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

In [2]:
```
df1 = pd.read_csv('Historical_Product_Demand.csv', parse_dates=['Date'])
df1.head()
```

Out[2]:

	Product_Code	Warehouse	Product_Category	Date	Order_Demand
0	Product_0993	Whse_J	Category_028	2012-07-27	100
1	Product_0979	Whse_J	Category_028	2012-01-19	500
2	Product_0979	Whse_J	Category_028	2012-02-03	500
3	Product_0979	Whse_J	Category_028	2012-02-09	500
4	Product_0979	Whse_J	Category_028	2012-03-02	500

In [3]:
```
df1.isnull().sum()/len(df1)*100
```

Out[3]:
```
Product_Code        0.000000
Warehouse           0.000000
Product_Category    0.000000
Date                1.071836
Order_Demand        0.000000
dtype: float64
```

In [4]:
```
df1.shape
```

Out[4]:
```
(1048575, 5)
```

In [5]:
```
df1.dropna(axis = 0, inplace = True)
```

In [6]:
```
df1.isnull().sum()/len(df1)*100
```

Out[6]:
```
Product_Code        0.0
Warehouse           0.0
Product_Category    0.0
Date                0.0
Order_Demand        0.0
dtype: float64
```

In [7]:
```
df1.dtypes
```

Out[7]:
```
Product_Code                object
Warehouse                   object
Product_Category            object
Date                datetime64[ns]
Order_Demand                object
dtype: object
```

In [8]:
```
df1.Product_Code.unique()
```

Out[8]:
```
array(['Product_0993', 'Product_0979', 'Product_1159', ...,
       'Product_0237', 'Product_0644', 'Product_0853'], dtype=object)
```

In [9]:
```
df1['Date'].min(), df1['Date'].max()
#There is 6 years of data
```

Out[9]:
```
(Timestamp('2011-01-08 00:00:00'), Timestamp('2017-01-09 00:00:00'))
```

In [10]:
```
#Lets start with 2012 and cap it 2016 december. Since the dates before 2012 have a lot of missing values - inspected and
df1 = df1[(df1['Date']>='2012-01-01') & (df1['Date']<='2014-12-31')]
```

In [11]:
```
df1
```

Out[11]:

	Product_Code	Warehouse	Product_Category	Date	Order_Demand
0	Product_0993	Whse_J	Category_028	2012-07-27	100
1	Product_0979	Whse_J	Category_028	2012-01-19	500
2	Product_0979	Whse_J	Category_028	2012-02-03	500
3	Product_0979	Whse_J	Category_028	2012-02-09	500
4	Product_0979	Whse_J	Category_028	2012-03-02	500
...	...	...	...	...	...
851417	Product_1723	Whse_S	Category_003	2014-12-28	1180
851424	Product_1723	Whse_S	Category_003	2014-12-15	820
852244	Product_1102	Whse_S	Category_004	2014-12-29	800
855387	Product_0158	Whse_S	Category_021	2014-12-22	600
856147	Product_2072	Whse_S	Category_009	2014-12-18	50

638337 rows × 5 columns

In [12]:
```
df1.shape
```

Out[12]:
```
(638337, 5)
```

In [13]:
```
df1.set_index('Date', inplace = True)
df1
```

Out[13]:

	Product_Code	Warehouse	Product_Category	Order_Demand
Date				
2012-07-27	Product_0993	Whse_J	Category_028	100
2012-01-19	Product_0979	Whse_J	Category_028	500
2012-02-03	Product_0979	Whse_J	Category_028	500
2012-02-09	Product_0979	Whse_J	Category_028	500
2012-03-02	Product_0979	Whse_J	Category_028	500
...	...	...	...	...
2014-12-28	Product_1723	Whse_S	Category_003	1180
2014-12-15	Product_1723	Whse_S	Category_003	820
2014-12-29	Product_1102	Whse_S	Category_004	800
2014-12-22	Product_0158	Whse_S	Category_021	600
2014-12-18	Product_2072	Whse_S	Category_009	50

638337 rows × 4 columns

In [14]:
```
df = df1.drop(['Warehouse', 'Product_Code', 'Product_Category'],axis = 1, inplace = True)
```

