Data analysis of coffee bean sales in Saudi Arabia

Project period Starting date : 2025 03 10 Expected closing data 2025 05 06

Overview

Let's assume that a company, Saudi bean, is coffee bean supplier in Saudi Arabia. The sales manager would like to improve inventory management for 2025 and maximize sales and customer satisfaction.

Project Plan

Plan - create report at each week including next steps

1. decide what tools to use - Python,
2. data - need data containing sale amount, location, product. found data in kaggle or
3. time frame

Phase	Periods	Starting date	Ending date	Task
Plan	1 week	03 10	03-17	data, tools, sample size
Analyze	2 weeks	03 18	04 01	Data Cleaning, Descriptive Statistics
Construct	2 weeks	04 02	04 15	A/B Testing , Model Building
Execute	3 weeks	04 16	05 06	Summary, Presentation, Report

Analyze

1. Data Cleaning
2. Descriptive statistics
3. visualization (Tableu and Python) ==results==: find key insights to develop more

Construct

1. Hypothesis Test
2. Model building Results: make sure find better way to do it, check model accuracy

Execute

1. presentaion
2. Markdown
3. Output: summary, presentaion, code file with markdown, tableau interactive visualization

As a data Analyst, sampled 730 sales in cities of Saudi Arabia.

Data import and initial data cleaning

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

In [2]:
```
# read data file and assign it to df
path = r"C:\Users\soonn\Documents\GitHub\CoffeeBeanSales\DatasetForCoffeeSales2.
df = pd.read_csv(path) # load data and assign data frame to df
```

In [3]:
```
# initially check 5 rows of data to have an idea of data.
df.head()
```

Out[3]:

	Date	Customer_ID	City	Category	Product	Unit Price	Quantity	Sales Amount	Use
0	1/1/2023	32	Riyadh	coffee beans	Colombian	40	14	560	
1	1/2/2023	49	Abha	coffee beans	Costa Rica	35	17	595	
2	1/3/2023	75	Tabuk	coffee beans	Costa Rica	35	19	665	
3	1/4/2023	80	Abha	coffee beans	Ethiopian	45	1	45	
4	1/5/2023	78	Hail	coffee beans	Colombian	40	46	1840	

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 730 entries, 0 to 729
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   Date             730 non-null    object
 1   Customer_ID      730 non-null    int64 
 2   City             730 non-null    object
 3   Category         730 non-null    object
 4   Product          730 non-null    object
 5   Unit Price       730 non-null    int64 
 6   Quantity         730 non-null    int64 
 7   Sales Amount     730 non-null    int64 
 8   Used_Discount    730 non-null    bool  
 9   Discount_Amount  730 non-null    int64 
 10  Final Sales      730 non-null    int64 
dtypes: bool(1), int64(6), object(4)
memory usage: 57.9+ KB
```

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

Out[5]:
```
(730, 11)
```

In [6]:
```
df.size
```

Out[6]:
```
8030
```

Insight

- Data type of 'Date' is objects. Changing data type to datetime is better.
- There is no missing values in the data and it has shpe of 730 rows and 11 columns.
- there is 8030 values in the data set.

In [7]:
```
df.describe(include = 'all')
```

Out[7]:

	Date	Customer_ID	City	Category	Product	Unit Price	Quantity	
count	730	730.000000	730	730	730	730.000000	730.000000	73
unique	730	NaN	10	1	5	NaN	NaN	
top	12/30/2024	NaN	Hail	coffee beans	Costa Rica	NaN	NaN	
freq	1	NaN	87	730	165	NaN	NaN	
mean	NaN	51.669863	NaN	NaN	NaN	36.794521	26.080822	9
std	NaN	29.014339	NaN	NaN	NaN	4.955104	14.480971	5
min	NaN	1.000000	NaN	NaN	NaN	30.000000	1.000000	
25%	NaN	27.250000	NaN	NaN	NaN	35.000000	14.000000	4
50%	NaN	52.000000	NaN	NaN	NaN	35.000000	27.000000	9
75%	NaN	77.000000	NaN	NaN	NaN	40.000000	39.000000	14
max	NaN	100.000000	NaN	NaN	NaN	45.000000	49.000000	22

Insight

- 'City' has 10 unique values, which means 10 cities are sampled and the city 'Hail' counted 87 showing in 730 cities.
- 'Category' contain only one unique value,'coffee_beans'.
- 'Product' has five different names and Costa Rica is the most frequent product in five products.
- 'Unit Price' range is from 30 to 45. the spread is narrow.
- 'Quantity' range is from 1 to 49. this spread is wide compare to unit price.
- Almost half of sales have discounts on sales.

- Sold in discounted price, so final sales price is settled.

In [8]:
```
# import the second data set
path = r"C:\Users\soonn\Documents\GitHub\CoffeeBeanSales\saudi_cities_geocoding.
df2 = pd.read_csv(path)
```

In [9]:
```
# take a look of data structure
df2.head(10)
```

Out[9]:

	City	Latitude	Longitude
0	Riyadh	24.7136	46.6753
1	Jeddah	21.2854	39.2376
2	Mecca	21.3891	39.8579
3	Medina	24.5247	39.5692
4	Dammam	26.3927	49.9777
5	Khobar	26.2172	50.1971
6	Abha	18.2465	42.5117
7	Tabuk	28.3835	36.5662
8	Hail	27.5114	41.7208
9	Buraidah	26.3259	43.9740

Insight

the dataset contains geographical information of 10 cities in Saudi Arabia. will combine df and df2 into one data set.

In [10]:
```
# create a new data 'df_combined' by merging df and df2 on the same city name.
df_combined = df.merge(df2, on = 'City')
```

In [11]:
```
df_combined.head(10)
```

Out[11]:

	Date	Customer_ID	City	Category	Product	Unit Price	Quantity	Sales Amount
0	1/1/2023	32	Riyadh	coffee beans	Colombian	40	14	560
1	1/2/2023	49	Abha	coffee beans	Costa Rica	35	17	595
2	1/3/2023	75	Tabuk	coffee beans	Costa Rica	35	19	665
3	1/4/2023	80	Abha	coffee beans	Ethiopian	45	1	45
4	1/5/2023	78	Hail	coffee beans	Colombian	40	46	1840
5	1/6/2023	1	Khobar	coffee beans	Colombian	40	32	1280
6	1/7/2023	95	Buraidah	coffee beans	Brazilian	30	1	30
7	1/8/2023	27	Medina	coffee beans	Colombian	40	33	1320
8	1/9/2023	73	Hail	coffee beans	Guatemala	35	47	1645
9	1/10/2023	82	Dammam	coffee beans	Costa Rica	35	4	140

In [12]:
```
df_combined.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 730 entries, 0 to 729
Data columns (total 13 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Date             730 non-null    object 
 1   Customer_ID      730 non-null    int64  
 2   City             730 non-null    object 
 3   Category         730 non-null    object 
 4   Product          730 non-null    object 
 5   Unit Price       730 non-null    int64  
 6   Quantity         730 non-null    int64  
 7   Sales Amount     730 non-null    int64  
 8   Used_Discount    730 non-null    bool   
 9   Discount_Amount  730 non-null    int64  
 10  Final Sales      730 non-null    int64  
 11  Latitude         730 non-null    float64
 12  Longitude        730 non-null    float64
dtypes: bool(1), float64(2), int64(6), object(4)
memory usage: 69.3+ KB
```

In [13]:
```
# check data duplication.
df_duplicated = df_combined[df_combined.duplicated()]
df_duplicated
```

```
# add month variable
df['Date'] = pd.to_datetime(df['Date']) # change datetime

df['month'] =df['Date'].dt.month
```

Insight

- The combined dataset does not have duplication.

In [14]:
```
# change data type of 'date'
df_combined['Date'] = pd.to_datetime(df_combined['Date'])
df_combined.info() # check data type changed successfully.
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 730 entries, 0 to 729
Data columns (total 13 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Date             730 non-null    datetime64[ns]
 1   Customer_ID      730 non-null    int64
 2   City             730 non-null    object
 3   Category         730 non-null    object
 4   Product          730 non-null    object
 5   Unit Price       730 non-null    int64
 6   Quantity         730 non-null    int64
 7   Sales Amount     730 non-null    int64
 8   Used_Discount    730 non-null    bool
 9   Discount_Amount  730 non-null    int64
 10  Final Sales      730 non-null    int64
 11  Latitude         730 non-null    float64
 12  Longitude        730 non-null    float64
dtypes: bool(1), datetime64[ns](1), float64(2), int64(6), object(3)
memory usage: 69.3+ KB
```

Optimizing inventory

Relevant variables

- Product - categorical variable
- Quantity - continuous variavble
- City - categorical variable

Reasoning

I would like to build a model with one continuous dependent variable and two categorical independent variables. Dependent variable : 'Quantity' Independent variables : 'City', 'Product' I will perform Multivariate ANOVA.

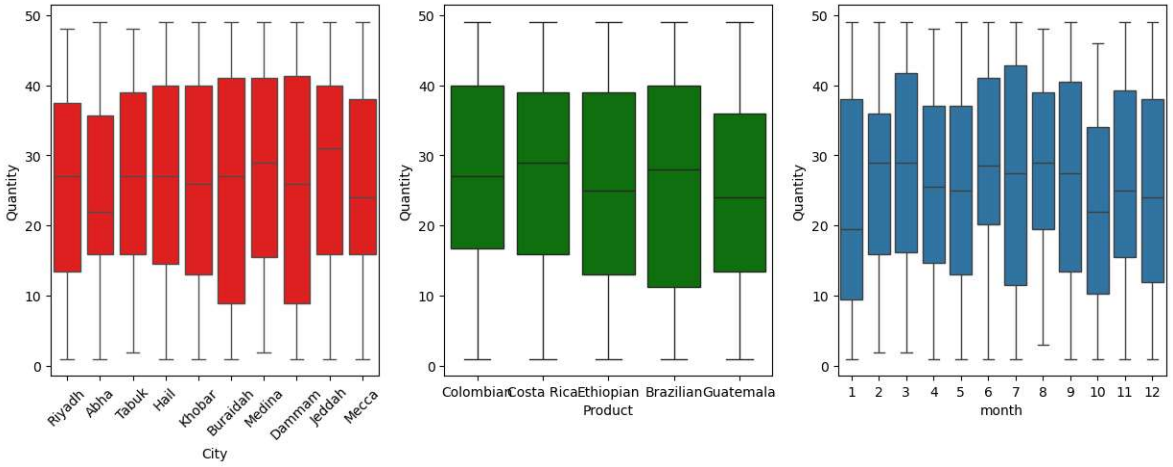
Steps

1. Build hyphothesis

2. fit the model
3. check model assumptions
4. evaluate model

In [15]:
```
# detect any difference on city-quantiy, product-quantity, product-month
fig, axes = plt.subplots(1,3, figsize = (15,5))
sns.boxplot(data = df, x = 'City', y = 'Quantity' , ax = axes[0], color = 'red')
axes[0].tick_params(axis='x', rotation=45)
sns.boxplot(data = df, x = 'Product', y = 'Quantity',ax = axes[1], color = 'gree

sns.boxplot(data = df, x = 'month', y = 'Quantity', ax = axes[2])
```

Out[15]: <Axes: xlabel='month', ylabel='Quantity'>



In [16]:
```
# Build hypothesis
h_null = 'The quantity of coffee bean sales is the same in different city'
h_alt = 'The quantity of coffee bean sleas is different in different city'
print("Null hypothesis : ", h_null)
print('Althernative hypothesis : ', h_alt)
```

```
Null hypothesis :  The quantity of coffee bean sales is the same in different cit
y
Althernative hypothesis :  The quantity of coffee bean sleas is different in diff
erent city
```

In [17]:
```
# build formula and holdout sample
from sklearn.model_selection import train_test_split
from statsmodels.formula.api import ols

ols_formula = "Quantity~ C(City)"
x = df['City']
y = df['Quantity']

OLS = ols(formula = ols_formula, data = df)
model = OLS.fit()
print(model.summary())

from statsmodels.stats.anova import anova_lm

anova_lm(model, typ = 2)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:               Quantity   R-squared:                       0.005
Model:                            OLS   Adj. R-squared:                 -0.008
Method:                 Least Squares   F-statistic:                    0.3823
Date:                Thu, 27 Mar 2025   Prob (F-statistic):              0.944
Time:                        11:35:01   Log-Likelihood:                -2984.8
No. Observations:                 730   AIC:                             5990.
Df Residuals:                     720   BIC:                             6035.
Df Model:                           9
Covariance Type:            nonrobust
================================================================================
======
                           coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
------
Intercept               24.4091      1.789     13.642      0.000      20.896
27.922
C(City)[T.Buraidah]     1.5184      2.503      0.607      0.544      -3.395
6.432
C(City)[T.Dammam]       0.8826      2.477      0.356      0.722      -3.981
5.746
C(City)[T.Hail]         2.5219      2.373      1.063      0.288      -2.137
7.181
C(City)[T.Jeddah]       3.3182      2.438      1.361      0.174      -1.469
8.105
C(City)[T.Khobar]       0.7279      2.469      0.295      0.768      -4.120
5.575
C(City)[T.Mecca]        1.7727      2.438      0.727      0.467      -3.015
6.560
C(City)[T.Medina]       3.0275      2.486      1.218      0.224      -1.852
7.907
C(City)[T.Riyadh]       1.1099      2.424      0.458      0.647      -3.649
5.869
C(City)[T.Tabuk]        1.4384      2.604      0.552      0.581      -3.675
6.552
==============================================================================
Omnibus:                      539.345   Durbin-Watson:                   1.973
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               45.298
Skew:                          -0.086   Prob(JB):                     1.46e-10
Kurtosis:                       1.792   Cond. No.                         11.4
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

Out[17]:

	sum_sq	df	F	PR(>F)
C(City)	727.007237	9.0	0.382275	0.943814
Residual	152143.224270	720.0	NaN	NaN

All pvalue of categorical variables, city have high and this indicate each coefficient is not statisticaly significant and fail to reject the hypothesis the quantity of each city is the same.

