Title of the Project:-Predicting Delivery Delays

Problem Statement-Timely delivery is crucial to customer satisfaction and business effectiveness. Delays in outbound deliveries can result in missed sales, higher costs, and lower consumer trust. Currently, delivery performance is influenced by factors such as invoice processing time, outbound processing time, and logistics restrictions.

Objective-The goal is to create a prediction model that can assess the possibility of delivery delays using past sales, delivery (IOD)data. By detecting key patterns and risk factors, the model enables firms to take preventative measures to increase delivery efficiency.

Importing Libraries

In [5]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

In [6]:
```
#import datsets
sales_data = pd.read_csv("Sales_Data_s.csv")
IOD = pd.read_csv("IOD.csv")
OBD = pd.read_csv("OBD.csv")
```

In [7]:
```
sales_data.head()
```

Out[7]:

	Product	Month	Supplying_Plant_Code	Channel	Sales_Office_ID	Invoice_No	Invoi
0	WM-200D	01-09-2024	833	MT	119	1000852	13-
1	LuxFridge R5000	01-01-2023	833	MT	119	1000864	12-
2	CoolBox R1000	01-03-2022	303	MT	123	1001005	24-
3	UltraCool R8000	01-01-2022	854	E-com	115	1001021	24-
4	TopCool R1000	01-03-2024	765	E-com	103	1001094	05-

In [8]:
```
IOD.head()
```

Out[8]:

	Invoice_No	OBD_No	Billing_Date	Billing_Amount	IOD_Date	Delivery_Days
0	7666954	62957949	2022-01-01	21.3235	2022-01-02	3
1	2738112	24956315	2022-01-01	39.1772	2022-01-22	19
2	8884625	44008664	2022-01-01	86.3094	2022-02-24	50
3	8382225	24117073	2022-01-01	19.4287	2022-01-02	1
4	7699816	47406640	2022-01-01	43.5204	2022-02-25	50

In [9]:
```
OBD.head()
```

Out[9]:

	OBD_No	OBD_Date	Goods_Issue_Time	Pick_Time	Pick_Date
0	10004793	18-09-2024	0 days 14:30:00	0 days 12:00:00	18-09-2024
1	10007168	NaN	0 days 00:00:00	0 days 00:00:00	NaN
2	10012470	23-11-2022	0 days 13:00:00	0 days 10:15:00	25-11-2022
3	10012511	25-06-2022	0 days 11:45:00	0 days 13:15:00	27-06-2022
4	10013474	20-01-2024	0 days 16:15:00	0 days 16:45:00	21-01-2024

merge datasets

In [11]:
```
sales_iod = sales_data.merge(IOD, on='Invoice_No', how='left')
```

In [12]:
```
sales_iod.head()
```

Out[12]:

	Product	Month	Supplying_Plant_Code	Channel	Sales_Office_ID	Invoice_No	Invoi
0	WM-200D	01-09-2024	833	MT	119	1000852	13-
1	LuxFridge R5000	01-01-2023	833	MT	119	1000864	12-
2	CoolBox R1000	01-03-2022	303	MT	123	1001005	24-
3	UltraCool R8000	01-01-2022	854	E-com	115	1001021	24-
4	TopCool R1000	01-03-2024	765	E-com	103	1001094	05-

In [ ]:
```

```

In [13]:
```
sales_iod.drop(columns=['OBD_No_y'], axis=1, inplace=True)
sales_iod.rename(columns={'OBD_No_x':'OBD_No'}, inplace=True)
```

In [14]:
```
df = sales_iod.merge(OBD, on='OBD_No', how='left')
```

In [15]:
```
df.head()
```

Out[15]:

	Product	Month	Supplying_Plant_Code	Channel	Sales_Office_ID	Invoice_No	Invoi
0	WM-200D	01-09-2024	833	MT	119	1000852	13-
1	LuxFridge R5000	01-01-2023	833	MT	119	1000864	12-
2	CoolBox R1000	01-03-2022	303	MT	123	1001005	24-
3	UltraCool R8000	01-01-2022	854	E-com	115	1001021	24-
4	TopCool R1000	01-03-2024	765	E-com	103	1001094	05-

5 rows × 21 columns

In [16]:
```
df.columns
```

Out[16]:
```
Index(['Product', 'Month', 'Supplying_Plant_Code', 'Channel',
       'Sales_Office_ID', 'Invoice_No', 'Invoice_Date', 'Ship_to_Code_City',
       'OBD_No', 'QTY', 'Sales_Value', 'Supplying_Plant_City',
       'Sales_Office_City', 'Billing_Date', 'Billing_Amount', 'IOD_Date',
       'Delivery_Days', 'OBD_Date', 'Goods_Issue_Time', 'Pick_Time',
       'Pick_Date'],
      dtype='object')
```

In [17]:
```
df.isnull().sum()
```

Out[17]:
```
Product                   0
Month                     0
Supplying_Plant_Code      0
Channel                   0
Sales_Office_ID           0
Invoice_No                0
Invoice_Date              0
Ship_to_Code_City         0
OBD_No                    0
QTY                       0
Sales_Value               0
Supplying_Plant_City      0
Sales_Office_City         0
Billing_Date          14742
Billing_Amount        14742
IOD_Date              14742
Delivery_Days         14742
OBD_Date               8928
Goods_Issue_Time       1661
Pick_Time              1661
Pick_Date             14742
dtype: int64
```

converting object datatype to date time datatype

In [19]:
```
df['Invoice_Date']=pd.to_datetime(df['Invoice_Date'])
```

In [20]:
```
df['OBD_Date']=pd.to_datetime(df['OBD_Date'])
```

In [21]:
```
df['Pick_Date']=pd.to_datetime(df['Pick_Date'])
```

In [22]:
```
df['Month']=pd.to_datetime(df['Month'])
```

In [23]:
```
df['Billing_Date']=pd.to_datetime(df['Billing_Date'])
```

In [24]:
```
df['IOD_Date']=pd.to_datetime(df['IOD_Date'])
```

Handling Null Values

In [26]:
```
df['OBD_Date'].fillna(df['OBD_Date'].ffill(),inplace=True)
```

In [27]:
```
df['Pick_Date'].fillna(df['Pick_Date'].ffill(),inplace=True)
```

In [28]:
```
df['Billing_Date'].fillna(df['Billing_Date'].bfill(),inplace=True)
```

In [29]:
```
df['IOD_Date'].fillna(df['IOD_Date'].bfill(),inplace=True)
```

In [30]:
```
df['Billing_Amount'].fillna(df['Billing_Amount'].median(),inplace=True)
```

In [31]:
```
df['Delivery_Days'].fillna(df['Delivery_Days'].median(),inplace=True)
```

In [32]:
```
#as we are predicting delivery delays so if they delivery is greater than 5,assi
#This will create a binary classification target.
```

In [33]:
```
delivery_delay_days = 5
df['Delivery_Delayed'] = np.where(df['Delivery_Days'] > delivery_delay_days, 1,
df = df.drop(columns=['Delivery_Days'])
print(df['Delivery_Delayed'].value_counts())
```

```
Delivery_Delayed
1    29343
0     8657
Name: count, dtype: int64
```

In [34]:
```
num_dataframe=df.select_dtypes(exclude=object)
num_dataframe.head()
```

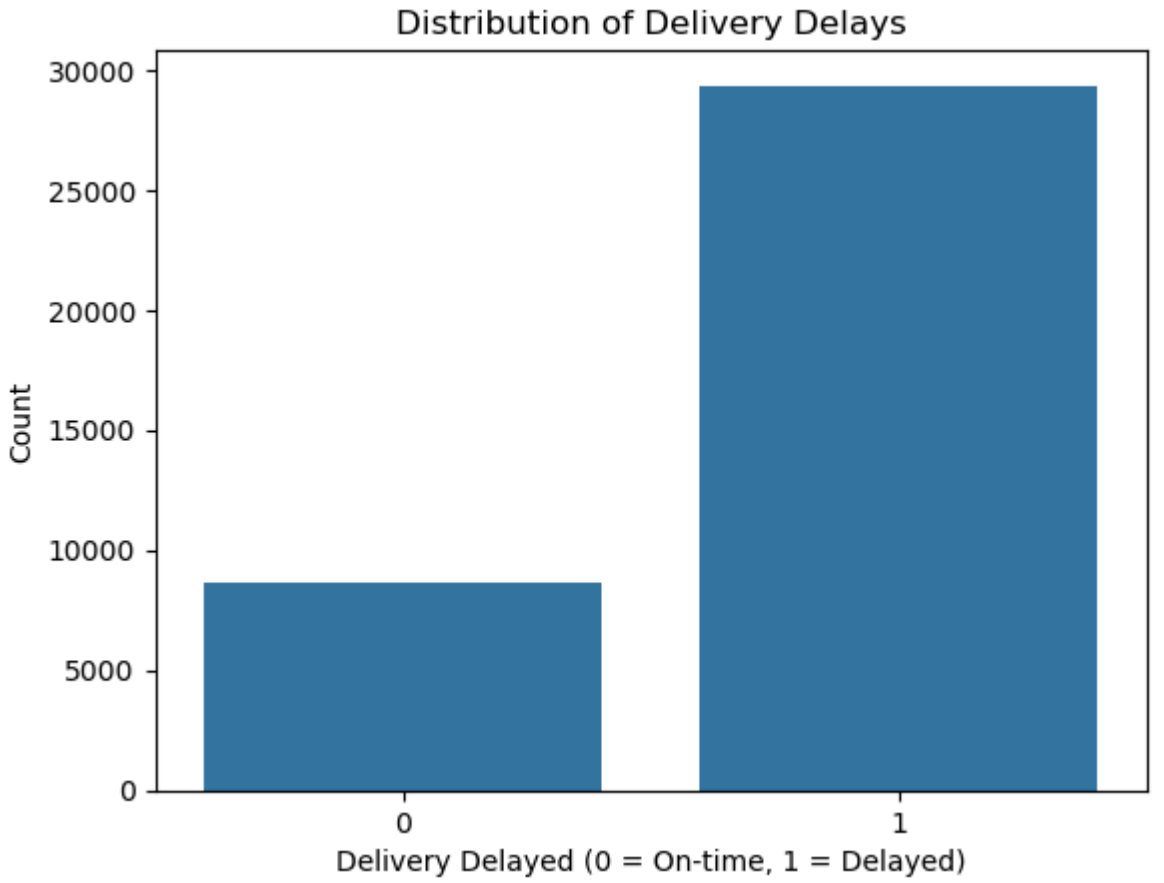
Out[34]:

	Month	Supplying_Plant_Code	Sales_Office_ID	Invoice_No	Invoice_Date	OBD_No
0	2024-01-09	833	119	1000852	2024-09-13	10062027
1	2023-01-01	833	119	1000864	2023-01-12	97593071
2	2022-01-03	303	123	1001005	2022-03-24	47670102
3	2022-01-01	854	115	1001021	2022-01-24	87111120
4	2024-01-03	765	103	1001094	2024-03-05	54107285

In [35]:
```
num_dataframe.drop(columns=['Invoice_No','Supplying_Plant_Code','Sales_Value','S
```

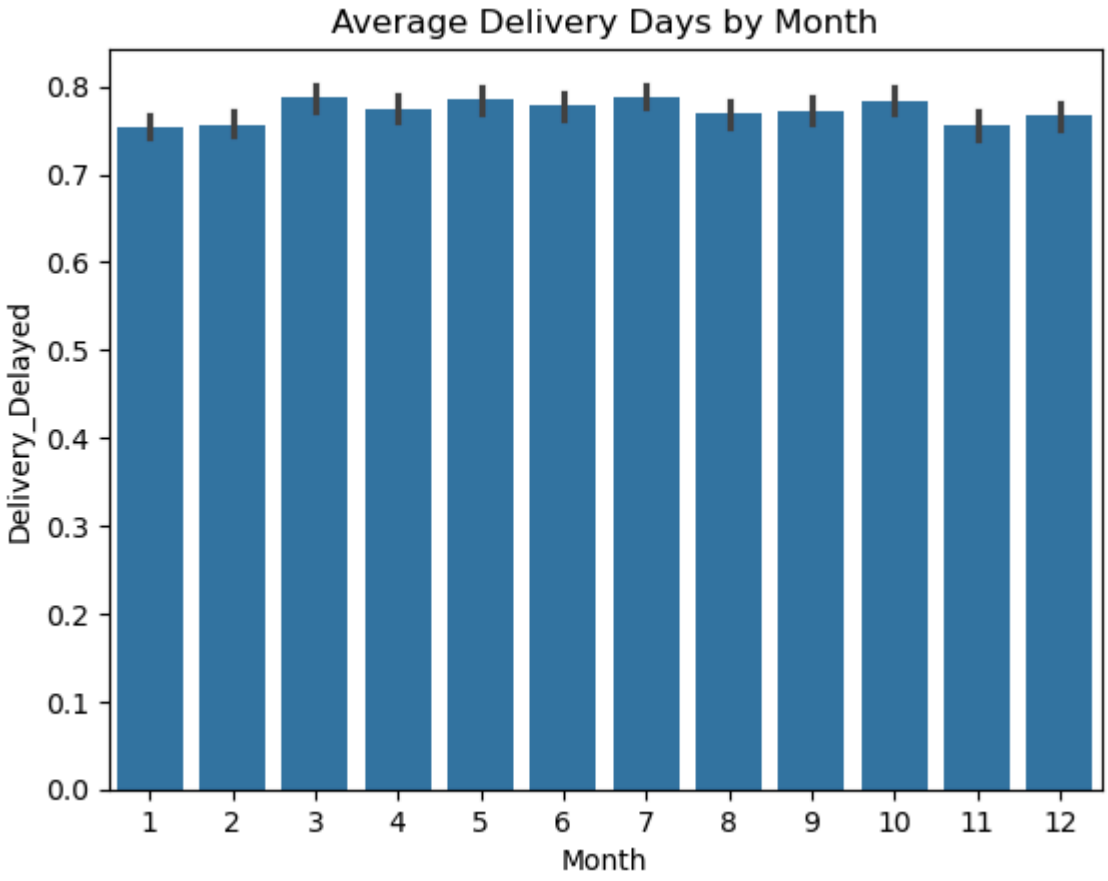
Data Visulatizations

In [37]:
```
sns.countplot(x='Delivery_Delayed', data=num_dataframe)
plt.title("Distribution of Delivery Delays")
plt.xlabel("Delivery Delayed (0 = On-time, 1 = Delayed)")
plt.ylabel("Count")
plt.show()
```



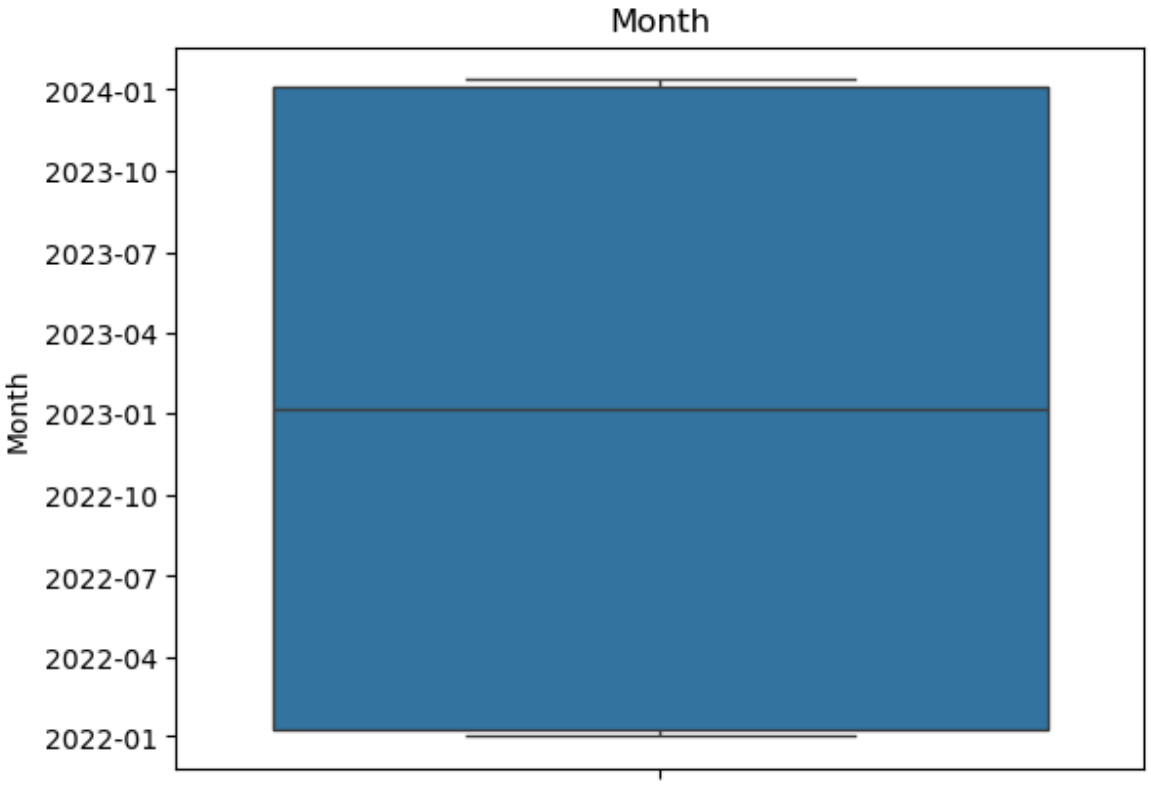
In [38]:
```
num_dataframe['Month'] = num_dataframe['OBD_Date'].dt.month
sns.barplot(x='Month', y='Delivery_Delayed', data=num_dataframe)
```

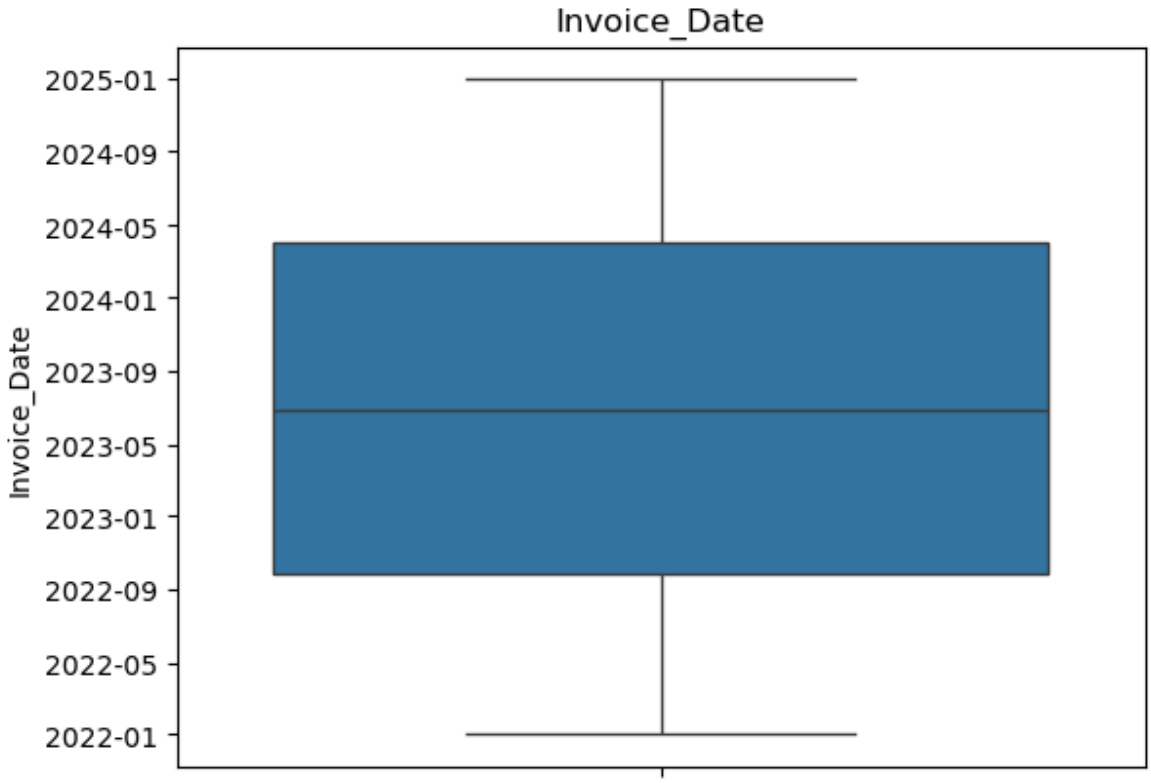
```
plt.title("Average Delivery Days by Month")
plt.show()
```

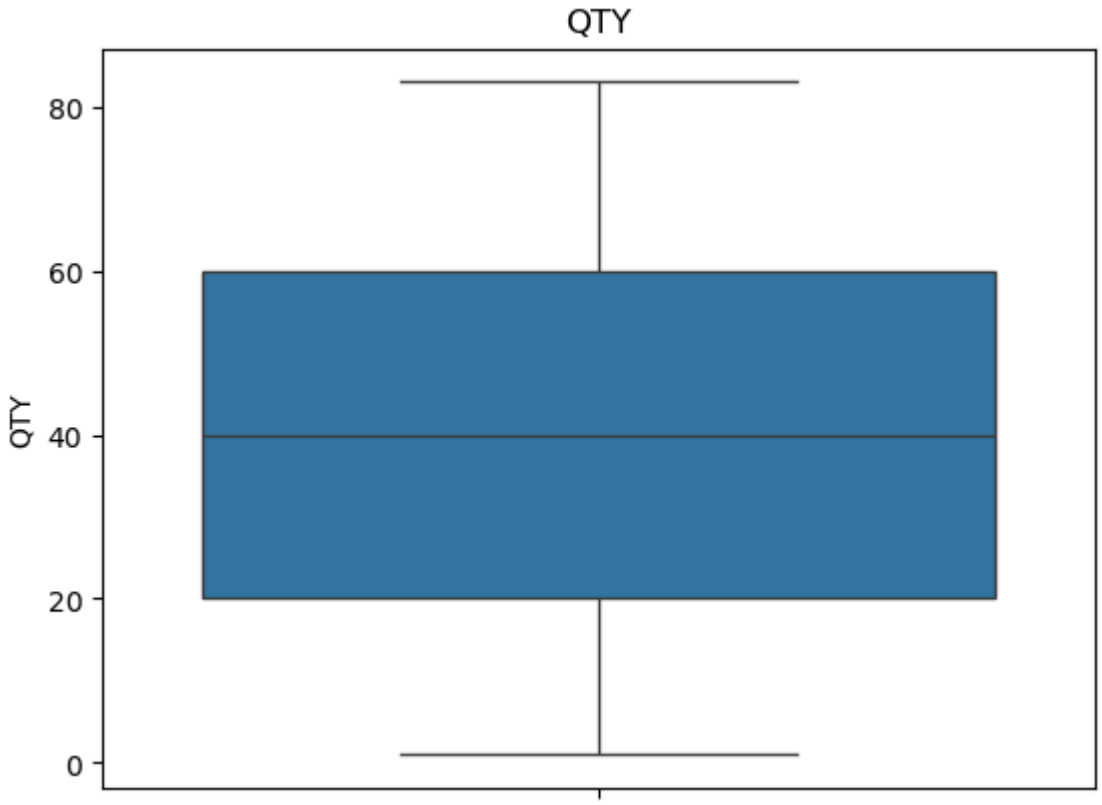


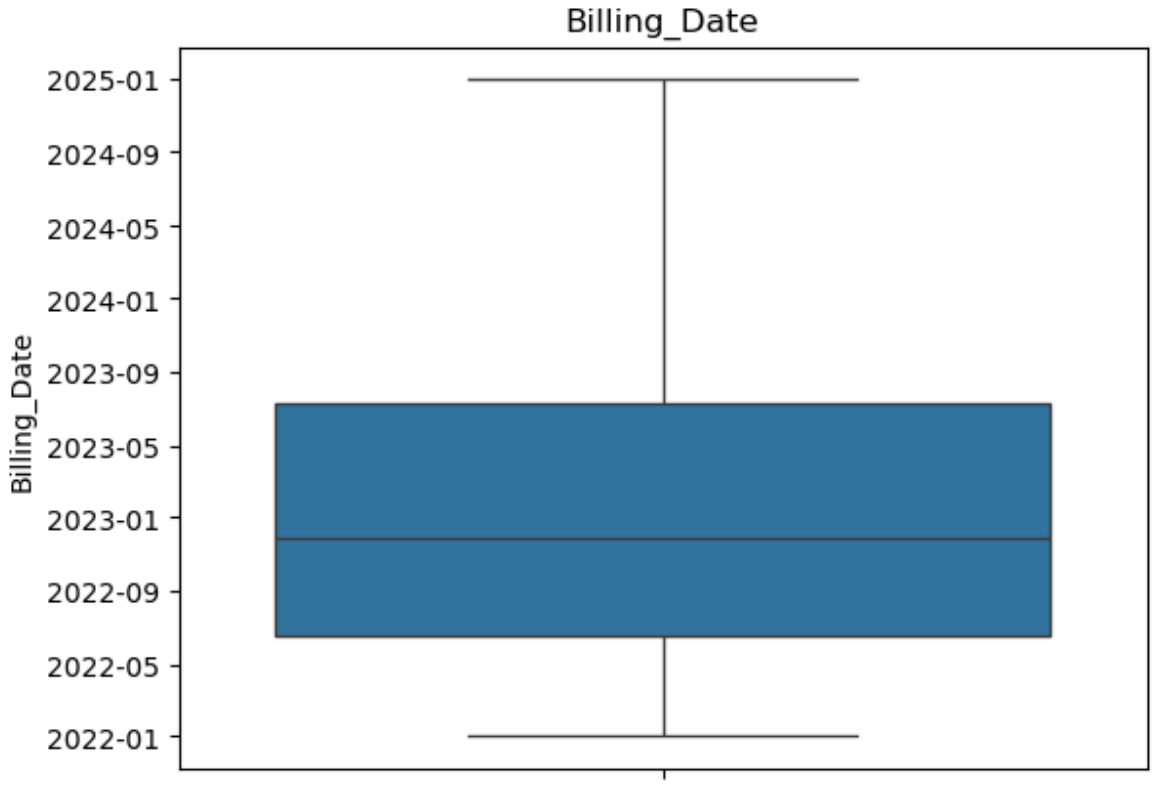
Outliers Detection

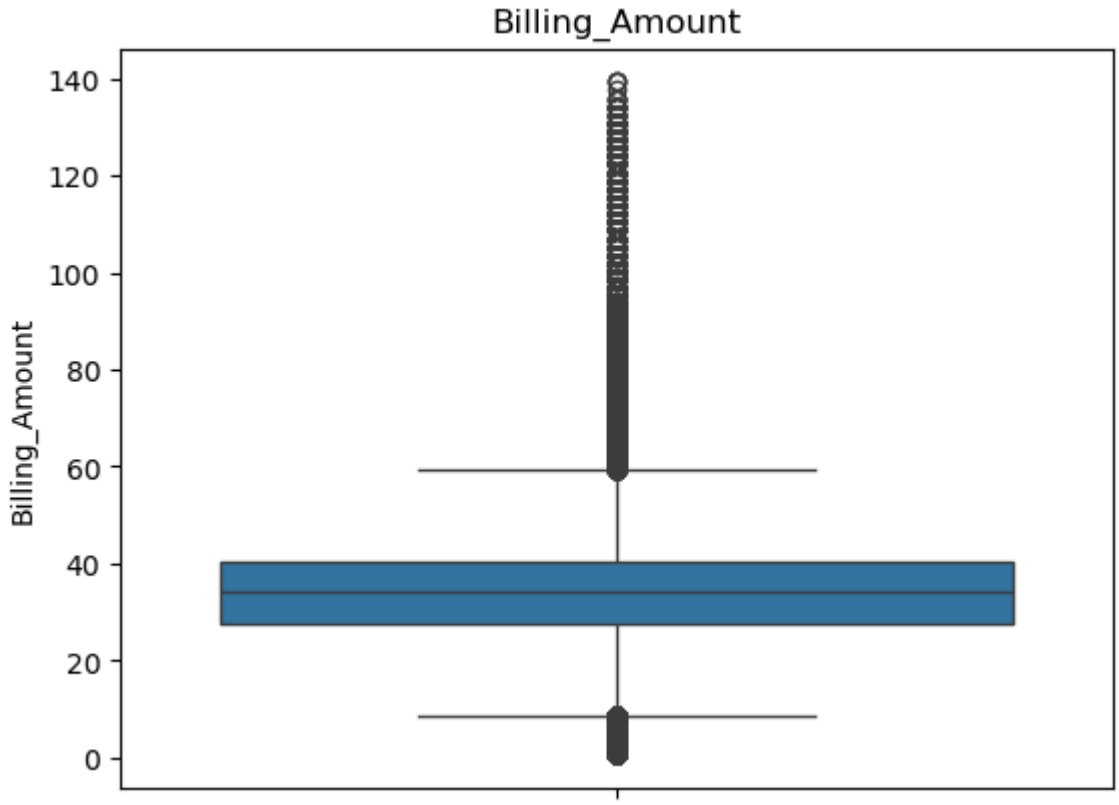
In [40]:
```
for column in num_dataframe:
    sns.boxplot(df[column])
    plt.title(column)
    plt.show()
```

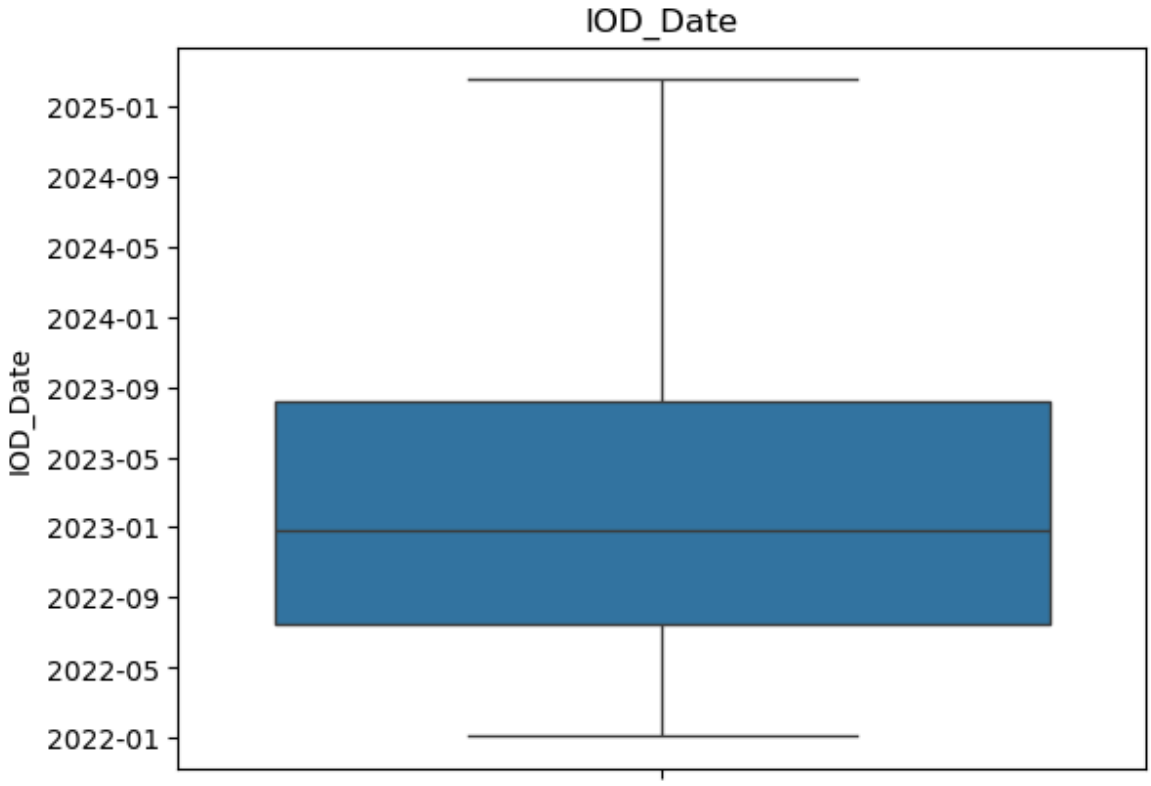

Month
Month
2024-01
2023-10
2023-07
2023-04
2023-01
2022-10
2022-07
2022-04
2022-01

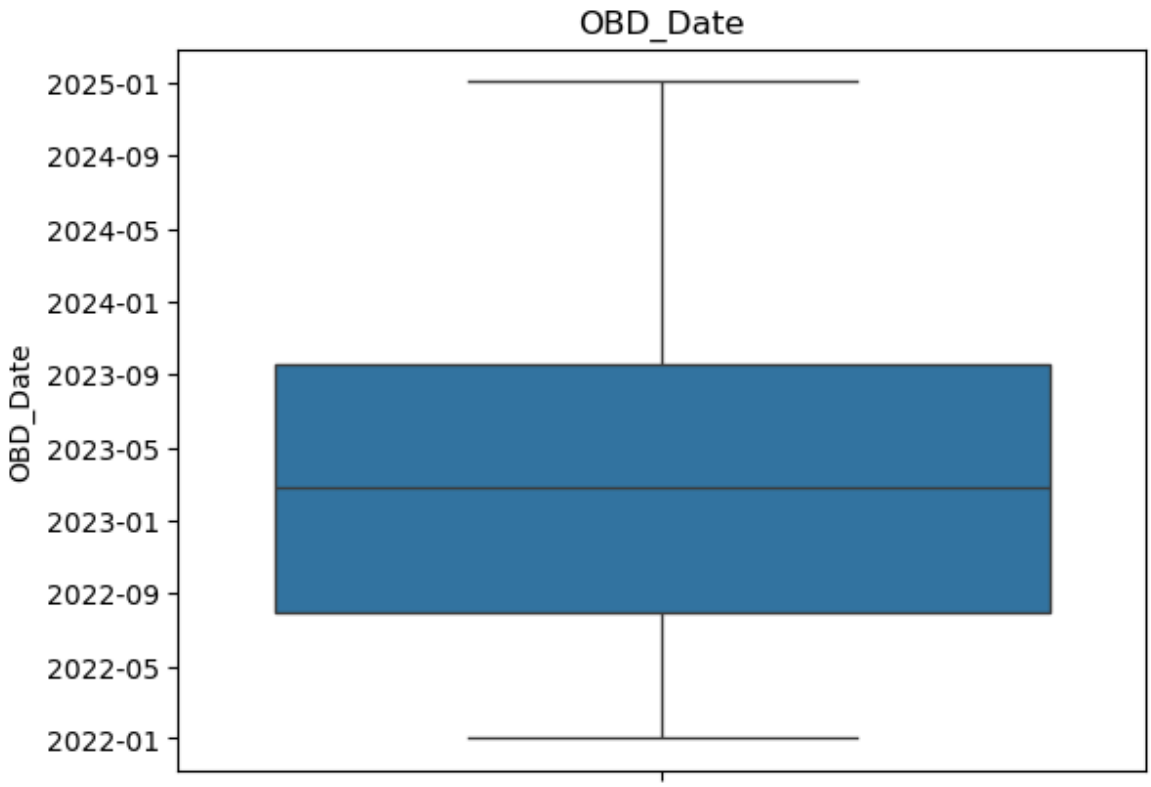

Invoice_Date
Invoice_Date
2025-01
2024-09
2024-05
2024-01
2023-09
2023-05
2023-01
2022-09
2022-05
2022-01


QTY
QTY
80
60
40
20
0


Billing_Date
Billing_Date
2025-01
2024-09
2024-05
2024-01
2023-09
2023-05
2023-01
2022-09
2022-05
2022-01


Billing_Amount
Billing_Amount
140
120
100
80
60
40
20
0


IOD_Date
IOD_Date
2025-01
2024-09
2024-05
2024-01
2023-09
2023-05
2023-01
2022-09
2022-05
2022-01



In [41]:
```
##outliers detection
```

In [42]:
```
cols=['Billing_Amount']
for column in cols:

    print(column,":")
    q1=num_dataframe[column].quantile(0.25)
    q3=num_dataframe[column].quantile(0.75)
    iqr=q3-q1
    upper_tail=q3+1.5*iqr
```

```
    lower_tail=q1-1.5*iqr

    print("q1 --->",q1)
    print("q3 --->",q3)
    print("iqr --->",iqr)
    print("upper tail --->",upper_tail)
    print("lower_tail --->",lower_tail)
    print("-"*50)
```

```
Billing_Amount :
q1 ---> 27.6357
q3 ---> 40.2985
iqr ---> 12.662799999999997
upper tail ---> 59.292699999999996
lower_tail ---> 8.641500000000004
--------------------------------------------------
```

In [43]:
```
num_dataframe.loc[num_dataframe['Billing_Amount']>upper_tail]#we are not handlin
```

Out[43]:

	Month	Invoice_Date	QTY	Billing_Date	Billing_Amount	IOD_Date	OBD_Date
6	5	2023-05-24	68	2023-05-24	83.1626	2023-07-15	2023-05-24
8	3	2024-03-05	41	2024-03-05	69.6652	2024-03-31	2024-03-05
15	12	2023-12-24	68	2023-12-24	83.1626	2024-02-14	2023-12-26
46	5	2022-04-29	43	2022-04-29	73.0635	2022-06-02	2022-05-01
53	7	2023-07-22	64	2023-07-22	78.2707	2023-09-11	2023-07-25
...	...	...	...	...	...	...	...
37935	11	2022-11-21	68	2022-11-21	76.3157	2023-01-11	2022-11-24
37938	2	2023-02-16	72	2023-02-16	80.8049	2023-04-09	2023-02-17
37940	1	2023-01-10	64	2023-01-10	72.6816	2023-03-05	2023-01-11
37955	12	2022-12-09	66	2022-12-09	112.1440	2023-01-31	2022-12-11
37988	10	2023-10-18	69	2023-10-18	84.3856	2023-12-11	2023-10-20

3567 rows × 8 columns

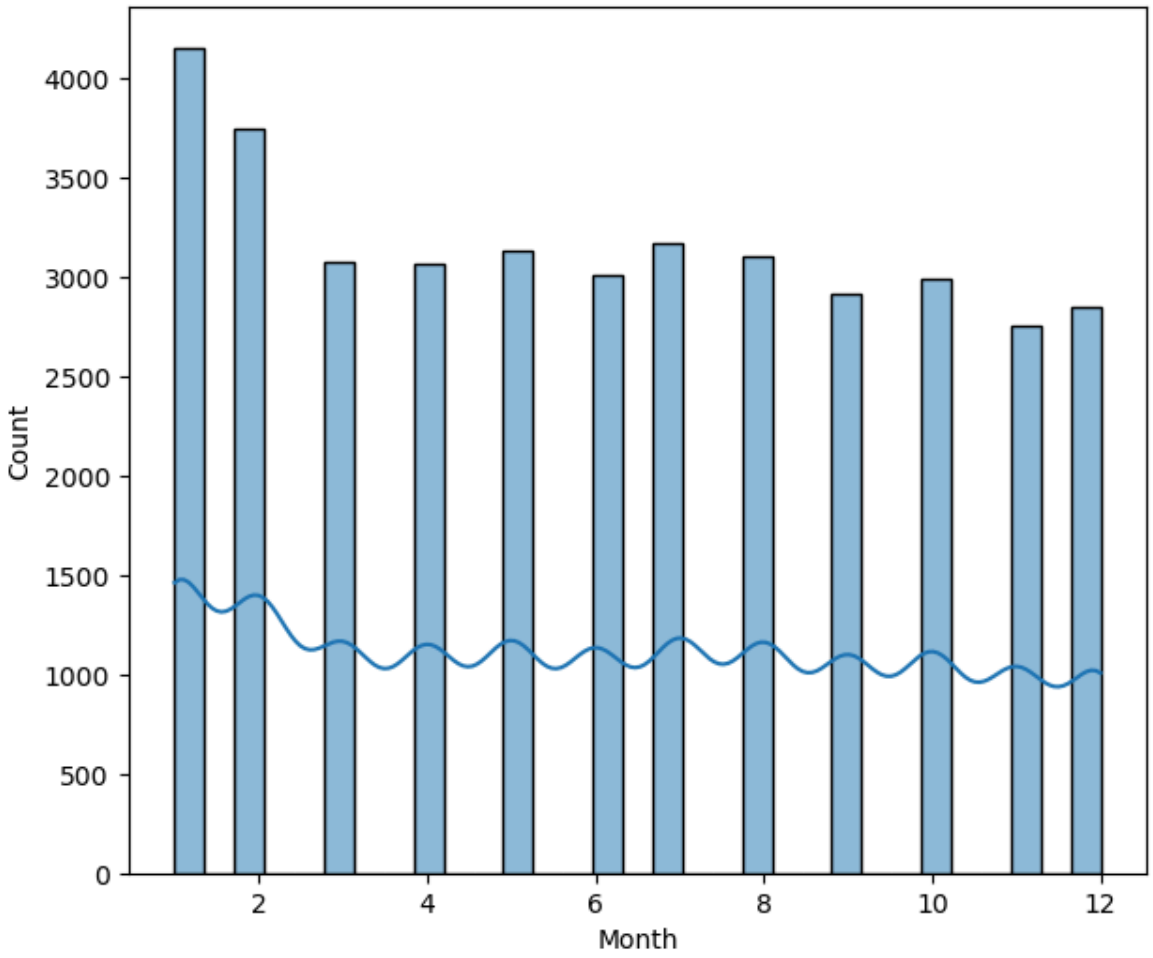
In [44]:
```
from scipy.stats import skew
```

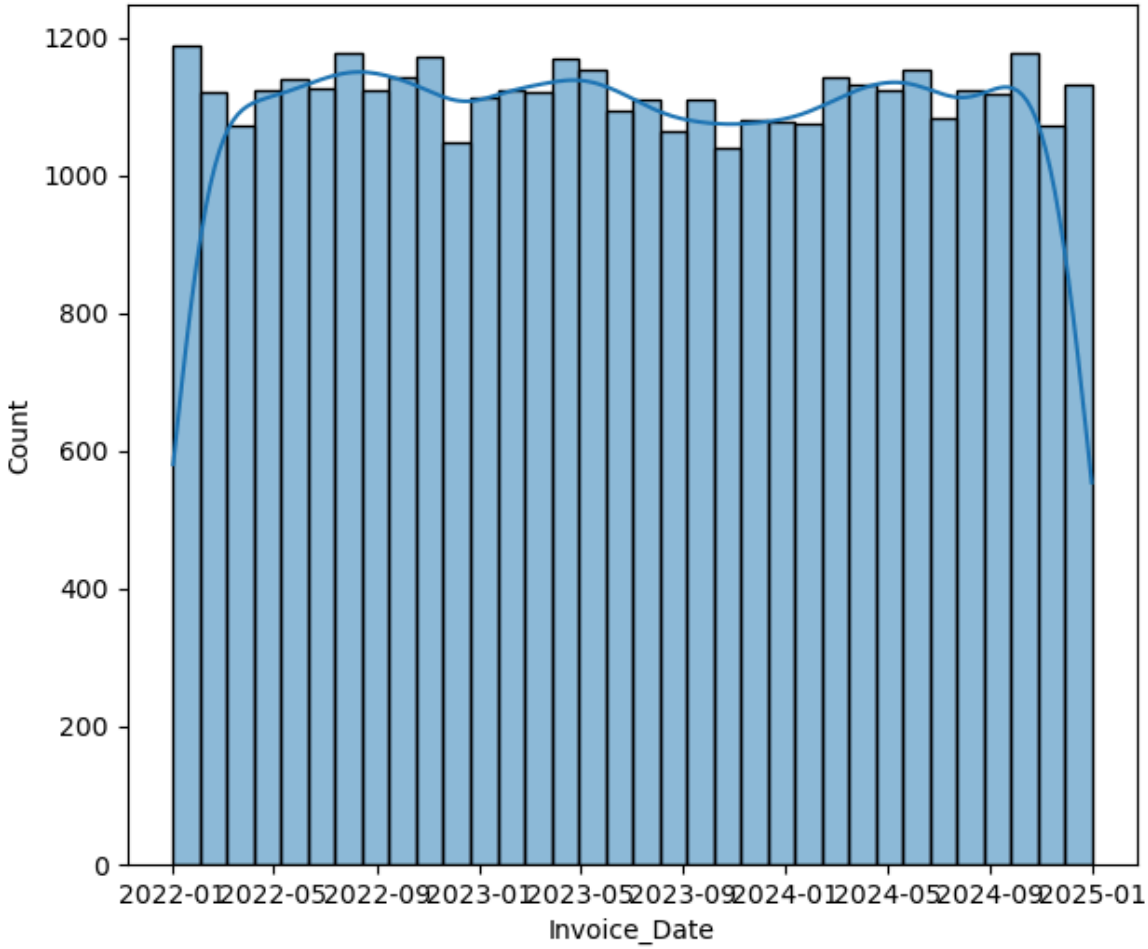
In [45]:
```
num_skew=['QTY','Billing_Amount']
for num_column in num_skew:
        print(num_column,"===>",skew(num_dataframe[num_column]))
```

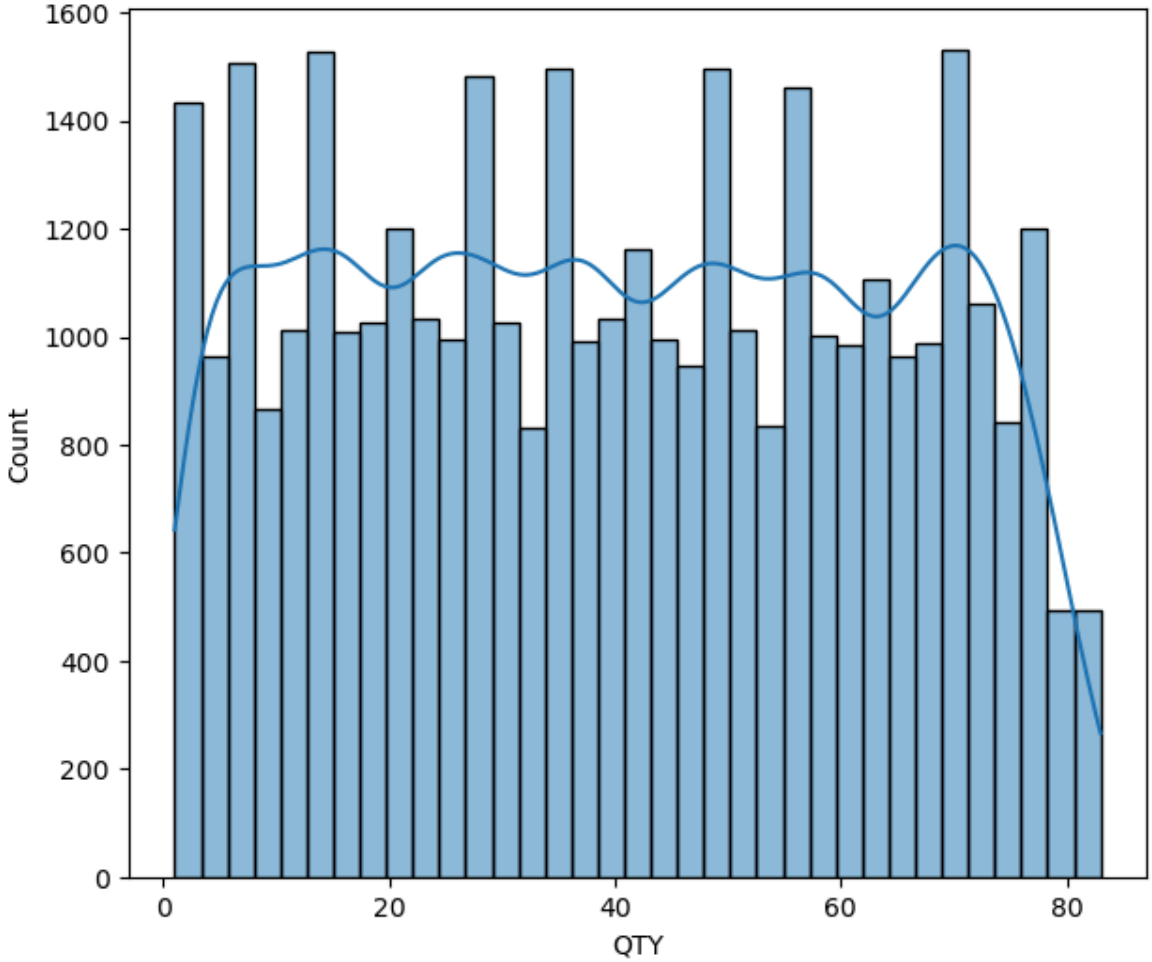
```
QTY ===> 0.027271167653722773
Billing_Amount ===> 1.1931918340284793
```

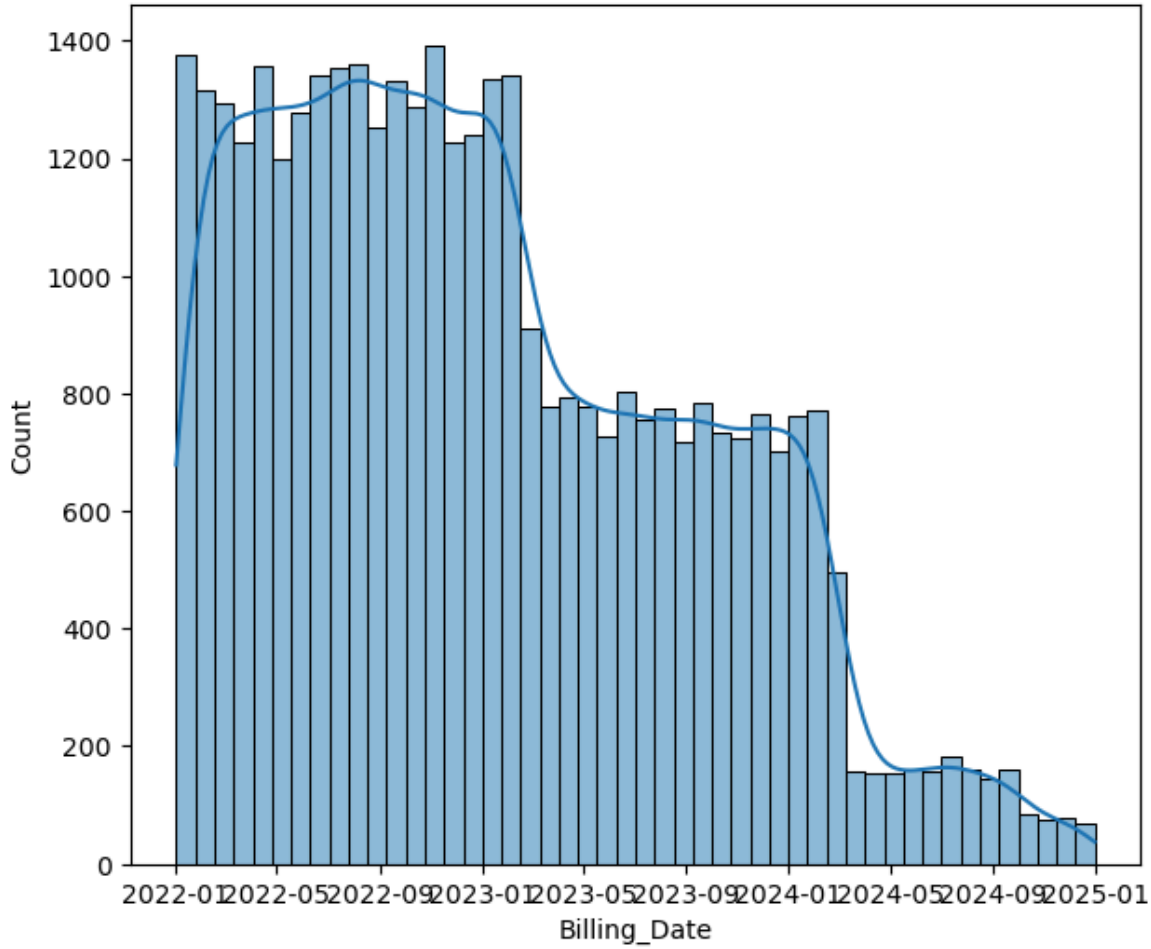
In [46]:
```
# graph of skewness visualization
```

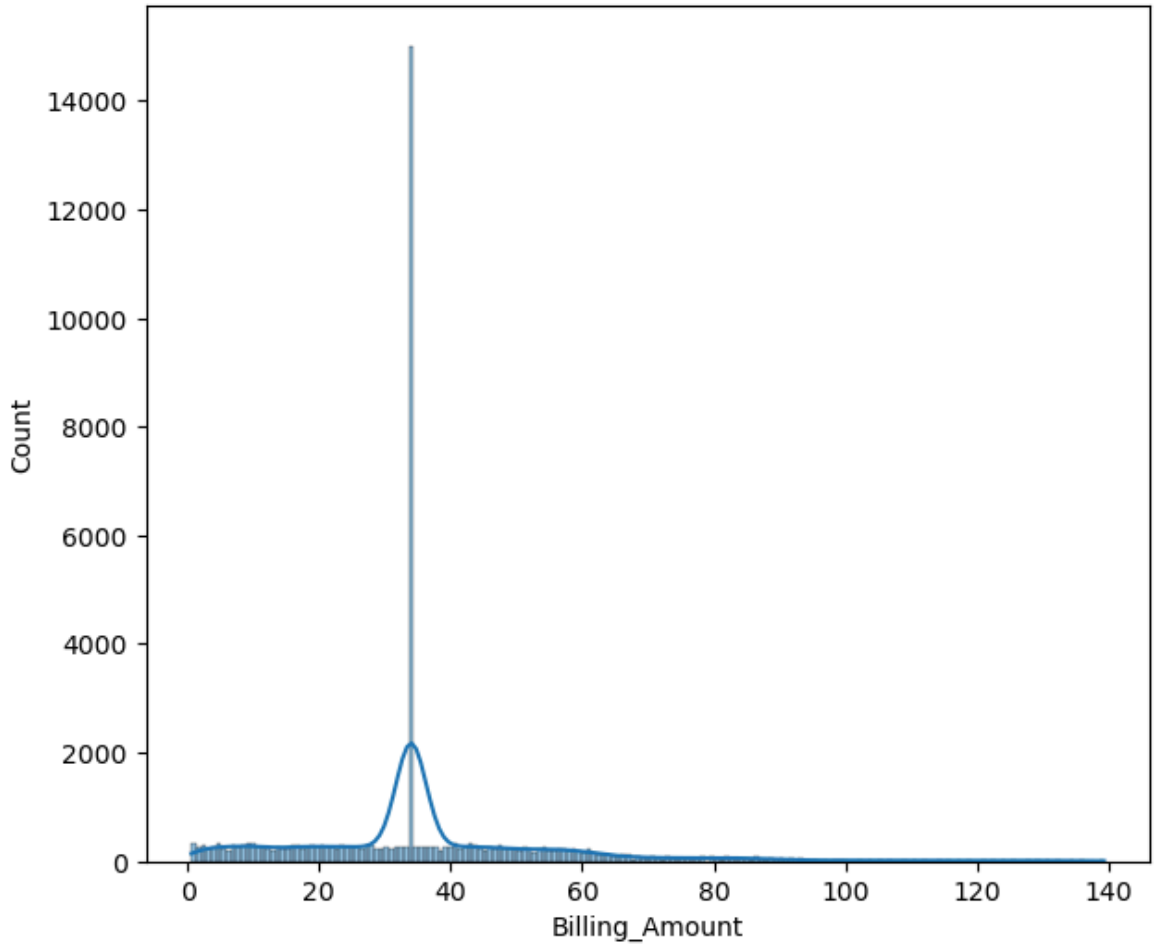
In [47]:
```
for num_column in num_dataframe:
    plt.figure(figsize=(7,6))
    sns.histplot(num_dataframe[num_column], kde=True)
    plt.show()
```

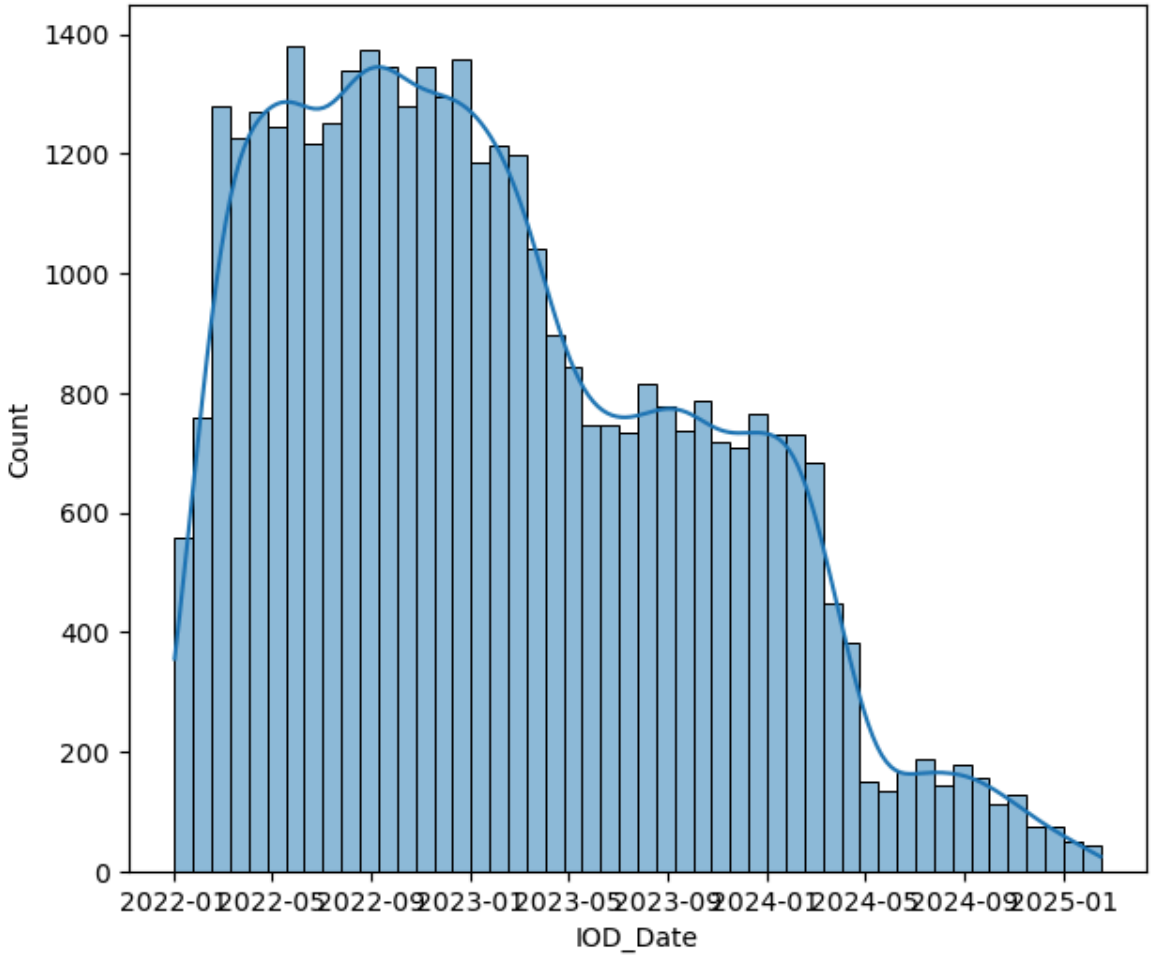


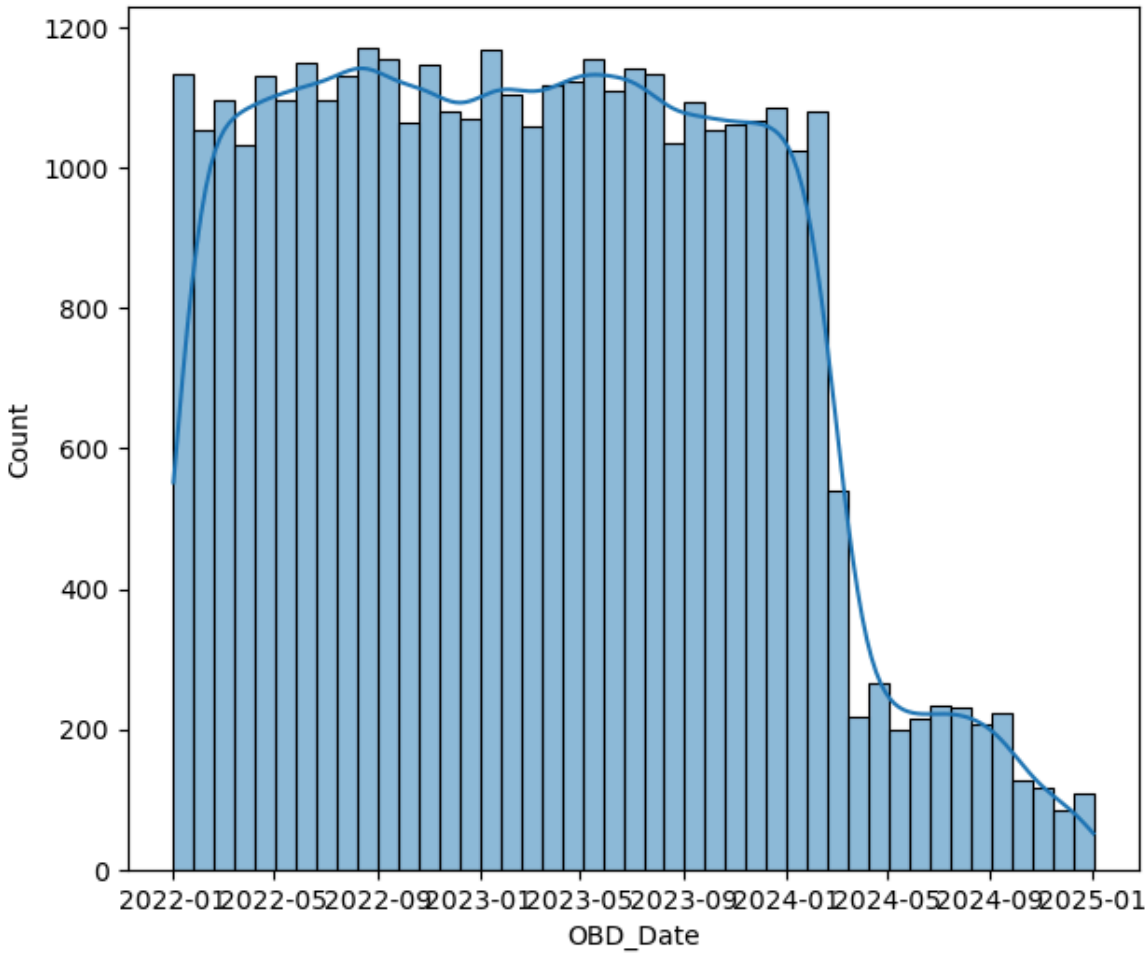

1200
1000
800
600
400
200
0
Count
2022-01
2022-05
2022-09
2023-01
2023-05
2023-09
2024-01
2024-05
2024-09
2025-01
Invoice_Date

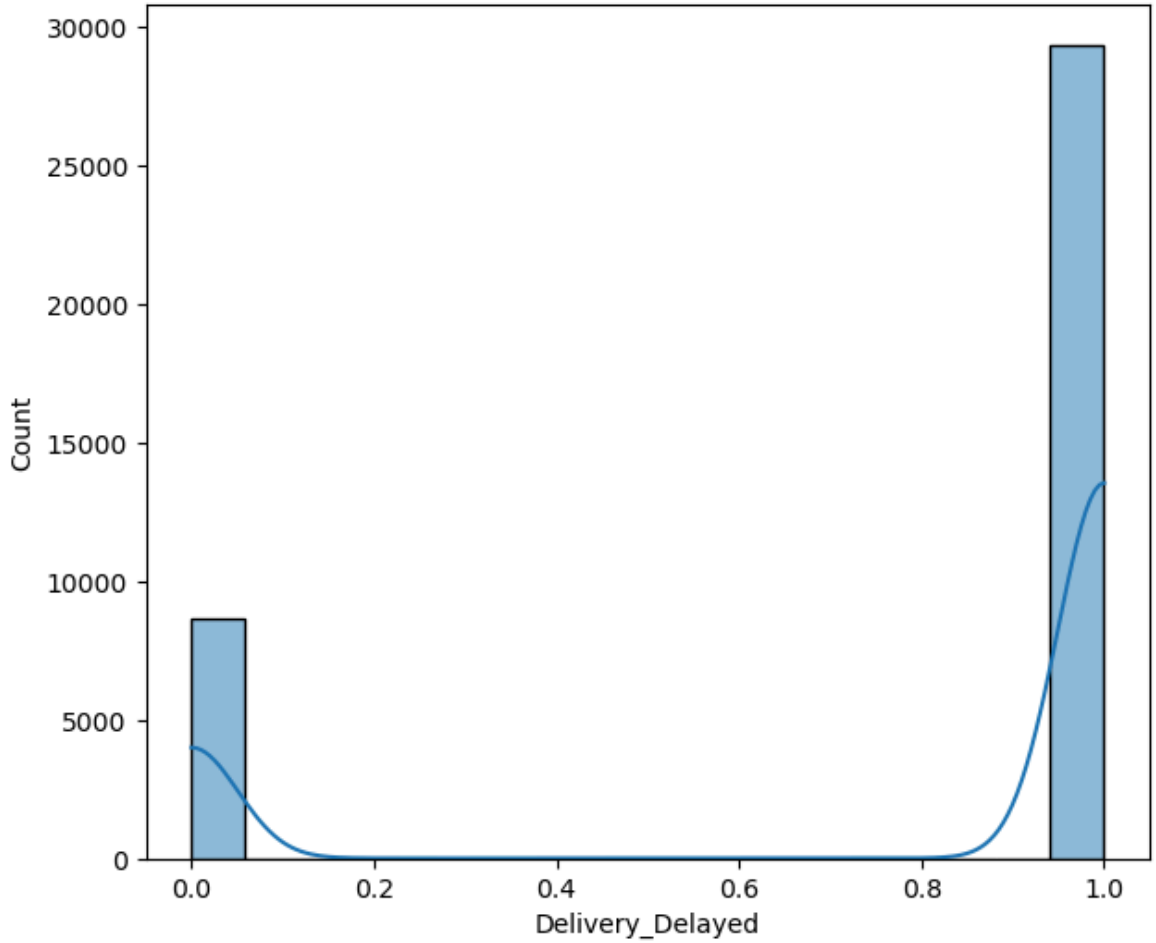

1600
1400
1200
1000
800
600
400
200
0
Count
0
20
40
60
80
QTY


1400
1200
1000
800
600
400
200
0
Count
2022-01
2022-05
2022-09
2023-01
2023-05
2023-09
2024-01
2024-05
2024-09
2025-01
Billing_Date


14000
12000
10000
8000
6000
4000
2000
0
Count
0
20
40
60
80
100
120
140
Billing_Amount


1400
1200
1000
800
600
400
200
0
Count
2022-01
2022-05
2022-09
2023-01
2023-05
2023-09
2024-01
2024-05
2024-09
2025-01
IOD_Date


1200
1000
800
600
400
200
0
Count
2022-01
2022-05
2022-09
2023-01
2023-05
2023-09
2024-01
2024-05
2024-09
2025-01
OBD_Date



In [48]:
```
# Feature Scaling
```

In [49]:
```
from sklearn.preprocessing import MinMaxScaler
```

In [50]:
```
minmax=MinMaxScaler()
```

In [51]:
```
minmax_cols=['QTY','Billing_Amount']
```

In [52]:
```
for i in minmax_cols:
    num_dataframe[i]=minmax.fit_transform(num_dataframe[[i]])
```

In [53]:
```
num_dataframe
```

Out[53]:

	Month	Invoice_Date	QTY	Billing_Date	Billing_Amount	IOD_Date	OBD_Da
0	9	2024-09-13	0.268293	2023-01-12	0.240625	2023-01-13	2024-(
1	1	2023-01-12	0.243902	2023-01-12	0.120803	2023-01-13	2023-(
2	3	2022-03-24	0.829268	2022-03-24	0.378414	2022-05-15	2022-(
3	1	2022-01-24	0.390244	2022-01-24	0.151624	2022-03-05	2022-(
4	3	2024-03-05	0.060976	2024-03-05	0.029620	2024-03-06	2024-(
...	...	...	...	...	...	...	
37995	9	2023-09-09	0.439024	2022-11-29	0.240625	2022-11-30	2023-(
37996	3	2023-03-31	0.536585	2022-11-29	0.240625	2022-11-30	2023-(
37997	8	2023-08-30	0.792683	2022-11-29	0.240625	2022-11-30	2023-(
37998	12	2022-11-29	0.182927	2022-11-29	0.126643	2022-11-30	2022-1
37999	6	2022-06-18	0.524390	2022-06-18	0.239737	2022-07-31	2022-(

38000 rows × 8 columns

Object Columns

In [55]:
```
object_dataframe = df.select_dtypes(include=object)
object_dataframe.head()
```

Out[55]:

	Product	Channel	Ship_to_Code_City	Supplying_Plant_City	Sales_Office_City	Goods
0	WM-200D	MT	Malad	Wada	Surat	0 c
1	LuxFridge R5000	MT	Shirgaon	Wada	Surat	0 c
2	CoolBox R1000	MT	Chennai	Vijayawada	Vijayawada	0 c
3	UltraCool R8000	E-com	Sadabad	Delhi	New Delhi	0 c
4	TopCool R1000	E-com	Hamirpur	Zirakpur	Chandigarh	0 c

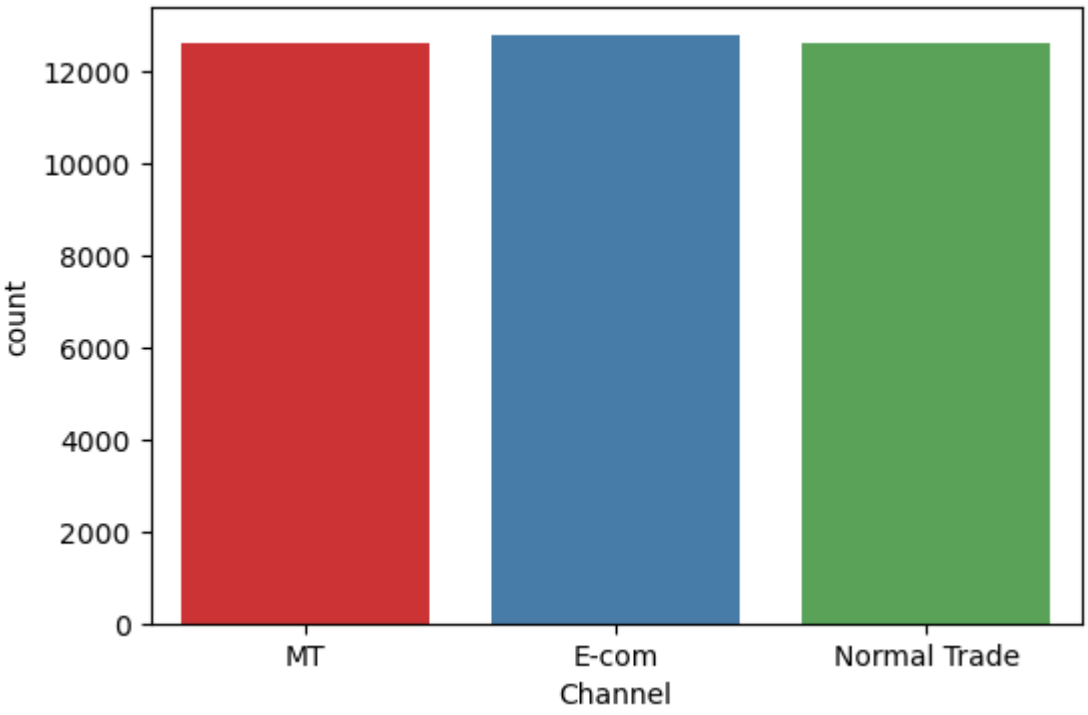
In [56]:
```
object_dataframe.drop(columns=['Goods_Issue_Time','Pick_Time'],axis=1,inplace=Tr
```

In [57]:
```
for column in object_dataframe:
    print(object_dataframe[column].value_counts())
    print("-"*50)
```

```
Product
CoolBox R1000         2796
WM-200D               2772
BakePro O3000         2770
CleanAir-C100         2741
WM-101SA              2731
TopCool R1000         2728
WM-TL10               2713
FreshBreeze-300       2707
PowerMix              2689
UltraCool R8000       2685
TurboConvection O9    2678
FreshFreeze R2800     2675
FreshCool R2000       2660
LuxFridge R5000       2655
Name: count, dtype: int64
--------------------------------------------------
Channel
E-com           12765
Normal Trade    12620
MT              12615
Name: count, dtype: int64
--------------------------------------------------
Ship_to_Code_City
Pune             232
Lucknow          221
Bilaspur         217
Aurangabad       205
Nagpur           203
                ...
Kallam             3
Hosur              3
Baramati           3
Pattukottai        3
Goregaon west      2
Name: count, Length: 1295, dtype: int64
--------------------------------------------------
Supplying_Plant_City
Cochin         4589
Ahmedabad      3068
Zirakpur       3037
Vijayawada     2965
Guwahati       1597
Pune           1576
Bangalore      1554
Wada           1546
Bhiwandi       1545
Chennai        1529
Nagpur         1529
Indore         1525
Hyderabad      1521
Goa            1519
Raipur         1519
Patna          1489
Lucknow        1487
Delhi          1485
Jaipur         1472
Bhubaneswar    1448
Name: count, dtype: int64
--------------------------------------------------
```

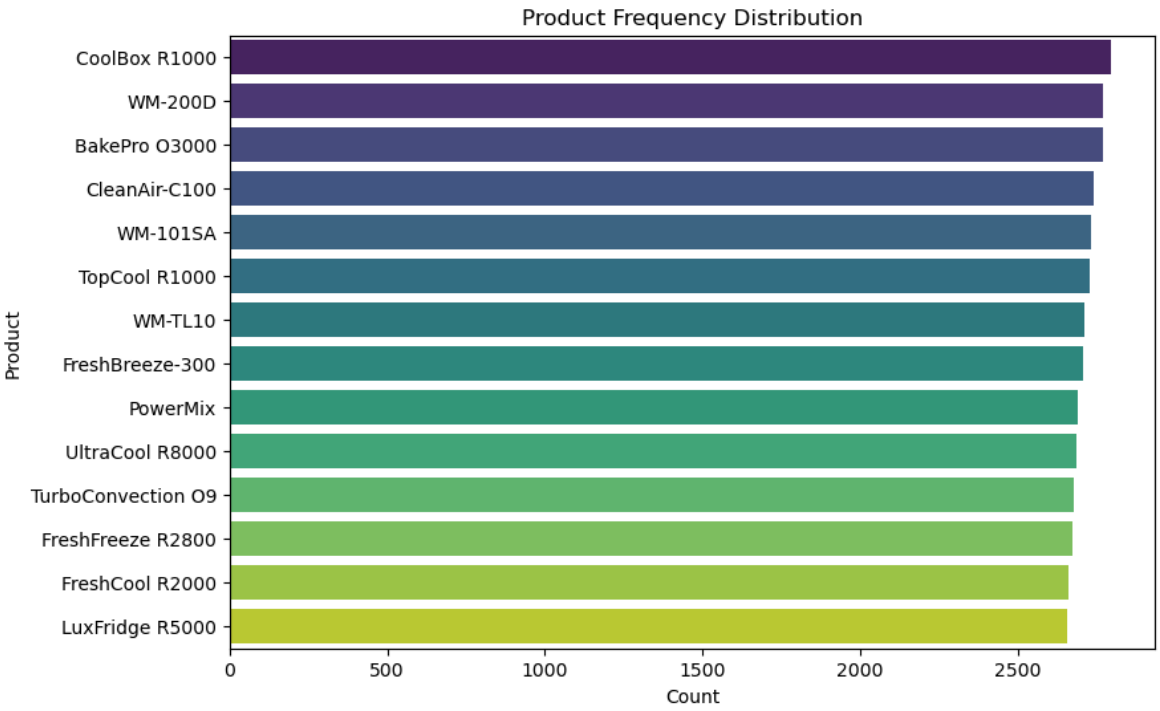
```
Sales_Office_City
Guwahati               1597
Pune                   1576
Bengaluru              1554
Coimbatore             1554
Surat                  1546
Thane                  1545
Vadodara               1542
Chennai                1529
Nagpur                 1529
Chandigarh             1527
Ahmedabad              1526
Indore                 1525
Kochi                  1524
Hyderabad              1521
Raipur                 1519
Goa                    1519
Thiruvananthapuram     1511
Ludhiana               1510
Vijayawada             1502
Patna                  1489
Lucknow                1487
New Delhi              1485
Jaipur                 1472
Visakhapatnam          1463
Bhubaneswar            1448
Name: count, dtype: int64
--------------------------------------------------
```

In [58]:
```
plt.figure(figsize=(6,4))
sns.countplot(x='Channel',data=df,palette = "Set1")
plt.show()
```



In [59]:
```
plt.figure(figsize=(9,6))
sns.countplot(y=df["Product"], order=df["Product"].value_counts().index, palette
plt.xlabel("Count")
plt.ylabel("Product")
```

```
plt.title("Product Frequency Distribution")
plt.show()
```



Data Encoding

In [61]:
```
from sklearn.preprocessing import LabelEncoder
```

In [62]:
```
label_encoder=LabelEncoder()
```

In [63]:
```
object_dataframe['Channel'] = label_encoder.fit_transform(object_dataframe['Chan
```

In [64]:
```
object_dataframe.head()
```

Out[64]:

	Product	Channel	Ship_to_Code_City	Supplying_Plant_City	Sales_Office_City
0	WM-200D	1	Malad	Wada	Surat
1	LuxFridge R5000	1	Shirgaon	Wada	Surat
2	CoolBox R1000	1	Chennai	Vijayawada	Vijayawada
3	UltraCool R8000	0	Sadabad	Delhi	New Delhi
4	TopCool R1000	0	Hamirpur	Zirakpur	Chandigarh

In [65]:
```
#one-hot Encoding(get dummies)
```

In [66]:
```
object_dataframe= pd.get_dummies(object_dataframe['Product']).astype(int)
```

In [67]:
```
object_dataframe.shape
```

Out[67]: (38000, 14)

In [68]:
```
concat_dataframe = pd.concat([object_dataframe,num_dataframe], axis=1)
concat_dataframe.head()
```

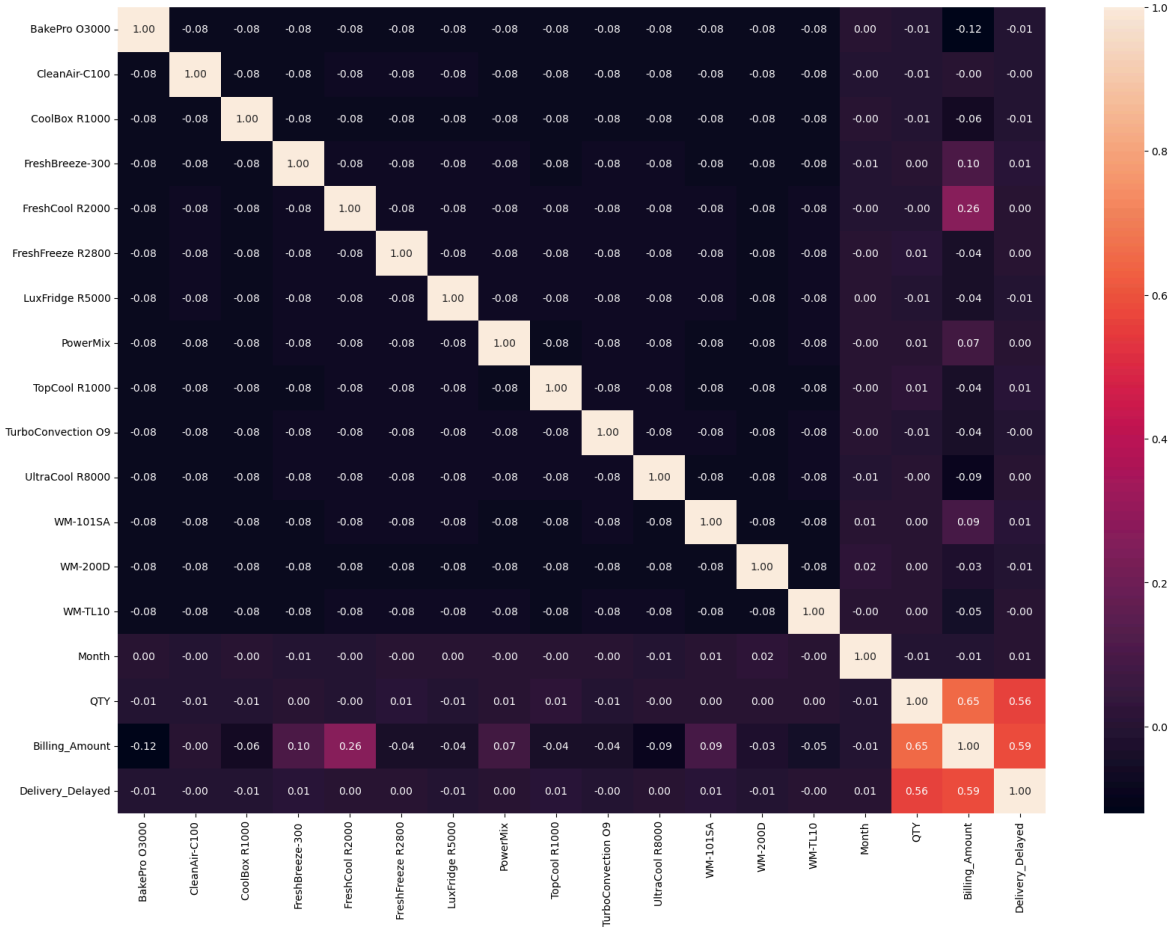
Out[68]:

	BakePro O3000	CleanAir-C100	CoolBox R1000	FreshBreeze-300	FreshCool R2000	FreshFreeze R2800	LuxFridge R5000	Powe
0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	1	
2	0	0	1	0	0	0	0	
3	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	

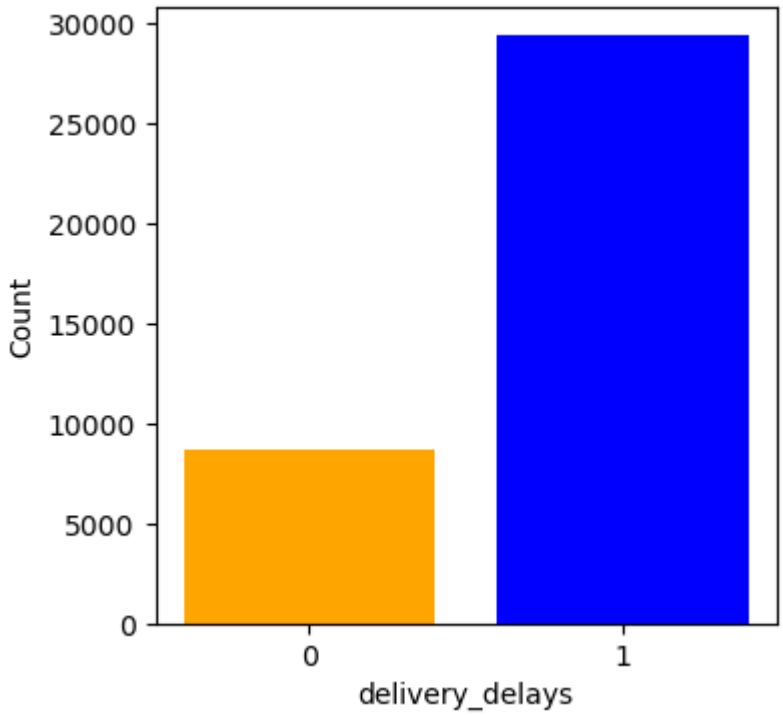
5 rows × 22 columns

In [69]:
```
concat_dataframe.drop(columns=['Invoice_Date','Billing_Date','IOD_Date','OBD_Dat
```

In [70]:
```
plt.figure(figsize=(20,14))
sns.heatmap(concat_dataframe.corr(),fmt = '.2f',annot = True)
plt.show()
```



In [71]:
```
target_count = num_dataframe['Delivery_Delayed'].value_counts()
plt.figure(figsize=(4,4))
plt.bar(x=target_count.index, height = target_count.values,color=['blue','orange
plt.xticks(ticks = [0,1])
plt.xlabel("delivery_delays")
plt.ylabel("Count")
plt.show()
```



In [72]:
```
x = concat_dataframe.drop('Delivery_Delayed', axis=1)
y = concat_dataframe['Delivery_Delayed']
```

train,test and split the dataset

In [74]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.30)
```

Applying Algorithms

In [76]:
```
from sklearn.tree import DecisionTreeClassifier,plot_tree
from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification_report, confusion_matrix,accuracy_sco
```

In [77]:
```
dt_clf=DecisionTreeClassifier(random_state=11)
dt_clf.fit(x_train,y_train)
```

Out[77]:
```
DecisionTreeClassifier(random_state=11)
```

In [78]:
```
y_pred_train1=dt_clf.predict(x_train)

cnf_matrix= confusion_matrix(y_train,y_pred_train1)
print("confusion matrix:\n",cnf_matrix)
print("-"*45)

accuracy=accuracy_score(y_train,y_pred_train1)
print("Accuracy:",accuracy)
print("-"*45)

clf_report=classification_report(y_train,y_pred_train1)
print("classification report:\n",clf_report)
```

```
confusion matrix:
 [[ 5869   197]
 [  277 20257]]
---------------------------------------------
Accuracy: 0.9821804511278196
---------------------------------------------
classification report:
               precision    recall  f1-score   support

           0       0.95      0.97      0.96      6066
           1       0.99      0.99      0.99     20534

    accuracy                           0.98     26600
   macro avg       0.97      0.98      0.97     26600
weighted avg       0.98      0.98      0.98     26600
```

In [79]:
```
y_pred_test1= dt_clf.predict(x_test)

cnf_matrix= confusion_matrix(y_test,y_pred_test1)
print("confusion matrix:\n",cnf_matrix)
```

```
print("-"*45)

accuracy=accuracy_score(y_test,y_pred_test1)
print("Accuracy:",accuracy)
print("-"*45)

clf_report=classification_report(y_test,y_pred_test1)
print("classification report:\n",clf_report)
```

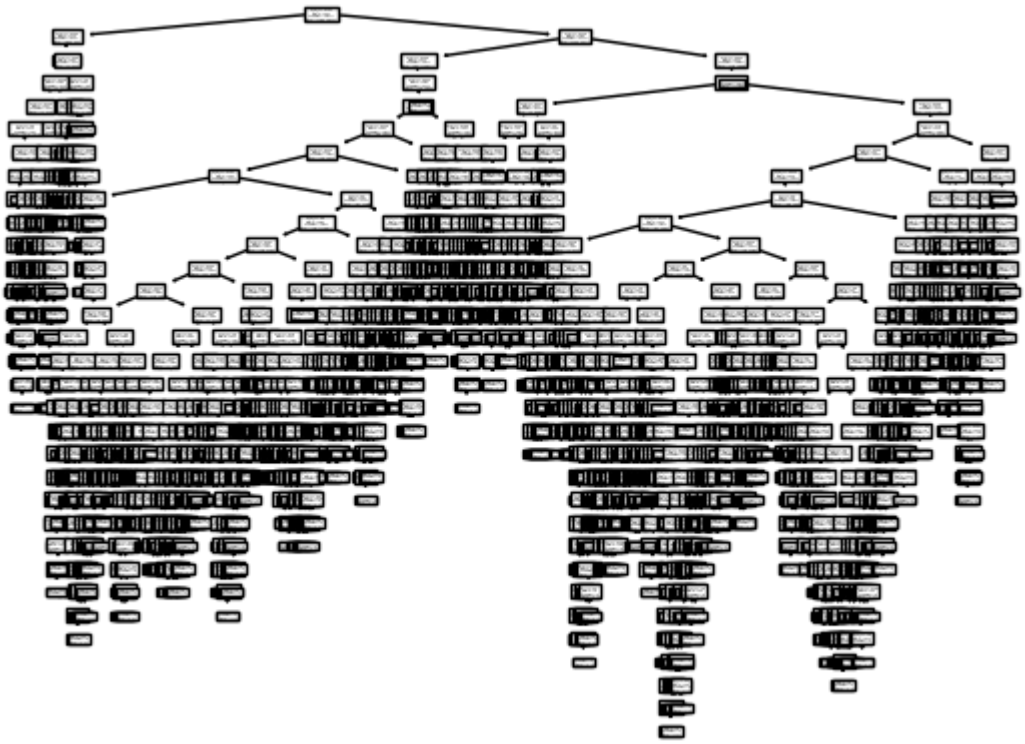
```
confusion matrix:
 [[2307  284]
 [ 336 8473]]
---------------------------------------------
Accuracy: 0.9456140350877194
---------------------------------------------
classification report:
               precision    recall  f1-score   support

           0       0.87      0.89      0.88      2591
           1       0.97      0.96      0.96      8809

    accuracy                           0.95     11400
   macro avg       0.92      0.93      0.92     11400
weighted avg       0.95      0.95      0.95     11400
```

In [155…

```
plot_tree(dt_clf)
print()
```



RANDOM FOREST CLASSIFIER

In [81]:

```
random_classify = RandomForestClassifier(n_estimators=100,criterion="entropy")
random_classify.fit(x_train,y_train)
```

Out[81]:
```
RandomForestClassifier
RandomForestClassifier(criterion='entropy')
```

In [82]:
```
y_pred_train2=random_classify.predict(x_train)

cnf_matrix= confusion_matrix(y_train,y_pred_train2)
print("confusion matrix:\n",cnf_matrix)
print("*"*45)

accuracy=accuracy_score(y_train,y_pred_train2)
print("Accuracy:",accuracy)
print("*"*45)

clf_report=classification_report(y_train,y_pred_train2)
print("classification report:\n",clf_report)
```

```
confusion matrix:
 [[ 5660   406]
 [   68 20466]]
*********************************************
Accuracy: 0.9821804511278196
*********************************************
classification report:
               precision    recall  f1-score   support

           0       0.99      0.93      0.96      6066
           1       0.98      1.00      0.99     20534

    accuracy                           0.98     26600
   macro avg       0.98      0.96      0.97     26600
weighted avg       0.98      0.98      0.98     26600
```

In [83]:
```
y_pred_test2= random_classify.predict(x_test)

cnf_matrix= confusion_matrix(y_test,y_pred_test2)
print("confusion matrix:\n",cnf_matrix)
print("-"*45)

accuracy=accuracy_score(y_test,y_pred_test2)
print("Accuracy:",accuracy)
print("-"*45)

clf_report=classification_report(y_test,y_pred_test2)
print("classification report:\n",clf_report)
```

```
confusion matrix:
 [[2275  316]
 [ 165 8644]]
--------------------------------------------
Accuracy: 0.9578070175438597
--------------------------------------------
classification report:
              precision    recall  f1-score   support

           0       0.93      0.88      0.90      2591
           1       0.96      0.98      0.97      8809

    accuracy                           0.96     11400
   macro avg       0.95      0.93      0.94     11400
weighted avg       0.96      0.96      0.96     11400
```