In [15]:
```
df1.head()
```

Out[15]:

	Order_Demand
Date	
2012-07-27	100
2012-01-19	500
2012-02-03	500
2012-02-09	500
2012-03-02	500

In [16]:
```
df1.describe()
```

Out[16]:

	Order_Demand
count	638337
unique	2982
top	1000
freq	69449

In [17]:
```
df1.dtypes
```

Out[17]:
```
Order_Demand    object
dtype: object
```

In [18]:
```
df1.sort_values('Date')[10:20]
```

Out[18]:

	Order_Demand
Date	
2012-01-02	3
2012-01-02	100
2012-01-02	5000
2012-01-02	58000
2012-01-02	500
2012-01-02	31250
2012-01-02	38000
2012-01-02	1000
2012-01-02	1000
2012-01-02	1000

In [19]:
```
df1['Order_Demand'] = df1['Order_Demand'].str.replace('(', '')
df1['Order_Demand'] = df1['Order_Demand'].str.replace(')', '')
```

```
C:\Users\Vaishu\AppData\Local\Temp/ipykernel_27968/3317173805.py:1: FutureWarning: The default value of regex will chan
ge from True to False in a future version. In addition, single character regular expressions will *not* be treated as l
iteral strings when regex=True.
  df1['Order_Demand'] = df1['Order_Demand'].str.replace('(', '')
C:\Users\Vaishu\AppData\Local\Temp/ipykernel_27968/3317173805.py:2: FutureWarning: The default value of regex will chan
ge from True to False in a future version. In addition, single character regular expressions will *not* be treated as l
iteral strings when regex=True.
  df1['Order_Demand'] = df1['Order_Demand'].str.replace(')', '')
```

In [20]:
```
df1['Order_Demand'] = df1['Order_Demand'].astype("int64")
```

In [21]:
```
df1.dtypes
```

Out[21]:
```
Order_Demand    int64
dtype: object
```

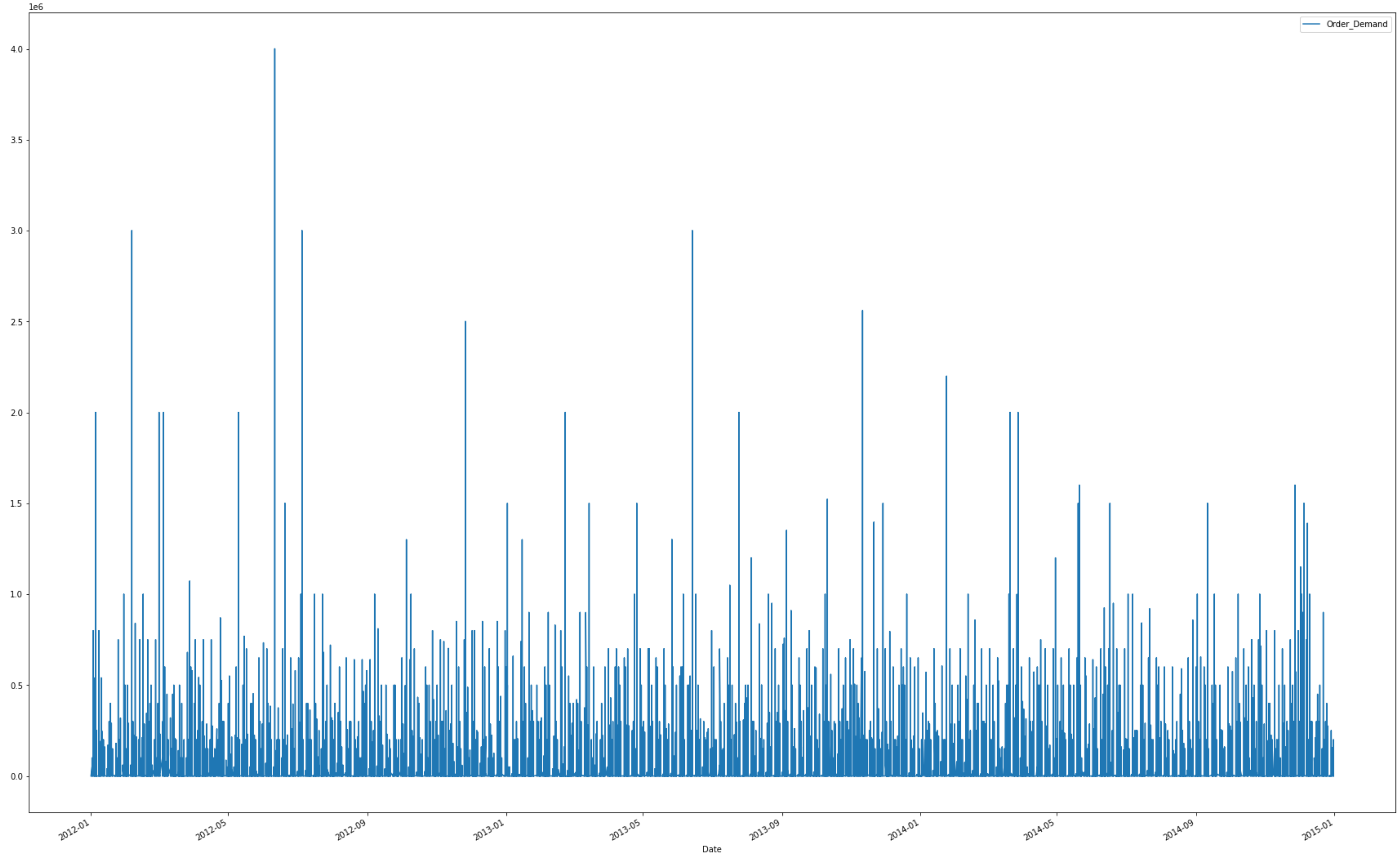
In [22]:
```
df1.describe()
```

Out[22]:

	Order_Demand
count	6.383370e+05
mean	4.753800e+03
std	2.817886e+04
min	0.000000e+00
25%	2.000000e+01
50%	3.000000e+02
75%	2.000000e+03
max	4.000000e+06

In [23]:
```
plt.rcParams["figure.figsize"] = (30,20)
df1.plot()
```

Out[23]: <AxesSubplot:xlabel='Date'>



In [24]:
```
#Testing for Stationarity

from statsmodels.tsa.stattools import adfuller
```

In [25]:
```
test_result = adfuller(df1['Order_Demand'])
```

In [26]:
```
print(test_result)
```

```
(-56.86465664577732, 0.0, 108, 638228, {'1%': -3.430360246066141, '5%': -2.8615445286427956, '10%': -2.56677241043075
7}, 14623495.040394)
```

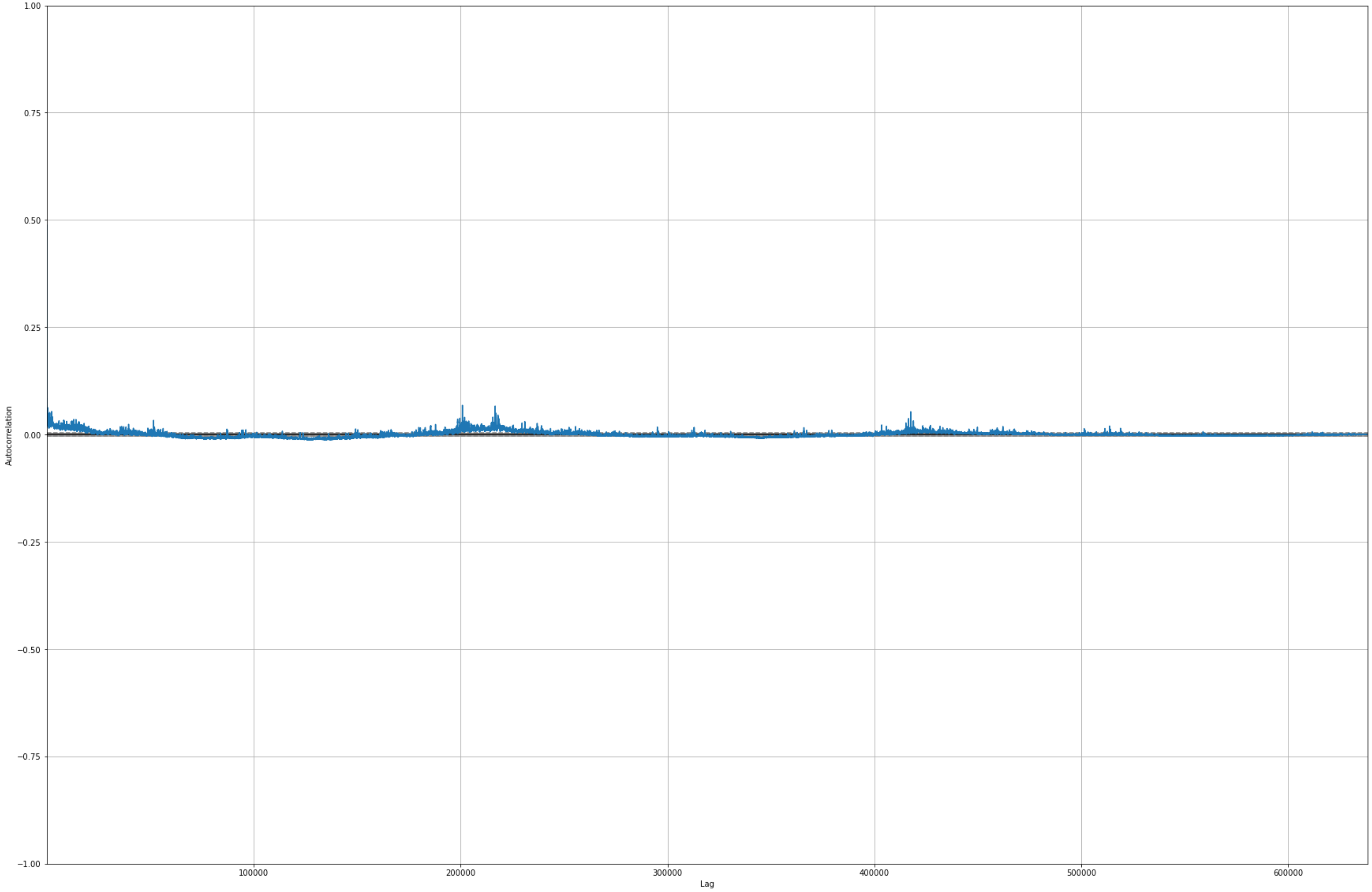
In [30]:
```
"""# Ho = Its not stationary
# Ha = It is stationary"""

print('ADF Statistic: %f' % test_result[0])
print('p-value: %f' % test_result[1])
print('Critical Values:')
for key, value in test_result[4].items():
    print('\t%s: %.3f' % (key, value))

if test_result[0] < test_result[4]["5%"]:
    print ("Reject Ho - Time Series is Stationary")
else:
    print ("Failed to Reject Ho - Time Series is Non-Stationary")
```

```
ADF Statistic: -56.864657
p-value: 0.000000
Critical Values:
	1%: -3.430
	5%: -2.862
	10%: -2.567
Reject Ho - Time Series is Stationary
```

In [28]:
```
# As Dataset is Stationary as p-value is less than 0.5, we do not need to perform furter steps like differencing
# If data is seasonal, then we have to perform seasonal difference i.e. shift the data by shift(12)
# If data is not seasonal, the shift(1) is enough.
```

AUTO REGRESSIVE MODEL

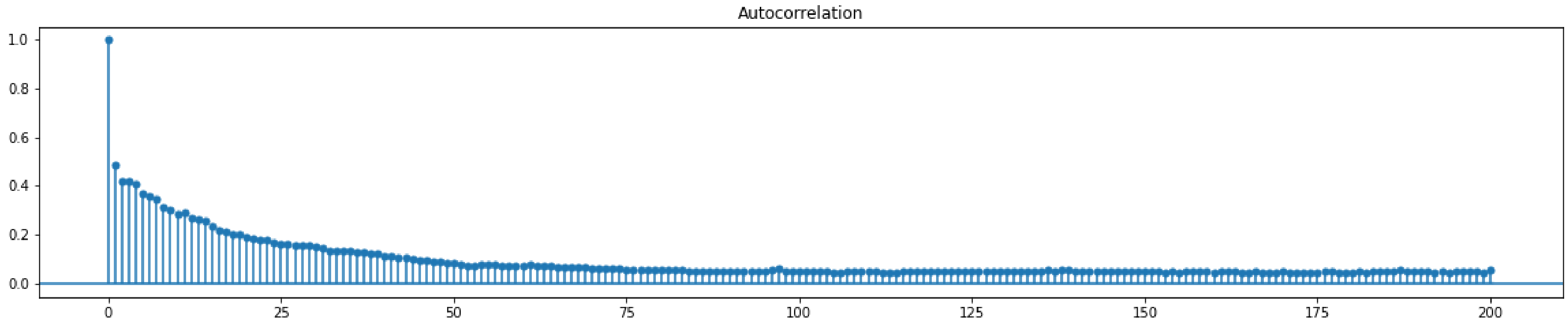
In [25]:
```
from pandas.plotting import autocorrelation_plot
autocorrelation_plot(df1['Order_Demand'])
plt.show()
```



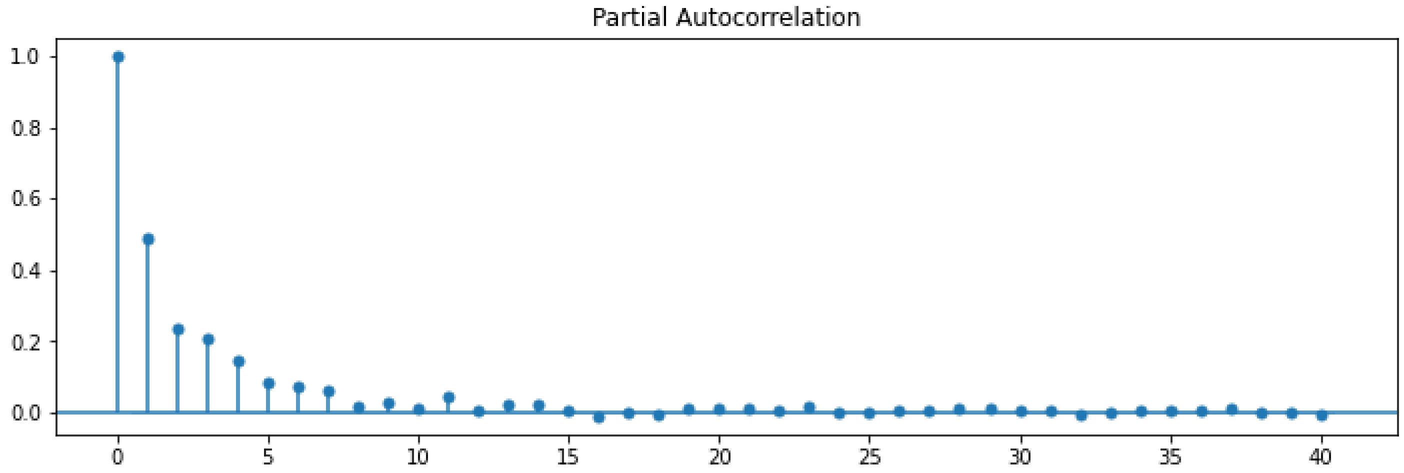
In [ ]:
```
#pd.plotting.autocorrelation_plot(df1['Order_Demand'])
```

In [29]:
```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
import statsmodels.api as sm
```

In [34]:
```
#For Moving Average Model
fig = plt.figure(figsize = (20,8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(df1['Order_Demand'], lags = 200 , ax = ax1)
```



In [33]:
```
#For Auto Regression Model
fig = plt.figure(figsize = (12,8))
ax1 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(df1['Order_Demand'], lags = 40 , ax = ax1)
```



In [ ]:
```
# p, d, q are important values to be considered in TSA model
# p = AR model lags
# d = Differencing
# q = MA model lags
# ARIMA model is used when data is not seasonal
# SARIMA is used for seasonal dataset.
```

In [ ]:
```
# ARIMA Model

p = 8 #From pack graph of AR model(where the value reached to zero)
d = 0 #no shifts are done as data is not seasonal.
q = 0
```

In [40]:
```
from statsmodels. tsa. arima_model import ARIMA
model = ARIMA(df1['Order_Demand'], order = (0,0,1))
model_fit = model.fit()
model_fit.summary()
```

```
C:\Users\Vaishu\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:581: ValueWarning: A date index has been provided, but it has no associated frequency information and so will be ignored when e.g. forecasting.
  warnings.warn('A date index has been provided, but it has no'
C:\Users\Vaishu\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:585: ValueWarning: A date index has been provided, but it is not monotonic and so will be ignored when e.g. forecasting.
  warnings.warn('A date index has been provided, but it is not'
```

Out[40]:

ARMA Model Results

Dep. Variable:	Order_Demand	**No. Observations:**	638337
Model:	ARMA(0, 1)	**Log Likelihood**	-7389402.510
Method:	css-mle	**S.D. of innovations**	25772.991
Date:	Fri, 15 Jul 2022	**AIC**	14778811.020
Time:	16:52:26	**BIC**	14778845.120
Sample:	0	**HQIC**	14778820.577

	coef	std err	z	P>|z|	[0.025	0.975]
const	4753.7998	43.752	108.653	0.000	4668.047	4839.552
ma.L1.Order_Demand	0.3563	0.001	360.167	0.000	0.354	0.358

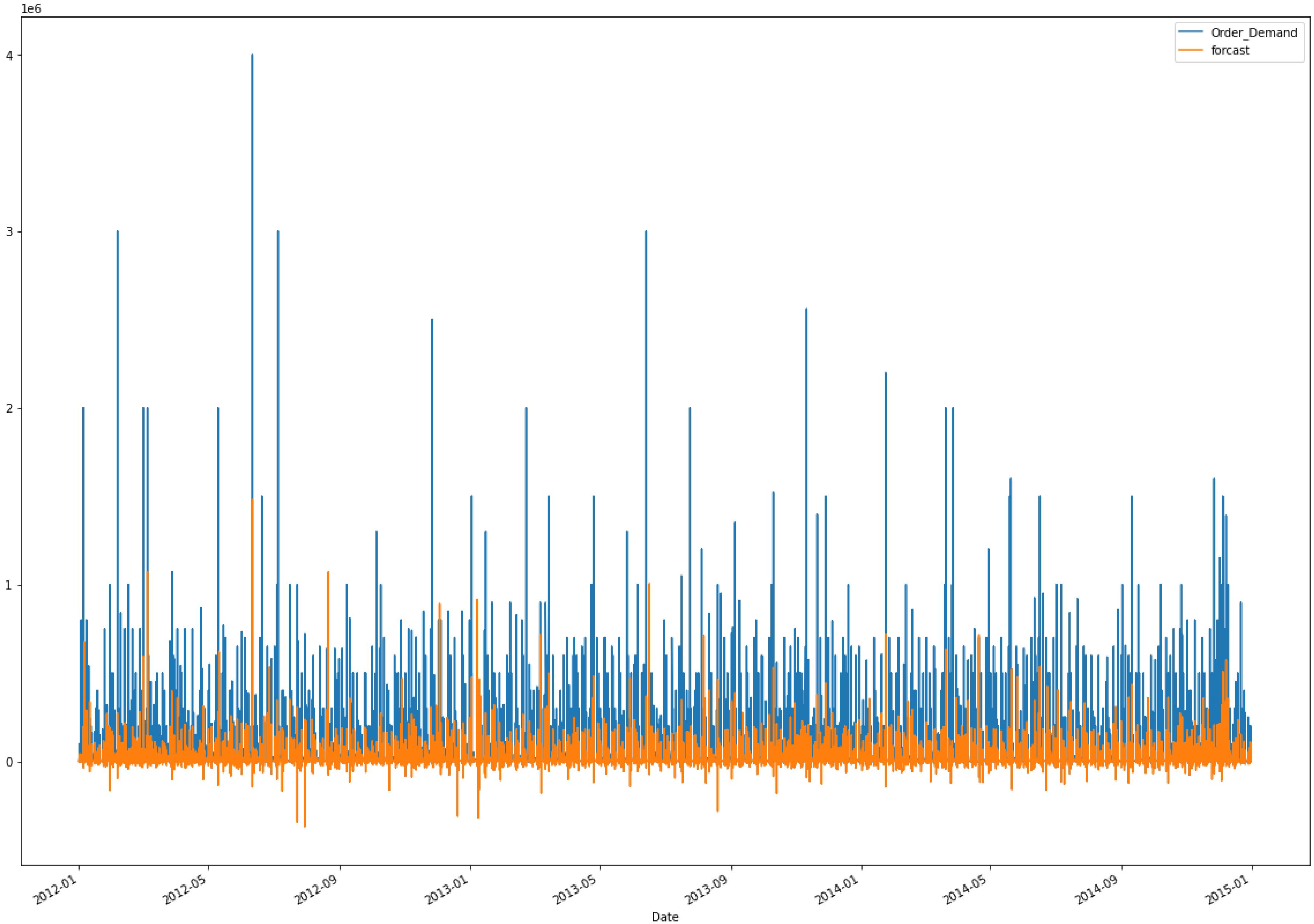
Roots

	Real	Imaginary	Modulus	Frequency
MA.1	-2.8065	+0.0000j	2.8065	0.5000

In [48]:
```
df1['forcast'] = model_fit.predict()
```

In [57]:
```
df1[['Order_Demand', 'forcast']].plot(figsize=(20,15))
```

Out[57]: <AxesSubplot:xlabel='Date'>



In [44]:
```
from statsmodels.tsa.statespace.sarimax import SARIMAX
```

In [45]:
```
model_s = sm.tsa.statespace.SARIMAX(df1['Order_Demand'], order=(0,0,1), seasonal_order=(0,0,1,12))
result_s = model_s.fit()
```

```
C:\Users\Vaishu\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:581: ValueWarning: A date index has been provided, but it has no associated frequency information and so will be ignored when e.g. forecasting.
  warnings.warn('A date index has been provided, but it has no'
C:\Users\Vaishu\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:585: ValueWarning: A date index has been provided, but it is not monotonic and so will be ignored when e.g. forecasting.
  warnings.warn('A date index has been provided, but it is not'
C:\Users\Vaishu\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:581: ValueWarning: A date index has been provided, but it has no associated frequency information and so will be ignored when e.g. forecasting.
  warnings.warn('A date index has been provided, but it has no'
C:\Users\Vaishu\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:585: ValueWarning: A date index has been provided, but it is not monotonic and so will be ignored when e.g. forecasting.
  warnings.warn('A date index has been provided, but it is not'
```

In [50]:
```
result_s.summary()
```

Out[50]:

SARIMAX Results

Dep. Variable:	Order_Demand	**No. Observations:**	638337
Model:	SARIMAX(0, 0, 1)x(0, 0, 1, 12)	**Log Likelihood**	-7389317.341
Date:	Fri, 15 Jul 2022	**AIC**	14778640.683
Time:	17:05:39	**BIC**	14778674.782
Sample:	0	**HQIC**	14778650.239
	- 638337		
Covariance Type:	opg		

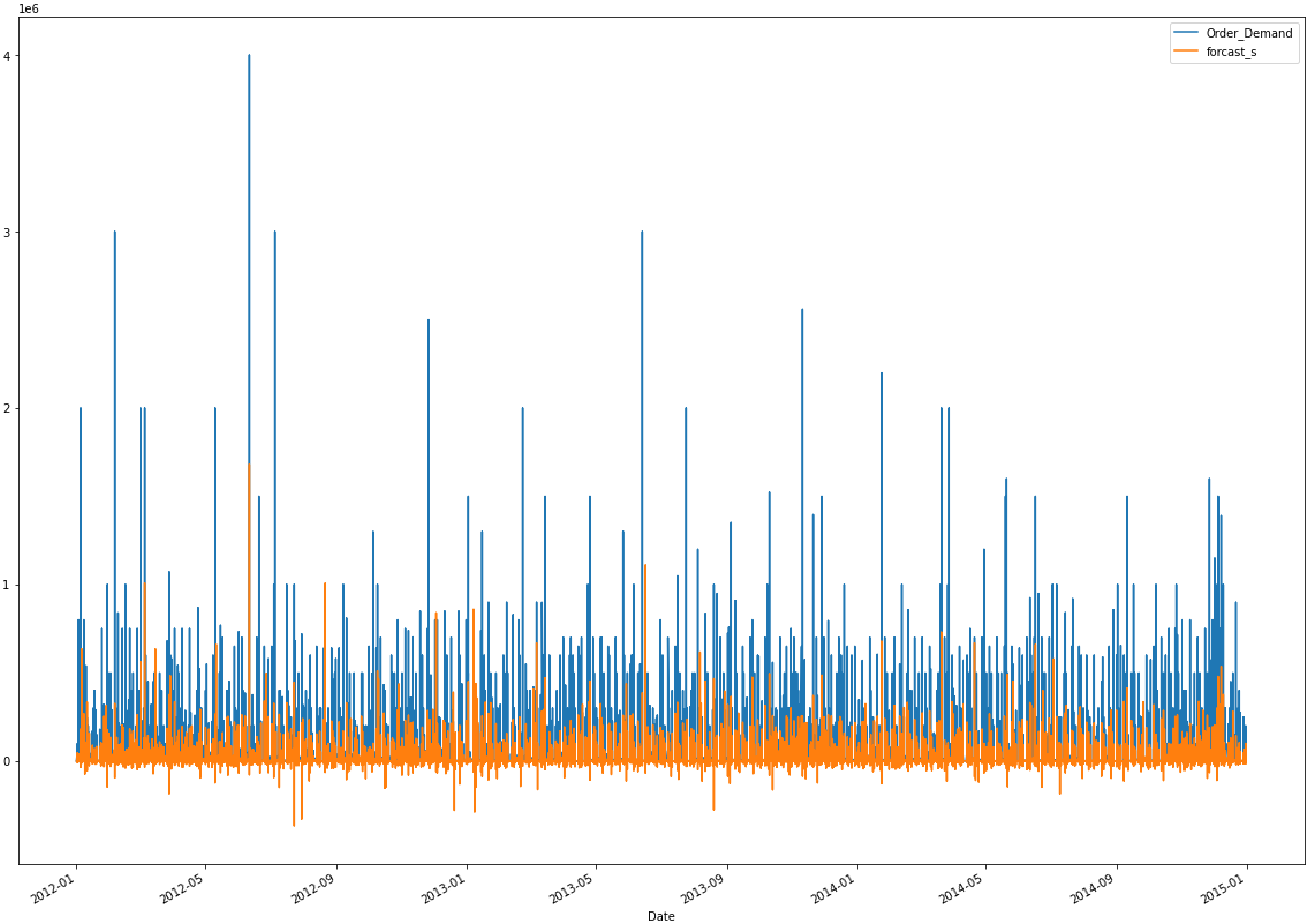
	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
ma.L1	0.3359	6.52e-05	5150.446	0.000	0.336	0.336
ma.S.L12	0.1556	0.000	1383.575	0.000	0.155	0.156
sigma2	7.546e+08	9.65e-14	7.82e+21	0.000	7.55e+08	7.55e+08

Ljung-Box (L1) (Q):	3185.33	**Jarque-Bera (JB):**	241358512378.84
Prob(Q):	0.00	**Prob(JB):**	0.00
Heteroskedasticity (H):	0.74	**Skew:**	35.63
Prob(H) (two-sided):	0.00	**Kurtosis:**	3014.55

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
[2] Covariance matrix is singular or near-singular, with condition number 1.21e+34. Standard errors may be unstable.

In [59]:
```
df1['forcast_s'] = result_s.predict()
df1[['Order_Demand', 'forcast_s']].plot(figsize=(20,15))
```

Out[59]: <AxesSubplot:xlabel='Date'>


1e6
4
3
2
1
0
Order_Demand
forcast_s
2012-01
2012-05
2012-09
2013-01
2013-05
2013-09
2014-01
2014-05
2014-09
2015-01
Date

In [ ]: