In [6]:

```
#pimporting libraries and data set
import pandas as pd
import numpy as np
df = pd.read_excel("C:\\Users\\maagalu\\Desktop\\arima.xlsx")
import warnings
warnings.filterwarnings('ignore')
```

In [8]:

```
# Checking for the data type
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 97936 entries, 0 to 97935
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Date           97936 non-null  object 
 1   Country        97936 non-null  object 
 2   Restaurant ID  97936 non-null  int64  
 3   Cuisine        97936 non-null  object 
 4   Items          97936 non-null  int64  
 5   Cost           97936 non-null  float64
 6   Delivery Time  97936 non-null  float64
 7   Month          97936 non-null  object 
 8   Week           97936 non-null  int64  
dtypes: float64(2), int64(3), object(4)
memory usage: 6.7+ MB
```

In [5]:

```
df.head(5)
```

Out[5]:

	Date	Country	Restaurant ID	Cuisine	Items	Cost	Delivery Time	Month	Week
0	1-1-2020	Portugal	8007	Sushi	1	46.00	18.2	January	1
1	1-1-2020	Portugal	7456	Fast food	1	23.50	11.5	January	1
2	1-1-2020	Portugal	9825	Sushi	3	10.91	10.7	January	1
3	1-1-2020	Portugal	6551	Chinese	1	6.45	7.4	January	1
4	1-1-2020	Ghana	9729	Fast food	2	10.92	22.1	January	1

In [9]:

```
#Converting Date column data type from Object to date data format
df['Date']=pd.to_datetime(df['Date'])
```

In [10]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 97936 entries, 0 to 97935
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype         
---  ------         --------------  -----         
 0   Date           97936 non-null  datetime64[ns]
 1   Country        97936 non-null  object        
 2   Restaurant ID  97936 non-null  int64         
 3   Cuisine        97936 non-null  object        
 4   Items          97936 non-null  int64         
 5   Cost           97936 non-null  float64       
 6   Delivery Time  97936 non-null  float64       
 7   Month          97936 non-null  object        
 8   Week           97936 non-null  int64         
dtypes: datetime64[ns](1), float64(2), int64(3), object(3)
memory usage: 6.7+ MB
```

In [23]:

```
#Grouping number of Orders placed by countries based on date
s = df.groupby(['Date'])['Items'].count()
```

In [24]:

```
s1= s.to_frame().rename(columns={'Items':'Orders'}).reset_index()
```

In [25]:

```
s1
```

Out[25]:

	Date	Orders
0	2020-01-01	1006
1	2020-01-02	1230
2	2020-01-03	1186
3	2020-01-04	953
4	2020-01-05	986
5	2020-01-06	1007
6	2020-01-07	1116
7	2020-01-08	1155
8	2020-01-09	1625
9	2020-01-10	1597
10	2020-01-11	1105
11	2020-01-12	1128
12	2020-01-13	1179
13	2020-01-14	1334
14	2020-01-15	1565
15	2020-01-16	1520
16	2020-01-17	1495
17	2020-01-18	1237
18	2020-01-19	1341
19	2020-01-20	1468
20	2020-01-21	1741
21	2020-01-22	1838
22	2020-01-23	1871
23	2020-01-24	1663
24	2020-01-25	1345
25	2020-01-26	1376
26	2020-01-27	1522
27	2020-01-28	1677
28	2020-01-29	2029
29	2020-01-30	2052
30	2020-01-31	1891
31	2020-02-01	1551
32	2020-02-02	1630
33	2020-02-03	1616
34	2020-02-04	1733
35	2020-02-05	2043
36	2020-02-06	2084
37	2020-02-07	1886
38	2020-02-08	1510
39	2020-02-09	1705
40	2020-02-10	1627
41	2020-02-11	1862
42	2020-02-12	2087
43	2020-02-13	2191
44	2020-02-14	2328
45	2020-02-15	1546
46	2020-02-16	1694
47	2020-02-17	1722
48	2020-02-18	1775
49	2020-02-19	2189
50	2020-02-20	2268
51	2020-02-21	1982

	Date	Orders
52	2020-02-22	1616
53	2020-02-23	1734
54	2020-02-24	1831
55	2020-02-25	2089
56	2020-02-26	2662
57	2020-02-27	2501
58	2020-02-28	2236

In [15]:

```
import matplotlib.pyplot as plt
import statsmodels.api as sms

%matplotlib inline
```

In [16]:

```
s1.set_index('Date',inplace=True)
```

In [18]:

```
s1.head()
```

Out[18]:

	Orders
Date	
2020-01-01	1006
2020-01-02	1230
2020-01-03	1186
2020-01-04	953
2020-01-05	986

ARIMA(Autoregressive Integrated Moving Averages) and Seasonal ARIMA

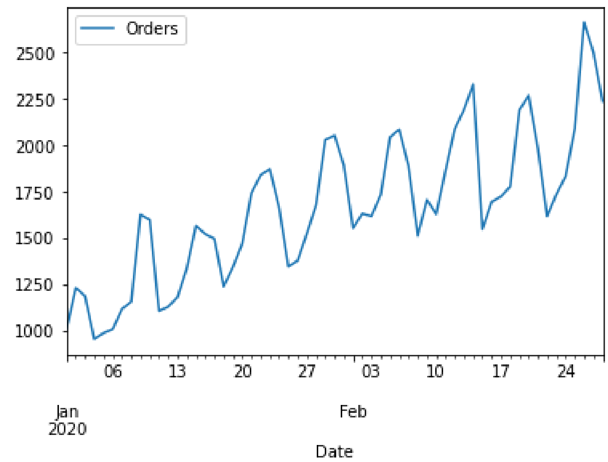
1) Visualize the time series data 2) If the data is seasonal then make the time series data stationary 3) Plot the correlation and autocorrelation charts 4) Construct the ARIMA model or Seasonal ARIMA based on the data 5) Use the model to make the predictions

In [19]:

```
s1.plot()
```

Out[19]:

<AxesSubplot:xlabel='Date'>



In [15]:

```
from statsmodels.tsa.stattools import adfuller
```

In [16]:

```
#Checking for seasonality by performing dickey fuller test
def adf_test(series):
    result=adfuller(series)
    print('ADF statistics: {}'.format(result[0]))
    print('p-value: {}'.format(result[1]))
    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis, rejecting the null hypothesis and data is stationary")
    else:
        print("Weak evidence against null hypothesis, indicating data is non stationary")
```

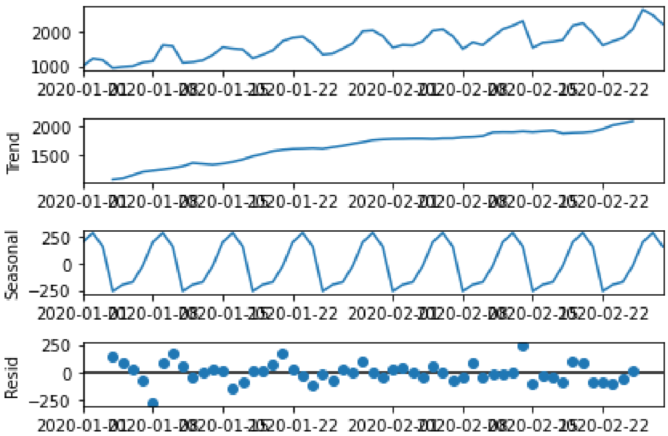
In [17]:

```
adf_test(s1['Orders'])
```

ADF statistics: -1.6160321018396893
p-value: 0.47481011817588453
Weak evidence against null hypothesis, indicating data is non stationary
As per the dickey fuller test I found out that the data is seasonal and the next steps involved in prediction are 1) Finding p,d,q values. Where 'p' can be determined by ploting Partial autocorrelation, which indicates AR model lags 'd' can be determined by how many times we diferentiated data to make it stationary 'q' can be determined by ploting autocorrelation, which indicates moving average lags 2) Based on the p,d,q values we can do the predictions

In [19]:

```
decomposition = sms.tsa.seasonal_decompose(s1, model='additive')
fig = decomposition.plot()
plt.show()
```



In [20]:

```
len(s1)
```

Out[20]:

59

In [21]:

```
train=s1[:45]
test=s1[45:]
```

In [22]:

```
#differentiating the data to make it stationary for forecasting
s1['Orders first difference'] = s1['Orders'] - s1['Orders'].shift(1)
```

In [23]:

```
s1.head()
```

Out[23]:

	Orders	Orders first difference
Date		
2020-01-01	1006	NaN
2020-01-02	1230	224.0
2020-01-03	1186	-44.0
2020-01-04	953	-233.0
2020-01-05	986	33.0

In [25]:

```
adf_test(s1['Orders first difference'].dropna())
```

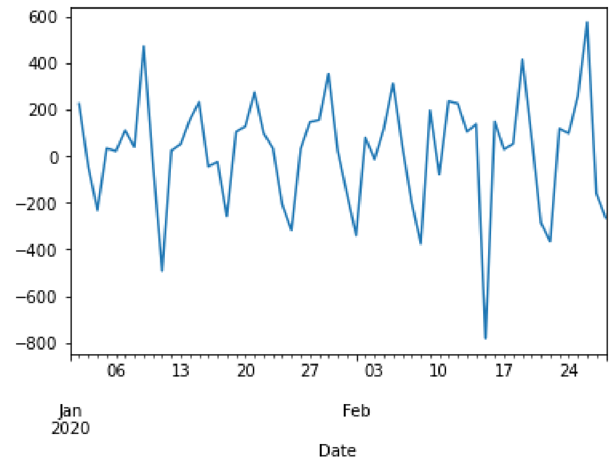
ADF statistics: -3.9379740884490B4
p-value: 0.0017724889713223088
strong evidence against the null hypothesis, rejecting the null hypothesis and data is stationary

In [26]:

```
s1['Orders first difference'].plot()
```

Out[26]:

<AxesSubplot:xlabel='Date'>

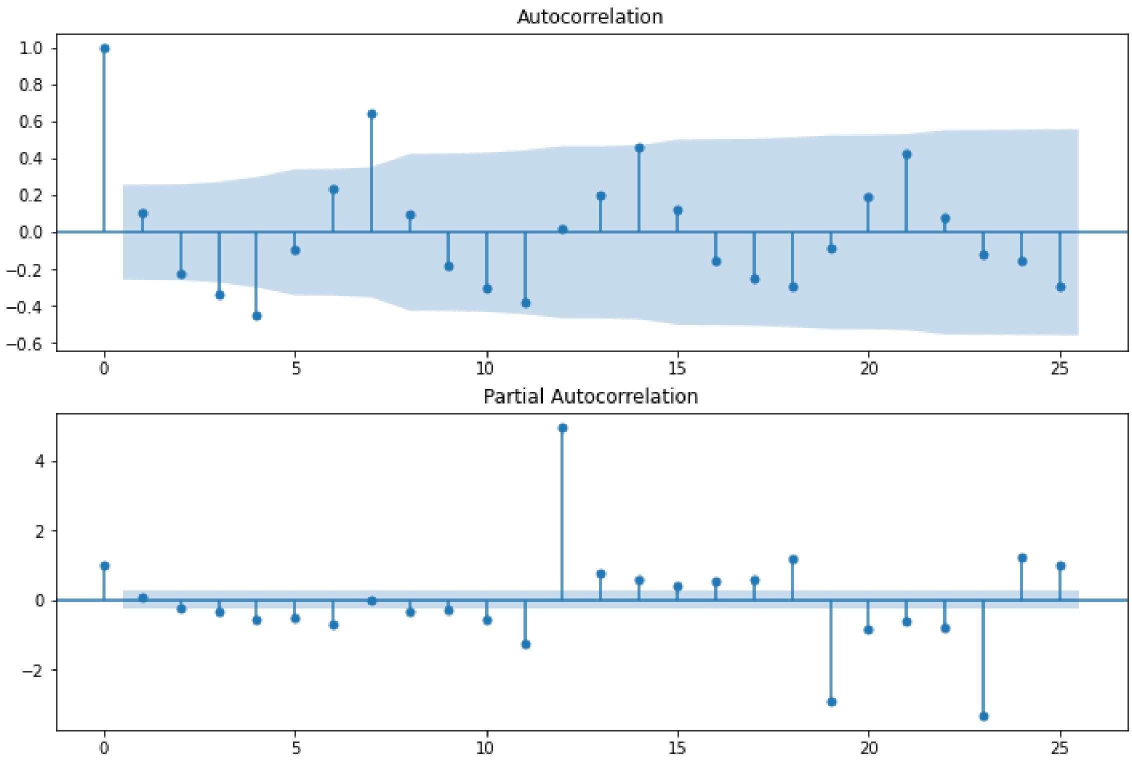


In [27]:

```
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
```

In [31]:

```
fig = plt.figure(figsize=(12,8))
ax1 = fig.add_subplot(211)
fig = sms.graphics.tsa.plot_acf(s1['Orders first difference'].dropna(),lags=25,ax=ax1)
ax2 = fig.add_subplot(212)
fig = sms.graphics.tsa.plot_pacf(s1['Orders first difference'].dropna(),lags=25,ax=ax2)
```



In [32]:

```
#p=1,d=1,q=1
from statsmodels.tsa.arima_model import ARIMA
```

In [34]:

```
model=ARIMA(s1['Orders'],order=(1,1,1))
model_fit=model.fit()
```

```
C:\Users\maagalu\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:524: ValueWarning: No frequency information was provided, so inferred f
requency D will be used.
  warnings.warn('No frequency information was'
C:\Users\maagalu\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:524: ValueWarning: No frequency information was provided, so inferred f
requency D will be used.
  warnings.warn('No frequency information was'
```

In [35]:

```
model_fit.summary()
```

Out[35]:

ARIMA Model Results

Dep. Variable:	D.Orders	**No. Observations:**	58
Model:	ARIMA(1, 1, 1)	**Log Likelihood**	-392.954
Method:	css-mle	**S.D. of innovations**	206.541
Date:	Mon, 09 May 2022	**AIC**	793.908
Time:	13:22:47	**BIC**	802.149
Sample:	01-02-2020	**HQIC**	797.118
	- 02-28-2020		

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
const	18.7577	3.081	6.088	0.000	12.719	24.796
ar.L1.D.Orders	0.5135	0.116	4.440	0.000	0.287	0.740
ma.L1.D.Orders	-0.9999	0.046	-21.619	0.000	-1.091	-0.909

Roots

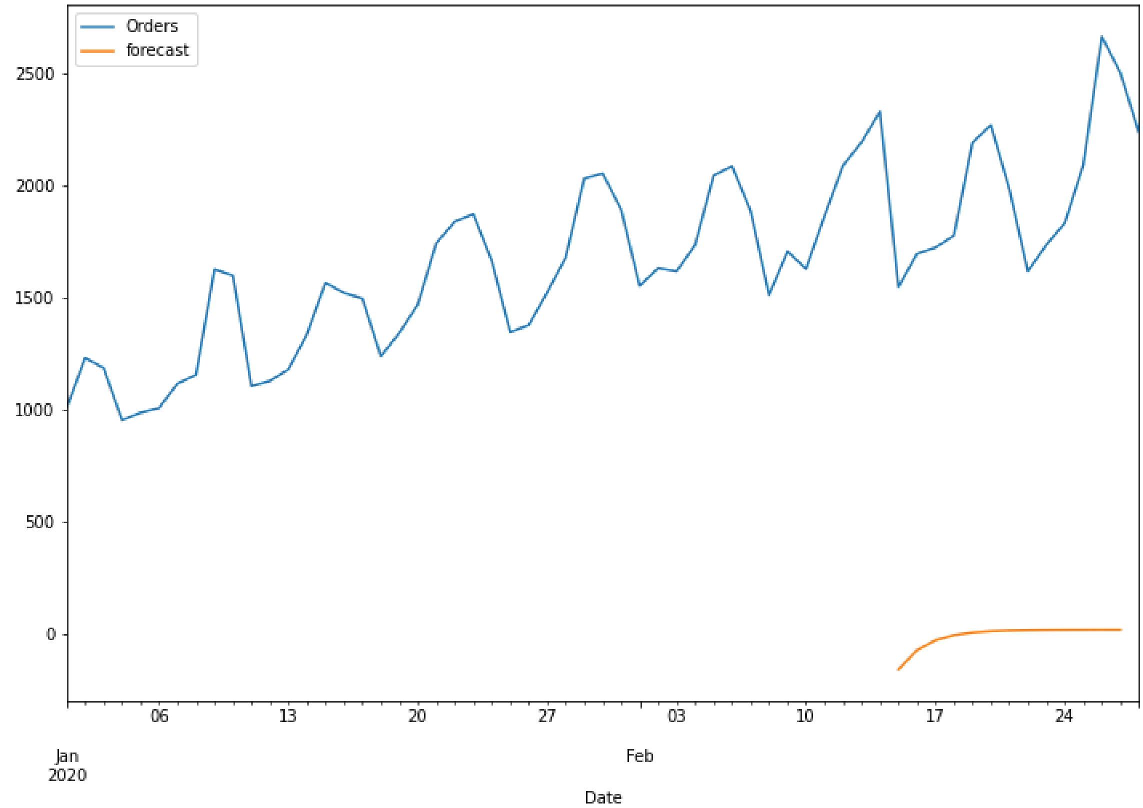
	Real	**Imaginary**	**Modulus**	**Frequency**
AR.1	1.9475	+0.0000j	1.9475	0.0000
MA.1	1.0001	+0.0000j	1.0001	0.0000

In [36]:

```
s1['forecast']=model_fit.predict(start=45,end=57,dynamic=True)
s1[['Orders','forecast']].plot(figsize=(12,8))
```

Out[36]:

<AxesSubplot:xlabel='Date'>



In [ ]:

```
# After ARIMA model evaluation, we can observe that the model has not given the proper prediction due to seasonality!
# That is why I am using SARIMA model for prediction by taking 7 days period as seasonal
```

In [38]:

```
model=sms.tsa.statespace.SARIMAX(s1['Orders'],order=(1, 1, 1),seasonal_order=(1,1,1,7))
results=model.fit()
```

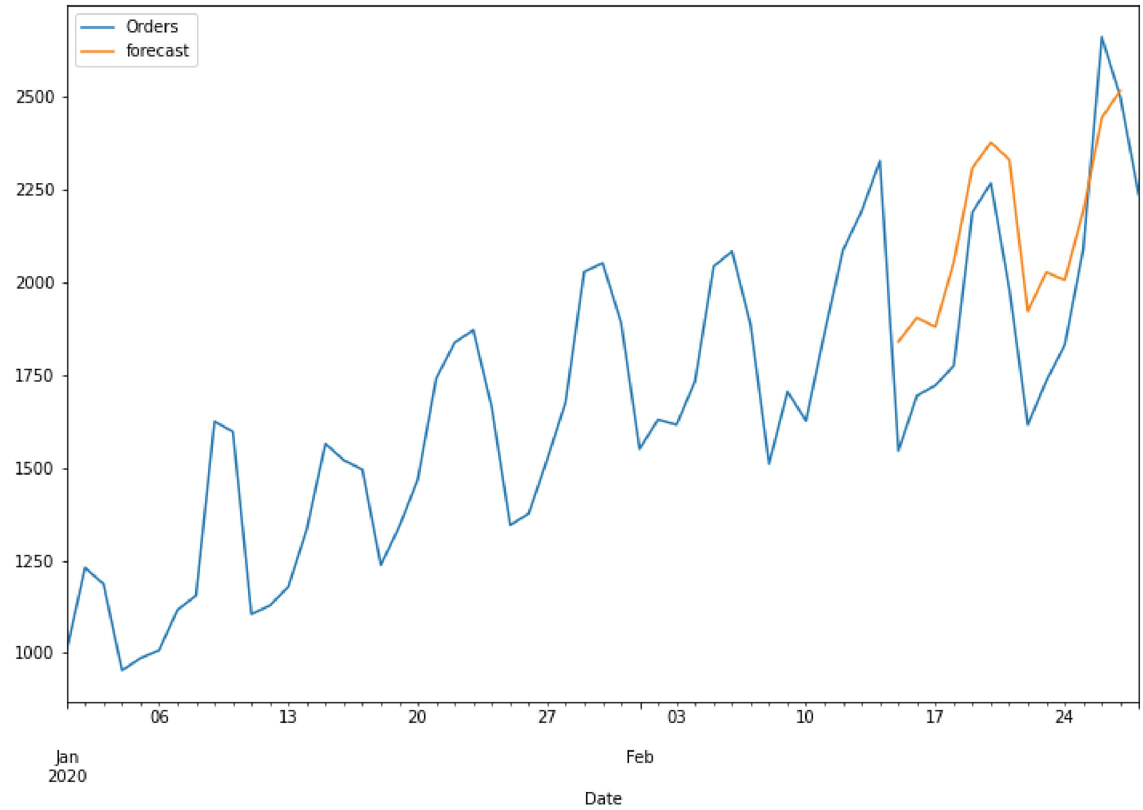
```
C:\Users\maagalu\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:524: ValueWarning: No frequency information was provided, so inferred f
requency D will be used.
  warnings.warn('No frequency information was'
C:\Users\maagalu\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:524: ValueWarning: No frequency information was provided, so inferred f
requency D will be used.
  warnings.warn('No frequency information was'
```

In [39]:

```
#Checking whether our predicted values are in line with the true values by spliting data into Train and Test data
s1['forecast']=results.predict(start=45,end=57,dynamic=True)
s1[['Orders','forecast']].plot(figsize=(12,8))
```

Out[39]:

<AxesSubplot:xlabel='Date'>



In [59]:

```
# From the above graph,It is confirmed that SARIMA model prediction is inline with the true values.
from pandas.tseries.offsets import DateOffset
future_dates=[s1.index[-1]+DateOffset(days=x)for x in range(0,32)]
```

In [60]:

```
future_datest_df=pd.DataFrame(index=future_dates[1:],columns=s1.columns)
```

In [61]:

```
future_datest_df.tail()
```

Out[61]:

	Orders	Orders first difference	forecast
2020-03-26	NaN	NaN	NaN
2020-03-27	NaN	NaN	NaN
2020-03-28	NaN	NaN	NaN
2020-03-29	NaN	NaN	NaN
2020-03-30	NaN	NaN	NaN

In [62]:

```
future_df=pd.concat([s1,future_datest_df])
```

In [65]:

```
#Continuing the SARIMA model test for the prediction of Orders for the month of March
future_df['forecast']=results.predict(start=58,end=90,dynamic=True)
future_df[['Orders','forecast']].plot(figsize=(12,8))
```

Out[65]:

<AxesSubplot:>