In [18]:
```
# build hypothesis
h_null = "The sales quantity of coffee bean is the same in product"
h_alt = "The sales quantity of coffee bean is different in product"
print("Null hypothesis : ", h_null)
print('Althernative hypothesis : ', h_alt)
```

```
Null hypothesis :  The sales quantity of coffee bean is the same in product
Althernative hypothesis :  The sales quantity of coffee bean is different in prod
uct
```

In [19]:
```
# build formula and ols
ols_formula = "Quantity ~ C(Product)"
OLS = ols(formula = ols_formula, data = df)
model = OLS.fit()
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:               Quantity   R-squared:                       0.003
Model:                            OLS   Adj. R-squared:                 -0.002
Method:                 Least Squares   F-statistic:                    0.6105
Date:                Thu, 27 Mar 2025   Prob (F-statistic):              0.655
Time:                        11:35:01   Log-Likelihood:                -2985.3
No. Observations:                 730   AIC:                             5981.
Df Residuals:                     725   BIC:                             6004.
Df Model:                           4
Covariance Type:            nonrobust
================================================================================
===========
                               coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
-----------
Intercept                   25.9726      1.200     21.649      0.000      23.617
28.328
C(Product)[T.Colombian]      0.8169      1.680      0.486      0.627      -2.481
4.115
C(Product)[T.Costa Rica]     1.0092      1.647      0.613      0.540      -2.224
4.243
C(Product)[T.Ethiopian]     -0.2070      1.755     -0.118      0.906      -3.653
3.239
C(Product)[T.Guatemala]     -1.3323      1.718     -0.776      0.438      -4.705
2.040
==============================================================================
Omnibus:                      548.281   Durbin-Watson:                   1.985
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               45.503
Skew:                          -0.089   Prob(JB):                     1.32e-10
Kurtosis:                       1.790   Cond. No.                         5.84
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

All pvalue of categorical variables, product have high and this indicate each coefficient is not statisticaly significant and fail to reject the hypothesis the quantity of each product is the same.

In [20]:
```
# adding mean of city,month,product
df['city_product_month'] = df['City'].astype(str) + ' ' + df['Product'].astype(s
grouped = df.groupby('city_product_month').mean(numeric_only = True)[['Quantity'
grouped_dict = grouped.to_dict()
grouped_dict = grouped_dict['Quantity']
df['mean_Quantity'] = df['city_product_month']
df['mean_Quantity'] = df['mean_Quantity'].map(grouped_dict)

df_corr = df[['Unit Price', 'Quantity', 'Discount_Amount','mean_Quantity']].corr
sns.heatmap(df_corr)
```

Out[20]: <Axes: >

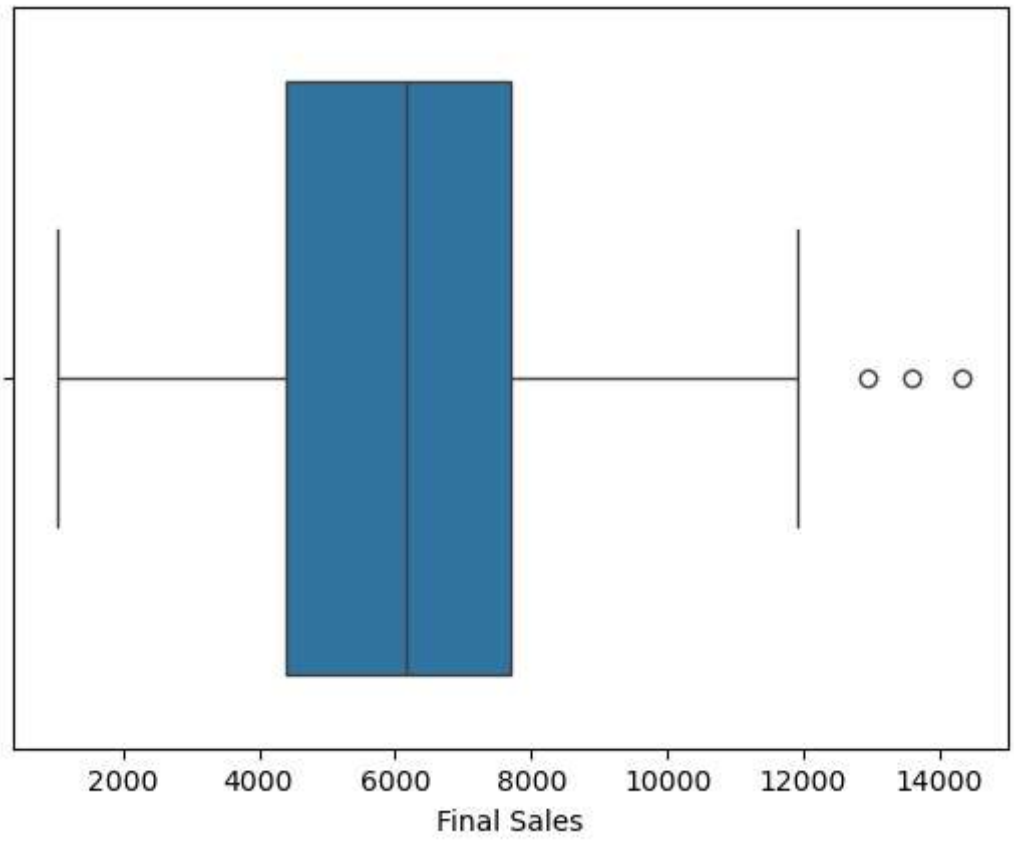
Customer_ID and Sales

1. Check Customer_ID and find big customers
2. check big customers' sales, frequency, order size and discount amount
3. build strategies for big customers.

In [21]:
```
df_sales_ID = df_combined[['Customer_ID', 'Final Sales']].groupby(['Customer_ID'
df_sales_ID = df_sales_ID.sort_values(by = 'Final Sales', ascending = False)
print(df_sales_ID)
print(df_sales_ID.describe())
sns.boxplot(data = df_sales_ID, x = 'Final Sales')
```

```
    Customer_ID  Final Sales
1             2        14334
96           97        13602
81           82        12932
80           81        11903
49           50        11285
..          ...          ...
11           12         2813
32           33         2548
19           20         1826
86           87         1440
69           70         1050

[100 rows x 2 columns]
       Customer_ID   Final Sales
count   100.000000    100.000000
mean     50.500000   6296.480000
std      29.011492   2586.040359
min       1.000000   1050.000000
25%      25.750000   4403.500000
50%      50.500000   6161.000000
75%      75.250000   7687.250000
max     100.000000  14334.000000
```

Out[21]: <Axes: xlabel='Final Sales'>



In [22]:
```
# there are three big customers
# a special promotion on the big three will and make a deal with them will boost
# find frequency of orders and date will help to prepare stocks to supply faster
# calculate discounts rate
```

In [23]:
```
big_three = df_sales_ID[df_sales_ID['Final Sales'] >= 12000] # make a data frame
big_three_ID = big_three['Customer_ID'].tolist() # save customer ids of top thre
big_three_df = df_combined[df_combined['Customer_ID'].isin(big_three_ID)] # buil
```

```
# add year and month to dataset
big_three_df['year'] = big_three_df['Date'].dt.year
big_three_df['month'] = big_three_df['Date'].dt.month_name().str[:3]
# check the columns are added to
big_three_df.head()
```

```
C:\Users\soonn\AppData\Local\Temp\ipykernel_22484\229075286.py:6: SettingWithCopy
Warning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  big_three_df['year'] = big_three_df['Date'].dt.year
C:\Users\soonn\AppData\Local\Temp\ipykernel_22484\229075286.py:7: SettingWithCopy
Warning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  big_three_df['month'] = big_three_df['Date'].dt.month_name().str[:3]
```

Out[23]:

	Date	Customer_ID	City	Category	Product	Unit Price	Quantity	Sales Amount	Use
9	2023-01-10	82	Dammam	coffee beans	Costa Rica	35	4	140	
19	2023-01-20	2	Riyadh	coffee beans	Ethiopian	45	18	810	
34	2023-02-04	82	Riyadh	coffee beans	Colombian	40	35	1400	
41	2023-02-11	2	Dammam	coffee beans	Colombian	40	43	1720	
51	2023-02-21	97	Dammam	coffee beans	Guatemala	35	36	1260	

Sales by month

In order to prepare inventory for the sales, want to know the sales of each month and based on this will build predictive model to answer question. From this process i will achieve

1. sum of quantity by month for each product with visualization
2. predictive model for 2020 to maintain inventory

In [24]:
```
df_combined['year'] = df_combined['Date'].dt.year
df_combined['month'] = df_combined['Date'].dt.month_name().str[:3]
products = df_combined['Product'].unique().tolist()
products # check products
```

```
print("Product Nama: ", str(products), "\nNumber of Product Type : ",len(product
df_combined.head()
```

```
Product Nama:  ['Colombian', 'Costa Rica', 'Ethiopian', 'Brazilian', 'Guatemala']
Number of Product Type :  5
```

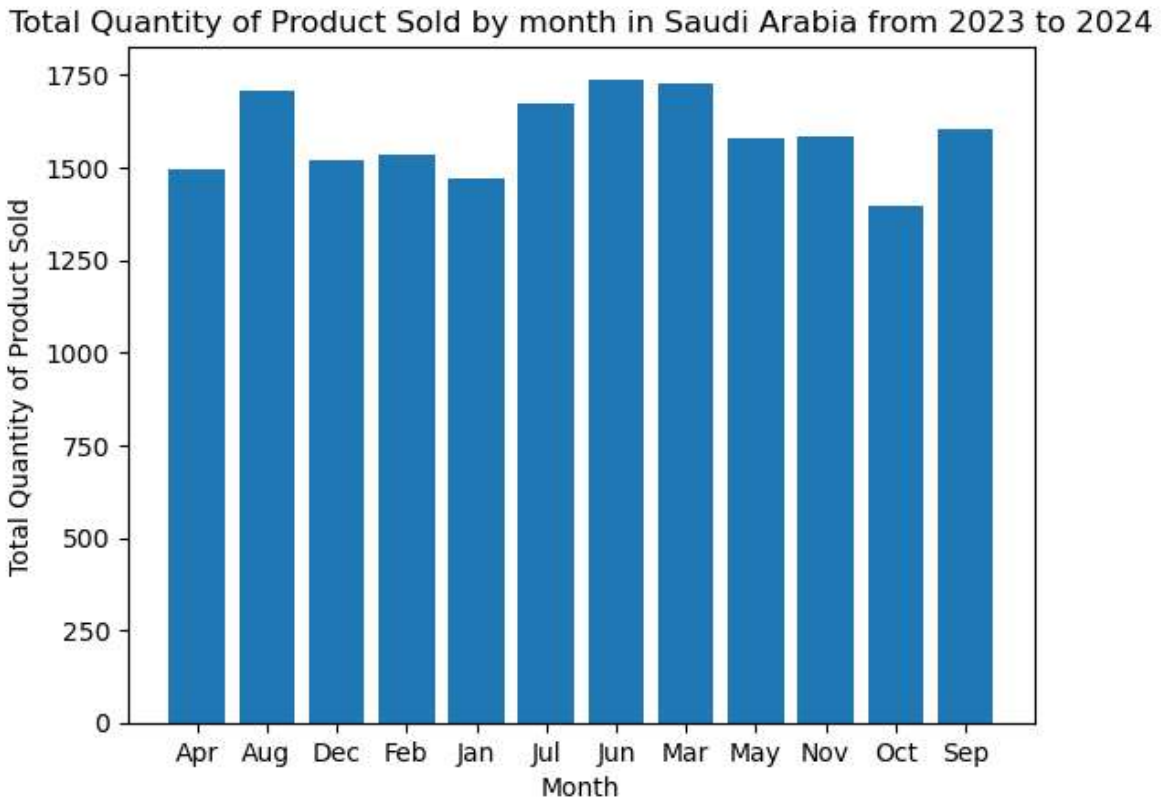
Out[24]:

	Date	Customer_ID	City	Category	Product	Unit Price	Quantity	Sales Amount	Used_D
0	2023-01-01	32	Riyadh	coffee beans	Colombian	40	14	560	
1	2023-01-02	49	Abha	coffee beans	Costa Rica	35	17	595	
2	2023-01-03	75	Tabuk	coffee beans	Costa Rica	35	19	665	
3	2023-01-04	80	Abha	coffee beans	Ethiopian	45	1	45	
4	2023-01-05	78	Hail	coffee beans	Colombian	40	46	1840	

In [ ]:

In [25]:
```
df_month = df_combined[['month','Quantity']].groupby('month').sum().reset_index(
df_month.head()
plt.bar(df_month['month'],df_month['Quantity'])
plt.xlabel('Month')
plt.ylabel('Total Quantity of Product Sold')
plt.title('Total Quantity of Product Sold by month in Saudi Arabia from 2023 to
```

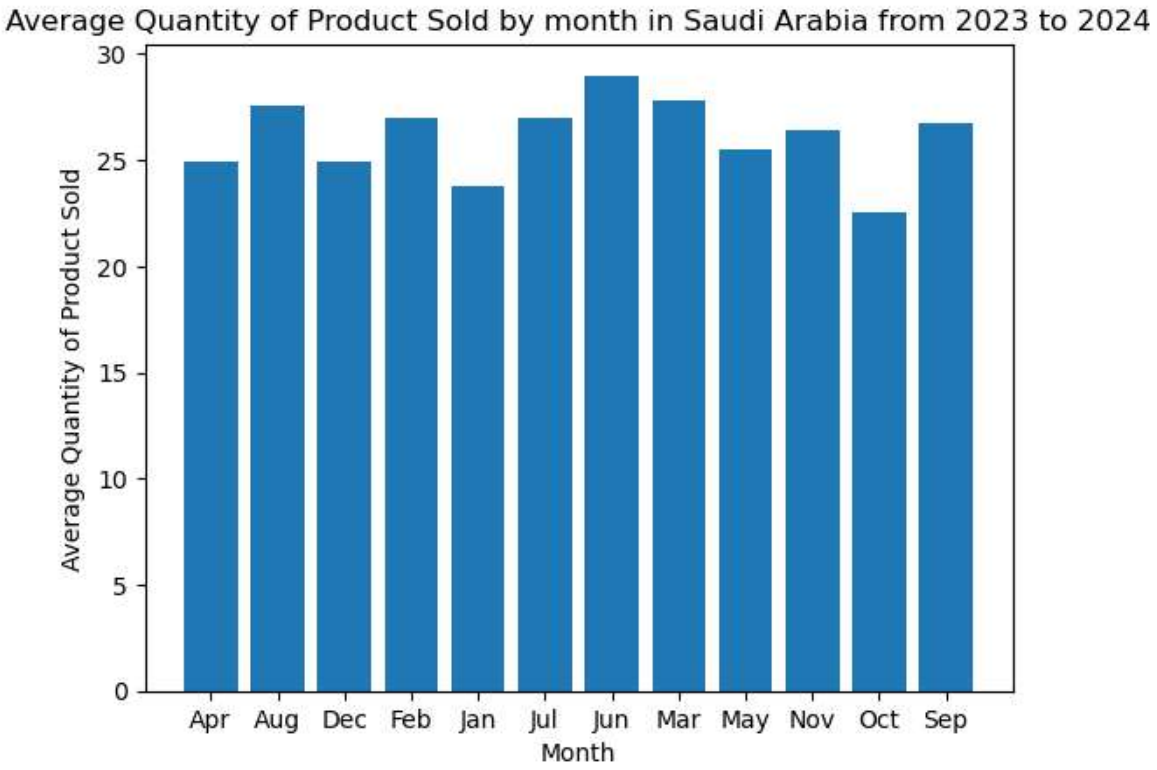
Out[25]:
```
Text(0.5, 1.0, 'Total Quantity of Product Sold by month in Saudi Arabia from 20
23 to 2024')
```



In [ ]:

In [26]:
```
df_month_mean = df_combined[['month','Quantity']].groupby('month').mean().reset_
df_month_mean.head()
plt.bar(df_month_mean['month'],df_month_mean['Quantity'])
plt.xlabel('Month')
plt.ylabel('Average Quantity of Product Sold')
plt.title('Average Quantity of Product Sold by month in Saudi Arabia from 2023 t
```

Out[26]:
```
Text(0.5, 1.0, 'Average Quantity of Product Sold by month in Saudi Arabia from
2023 to 2024')
```



Null hypothesis: expected quantity of each month is not differet from expected value of Quanity over year. Alternative hypothesis: expected quantity of each month is differet from expected value of Quanity over year.

Significance level: 0.05

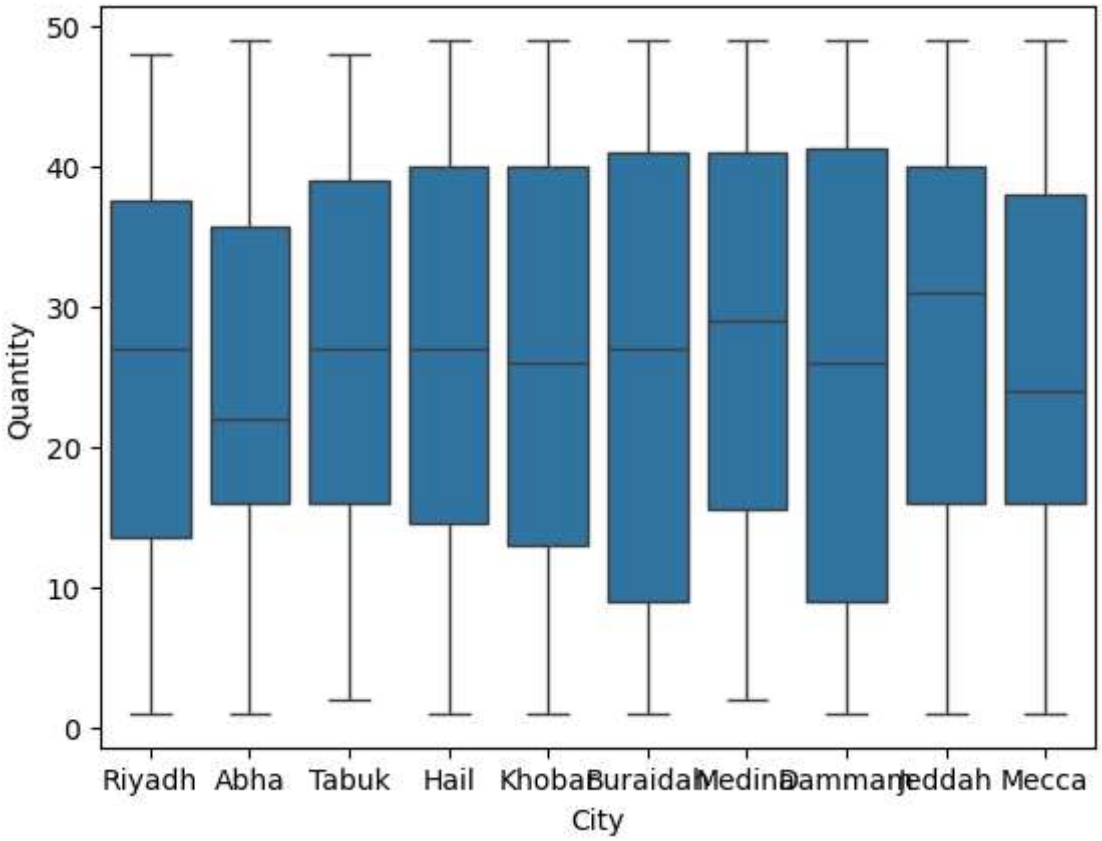
In [27]:
```
df_combined['month'] = df_combined['Date'].dt.month
from scipy.stats import ttest_1samp
q_mean = df_combined['Quantity'].mean()
print("The average value of Quantity : ", q_mean)
for i  in range(len(df_combined['month'].unique())+1):
    if i ==0:
        print()
    else:
        m = df_combined[df_combined['month'] == i]
        t_statitics, pval = ttest_1samp( m['Quantity'],q_mean)
        print("t_statistics of month {}".format(i), t_statitics,
          "\n P value of month {} : ".format(i), pval)
        print("")
```

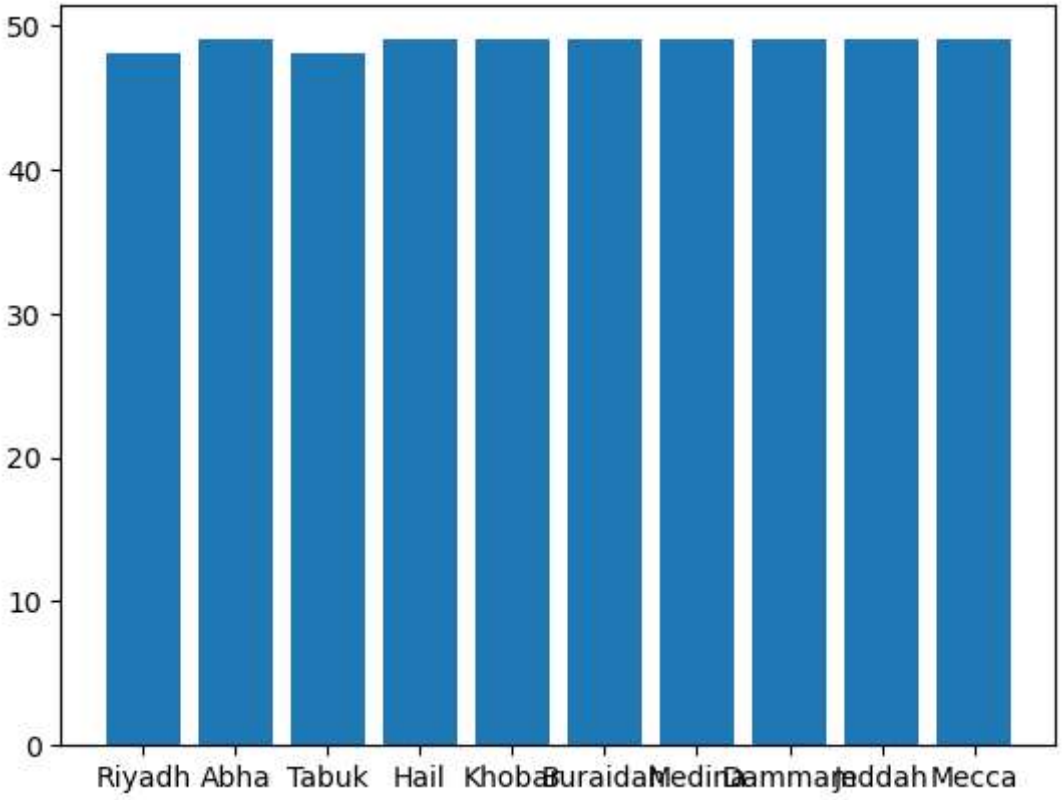
```
The average value of Quantity :  26.08082191780822

t_statistics of month 1 -1.1609607374750097
 P value of month 1 :  0.2501810118116579

t_statistics of month 2 0.504358591195905
 P value of month 2 :  0.615988197284021

t_statistics of month 3 0.9055409606551041
 P value of month 3 :  0.368741774957579

t_statistics of month 4 -0.6673483658000391
 P value of month 4 :  0.5071515059120049

t_statistics of month 5 -0.3229309033146611
 P value of month 5 :  0.7478519360413689

t_statistics of month 6 1.693154605061607
 P value of month 6 :  0.0957003622508075

t_statistics of month 7 0.4437180812089893
 P value of month 7 :  0.6588163899701915

t_statistics of month 8 0.9066707766955214
 P value of month 8 :  0.3681486174937507

t_statistics of month 9 0.31722394903817847
 P value of month 9 :  0.7521935547436993

t_statistics of month 10 -1.9782319906710595
 P value of month 10 :  0.0524237519856854

t_statistics of month 11 0.17035780170193698
 P value of month 11 :  0.8653116019052833

t_statistics of month 12 -0.594192937039002
 P value of month 12 :  0.5546169836055441
```

Calculated t-statistics and pvalue of each month to check significance difference between expected quantity of month and expected quantity of year. Month 6 and 10 have the lower p value compare to other month, but they are insignificant with 5% significant level. Fail to reject the Null hypothesis: expected quantity of each month is not differet from expected value of Quanity over year and average quantity sold in a u

Quantity by city

In [28]:
```
sns.boxplot(data = df_combined, x = 'City', y = 'Quantity')
plt.show()
plt.bar( df_combined['City'],  df_combined['Quantity'])
plt.show()
```

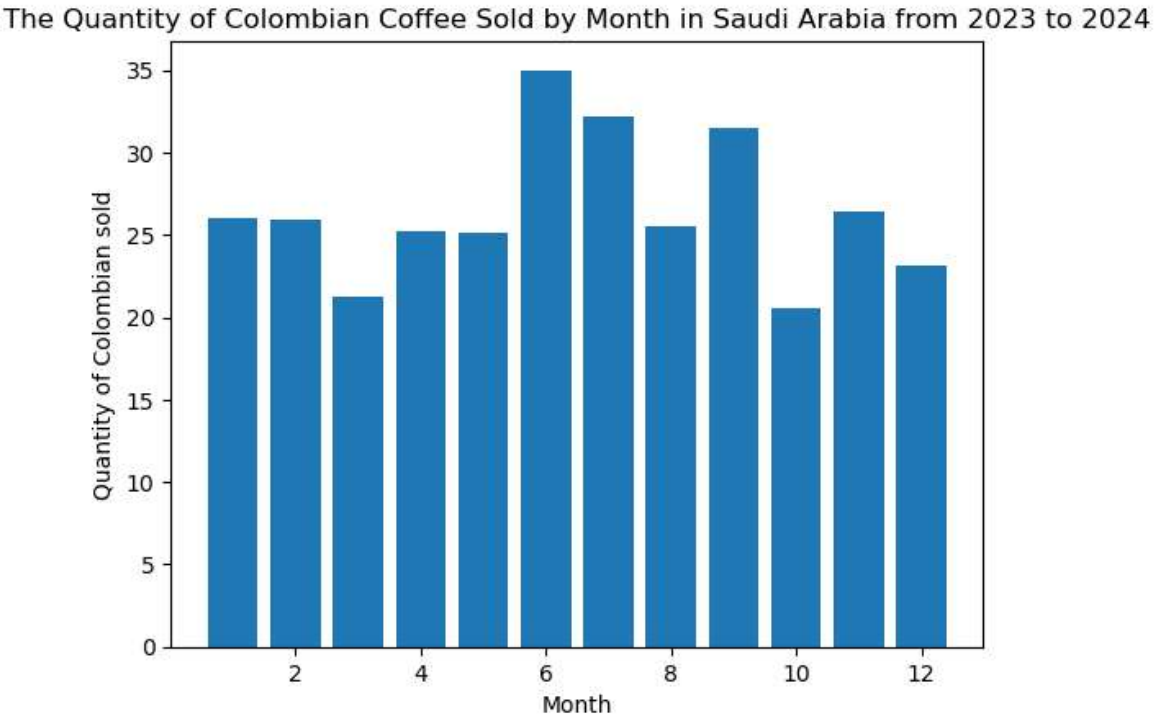


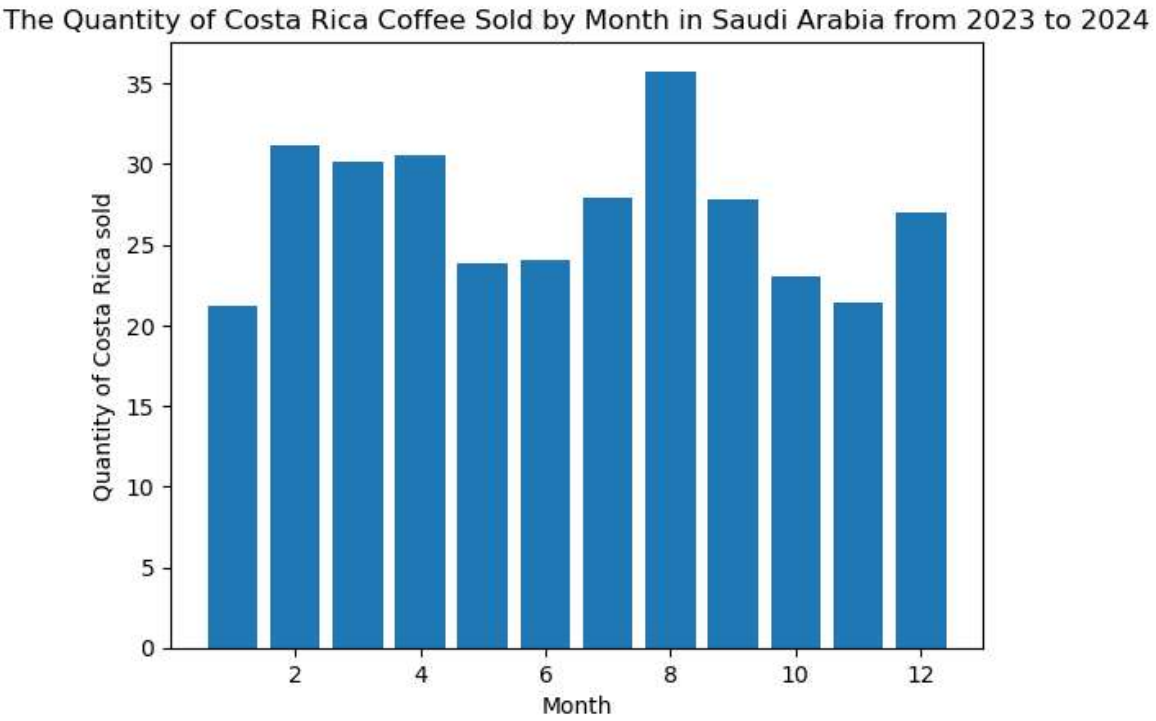


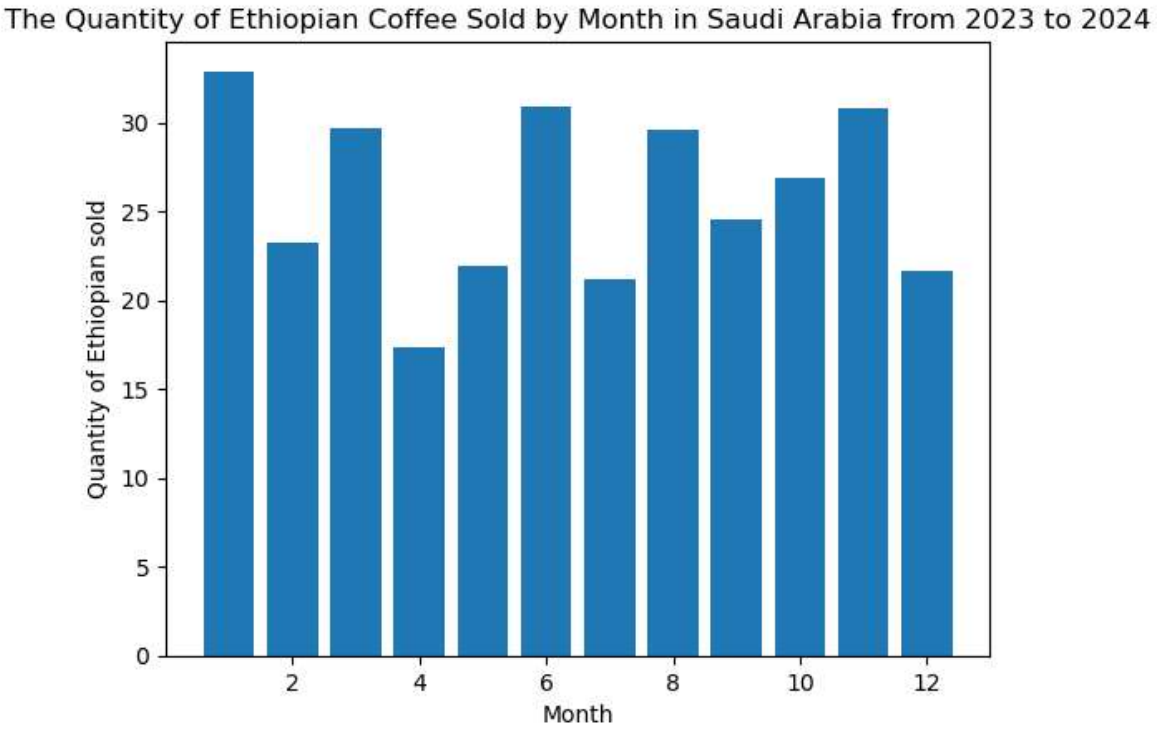
Quantity by product and month

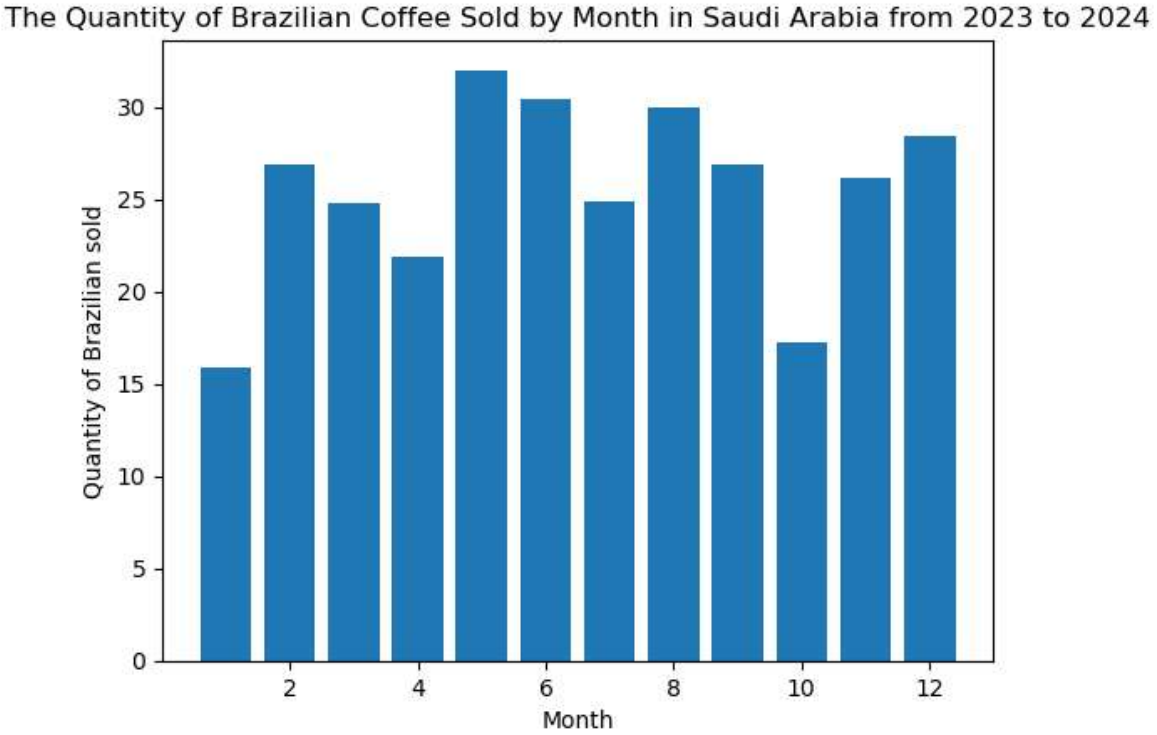
In [29]:
```
q_p_m = df_combined[['Quantity','Product','month']]
q_p_m.head()
q_p_m.groupby(['Product','month']).mean().reset_index()

df_month_product = df_combined[['Product','month','Quantity']].groupby(['Product
```

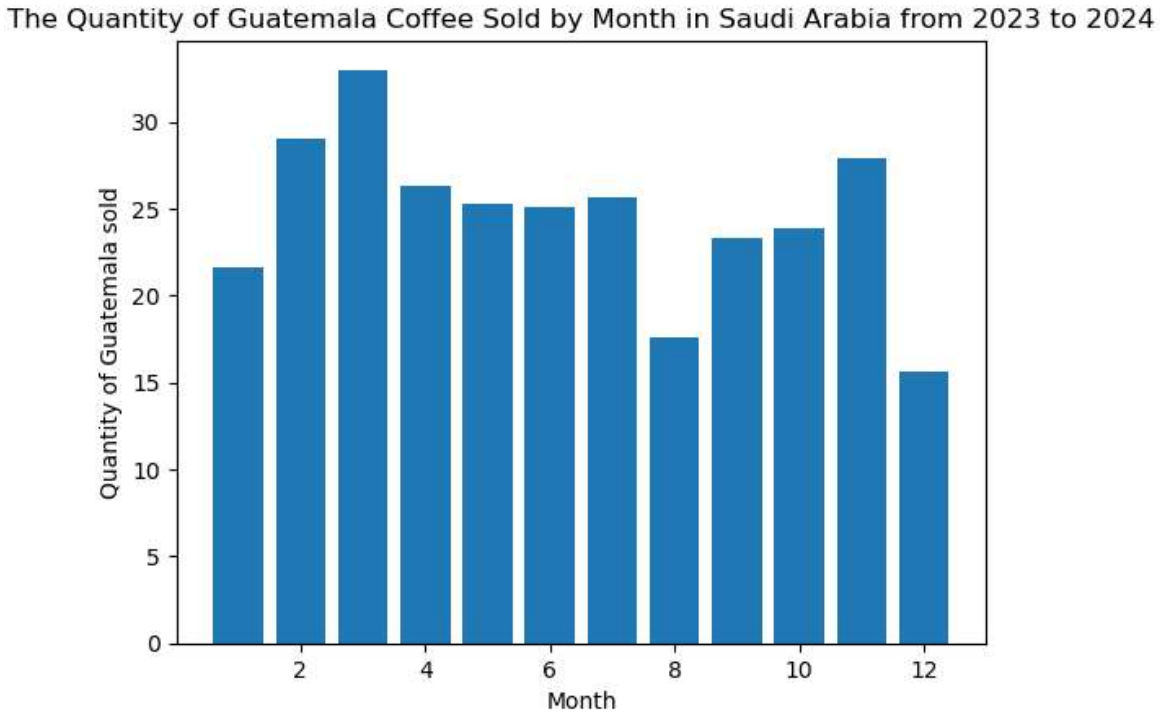
```
for i in range(len(products)):
    df_products = df_month_product[df_month_product['Product'] == products[i]]
    plt.bar(df_products['month'], df_products['Quantity'])
    plt.xlabel("Month")
    plt.ylabel('Quantity of {} sold'.format(products[i]))
    plt.title('The Quantity of {} Coffee Sold by Month in Saudi Arabia from 2023
    plt.show()
```






The Quantity of Ethiopian Coffee Sold by Month in Saudi Arabia from 2023 to 2024
Quantity of Ethiopian sold
0
5
10
15
20
25
30
2
4
6
8
10
12
Month


The Quantity of Brazilian Coffee Sold by Month in Saudi Arabia from 2023 to 2024
Quantity of Brazilian sold
0
5
10
15
20
25
30
2
4
6
8
10
12
Month



Quanity by City and Product

In [30]:
```
df_qpc = df_combined[['Quantity','City','Product']]
df_qpc.head()
df_col = df_qpc[df_qpc['Product'] == products[0]]
df_col_sum = df_col[['Quantity','City']].groupby('City').sum().reset_index()
df_col_sum.rename(columns = {'Quantity' : 'Colombia_Quantity'}, inplace = True)
print(df_col_sum)

df_col = df_qpc[df_qpc['Product'] == products[1]]
df_1_sum = df_col[['Quantity','City']].groupby('City').sum().reset_index()
df_1_sum.rename(columns = {'Quantity' : '{}_Quantity'.format(str(products[1]))},
print(df_1_sum)
df_col_sum = pd.concat([df_col_sum,df_1_sum['Costa Rica_Quantity']], axis = 1)

df_col = df_qpc[df_qpc['Product'] == products[2]]
df_1_sum = df_col[['Quantity','City']].groupby('City').sum().reset_index()
df_1_sum.rename(columns = {'Quantity' : '{}_Quantity'.format(str(products[2]))},
df_col_sum = pd.concat([df_col_sum,df_1_sum['{}_Quantity'.format(str(products[2]

df_col = df_qpc[df_qpc['Product'] == products[3]]
df_1_sum = df_col[['Quantity','City']].groupby('City').sum().reset_index()
df_1_sum.rename(columns = {'Quantity' : '{}_Quantity'.format(str(products[3]))},
df_col_sum = pd.concat([df_col_sum,df_1_sum['{}_Quantity'.format(str(products[3]

df_col = df_qpc[df_qpc['Product'] == products[4]]
df_1_sum = df_col[['Quantity','City']].groupby('City').sum().reset_index()
df_1_sum.rename(columns = {'Quantity' : '{}_Quantity'.format(str(products[4]))},
df_col_sum = pd.concat([df_col_sum,df_1_sum['{}_Quantity'.format(str(products[4]


df_col_sum
```

```
       City  Colombia_Quantity
0      Abha                286
1  Buraidah                291
2    Dammam                590
3      Hail                497
4    Jeddah                212
5    Khobar                465
6     Mecca                369
7    Medina                502
8    Riyadh                374
9     Tabuk                486
       City  Costa Rica_Quantity
0      Abha                 591
1  Buraidah                 304
2    Dammam                 456
3      Hail                 409
4    Jeddah                 520
5    Khobar                 293
6     Mecca                 326
7    Medina                 592
8    Riyadh                 632
9     Tabuk                 329
```

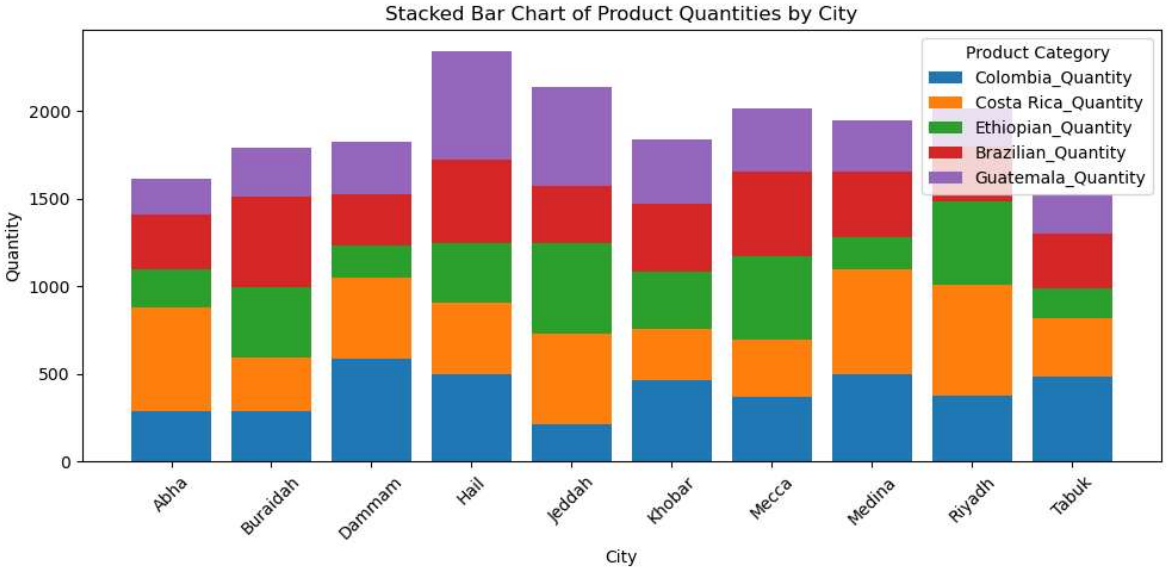
Out[30]:

	City	Colombia_Quantity	Costa Rica_Quantity	Ethiopian_Quantity	Brazilian_Quantity
0	Abha	286	591	220	312
1	Buraidah	291	304	401	516
2	Dammam	590	456	185	295
3	Hail	497	409	339	477
4	Jeddah	212	520	515	323
5	Khobar	465	293	327	386
6	Mecca	369	326	475	484
7	Medina	502	592	187	370
8	Riyadh	374	632	475	317
9	Tabuk	486	329	174	312

In [31]:

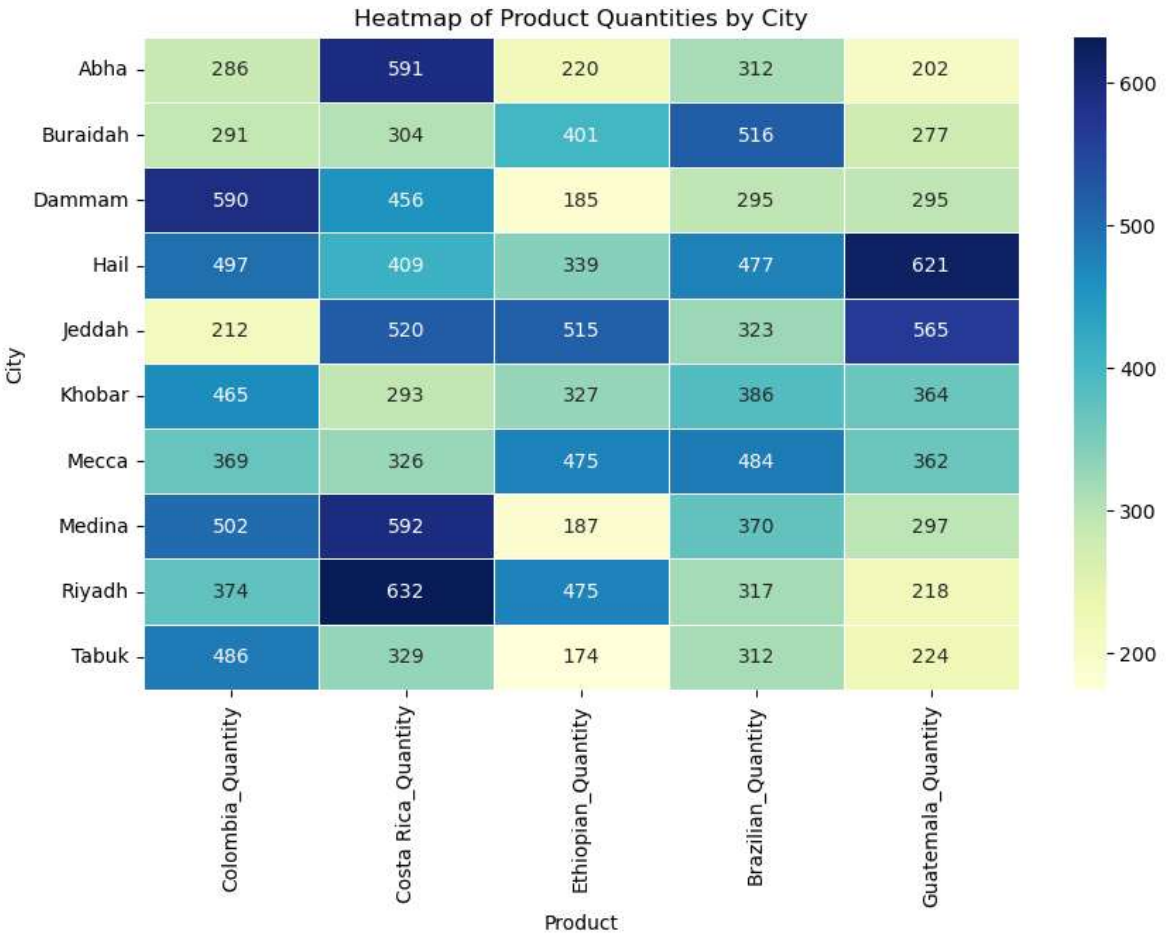
```
# Set up the figure and axis
plt.figure(figsize=(10, 5))

# Plot the stacked bar chart
cities = df_col_sum['City']
cities
categories = ["Colombia_Quantity", "Costa Rica_Quantity", "Ethiopian_Quantity",

# Stacking the bars
bottom_values = [0] * len(cities)
for category in categories:
    plt.bar(cities, df_col_sum[category], label=category, bottom=bottom_values)
    bottom_values = [bottom_values[i] + df_col_sum[category][i] for i in range(l
```

```
# Customizing the chart
plt.title("Stacked Bar Chart of Product Quantities by City")
plt.xlabel("City")
plt.ylabel("Quantity")
plt.xticks(rotation=45)
plt.legend(title="Product Category")
plt.tight_layout()
```



In [ ]:

In [32]:
```
# Set 'City' as the index
df_col_sum.set_index("City", inplace=True)

# Create the heatmap
plt.figure(figsize=(10, 6))  # Adjust the size of the plot
sns.heatmap(df_col_sum, annot=True, fmt="d", cmap="YlGnBu", linewidths=0.5)
plt.title("Heatmap of Product Quantities by City")
plt.xlabel("Product")
plt.ylabel("City")
plt.show()
```



In [33]:
```
df_col_sum = df_col_sum.T
for col in df_col_sum.columns.tolist() :
   print((df_col_sum.loc[df_col_sum[col] >= 500, col]))
   print("")
```

```
Costa Rica_Quantity    591
Name: Abha, dtype: int64

Brazilian_Quantity    516
Name: Buraidah, dtype: int64

Colombia_Quantity    590
Name: Dammam, dtype: int64

Guatemala_Quantity    621
Name: Hail, dtype: int64

Costa Rica_Quantity    520
Ethiopian_Quantity     515
Guatemala_Quantity     565
Name: Jeddah, dtype: int64

Series([], Name: Khobar, dtype: int64)

Series([], Name: Mecca, dtype: int64)

Colombia_Quantity      502
Costa Rica_Quantity    592
Name: Medina, dtype: int64

Costa Rica_Quantity    632
Name: Riyadh, dtype: int64

Series([], Name: Tabuk, dtype: int64)
```

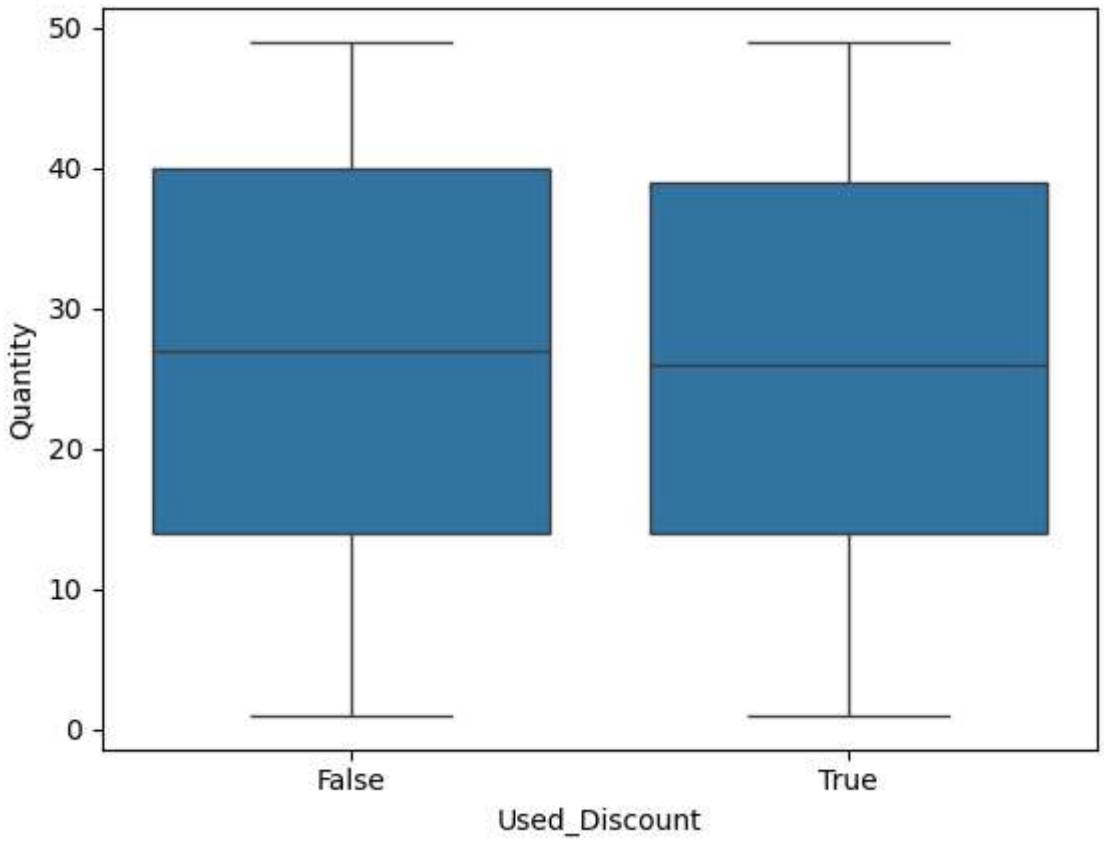
Each city has popular products, so manage invenory well.

Quanity and discunts

In [34]:
```
print(df[['Quantity','Used_Discount']].groupby('Used_Discount').sum())
sns.boxplot(x= df['Used_Discount'],y= df['Quantity'])
```

```
                Quantity
Used_Discount
False               9398
True                9641
```

Out[34]: <Axes: xlabel='Used_Discount', ylabel='Quantity'>



In [35]:
```
df[['Final Sales','Used_Discount']].groupby('Used_Discount').sum()
#sns.boxplot(x= df['Used_Discount'],y= df['Final Sales'])
```

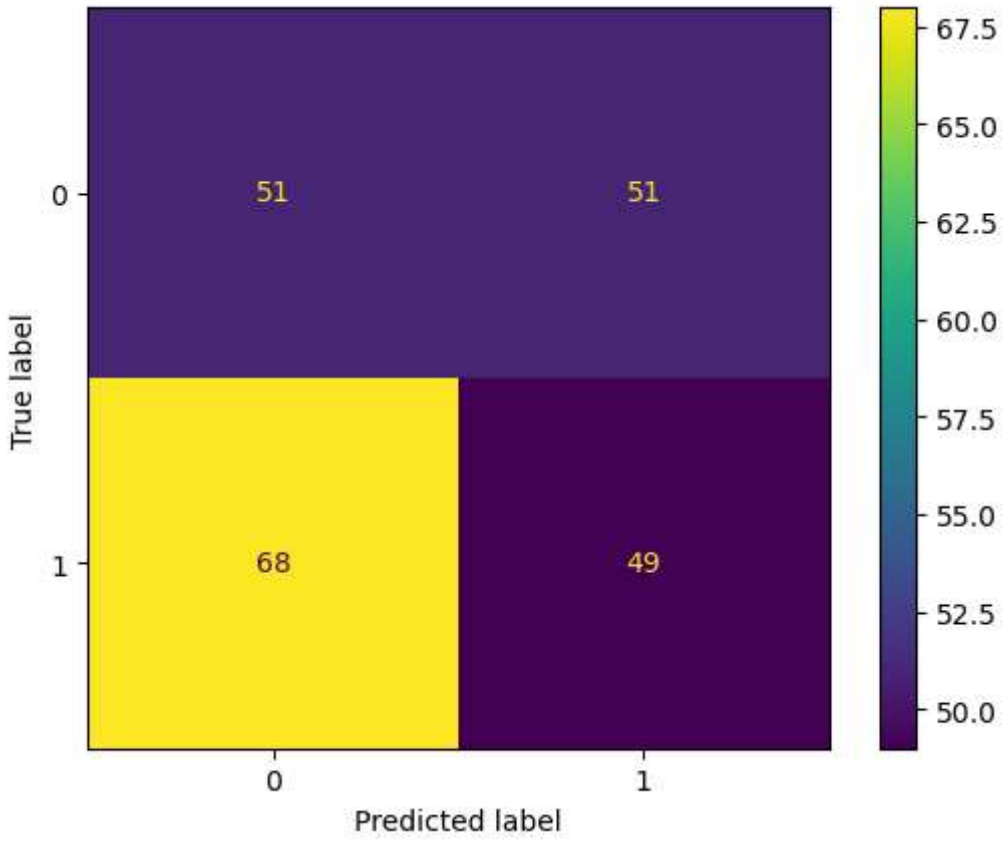
Out[35]:

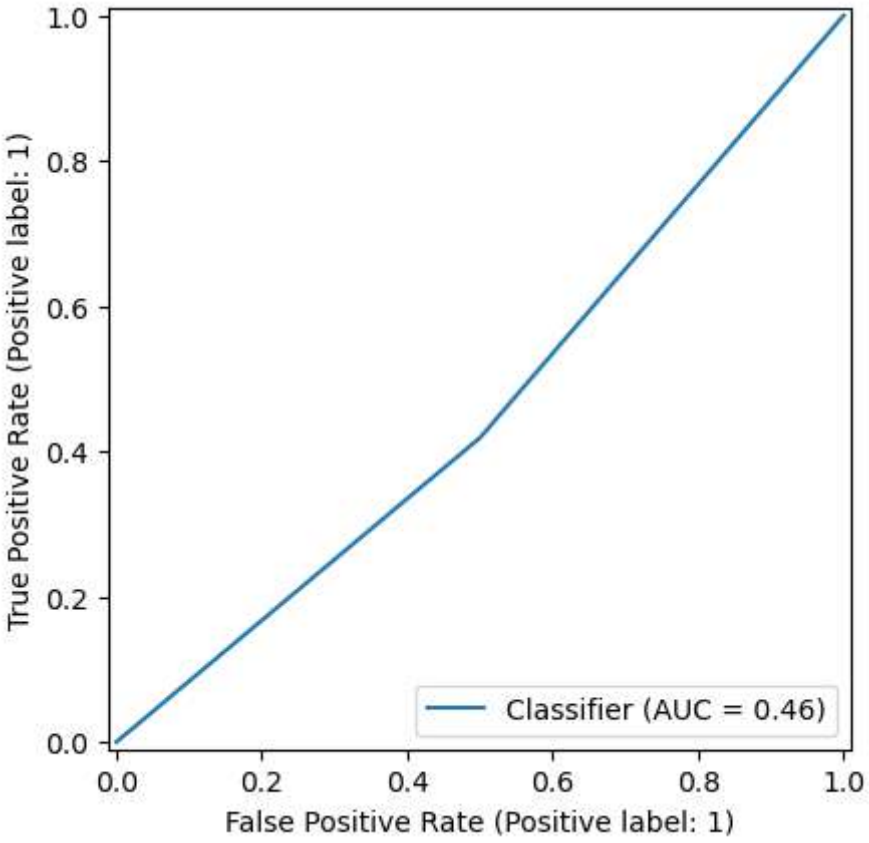
Used_Discount	Final Sales
False	345260
True	284388

In [36]:
```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression()
x = df[['Quantity']]
y = df['Used_Discount'].astype(int)


x_train,x_test, y_train,y_test = train_test_split(x, y, test_size = 0.3, random_

clf = clf.fit(x_train,y_train)

print(clf.coef_)
print(clf.intercept_)

# sns.regplot(x= x,y = y, logistic = True)

y_pred = clf.predict(x_test)

y_pred_pro = clf.predict_proba(x_test)[::,-1]

import sklearn.metrics as metrics

cm = metrics.confusion_matrix(y_test, y_pred, labels = clf.classes_)
```

```
disp = metrics.ConfusionMatrixDisplay(confusion_matrix = cm, display_labels = cl
disp.plot()

precision = metrics.precision_score(y_test,y_pred)
recall = metrics.recall_score(y_test,y_pred)
accuracy = metrics.accuracy_score(y_test,y_pred)

print( precision, recall, accuracy)

import matplotlib.pyplot as plt
from sklearn.metrics import RocCurveDisplay

RocCurveDisplay.from_predictions(y_test, y_pred)
plt.show()
```

```
[[-0.00584148]]
[0.1419348]
0.49 0.4188034188034188 0.45662100456621
```



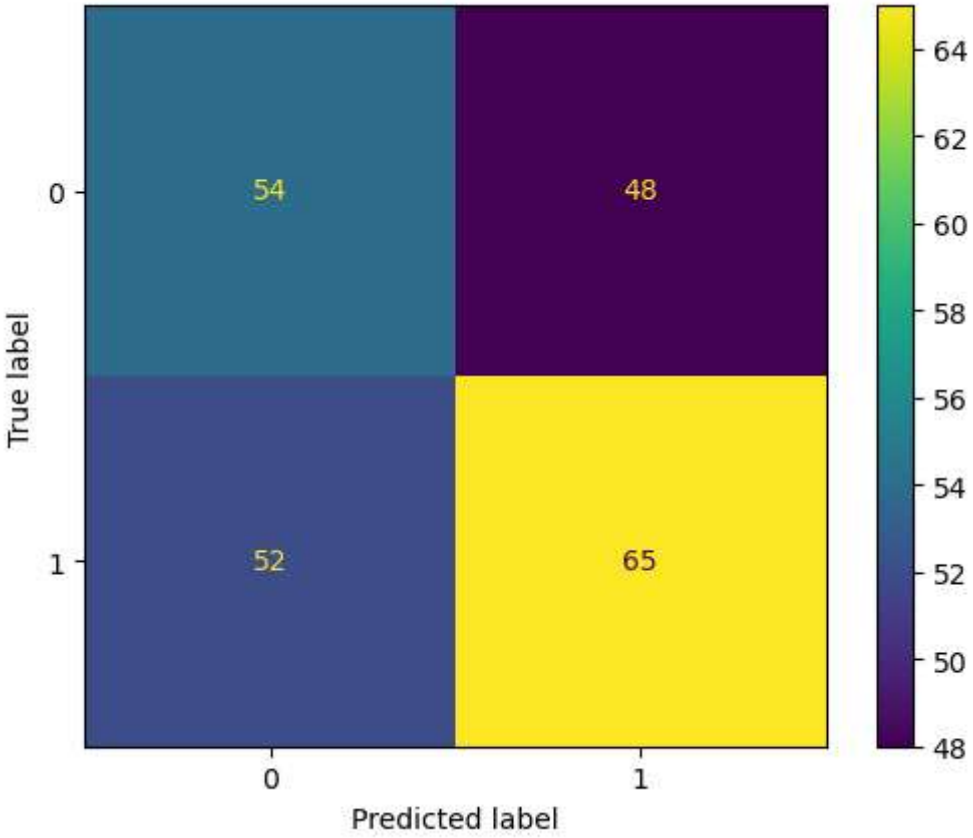


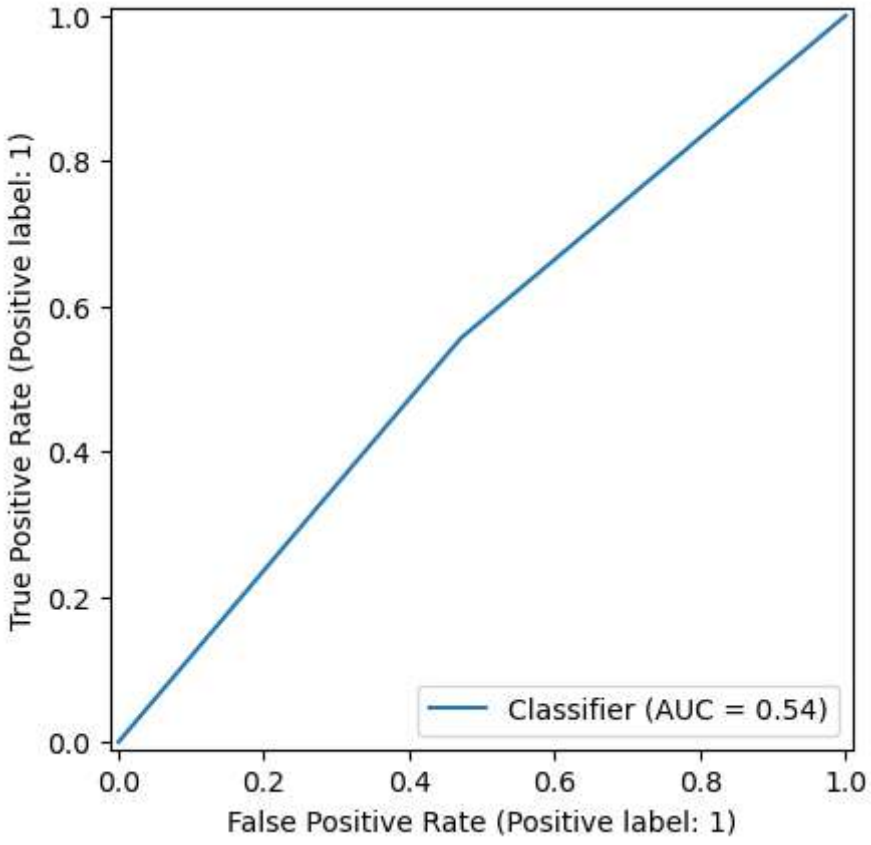
In [37]:

```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression()
x = df[['Final Sales']]
y = df['Used_Discount'].astype(int)


x_train,x_test, y_train,y_test = train_test_split(x, y, test_size = 0.3, random_

clf = clf.fit(x_train,y_train)

print(clf.coef_)
print(clf.intercept_)

# sns.regplot(x= x,y = y, logistic = True)

y_pred = clf.predict(x_test)

y_pred_pro = clf.predict_proba(x_test)[::,-1]

import sklearn.metrics as metrics

cm = metrics.confusion_matrix(y_test, y_pred, labels = clf.classes_)
disp = metrics.ConfusionMatrixDisplay(confusion_matrix = cm, display_labels = cl
disp.plot()

precision = metrics.precision_score(y_test,y_pred)
recall = metrics.recall_score(y_test,y_pred)
accuracy = metrics.accuracy_score(y_test,y_pred)

print( precision, recall, accuracy)

import matplotlib.pyplot as plt
from sklearn.metrics import RocCurveDisplay
```

```
RocCurveDisplay.from_predictions(y_test, y_pred)
plt.show()
```

```
[[-0.00088866]]
[0.76885301]
0.5752212389380531 0.5555555555555556 0.54337899543379
```

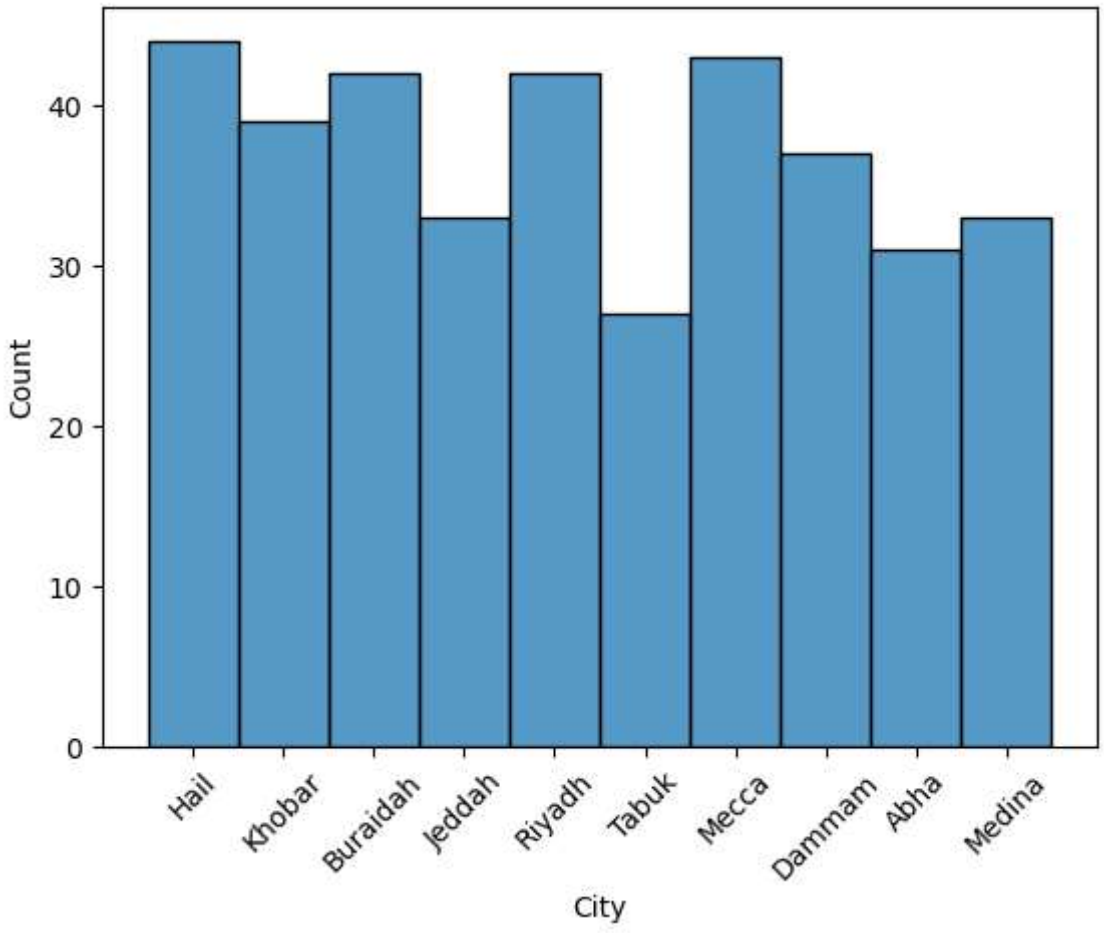




Now i want to figure out why?

1. expary date is coming. so give discount price. -> check the which city have the high discount rate
2. big sum of purchase.

In [38]:
```
df_dis = df_combined[['Used_Discount','City']]

df_dis = df_dis[df_dis['Used_Discount'] == True]
df_dis.head()

sns.histplot(df_dis['City'])
plt.xticks(rotation = 45)
plt.show()
city_fre = df_dis.value_counts().tolist()
city_mean = np.mean(city_fre)

df_grp = df_dis.groupby('City').sum().reset_index()
mask = df_grp['Used_Discount'] >= city_mean
df_grp= df_grp[mask]
df_grp = df_grp['City'].tolist()

print("Cities with high discount number : ", df_grp)
#for column in df_dis['City'].tolist():
#    if city_fre[df_dis['City'] == column] >= city_mean:
#        print("")
#        print(city_fre[city_fre['City'] == column].mean())
```




```
Cities with high discount number :  ['Buraidah', 'Hail', 'Khobar', 'Mecca', 'Riya
dh']
```

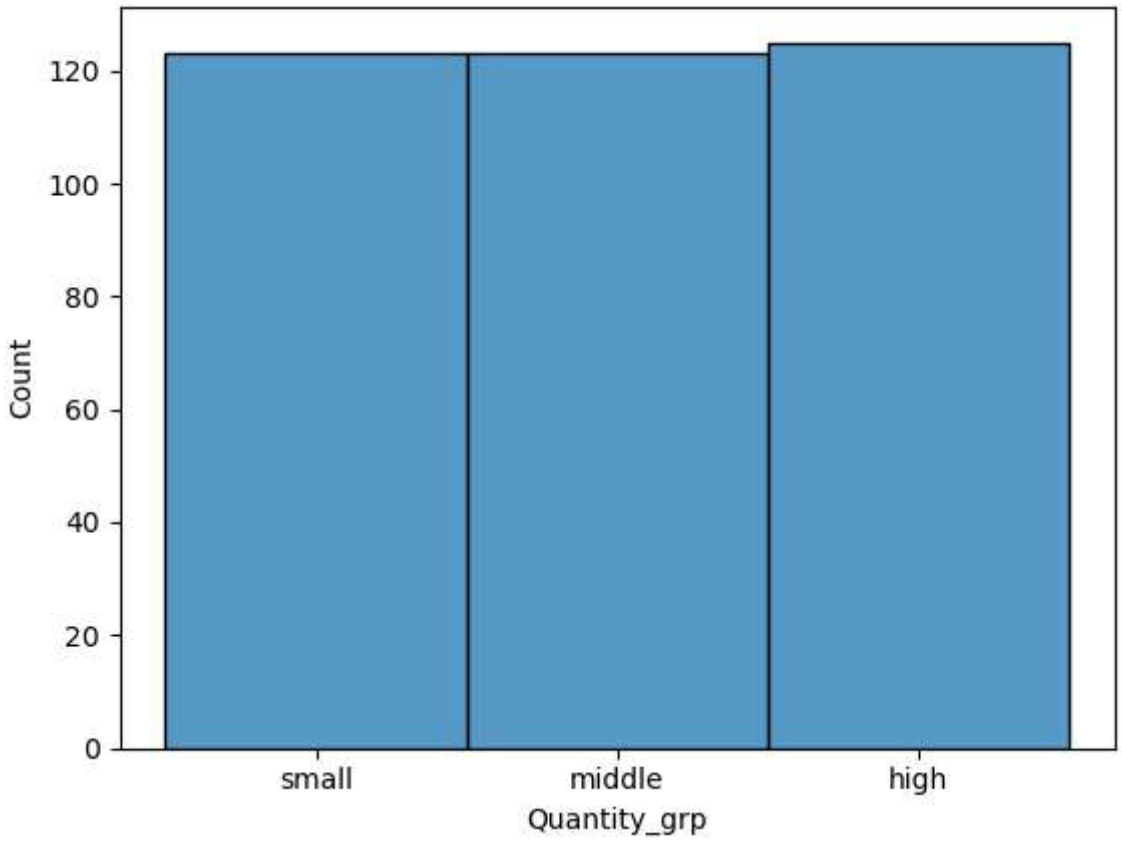
['Buraidah', 'Hail', 'Khobar', 'Mecca', 'Riyadh'] have high discount rate and It is possible the might have more products in stock.

2. big sum of purchase.
3.

In [39]:
```
df_grp = df_combined[['Quantity','Used_Discount']]
bins = np.linspace(min(df_grp['Quantity']), max(df_grp['Quantity']), 4)
head = ['small','middle','high']
df_grp['Quantity_grp'] = pd.cut(df_grp['Quantity'], bins = bins, labels = head,

df_grp.head()
df_grp = df_grp[df_grp['Used_Discount'] == True]

sns.histplot(df_grp['Quantity_grp'])
```

```
C:\Users\soonn\AppData\Local\Temp\ipykernel_22484\1719746665.py:4: SettingWithCop
yWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_grp['Quantity_grp'] = pd.cut(df_grp['Quantity'], bins = bins, labels = head,
include_lowest = True)
```

Out[39]: <Axes: xlabel='Quantity_grp', ylabel='Count'>



Althohgh the quantity of sales at a time is small, the discount was applied, which means the discount is not applied by quantity of order.

In [40]:
```python
# hypothersis
# Null hypothesis : sales of bean from Jun to Sep is higher than other month
# alternative hypothesis : sales of bean from Jun to Sep is the same or less tha
df['month'] = df['Date'].dt.month # assign month with integer
df['compare'] = df['month']
df.loc[df['compare'].isin([6,7,8,9]),'compare'] = 99
df.loc[df['compare'].isin([1,2,3,4,5,10,11,12]),'compare'] = 1
df['compare'] = df['compare'].astype(str)
df.loc[df['compare'] == '99','compare'] = 'high'
df.loc[df['compare'] == '1','compare'] = 'others'

df.head()

import pandas as pd
from scipy.stats import ttest_ind

# Split data by city
summer_q = df[df['compare'] == 'high']['Quantity']
other_q = df[df['compare'] == 'others']['Quantity']

# Perform independent t-test
t_stat, p_value = ttest_ind(summer_q, other_q)
print(f"T-statistic: {t_stat}, P-value: {p_value}")

# Interpret results
if p_value < 0.05:
    print("There is a significant difference in sales between summer months and
else:
    print("No significant difference in sales between summer months and other mo
```

```
T-statistic: 1.955825526815722, P-value: 0.05086781138757017
No significant difference in sales between summer months and other months.
```

Sales trends from 2023 Jan to 2024 Dec

Overview : find sales trends of sales and see any increase and decrease. consider model variables

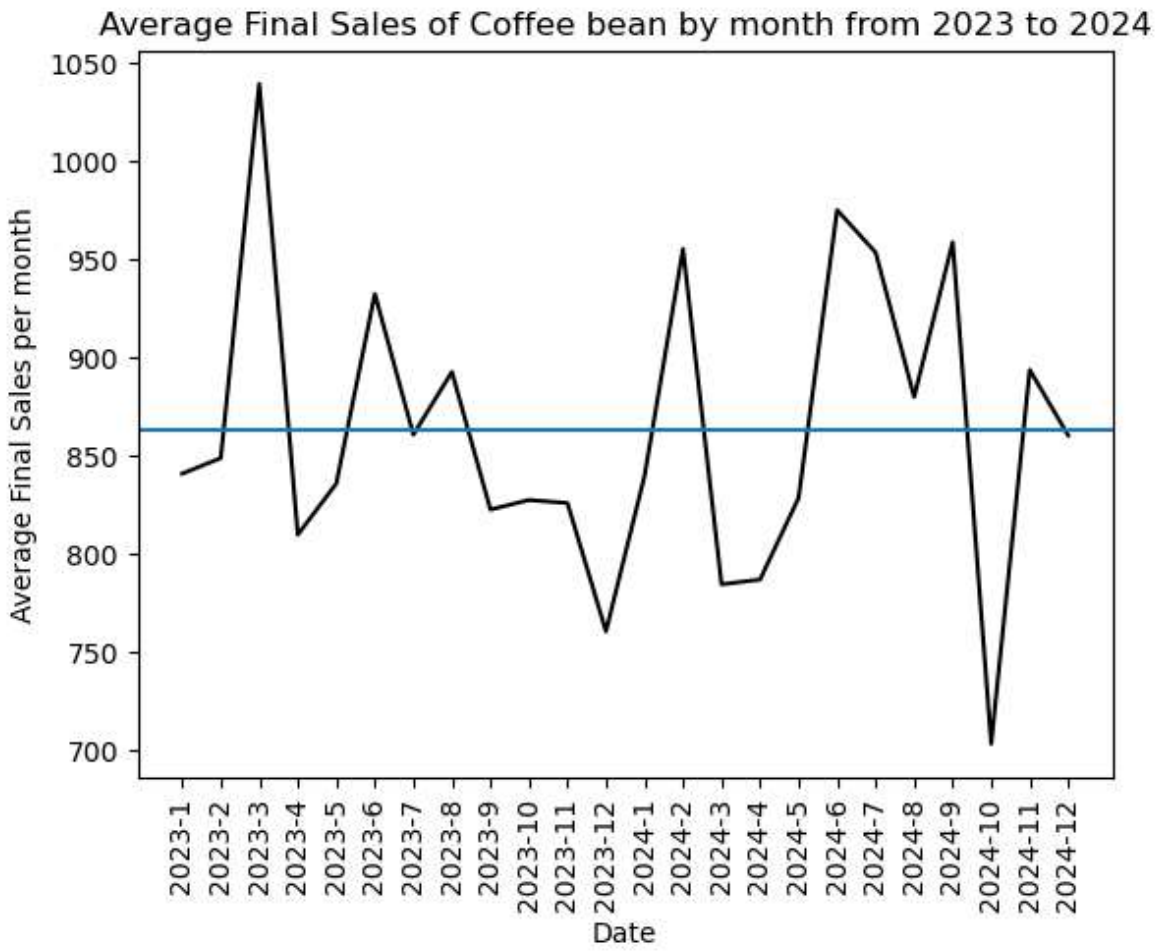
In [41]:
```python
df.head()
```

Out[41]:

	Date	Customer_ID	City	Category	Product	Unit Price	Quantity	Sales Amount	Used_D
0	2023-01-01	32	Riyadh	coffee beans	Colombian	40	14	560	
1	2023-01-02	49	Abha	coffee beans	Costa Rica	35	17	595	
2	2023-01-03	75	Tabuk	coffee beans	Costa Rica	35	19	665	
3	2023-01-04	80	Abha	coffee beans	Ethiopian	45	1	45	
4	2023-01-05	78	Hail	coffee beans	Colombian	40	46	1840	

In [71]:
```
df['year'] = df['Date'].dt.year
mean_sales_date = df.groupby(['year','month']).mean(numeric_only = True)[['Final
mean_sales_date = mean_sales_date.reset_index()

mean_sales_date['year_month'] = mean_sales_date['year'].astype(str)+'-'+mean_sal
mean_sales_date.drop(['month','year'], axis = 1)
mean_sales = mean_sales_date['Final Sales'].mean()
sns.lineplot(data = mean_sales_date, x = 'year_month', y = 'Final Sales', color
plt.xticks(rotation = 90)
plt.axhline(y = mean_sales)
plt.xlabel('Date')
plt.ylabel('Average Final Sales per month')
plt.title('Average Final Sales of Coffee bean by month from 2023 to 2024')
```

Out[71]:
```
Text(0.5, 1.0, 'Average Final Sales of Coffee bean by month from 2023 to 2024')
```



the averaage sales of coffee beans has fluctuate over the overs. the sales tend to go up in the beginnig of the year go down and goes up in summer time then go down again. one possible reason of decrease from march to april is Ramadan, which people does not eat daytime.

Consider variables for model

Tager variables: Quantity or Final sales independent variables : other variabless

In [43]:
```
df.head()
df_model = df[['Date', 'Unit Price', 'Sales Amount', 'Discount_Amount', 'Final S
```

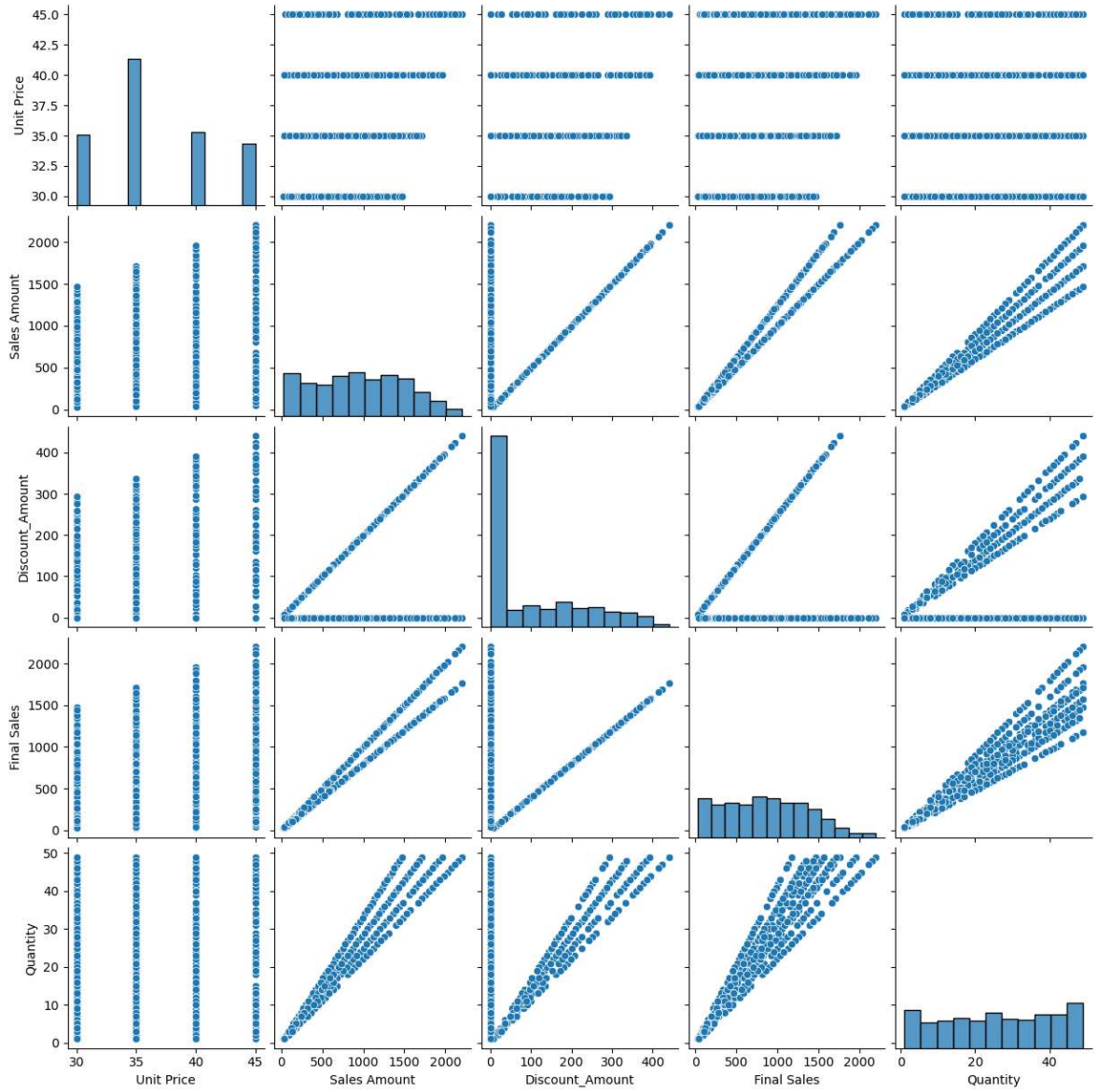
In [44]:
```
df_model.head()
```

Out[44]:

	Date	Unit Price	Sales Amount	Discount_Amount	Final Sales	Quantity
0	2023-01-01	40	560	0	560	14
1	2023-01-02	35	595	0	595	17
2	2023-01-03	35	665	0	665	19
3	2023-01-04	45	45	0	45	1
4	2023-01-05	40	1840	368	1472	46

In [45]:
```
sns.pairplot(df_model)
```

Out[45]:
```
<seaborn.axisgrid.PairGrid at 0x1ef59d04aa0>
```



In [46]:
```
df_model['Discount_Rate'] = df_model['Discount_Amount']/df_model['Final Sales']
df_model.head()
```

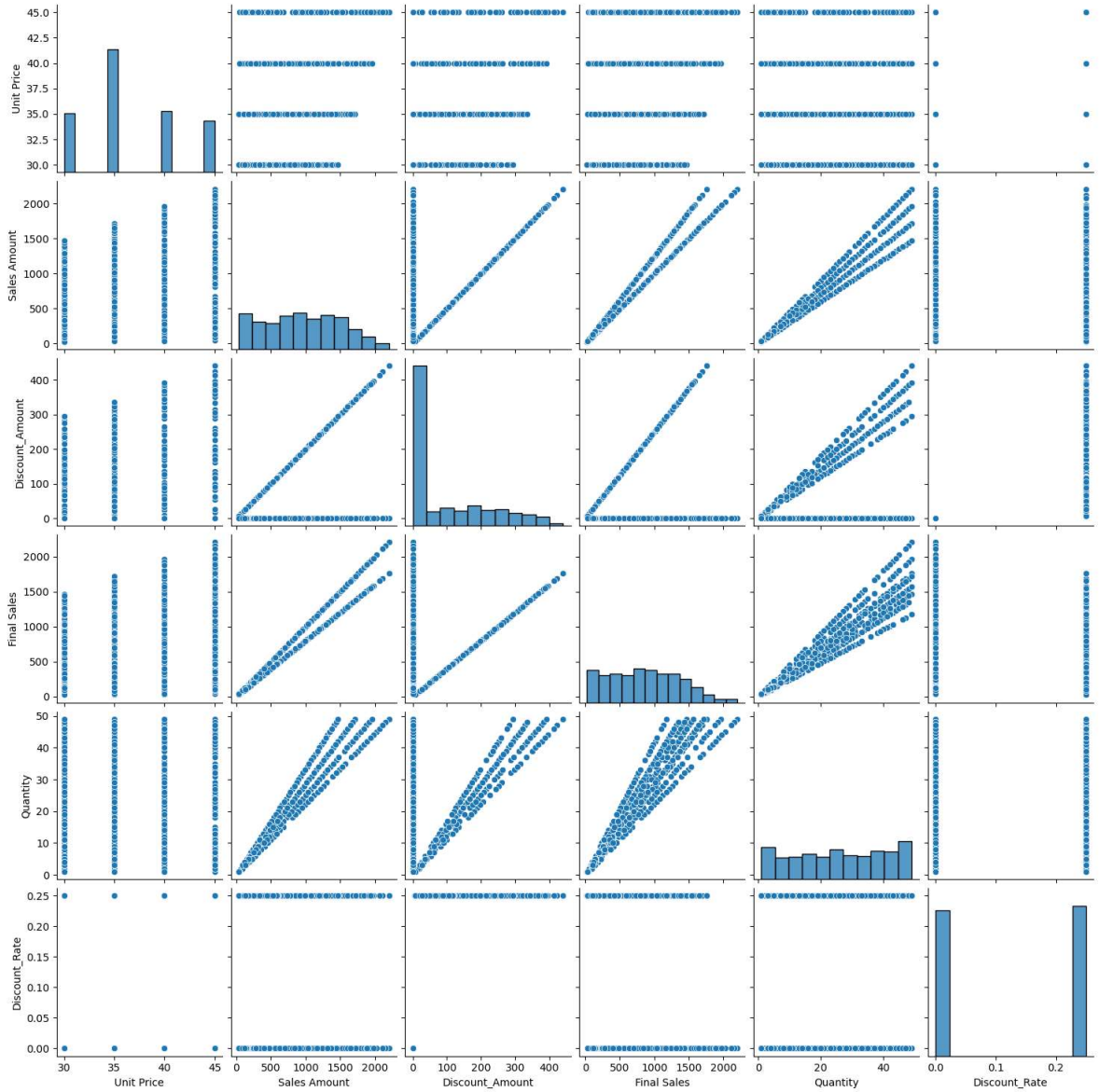
```
C:\Users\soonn\AppData\Local\Temp\ipykernel_22484\1974359409.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_model['Discount_Rate'] = df_model['Discount_Amount']/df_model['Final Sales']
```

Out[46]:

	Date	Unit Price	Sales Amount	Discount_Amount	Final Sales	Quantity	Discount_Rate
0	2023-01-01	40	560	0	560	14	0.00
1	2023-01-02	35	595	0	595	17	0.00
2	2023-01-03	35	665	0	665	19	0.00
3	2023-01-04	45	45	0	45	1	0.00
4	2023-01-05	40	1840	368	1472	46	0.25

In [47]:
```
sns.pairplot(df_model)
```

Out[47]:
```
<seaborn.axisgrid.PairGrid at 0x1ef5965d8e0>
```



In [48]:
```
df_model['year'] = df_model['Date'].dt.year
df_model_2023 = df_model[df_model['year'] == 2023]
```

```
df_model_2024 = df_model[df_model['year'] == 2024]
```

```
C:\Users\soonn\AppData\Local\Temp\ipykernel_22484\670021762.py:1: SettingWithCopy
Warning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_model['year'] = df_model['Date'].dt.year
```

In [49]:
```
x = df_model[['Quantity','Sales Amount']]
y = df_model['Final Sales']
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.3, random_sta
df_model.rename(columns = {'Sales Amount': 'Sales_Amount', 'Final Sales': 'Final
df_model.info()
```

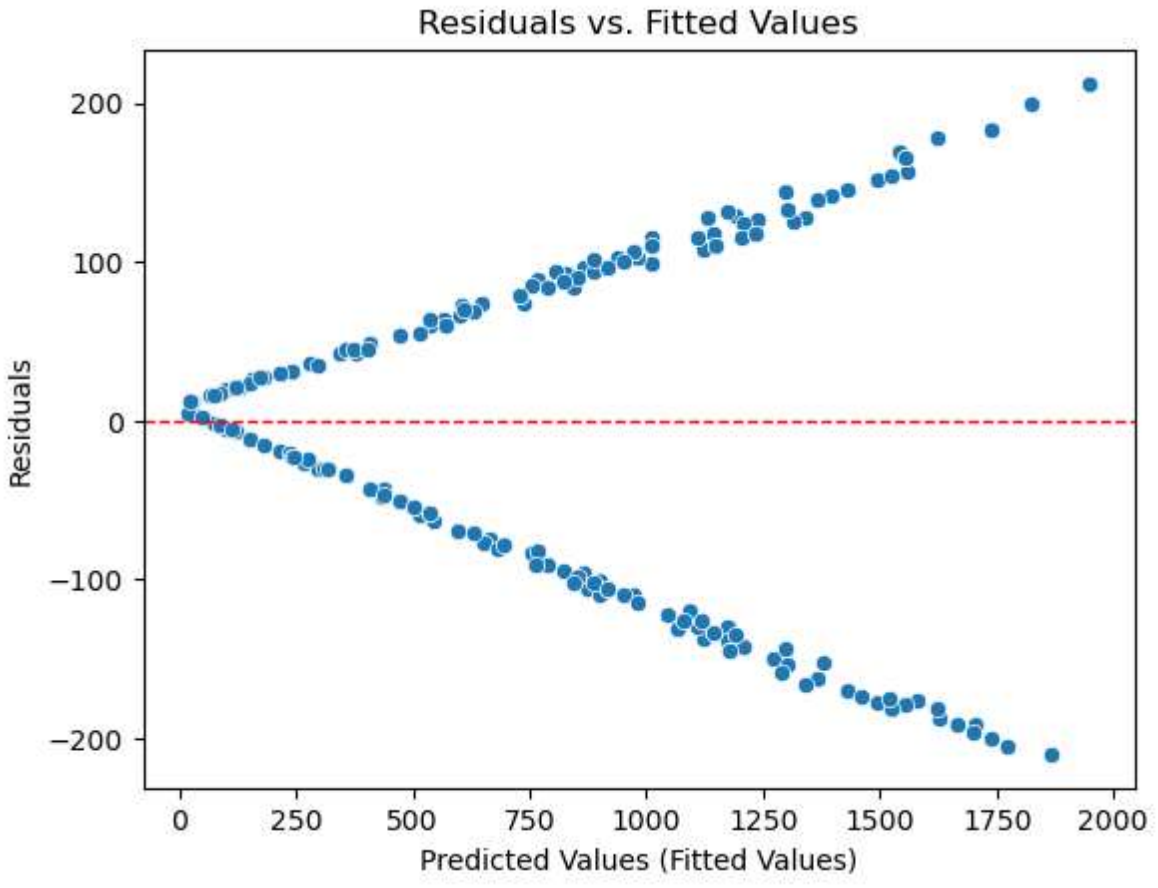
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 730 entries, 0 to 729
Data columns (total 8 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Date             730 non-null    datetime64[ns]
 1   Unit Price       730 non-null    int64
 2   Sales Amount     730 non-null    int64
 3   Discount_Amount  730 non-null    int64
 4   Final Sales      730 non-null    int64
 5   Quantity         730 non-null    int64
 6   Discount_Rate    730 non-null    float64
 7   year             730 non-null    int32
dtypes: datetime64[ns](1), float64(1), int32(1), int64(5)
memory usage: 42.9 KB
```

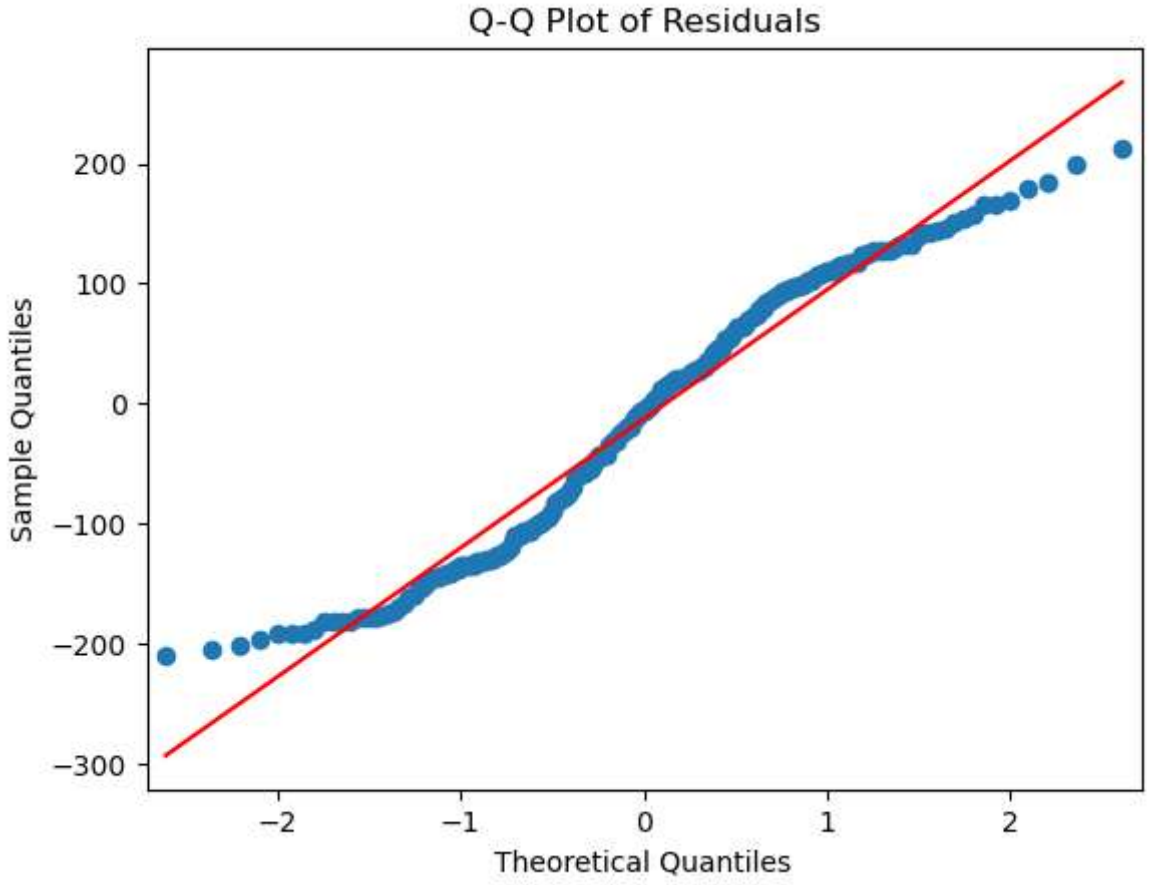
In [64]:
```
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error

# Initialize the scaler
scaler = StandardScaler()

# Scale the training and test data
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)

# Fit the Linear Regression model on the scaled training data
lr = LinearRegression()
lr.fit(x_train_scaled, y_train)

# Predict on the scaled test data
y_pred_test = lr.predict(x_test_scaled)

# Calculate residuals for the test set
residuals_test = y_test - y_pred_test

# Display residuals
print("Residuals for test set:", residuals_test)
```

```
Residuals for test set: 468     89.410477
148   -134.029513
302     48.253055
355    127.756439
515     42.190670
          ...
332    -91.204513
532    -54.653367
558    165.181505
137     20.972321
314     27.025917
Name: Final Sales, Length: 219, dtype: float64
```

In [69]:

```
import seaborn as sns
import matplotlib.pyplot as plt

# Residuals vs Fitted Values (Linearity & Homoscedasticity Test)
sns.scatterplot(x=y_pred_test, y=residual)  # Predicted values vs residuals
plt.axhline(0, color='red', linestyle='--', linewidth=1)  # Reference line at y=
plt.xlabel('Predicted Values (Fitted Values)')
plt.ylabel('Residuals')
plt.title('Residuals vs. Fitted Values')
plt.show()


import statsmodels.api as sm
import matplotlib.pyplot as plt

# Q-Q Plot for Normality Check
sm.qqplot(residual, line='s')  # Residuals Q-Q plot
plt.title('Q-Q Plot of Residuals')
plt.show()
```





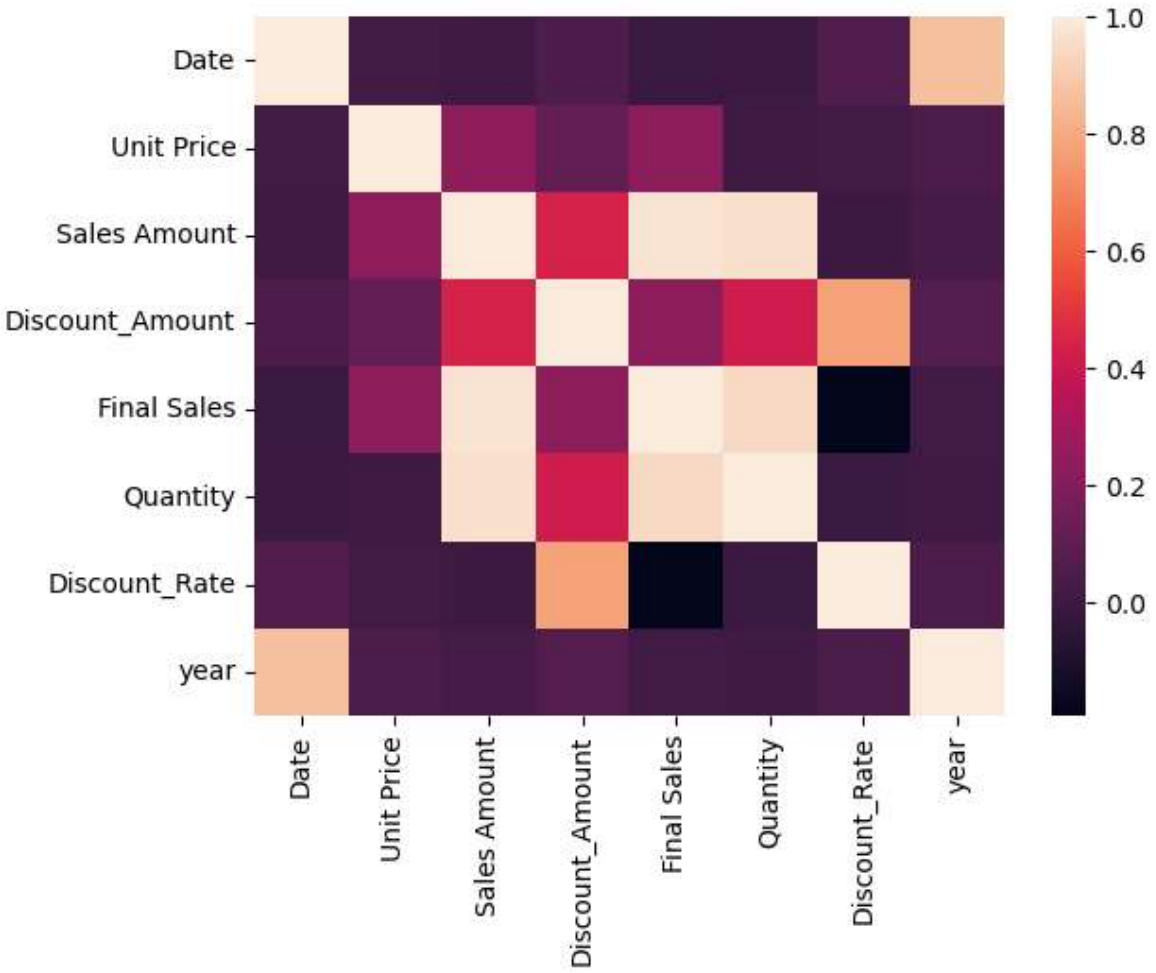
In [68]:
```
lr.score(x_test_scaled,y_test)
```

Out[68]: 0.951575747121218

In [57]:
```
print(df_model.corr())
sns.heatmap(df_model.corr())
```

```
                    Date  Unit Price  Sales Amount  Discount_Amount  \
Date            1.000000    0.014802      0.006382         0.053539
Unit Price      0.014802    1.000000      0.237121         0.114109
Sales Amount    0.006382    0.237121      1.000000         0.441184
Discount_Amount 0.053539    0.114109      0.441184         1.000000
Final Sales    -0.006068    0.229140      0.976048         0.235380
Quantity       -0.002309    0.004093      0.963352         0.420565
Discount_Rate   0.055762    0.013410     -0.003218         0.776902
year            0.866026    0.047030      0.022740         0.064374

                Final Sales  Quantity  Discount_Rate      year
Date              -0.006068 -0.002309       0.055762  0.866026
Unit Price         0.229140  0.004093       0.013410  0.047030
Sales Amount       0.976048  0.963352      -0.003218  0.022740
Discount_Amount    0.235380  0.420565       0.776902  0.064374
Final Sales        1.000000  0.941357      -0.191825  0.009021
Quantity           0.941357  1.000000      -0.006624  0.004070
Discount_Rate     -0.191825 -0.006624       1.000000  0.046582
year               0.009021  0.004070       0.046582  1.000000
```

Out[57]: <Axes: >



model building variales

I worked extensively on both binomial logistic regression and linear regression models. For the logistic regression model, I used Used_discount as the dependent variable. Initially, I built a model with Quantity as the independent variable, but the results were not promising. I then created a second model using Final_Sales as the independent variable, which yielded a slightly better confusion matrix. However, the AUC remained at 0.54, with recall, accuracy, and precision indicators hovering around 0.55. Despite these efforts, I was unable to build a robust logistic regression model with the available data. Next, I shifted my focus to building a multivariable linear regression model to predict Final_Sales. The variables Quantity and Sales_Amount showed clear relationships with Final_Sales, resulting in a model with an impressive R-squared value of 0.95—indicating that 95% of the variance in the dependent variable was explained by the independent variables. However, the model failed to satisfy critical linear regression assumptions, such as linearity and normality. Thus, the model building process was ultimately unsuccessful. Recognizing the limitations of the given data, I sought open data sources related to coffee sales in Saudi Arabia from 2023 to 2024 but was unable to locate relevant datasets. Given this situation, I recommend collecting additional data to enhance the modeling process. Specifically, collecting sales/inventory data, market/customer data, and geographic/location data could prove valuable for building better predictive models in the future.

a) Sales and Inventory Data

1. Historical Inventory Levels: Add information on past inventory amounts and stock-outs to capture supply-demand dynamics.
2. Waste/Expiration Data: Include the number of products that spoiled or went unsold to refine inventory predictions.
3. Sales Time Data: Record timestamps more granularly (e.g., daily/hourly sales) to account for short-term trends.

b) Market/Customer Data

1. Promotions/Events: Track promotional activities and local events, which could impact sales spikes.
2. Customer Feedback: Add review/rating data or customer preferences for specific products

c) Geographic/Location Data

1. Foot Traffic: If sales locations are physical stores, include data on customer foot traffic
2. Regional Patterns: Consider adding broader regional economic indicators, such as population or income