March	20898.5805	13698.7620
Naypyitaw	February	21950.5020	10984.4805
	January	30470.0550	9964.6260
	March	23041.4310	14157.6120
Yangon	February	20417.2395	9442.8810
	January	25736.4870	12944.6415
	March	26274.4125	11384.7090

In [60]:

```
city_name=["Yangon", 'Naypyitaw', 'Mandalay']
pos=1
plt.figure(figsize=(20,5))
for city in cities:
    #print(city)
    plt.subplot(1,3,pos)
    sns.barplot(x="month",y="Total",data=city,estimator=sum,ci=False,hue=df["week
    plt.title(f"the trend of city {city_name[pos-1]} in weekday/weekend",fontweig
    #plt.xticks(rotation=90)
    pos=pos+1
```

```
C:\Users\Rahul\AppData\Local\Temp\ipykernel_15336\2637785667.py:7: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=('ci', False)` for the same effec
t.

  sns.barplot(x="month",y="Total",data=city,estimator=sum,ci=False,hue=df["weekda
y/weekend"])
C:\Users\Rahul\AppData\Local\Temp\ipykernel_15336\2637785667.py:7: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=('ci', False)` for the same effec
t.

  sns.barplot(x="month",y="Total",data=city,estimator=sum,ci=False,hue=df["weekda
y/weekend"])
C:\Users\Rahul\AppData\Local\Temp\ipykernel_15336\2637785667.py:7: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=('ci', False)` for the same effec
t.

  sns.barplot(x="month",y="Total",data=city,estimator=sum,ci=False,hue=df["weekda
y/weekend"])
```

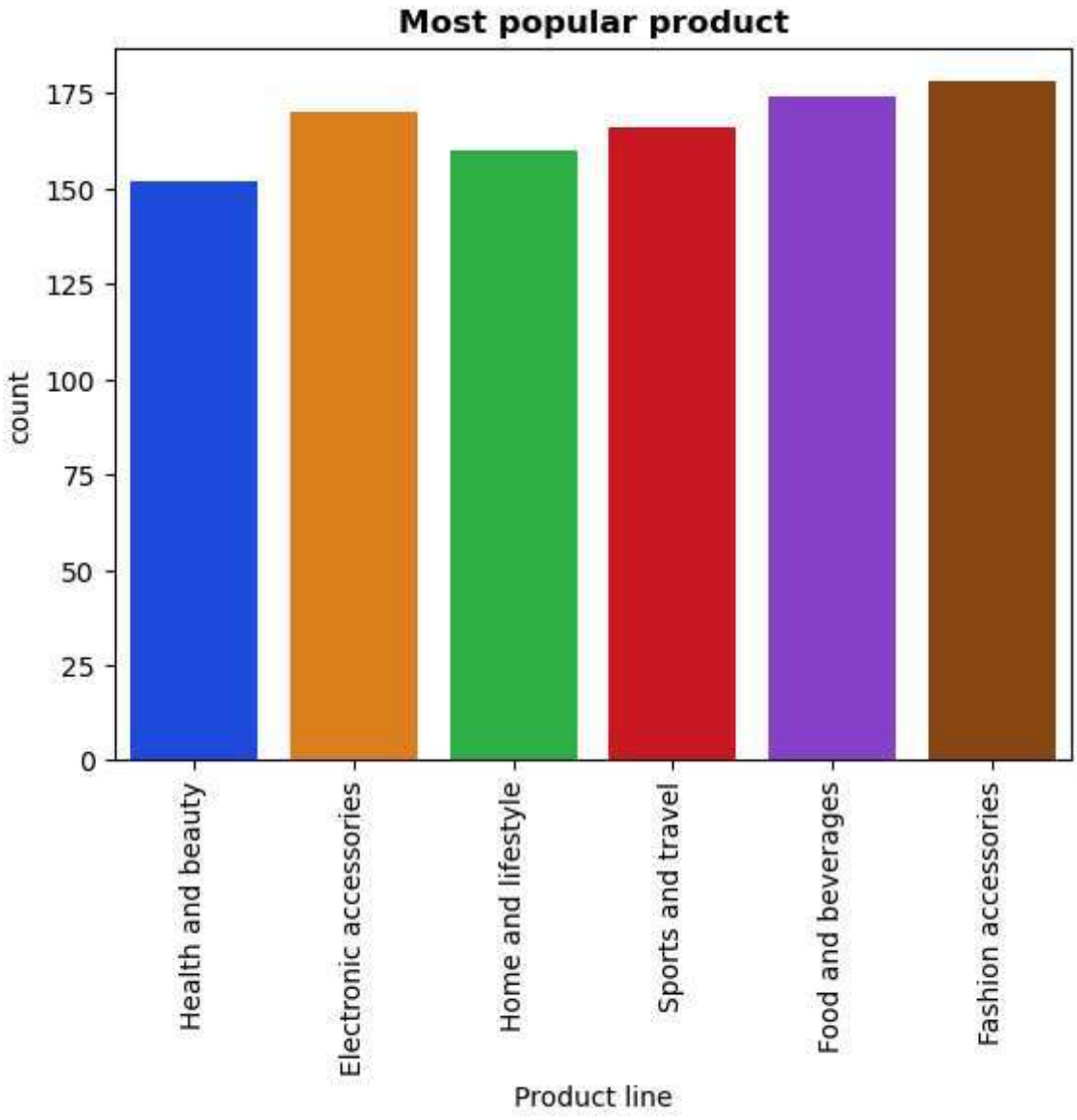


Observation: Examining sales data by city and month reveals varying patterns across different time periods. In Mandalay, January sees the highest weekday sales at 27,634.887, while February records the lowest weekday sales at 25,964.4525. Conversely, March witnesses the highest weekend sales in Mandalay at 13,698.762. Naypyitaw, on the other hand, experiences the highest weekday sales in January, totaling 30,470.055, and the highest weekend sales in March, reaching 14,157.612. In Yangon, January demonstrates the highest weekday sales at 25,736.487, while March shows the highest weekend sales at 11,384.709. These fluctuations underscore the importance of analyzing sales trends at a granular level to inform strategic decision-making and optimize revenue generation.

Most popular or demanding product of the company

In [61]:
```
sns.countplot(x="Product line",data=df,palette="bright")
plt.xticks(rotation=90);
plt.title("Most popular product",fontweight="bold")
```

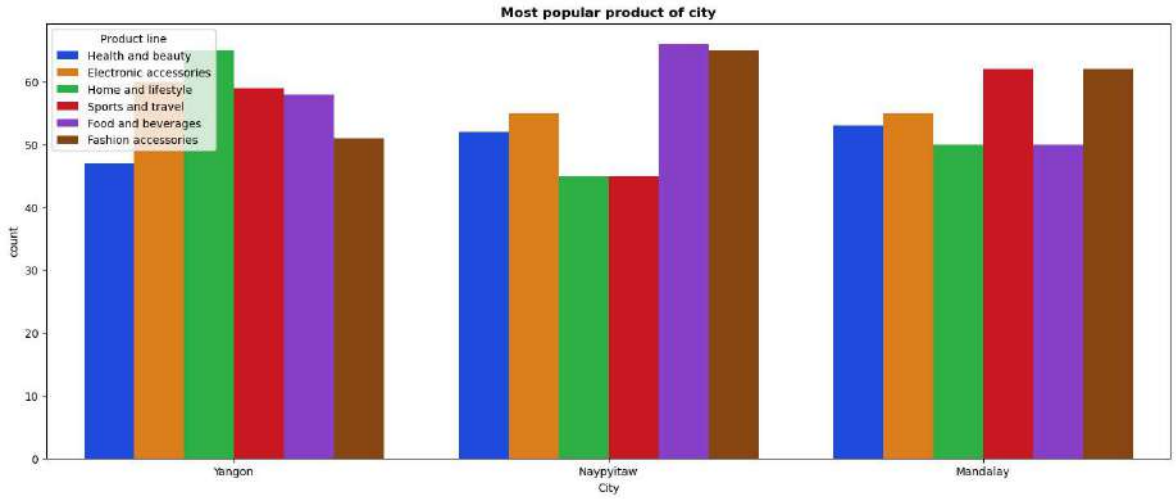
Out[61]:
```
Text(0.5, 1.0, 'Most popular product')
```



Observation : Fashion accessories is the most popular product of company

Find most popular or demanding product of each city/branch

In [62]:
```
plt.figure(figsize=(18,7))
sns.countplot(x="City",hue="Product line",data=df,palette="bright")
plt.title("Most popular product of city",fontweight="bold");
```

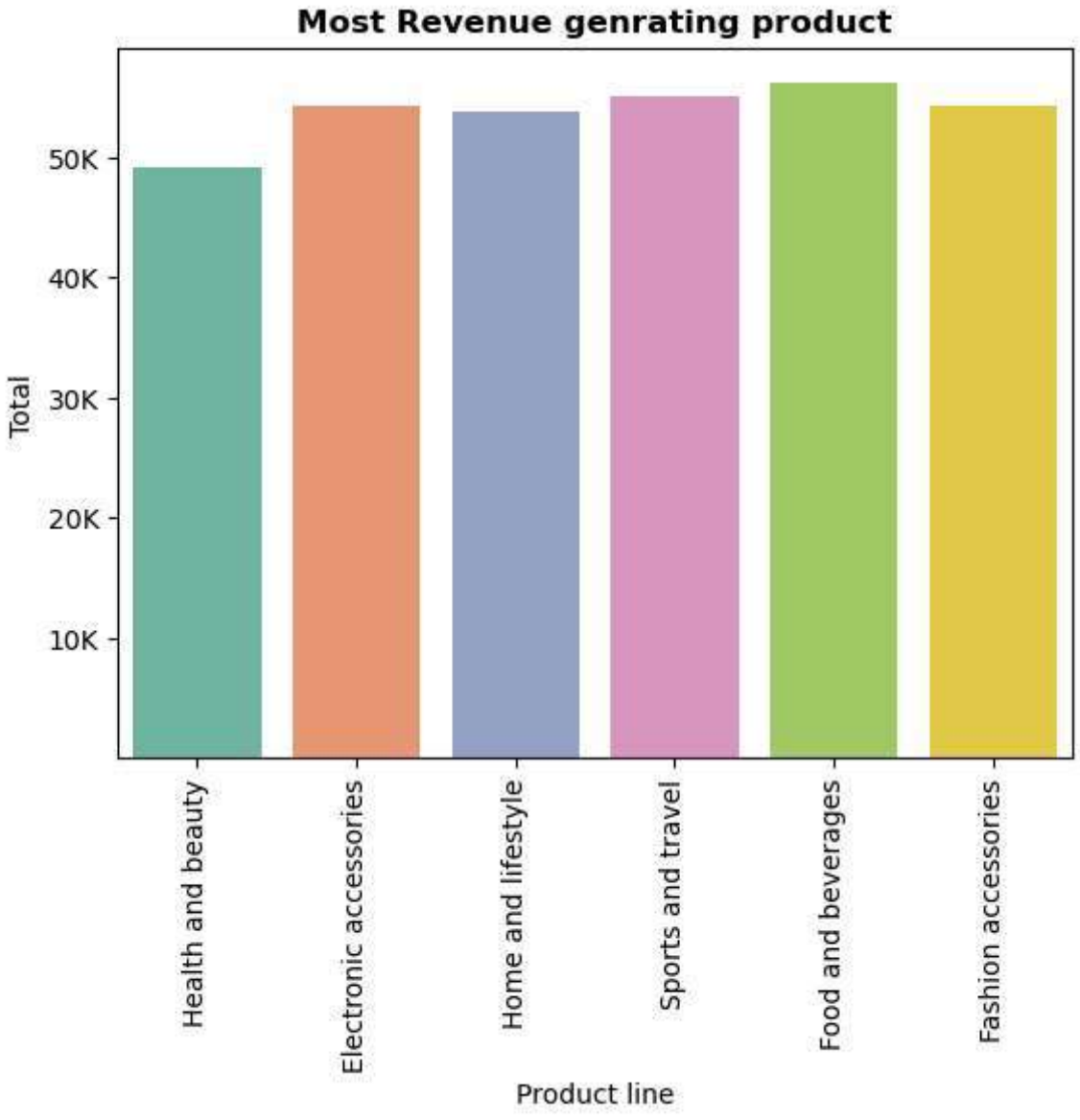


Observation : The preferred product categories vary across cities: Yangon favors Home and Lifestyle items, Naypyitaw leans towards Food and Beverages, and Mandalay boasts two prominent product categories: Sports and Travel, alongside Fashion Accessories.

Find the most revenue genrating product of company

In [63]:
```
sns.barplot(x="Product line",y="Total",data=df,estimator=sum,ci=False,palette="Se
plt.xticks(rotation=90)
plt.yticks(list(range(10000,60000,10000)),["10K","20K","30K","40K","50K"])
plt.title("Most Revenue genrating product",fontweight="bold");
```

```
C:\Users\Rahul\AppData\Local\Temp\ipykernel_15336\3131632383.py:1: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=('ci', False)` for the same effec
t.

  sns.barplot(x="Product line",y="Total",data=df,estimator=sum,ci=False,palette
="Set2")
```



In [64]:
```
df.groupby(["Product line"])["Total"].sum()
```

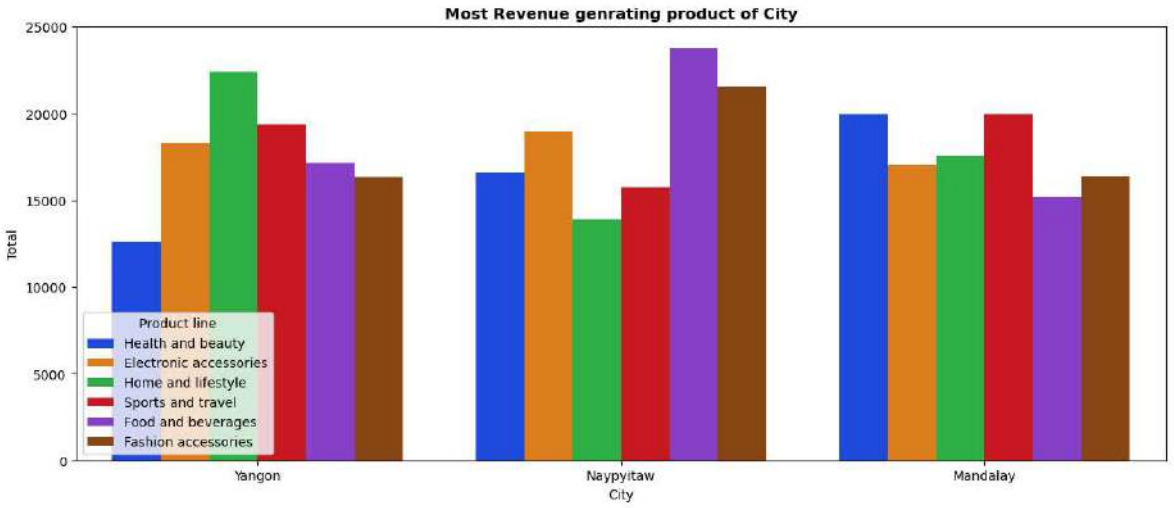
Out[64]:
```
Product line
Electronic accessories    54337.5315
Fashion accessories       54305.8950
Food and beverages        56144.8440
Health and beauty         49193.7390
Home and lifestyle        53861.9130
Sports and travel         55122.8265
Name: Total, dtype: float64
```

Observation : The most revenue genrating product is Food and beverages whose total sales is $56144.84

Most revenue genrating product of each city

In [65]:
```
plt.figure(figsize=(15,6))
sns.barplot(x="City",hue="Product line",y="Total",data=df,estimator=sum,ci=False,

plt.title("Most Revenue genrating product of City",fontweight="bold");
```

```
C:\Users\Rahul\AppData\Local\Temp\ipykernel_15336\2773156030.py:2: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=('ci', False)` for the same effec
t.

  sns.barplot(x="City",hue="Product line",y="Total",data=df,estimator=sum,ci=Fals
e,palette="bright")
```



In [66]:
```
df.groupby(["City","Product line"])["Total"].agg(["sum"]).unstack()
```

Out[66]:

	sum					
Product line	**Electronic accessories**	**Fashion accessories**	**Food and beverages**	**Health and beauty**	**Home and lifestyle**	**Sports and travel**
City						
Mandalay	17051.4435	16413.3165	15214.8885	19980.660	17549.1645	19988.1990
Naypyitaw	18968.9745	21560.0700	23766.8550	16615.326	13895.5530	15761.9280
Yangon	18317.1135	16332.5085	17163.1005	12597.753	22417.1955	19372.6995

Observation :

- The most revenue genrating product of Mandalay is Sports and travel with 19988.1990.
- The most revenue genrating product of Naypyitaw is Food and beverages with 23766.8550.
- The most revenue genrating product of Yangon is Home and lifestyle with 22417.1955.

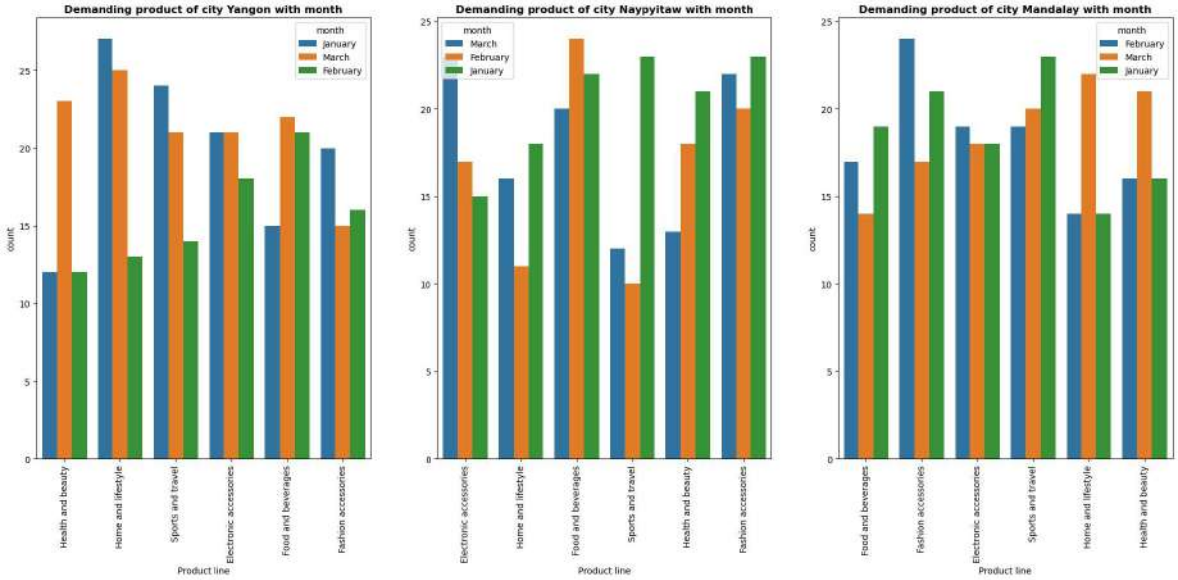
Find the most demanding product of each city of each month

In [67]:
```
df.groupby(["City","month"])["Product line"].agg(["value_counts"]).unstack()
```

Out[67]:

		value_counts					
	Product line	Electronic accessories	Fashion accessories	Food and beverages	Health and beauty	Home and lifestyle	Sports and travel
City	month						
Mandalay	February	19	24	17	16	14	19
	January	18	21	19	16	14	23
	March	18	17	14	21	22	20
Naypyitaw	February	17	20	24	18	11	10
	January	15	23	22	21	18	23
	March	23	22	20	13	16	12
Yangon	February	18	16	21	12	13	14
	January	21	20	15	12	27	24
	March	21	15	22	23	25	21

In [68]:
```
city_name=["Yangon", 'Naypyitaw', 'Mandalay']
pos=1
plt.figure(figsize=(23,9))
for city in cities:
    #print(city)
    plt.subplot(1,3,pos)
    sns.countplot(x="Product line",data=city,hue="month")
    plt.title(f"Demanding product of city {city_name[pos-1]} with month",fontweig
    plt.xticks(rotation=90)
    pos=pos+1
```



Observation :

- The plot represents a detailed insight into the demand for various product categories across different cities and months. The counts provided in the table reflect the sales volume or demand for each product line within Mandalay, Naypyitaw, and Yangon over the course of three months.
- Mandalay consistently demonstrates a balanced demand across multiple product categories, with Fashion Accessories and Sports and Travel showing notable popularity. Naypyitaw showcases a strong inclination towards Food and Beverages, maintaining consistently high counts across all months. Meanwhile, Yangon exhibits a diverse demand, with fluctuations observed across different product categories and months.

Find the monthly sale of each product of all branches

In [69]:
```
df.groupby(["City","month","Product line"])["Total"].agg(["sum"]).unstack()
```

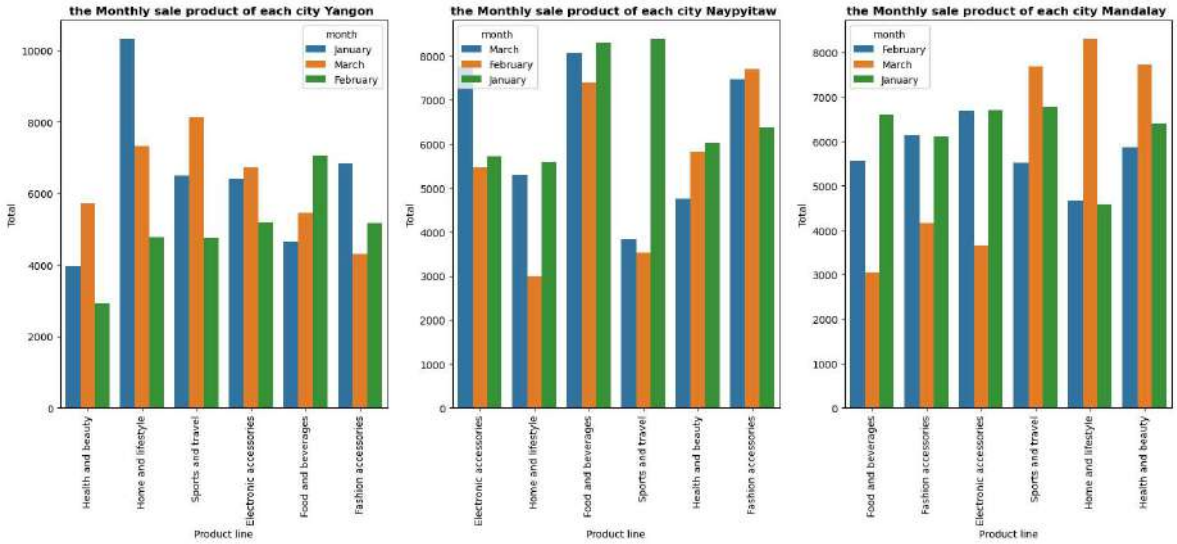
Out[69]:

		sum					
	Product line	Electronic accessories	Fashion accessories	Food and beverages	Health and beauty	Home and lifestyle	Sports and travel
City	month						
Mandalay	February	6686.2530	6137.1135	5554.8150	5856.4275	4659.8475	5529.8145
	January	6699.7770	6112.5960	6609.2775	6399.8865	4586.4420	6768.0795
	March	3665.4135	4163.6070	3050.7960	7724.3460	8302.8750	7690.3050
Naypyitaw	February	5473.8810	7699.1145	7391.3175	5830.3455	3002.9055	3537.4185
	January	5730.2385	6385.0290	8315.0235	6020.6895	5594.7045	8388.9960
	March	7764.8550	7475.9265	8060.5140	4764.2910	5297.9430	3835.5135
Yangon	February	5202.7710	5173.6335	7054.2255	2915.4825	4771.6305	4742.3775
	January	6401.2725	6847.4910	4646.2290	3962.5950	10313.5935	6509.9475
	March	6713.0700	4311.3840	5462.6460	5719.6755	7331.9715	8120.3745

In [70]:

```
city_name=["Yangon", 'Naypyitaw', 'Mandalay']
pos=1
plt.figure(figsize=(20,7))
for city in cities:
    #print(city)
    plt.subplot(1,3,pos)
    sns.barplot(x="Product line",y="Total",data=city,estimator=sum,ci=False,hue="
    plt.title(f"the Monthly sale product of each city {city_name[pos-1]} ",fontwe
    plt.xticks(rotation=90)
    pos=pos+1
```

```
C:\Users\Rahul\AppData\Local\Temp\ipykernel_15336\83427922.py:7: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=('ci', False)` for the same effec
t.

  sns.barplot(x="Product line",y="Total",data=city,estimator=sum,ci=False,hue="mo
nth")
C:\Users\Rahul\AppData\Local\Temp\ipykernel_15336\83427922.py:7: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=('ci', False)` for the same effec
t.

  sns.barplot(x="Product line",y="Total",data=city,estimator=sum,ci=False,hue="mo
nth")
C:\Users\Rahul\AppData\Local\Temp\ipykernel_15336\83427922.py:7: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=('ci', False)` for the same effec
t.

  sns.barplot(x="Product line",y="Total",data=city,estimator=sum,ci=False,hue="mo
nth")
```

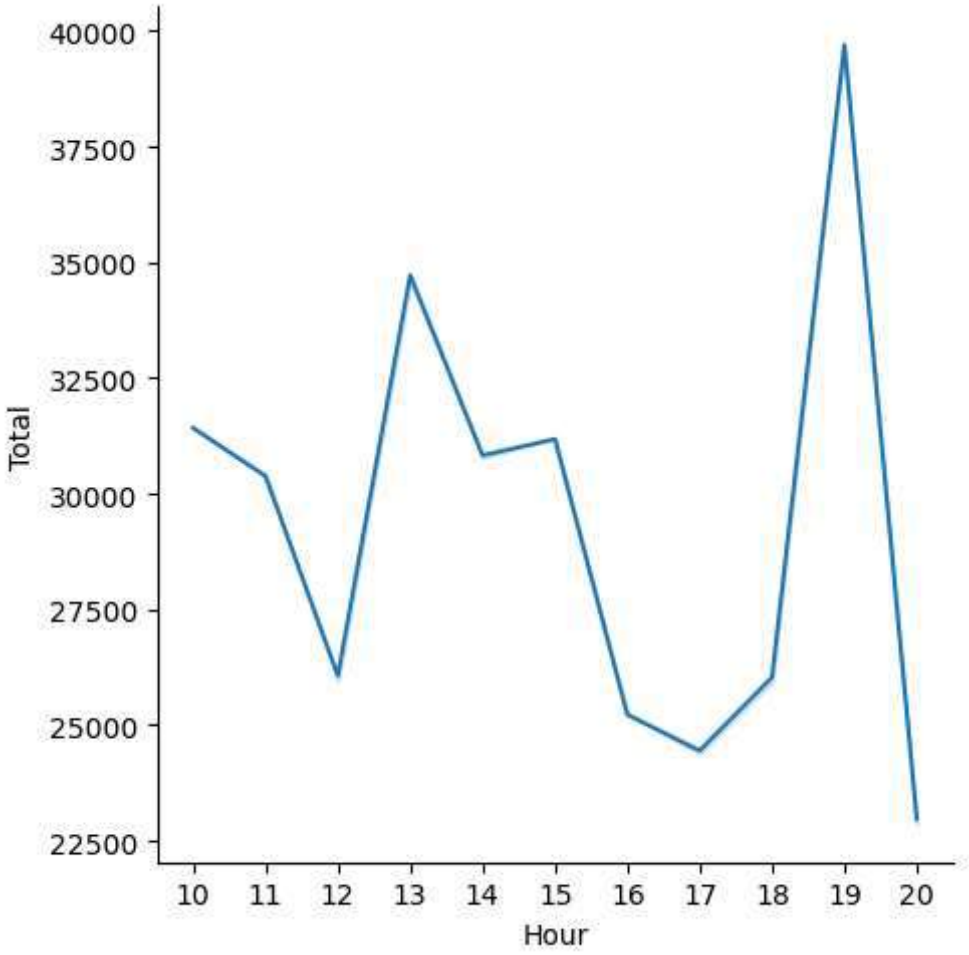


Observation : Analyzing the monthly sales performance of various product categories across different cities provides valuable insights into consumer preferences and market dynamics. Businesses can leverage this information to tailor their marketing efforts, optimize inventory levels, and capitalize on emerging trends to drive revenue growth. By understanding the unique characteristics of each market and monitoring seasonal variations, companies can make informed decisions to maximize profitability and enhance customer satisfaction.

Peak time of customer visit at supermarket

In [71]:
```
sns.relplot(x="Hour",y="Total",data=df,estimator=sum,ci=False,kind="line")
plt.xticks(list(range(10,21,1)));
```

```
C:\Users\Rahul\anaconda3\Lib\site-packages\seaborn\axisgrid.py:848: FutureWarnin
g:

The `ci` parameter is deprecated. Use `errorbar=('ci', False)` for the same effec
t.

  func(*plot_args, **plot_kwargs)
C:\Users\Rahul\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning:
The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



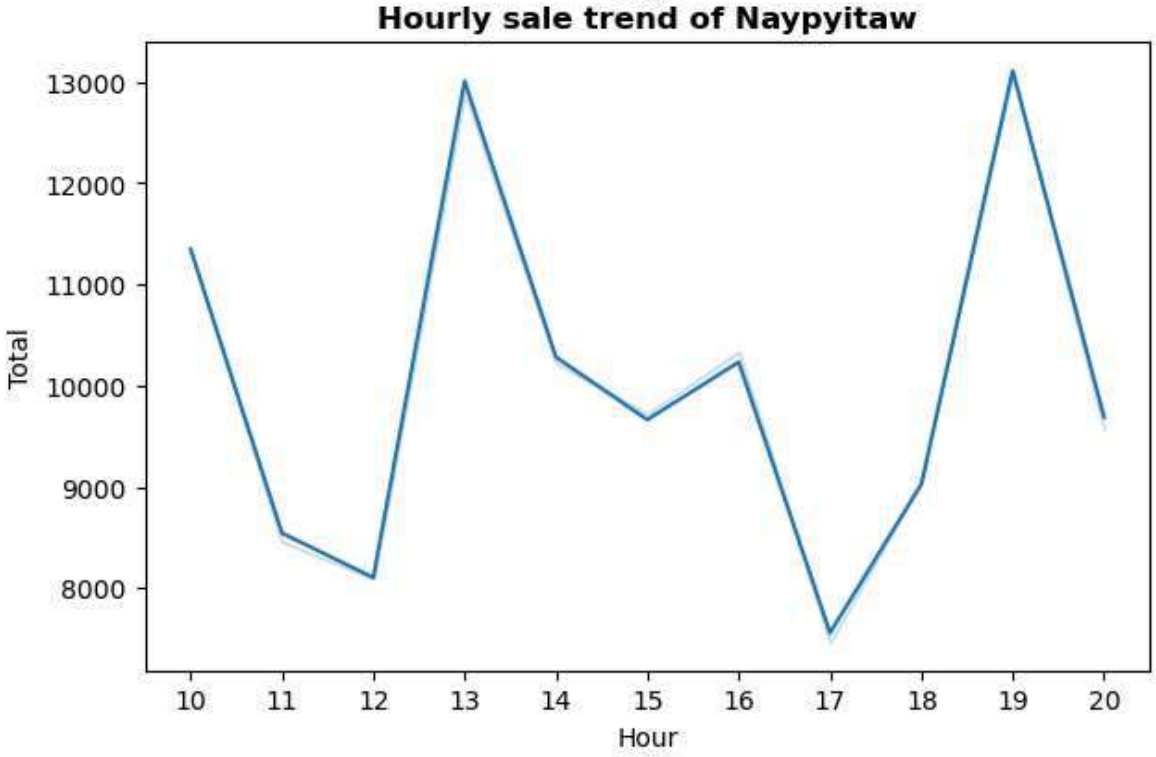
Observation :The peak hour of sales occurs consistently at **7:00 PM**. This observation suggests that there is a significant surge in consumer activity during this specific hour, potentially indicating a time when customers are more likely to make purchases. Understanding this peak hour can inform businesses about the optimal timing for promotions, staffing, and inventory management to capitalize on increased demand and maximize sales opportunities.

Peak hour of each city

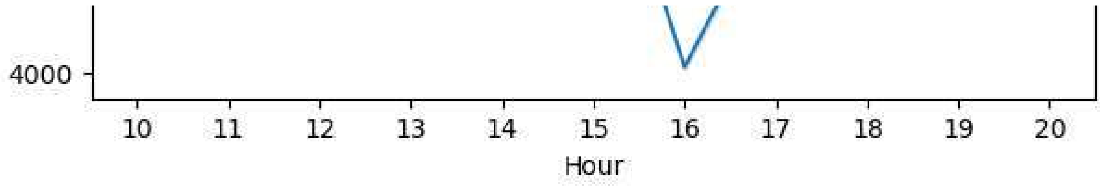
In [72]:
```
city_name=["Yangon", 'Naypyitaw', 'Mandalay']
pos=1
plt.figure(figsize=(7,15))
for city in cities:
    #print(city)
    plt.subplot(3,1,pos)
    sns.lineplot(x="Hour",y="Total",data=city,estimator=sum,ci=False)
    plt.title(f"Hourly sale trend of {city_name[pos-1]}",fontweight="bold")
    #plt.xticks(rotation=90)
    plt.xticks(list(range(10,21,1)))
    pos=pos+1
```

```
C:\Users\Rahul\AppData\Local\Temp\ipykernel_15336\2820857270.py:7: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=('ci', False)` for the same effec
t.

  sns.lineplot(x="Hour",y="Total",data=city,estimator=sum,ci=False)
C:\Users\Rahul\AppData\Local\Temp\ipykernel_15336\2820857270.py:7: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=('ci', False)` for the same effec
t.

  sns.lineplot(x="Hour",y="Total",data=city,estimator=sum,ci=False)
C:\Users\Rahul\AppData\Local\Temp\ipykernel_15336\2820857270.py:7: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=('ci', False)` for the same effec
t.

  sns.lineplot(x="Hour",y="Total",data=city,estimator=sum,ci=False)
```


Hourly sale trend of Yangon
Total
Hour
11000
10000
9000
8000
7000
6000
10
11
12
13
14
15
16
17
18
19
20


Hourly sale trend of Naypyitaw
Total
Hour
13000
12000
11000
10000
9000
8000
10
11
12
13
14
15
16
17
18
19
20


Hourly sale trend of Mandalay
Total
16000
14000
12000
10000
8000
6000



Observation :

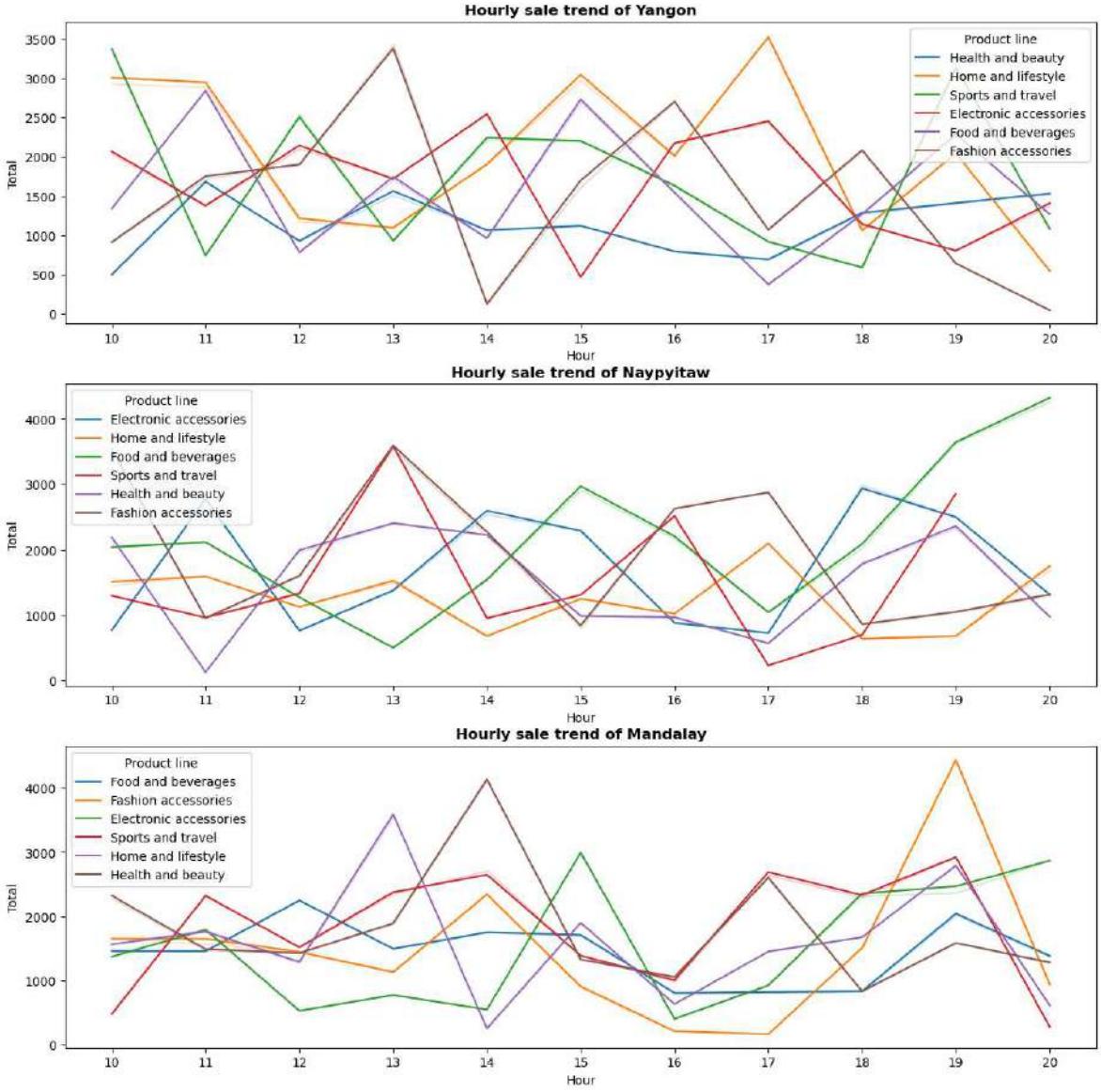
- The peak hour of sales of city Mandalay is **7 PM**.
- The peak hour of sales of city Naypyitaw is **1 and 7 PM**.
- The peak hour of sales of city Yangon is **11 AM**.
- This observation suggests that there is a significant surge in consumer activity during this specific hour, potentially indicating a time when customers are more likely to make purchases. Understanding this peak hour can inform businesses about the optimal timing for promotions, staffing, and inventory management to capitalize on increased demand and maximize sales opportunities.

Hourly sale trend of each city with product wise

In [73]:

```
city_name=["Yangon", 'Naypyitaw', 'Mandalay']
pos=1
plt.figure(figsize=(15,15))
for city in cities:
    #print(city)
    plt.subplot(3,1,pos)
    sns.lineplot(x="Hour",y="Total",data=city,estimator=sum,ci=False,hue="Product
    plt.title(f"Hourly sale trend of {city_name[pos-1]}",fontweight="bold")
    #plt.xticks(rotation=90)
    plt.xticks(list(range(10,21,1)))
    pos=pos+1
```

```
C:\Users\Rahul\AppData\Local\Temp\ipykernel_15336\3389984275.py:7: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=('ci', False)` for the same effec
t.

  sns.lineplot(x="Hour",y="Total",data=city,estimator=sum,ci=False,hue="Product l
ine")
C:\Users\Rahul\AppData\Local\Temp\ipykernel_15336\3389984275.py:7: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=('ci', False)` for the same effec
t.

  sns.lineplot(x="Hour",y="Total",data=city,estimator=sum,ci=False,hue="Product l
ine")
C:\Users\Rahul\AppData\Local\Temp\ipykernel_15336\3389984275.py:7: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=('ci', False)` for the same effec
t.

  sns.lineplot(x="Hour",y="Total",data=city,estimator=sum,ci=False,hue="Product l
ine")
```



Observation : The analysis of hourly sales trends across cities and product categories reveals distinct patterns. Peak sales hours vary by city, with unique fluctuations observed for each product category. Insights suggest opportunities for businesses to optimize staffing, promotions, and inventory management. Recommendations include capitalizing on peak hours and mitigating low activity periods. Further research could explore the impact of external factors on sales trends, refining strategies for maximizing sales effectiveness and enhancing overall performance.

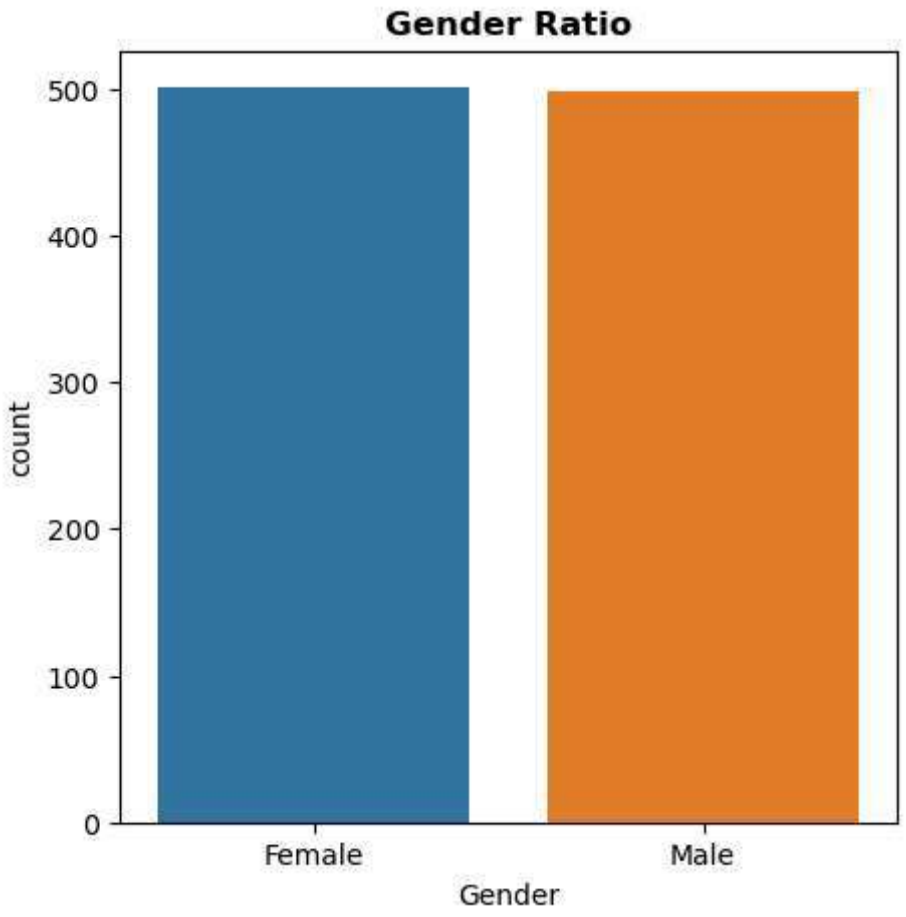
Find the total number of cutomers

In [74]:
```
print("Total Customer :",df["Gender"].count())
print(df["Gender"].value_counts())
```

```
Total Customer : 1000
Gender
Female    501
Male      499
Name: count, dtype: int64
```

In [75]:
```
plt.figure(figsize=(5,5))
sns.countplot(x="Gender",data=df)
plt.title("Gender Ratio",fontweight="bold")
```

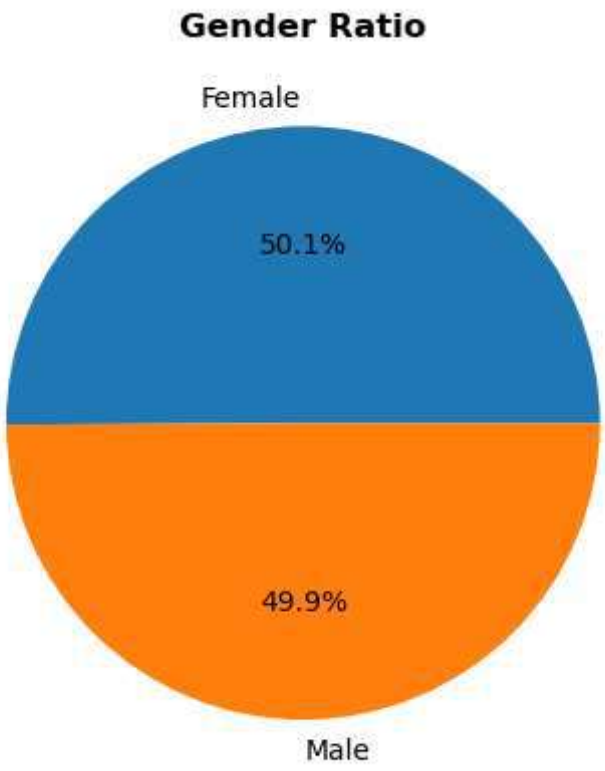
Out[75]: Text(0.5, 1.0, 'Gender Ratio')



Observation : Total number of cutomer is 1000 in which males are 499 and females are 501

Gender ratio of customers

In [76]:
```
plt.pie(df["Gender"].value_counts().values,labels=df["Gender"].value_counts().ind
plt.title("Gender Ratio",fontweight="bold");
```



Observation : In Gender ratio , ratio of female is 50.1% and male is 49.9%

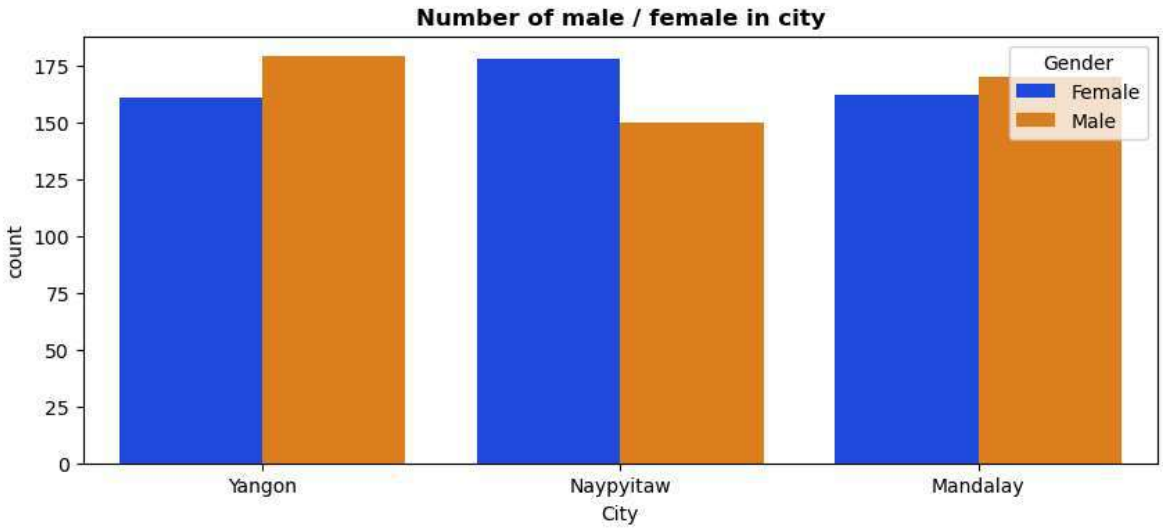
Total male and female customers in each city

In [77]:
```
df.groupby(["City"])["Gender"].agg(["value_counts"]).unstack()
```

Out[77]:

	value_counts	
Gender	Female	Male
City		
Mandalay	162	170
Naypyitaw	178	150
Yangon	161	179

In [78]:
```
plt.figure(figsize=(10,4))
sns.countplot(x="City",data=df,hue="Gender",palette="bright")
plt.title("Number of male / female in city",fontweight="bold");
```

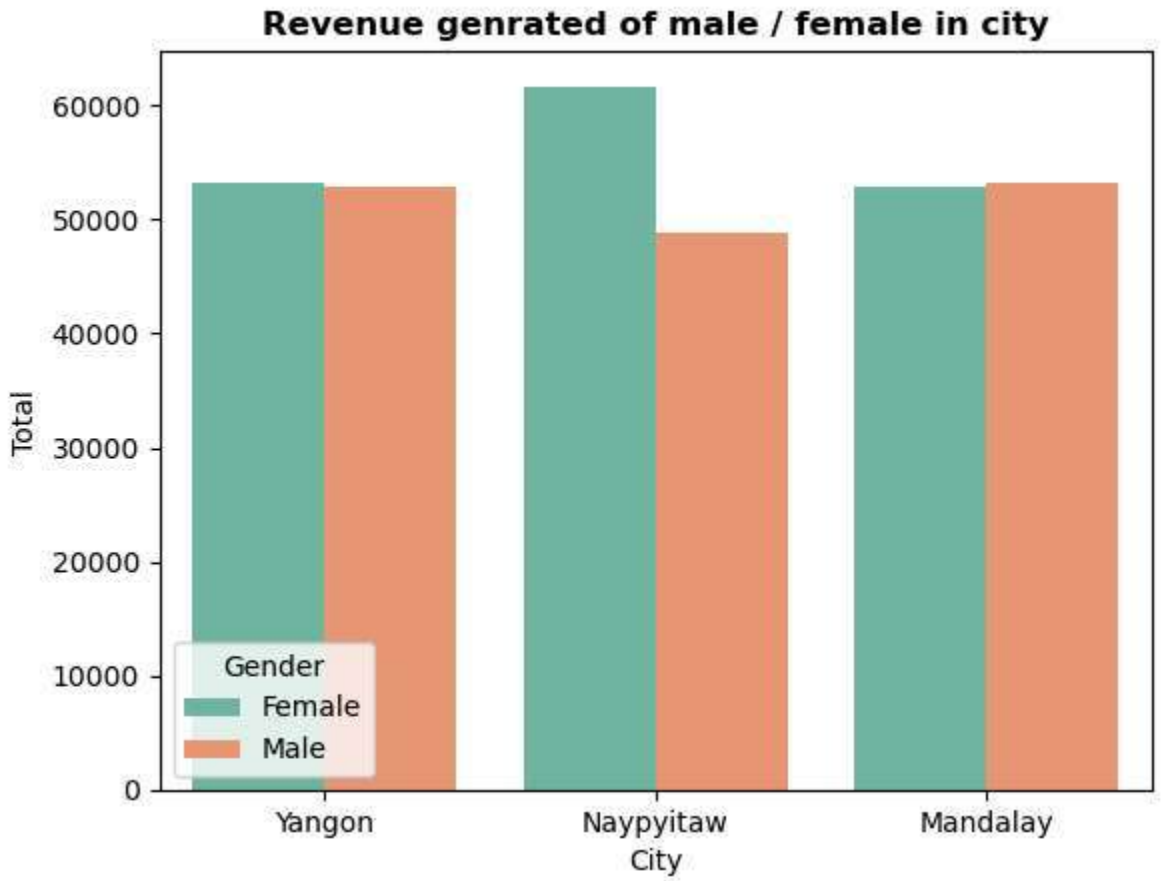


Observation : In Yangon and Mandalay, male customers predominate, whereas in Naypyitaw, female customers are more prevalent.

Who contributed most in each city

In [79]:
```
sns.barplot(x="City",y="Total",data=df,hue="Gender",estimator=sum,ci=False,palett
plt.title("Revenue genrated of male / female in city",fontweight="bold");
```

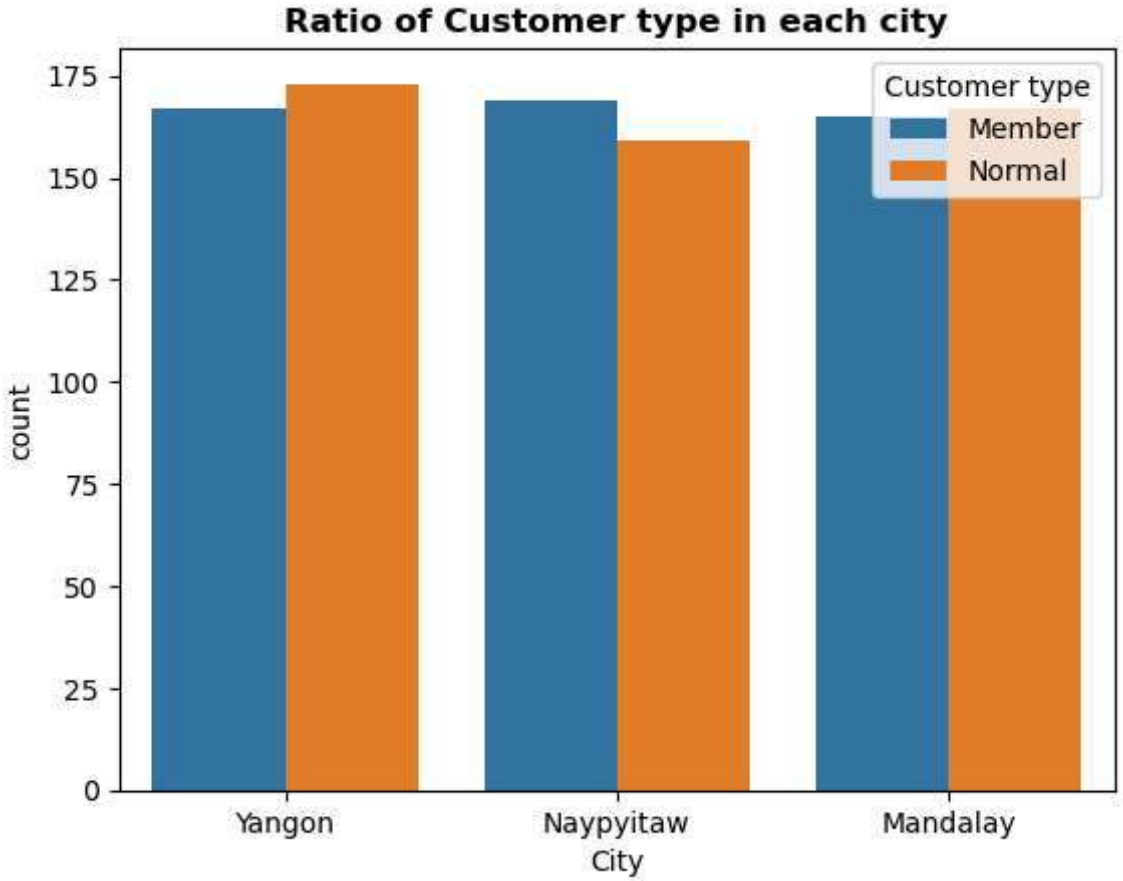
```
C:\Users\Rahul\AppData\Local\Temp\ipykernel_15336\3610870298.py:1: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=('ci', False)` for the same effec
t.

  sns.barplot(x="City",y="Total",data=df,hue="Gender",estimator=sum,ci=False,pale
tte="Set2")
```



Observation : In Yangon and Mandalay, the revenue generated by male and female customers is approximately equal, whereas in Naypyitaw, females lead in revenue generation.

Which type of customer visiting most in each city

In [80]:
```
sns.countplot(hue="Customer type",data=df,x="City")
plt.title("Ratio of Customer type in each city",fontweight="bold");
```



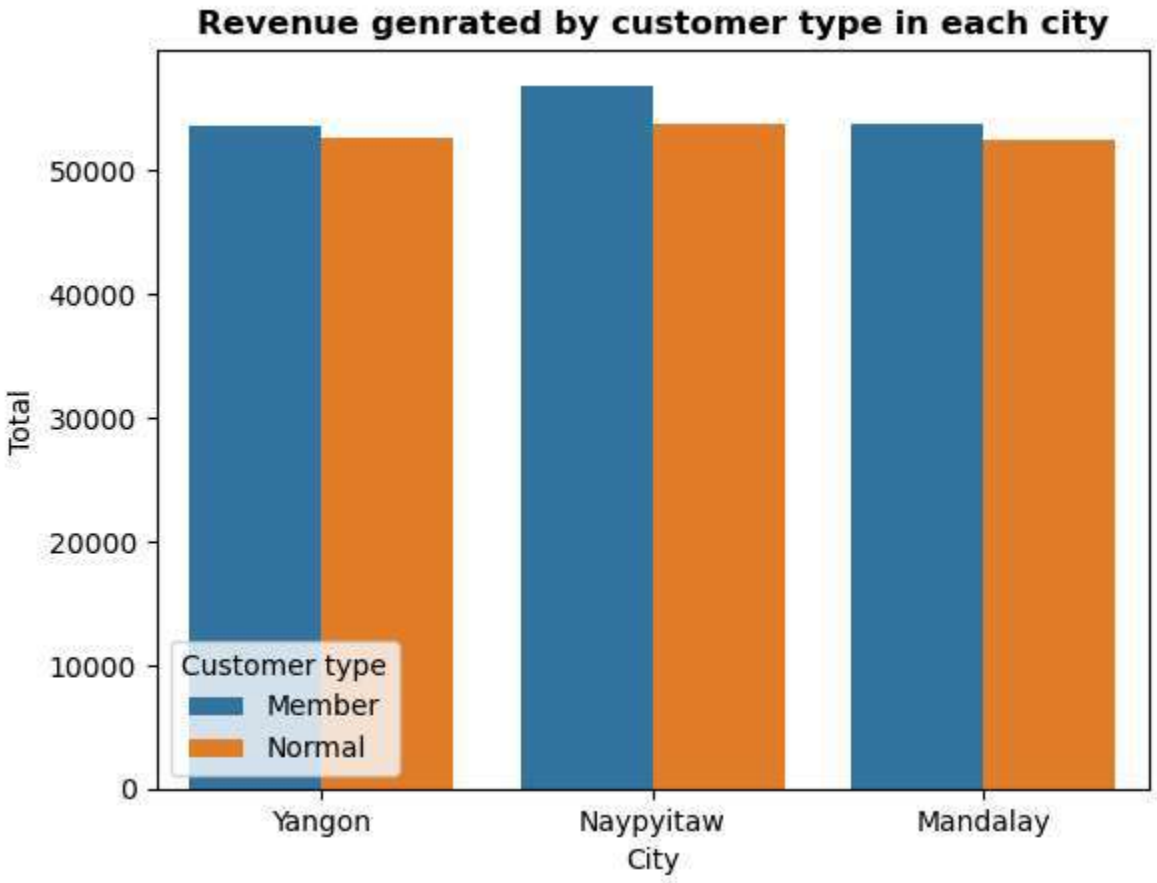
Observation :

- In yangon , Normal customer type are more than member type.
- In Naypyitaw, Member customer type are more than Normal type.
- In Mandalay,Both member and normal cutomer are aprrox equal.

which customer type genrate most business revenue

In [81]:
```
sns.barplot(x="City",hue="Customer type",y="Total",data=df,estimator=sum,ci=False
plt.title("Revenue genrated by customer type in each city",fontweight="bold");
```

```
C:\Users\Rahul\AppData\Local\Temp\ipykernel_15336\3395703966.py:1: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=('ci', False)` for the same effec
t.

  sns.barplot(x="City",hue="Customer type",y="Total",data=df,estimator=sum,ci=Fal
se)
```



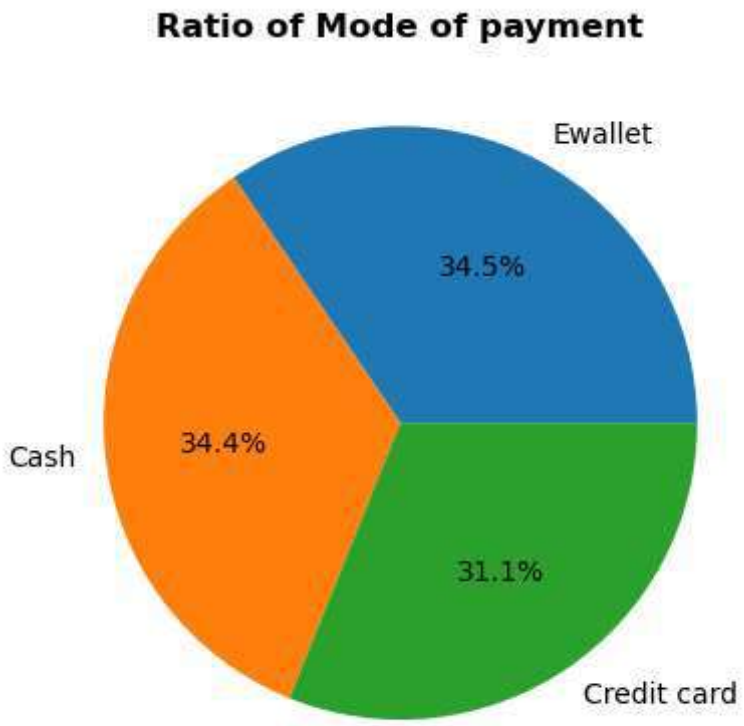
Observation : The revenue genrated by member type customer is more than the normal customer in each city.

Mode of payment ratio

In [82]:
```
df["Payment"].value_counts()
```

Out[82]:
```
Payment
Ewallet        345
Cash           344
Credit card    311
Name: count, dtype: int64
```

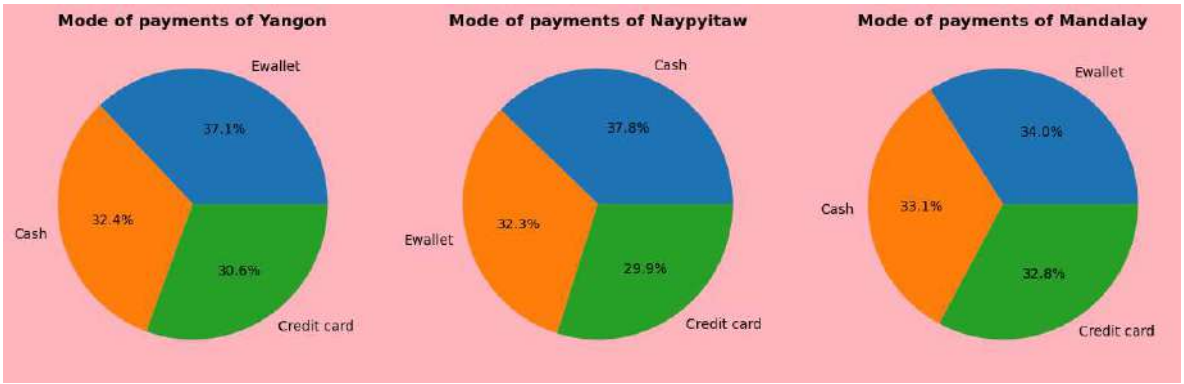
In [83]:
```
plt.pie(x=df["Payment"].value_counts().values,labels=df["Payment"].value_counts()
plt.title("Ratio of Mode of payment",fontweight="bold");
```



Observation : The distribution of payment methods shows similar ratios across all modes, with credit card payments being comparatively lower in frequency.

Mode of payment ratio of each city

In [84]:
```
city_name=["Yangon", 'Naypyitaw', 'Mandalay']
pos=1
plt.figure(figsize=(15,15),facecolor="lightpink")
for city in cities:
    plt.subplot(1,3,pos)
    plt.pie(x=city["Payment"].value_counts().values,labels=city["Payment"].value_
    plt.title(f"Mode of payments of {city_name[pos-1]}",fontweight="bold")
    pos=pos+1
```



Observation : In Naypyitaw, customers predominantly utilize cash as their preferred payment method. To foster growth and enhance customer engagement, a strategic shift towards encouraging credit card usage could be beneficial. Offering enticing incentives and promotions for credit card transactions can incentivize customers to adopt this mode of payment, thereby driving revenue growth and enhancing the overall customer experience.

Rating for cutomer satisfaction

Find the highest ,lowest ,average rating of company

In [85]:
```
df["Rating"].agg(["max","min","mean"])
```

Out[85]:
```
max     10.0000
min      4.0000
mean     6.9727
Name: Rating, dtype: float64
```

How many cutomer given maximum rating that is 10 ?

In [86]:
```
print(len(df[df["Rating"]==10]),"Customer gives 10 rating")
print(len(df[df["Rating"]==10])*100/len(df["Rating"]),"% Customer gives 10 rating
```

```
5 Customer gives 10 rating
0.5% Customer gives 10 rating
```

How many cutomer given minimum rating that is 4 ?

In [87]:
```
print(len(df[df["Rating"]==4]),"Customer gives 4 rating")
print(len(df[df["Rating"]==4])*100/len(df["Rating"]),"% Customer gives 4 rating",
```

```
11 Customer gives 4 rating
1.1% Customer gives 4 rating
```

How many cutomer given below or equal 5 rating ?

In [88]:
```
print(len(df[df["Rating"]<=5]),"Customer gives below or equal 5 rating")
print(len(df[df["Rating"]<=5])*100/len(df["Rating"]),"% Customer gives gives belo
```

```
174 Customer gives below or equal 5 rating
17.4% Customer gives gives below or equal 5 rating
```

rating of cutomers of each city

In [89]:
```
df.groupby(["City"])["Rating"].agg(["max","min","mean"])
```

Out[89]:

City	max	min	mean
Mandalay	10.0	4.0	6.818072
Naypyitaw	10.0	4.0	7.072866
Yangon	10.0	4.0	7.027059

Rating w.r.t each product

In [90]:
```
df.groupby(["Product line"])["Rating"].agg(["max","min","mean"])
```

Out[90]:

Product line	max	min	mean
Electronic accessories	10.0	4.0	6.924706
Fashion accessories	9.9	4.0	7.029213
Food and beverages	9.9	4.0	7.113218
Health and beauty	10.0	4.0	7.003289
Home and lifestyle	9.9	4.1	6.837500
Sports and travel	10.0	4.0	6.916265

Rating of each product of each city

In [91]:
```
df.groupby(["City","Product line"])["Rating"].agg(["max","min","mean"]).T
```

Out[91]:

City						Mandalay		
Product line	Electronic accessories	Fashion accessories	Food and beverages	Health and beauty	Home and lifestyle	Sports and travel	Electronic accessories	Fashion accessories
max	10.000000	9.900000	9.900	9.9	9.800	10.000000	9.800000	9.9
min	4.000000	4.100000	4.000	4.0	4.100	4.100000	4.100000	4.0
mean	7.116364	6.722581	6.994	7.1	6.516	6.509677	6.747273	7.4

Observation : The overall company rating reflects high satisfaction levels among customers, indicating positive perceptions of both product quality and pricing across all branches.

Thank YOU For Reading the Notebook!!!

In [ ]: