Ad Ease Website Views forecasting

About the company

Ad Ease is an ads and marketing based company helping businesses elicit maximum clicks @ minimum cost. AdEase is an ad infrastructure to help businesses promote themselves easily, effectively, and economically. The interplay of 3 AI modules - Design, Dispense, and Decipher, come together to make it this an end-to-end 3 step process digital advertising solution for all.

Business Problem

You are working in the Data Science team of Ad ease trying to understand the per page view report for different wikipedia pages for 550 days, and forecasting the number of views so that you can predict and optimize the ad placement for your clients. You are provided with the data of 145k wikipedia pages and daily view count for each of them. Your clients belong to different regions and need data on how their ads will perform on pages in different languages.

EDA + Preprocessing

In [31]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

In [32]:

```
import warnings

# Ignore all warnings
warnings.filterwarnings("ignore")
```

In [33]:

```
#Load train_1.csv
train=pd.read_csv("train_1.csv")
train
```

Out[33]:

	Page	2015-07-01	2015-07-02	2015-07-03	2015-07-04	2015-07-05	2015-07-06	2015-07-07	2015-07-08	2015-07-09	...
0	2NE1_zh.wikipedia.org_all-access_spider	18.0	11.0	5.0	13.0	14.0	9.0	9.0	22.0	26.0	...
1	2PM_zh.wikipedia.org_all-access_spider	11.0	14.0	15.0	18.0	11.0	13.0	22.0	11.0	10.0	...
2	3C_zh.wikipedia.org_all-access_spider	1.0	0.0	1.0	1.0	0.0	4.0	0.0	3.0	4.0	...
3	4minute_zh.wikipedia.org_all-access_spider	35.0	13.0	10.0	94.0	4.0	26.0	14.0	9.0	11.0	...
4	52_Hz_I_Love_You_zh.wikipedia.org_all-access_s...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
...	...	...	...	...	...	...	...	...	...	...	...
145058	Underworld_(serie_de_películas)_es.wikipedia.o...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
145059	Resident_Evil:_Capítulo_Final_es.wikipedia.org...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
145060	Enamorándome_de_Ramón_es.wikipedia.org_all-acc...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...

	Page	2015-07-01	2015-07-02	2015-07-03	2015-07-04	2015-07-05	2015-07-06	2015-07-07	2015-07-08	2015-07-09	...	2 1:
145061	Hasta_el_último_hombre_es.wikipedia.org_all-ac...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	
145062	Francisco_el_matemático_(serie_de_televisión_d...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	

145063 rows × 551 columns

In [34]:

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 145063 entries, 0 to 145062
Columns: 551 entries, Page to 2016-12-31
dtypes: float64(550), object(1)
memory usage: 609.8+ MB
```

In [35]:

```
train.describe()
```

Out[35]:

	2015-07-01	2015-07-02	2015-07-03	2015-07-04	2015-07-05	2015-07-06	2015-07-07	2015-07-08	
count	1.243230e+05	1.242470e+05	1.245190e+05	1.244090e+05	1.244040e+05	1.245800e+05	1.243990e+05	1.247690e+05	1.
mean	1.195857e+03	1.204004e+03	1.133676e+03	1.170437e+03	1.217769e+03	1.290273e+03	1.239137e+03	1.193092e+03	1.
std	7.275352e+04	7.421515e+04	6.961022e+04	7.257351e+04	7.379612e+04	8.054448e+04	7.576288e+04	6.820002e+04	7.
min	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.
25%	1.300000e+01	1.300000e+01	1.200000e+01	1.300000e+01	1.400000e+01	1.100000e+01	1.300000e+01	1.300000e+01	1.
50%	1.090000e+02	1.080000e+02	1.050000e+02	1.050000e+02	1.130000e+02	1.130000e+02	1.150000e+02	1.170000e+02	1.
75%	5.240000e+02	5.190000e+02	5.040000e+02	4.870000e+02	5.400000e+02	5.550000e+02	5.510000e+02	5.540000e+02	5.
max	2.038124e+07	2.075219e+07	1.957397e+07	2.043964e+07	2.077211e+07	2.254467e+07	2.121089e+07	1.910791e+07	1.

8 rows × 550 columns

In [36]:

```
train.shape
```

Out[36]:

```
(145063, 551)
```

In [37]:

```
train.head(5)
```

Out[37]:

	Page	2015-07-01	2015-07-02	2015-07-03	2015-07-04	2015-07-05	2015-07-06	2015-07-07	2015-07-08	2015-07-09	...	2016-12-22	2016-12-23	20 12-
0	2NE1_zh.wikipedia.org_all-access_spider	18.0	11.0	5.0	13.0	14.0	9.0	9.0	22.0	26.0	...	32.0	63.0	1
1	2PM_zh.wikipedia.org_all-access_spider	11.0	14.0	15.0	18.0	11.0	13.0	22.0	11.0	10.0	...	17.0	42.0	2
2	3C_zh.wikipedia.org_all-access_spider	1.0	0.0	1.0	1.0	0.0	4.0	0.0	3.0	4.0	...	3.0	1.0	
3	4minute_zh.wikipedia.org_all-access_spider	35.0	13.0	10.0	94.0	4.0	26.0	14.0	9.0	11.0	...	32.0	10.0	2
4	52_Hz_I_Love_You_zh.wikipedia.org_all-access_s...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	48.0	9.0	2

5 rows × 551 columns

In [38]:

```
train.dtypes
```

Out[38]:

	0
Page	object
2015-07-01	float64
2015-07-02	float64
2015-07-03	float64
2015-07-04	float64
...	...
2016-12-27	float64
2016-12-28	float64
2016-12-29	float64
2016-12-30	float64
2016-12-31	float64

551 rows × 1 columns

dtype: object

In [39]:

```
# If 'Page' column contains NaN values, fill them with an empty string or handle missing
values appropriately
train['Page'].fillna('', inplace=True)

# Split the 'Page' column into multiple columns
split_columns = train['Page'].str.split('_', expand=True)

# Assign the split columns to new columns in the original DataFrame
train['Specific_Name'] = split_columns[0]
train['Language_Domain'] = split_columns[1]
train['Access_Type'] = split_columns[2]
train['Access_Origin'] = split_columns[3]

# Drop the original 'Page' column if you no longer need it
train.drop(columns=['Page'], inplace=True)
train
```

Out[39]:

	2015-07-01	2015-07-02	2015-07-03	2015-07-04	2015-07-05	2015-07-06	2015-07-07	2015-07-08	2015-07-09	2015-07-10	...	2016-12-26	2016-12-27	2016-12-28	2016-12-29	2016-12-30	2016-12-31
0	18.0	11.0	5.0	13.0	14.0	9.0	9.0	22.0	26.0	24.0	...	14.0	20.0	22.0	19.0	18.0	20.0
1	11.0	14.0	15.0	18.0	11.0	13.0	22.0	11.0	10.0	4.0	...	9.0	30.0	52.0	45.0	26.0	20.0
2	1.0	0.0	1.0	1.0	0.0	4.0	0.0	3.0	4.0	4.0	...	4.0	4.0	6.0	3.0	4.0	17.0
3	35.0	13.0	10.0	94.0	4.0	26.0	14.0	9.0	11.0	16.0	...	16.0	11.0	17.0	19.0	10.0	11.0
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	3.0	11.0	27.0	13.0	36.0	10.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
145058	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	13.0	12.0	13.0	3.0	5.0	10.0
145059	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN

	2015-07-01	2015-07-02	2015-07-03	2015-07-04	2015-07-05	2015-07-06	2015-07-07	2015-07-08	2015-07-09	2015-07-10	...	2016-12-26	2016-12-27	2016-12-28	2016-12-29	2016-12-30	2016-12-31	E
145060	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	
145061	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	
145062	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	

145063 rows × 554 columns

In [40]:

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 145063 entries, 0 to 145062
Columns: 554 entries, 2015-07-01 to Access_Origin
dtypes: float64(550), object(4)
memory usage: 613.1+ MB
```

In [41]:

```
train.isnull().sum()
```

Out[41]:

	0
2015-07-01	20740
2015-07-02	20816
2015-07-03	20544
2015-07-04	20654
2015-07-05	20659
...	...
2016-12-31	3465
Specific_Name	0
Language_Domain	0
Access_Type	0
Access_Origin	0

554 rows × 1 columns

dtype: int64

In [42]:

```
# Get the column names
columns = train.columns

# Move the last 4 columns to the top
new_order = list(columns[-4:]) + list(columns[:-4])

# Reorder the columns
train = train[new_order]
train
```

Out[42]:

	Specific_Name	Language_Domain	Access_Type	Access_Origin	2015-07-01	2015-07-02	2015-07-03	2015-07-04	2015-07-05	2015-07-06	...	2016-12-2
0	2NE1	zh.wikipedia.org	all-access	spider	18.0	11.0	5.0	13.0	14.0	9.0	...	32.
1	2PM	zh.wikipedia.org	all-access	spider	11.0	14.0	15.0	18.0	11.0	13.0	...	17.
2	3C	zh.wikipedia.org	all-access	spider	1.0	0.0	1.0	1.0	0.0	4.0	...	3.
3	4minute	zh.wikipedia.org	all-access	spider	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	...	[illegible]

	Specific_Name	Language_Domain	Access_Type	Access_Origin	2015-07-01	2015-07-02	2015-07-03	2015-07-04	2015-07-05	2015-07-06	...	2016-12-2
3	4minute	zh.wikipedia.org	all-access	spider	35.0	13.0	10.0	94.0	4.0	26.0	...	32.
4	52	Hz	1	Love	NaN	NaN	NaN	NaN	NaN	NaN	...	48
...	...	...	...	...	...	...	...	...	...	...	...	.
145058	Underworld	(serie	de	películas)	NaN	NaN	NaN	NaN	NaN	NaN	...	Nal
145059	Resident	Evil:	Capítulo	Final	NaN	NaN	NaN	NaN	NaN	NaN	...	Nal
145060	Enamorándome	de	Ramón	es.wikipedia.org	NaN	NaN	NaN	NaN	NaN	NaN	...	Nal
145061	Hasta	el	último	hombre	NaN	NaN	NaN	NaN	NaN	NaN	...	Nal
145062	Francisco	el	matemático	(serie	NaN	NaN	NaN	NaN	NaN	NaN	...	Nal

145063 rows × 554 columns

In [43]:

```
train.isnull().sum()
```

Out[43]:

	0
Specific_Name	0
Language_Domain	0
Access_Type	0
Access_Origin	0
2015-07-01	20740
...	...
2016-12-27	3701
2016-12-28	3822
2016-12-29	3826
2016-12-30	3635
2016-12-31	3465

554 rows × 1 columns

dtype: int64

In [44]:

```
train1 = train.select_dtypes(include='float64')
train1
```

Out[44]:

	2015-07-01	2015-07-02	2015-07-03	2015-07-04	2015-07-05	2015-07-06	2015-07-07	2015-07-08	2015-07-09	2015-07-10	...	2016-12-22	2016-12-23	2016-12-24	2016-12-25	2016-12-26	2016-12-27	2(1:
0	18.0	11.0	5.0	13.0	14.0	9.0	9.0	22.0	26.0	24.0	...	32.0	63.0	15.0	26.0	14.0	20.0	
1	11.0	14.0	15.0	18.0	11.0	13.0	22.0	11.0	10.0	4.0	...	17.0	42.0	28.0	15.0	9.0	30.0	
2	1.0	0.0	1.0	1.0	0.0	4.0	0.0	3.0	4.0	4.0	...	3.0	1.0	1.0	7.0	4.0	4.0	
3	35.0	13.0	10.0	94.0	4.0	26.0	14.0	9.0	11.0	16.0	...	32.0	10.0	26.0	27.0	16.0	11.0	
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	48.0	9.0	25.0	13.0	3.0	11.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
145058	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	13.0	12.0	
145059	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	I
145060	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	I

145061	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN
145062	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN

145063 rows × 550 columns

Working with Missing Values

In [45]:

```
train1.isnull().sum()
```

Out[45]:

	0
2015-07-01	20740
2015-07-02	20816
2015-07-03	20544
2015-07-04	20654
2015-07-05	20659
...	...
2016-12-27	3701
2016-12-28	3822
2016-12-29	3826
2016-12-30	3635
2016-12-31	3465

550 rows × 1 columns

dtype: int64

In [46]:

```
train1.interpolate(method='linear', inplace=True)
train1
```

Out[46]:

	2015-07-01	2015-07-02	2015-07-03	2015-07-04	2015-07-05	2015-07-06	2015-07-07	2015-07-08	2015-07-09	2015-07-10	...	2016-12-22	2016-12-23	2016-12-24	2016-12-25	2016-12-26	2016-12-27
0	18.0	11.0	5.0	13.0	14.0	9.0	9.0	22.0	26.0	24.0	...	32.0	63.0	15.0	26.0	14.0	20.0
1	11.0	14.0	15.0	18.0	11.0	13.0	22.0	11.0	10.0	4.0	...	17.0	42.0	28.0	15.0	9.0	30.0
2	1.0	0.0	1.0	1.0	0.0	4.0	0.0	3.0	4.0	4.0	...	3.0	1.0	1.0	7.0	4.0	4.0
3	35.0	13.0	10.0	94.0	4.0	26.0	14.0	9.0	11.0	16.0	...	32.0	10.0	26.0	27.0	16.0	11.0
4	23.5	10.0	7.0	49.5	12.0	17.0	9.5	13.0	17.5	11.5	...	48.0	9.0	25.0	13.0	3.0	11.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
145058	3.0	10.0	0.0	2.0	2.0	0.0	7.0	45.0	1.0	19.0	...	0.0	0.0	1.0	1.0	13.0	12.0
145059	3.0	10.0	0.0	2.0	2.0	0.0	7.0	45.0	1.0	19.0	...	0.0	0.0	1.0	1.0	13.0	12.0
145060	3.0	10.0	0.0	2.0	2.0	0.0	7.0	45.0	1.0	19.0	...	0.0	0.0	1.0	1.0	13.0	12.0
145061	3.0	10.0	0.0	2.0	2.0	0.0	7.0	45.0	1.0	19.0	...	0.0	0.0	1.0	1.0	13.0	12.0
145062	3.0	10.0	0.0	2.0	2.0	0.0	7.0	45.0	1.0	19.0	...	0.0	0.0	1.0	1.0	13.0	12.0

145063 rows × 550 columns

In [47]:

```
train1= train1.T
train1
```

Out[47]:

	0	1	2	3	4	5	6	7	8	9	...	145053	145054	145055	145056	145057	145058	14
2015-07-01	18.0	11.0	1.0	35.0	23.5	12.0	65.0	118.0	5.0	6.0	...	3.0	3.0	3.0	3.000000	3.000000	3.0	
2015-07-02	11.0	14.0	0.0	13.0	10.0	7.0	16.5	26.0	23.0	3.0	...	10.0	10.0	10.0	10.000000	10.000000	10.0	
2015-07-03	5.0	15.0	1.0	10.0	7.0	4.0	17.0	30.0	14.0	5.0	...	0.0	0.0	0.0	0.000000	0.000000	0.0	
2015-07-04	13.0	18.0	1.0	94.0	49.5	5.0	14.5	24.0	12.0	12.0	...	2.0	2.0	2.0	2.000000	2.000000	2.0	
2015-07-05	14.0	11.0	0.0	4.0	12.0	20.0	24.5	29.0	9.0	6.0	...	2.0	2.0	2.0	2.000000	2.000000	2.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
2016-12-27	20.0	30.0	4.0	11.0	11.0	19.0	4.0	23.0	30.0	29.0	...	8.0	7.0	4.0	6.666667	9.333333	12.0	
2016-12-28	22.0	52.0	6.0	17.0	27.0	23.0	15.0	32.0	36.0	35.0	...	21.0	13.0	2.0	5.666667	9.333333	13.0	
2016-12-29	19.0	45.0	3.0	19.0	13.0	17.0	6.0	39.0	38.0	44.0	...	14.0	12.0	4.0	3.666667	3.333333	3.0	
2016-12-30	18.0	26.0	4.0	10.0	36.0	17.0	8.0	32.0	31.0	26.0	...	24.0	31.0	4.0	4.333333	4.666667	5.0	
2016-12-31	20.0	20.0	17.0	11.0	10.0	50.0	6.0	17.0	97.0	41.0	...	37.0	11.0	3.0	51.000000	30.500000	10.0	

550 rows × 145063 columns

In [48]:

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 145063 entries, 0 to 145062
Columns: 554 entries, Specific_Name to 2016-12-31
dtypes: float64(550), object(4)
memory usage: 613.1+ MB
```

In [49]:

```
train1.dtypes
```

Out[49]:

	0
0	float64
1	float64
2	float64
3	float64
4	float64
...	...
145058	float64
145059	float64

	0
145060	float64
145061	float64
145062	float64

145063 rows × 1 columns

dtype: object

In [50]:

```
train1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 550 entries, 2015-07-01 to 2016-12-31
Columns: 145063 entries, 0 to 145062
dtypes: float64(145063)
memory usage: 608.7+ MB
```

In [51]:

```
train['Language_Domain'].nunique()
```

Out[51]:

```
16509
```

In [52]:

```
train_columns = ['Specific_Name', 'Language_Domain', 'Access_Type', 'Access_Origin']

for col in train_columns:
    print(f"Number of unique values in '{col}': {train[col].nunique()}")
```

```
Number of unique values in 'Specific_Name': 33805
Number of unique values in 'Language_Domain': 16509
Number of unique values in 'Access_Type': 7375
Number of unique values in 'Access_Origin': 4085
```

In [53]:

```
for col in train_columns:
    print(f"Number of value_counts in '{col}': {train[col].value_counts()}")
```

```
Number of value_counts in 'Specific_Name': Specific_Name
The                 1501
List                 687
Liste                592
How                  438
La                   372
                    ...
File:Auschwitz         1
File:Auto              1
File:Ayrton            1
File:BYR               1
File:Tyto              1
Name: count, Length: 33805, dtype: int64
Number of value_counts in 'Language_Domain': Language_Domain
ja.wikipedia.org     18085
zh.wikipedia.org     15189
de.wikipedia.org      4532
www.mediawiki.org     3810
es.wikipedia.org      3473
                     ...
Kharbanda                1
Jadhav                   1
Tere                     1
Ranaut                   1
Þór                      1
Name: count, Length: 16509, dtype: int64
Number of value_counts in 'Access_Type': Access_Type
```

```
Number of value_counts in 'Access_Type': Access_Type
all-access          27537
desktop             14274
mobile-web          13155
en.wikipedia.org    12099
de.wikipedia.org     9487
                    ...
Naked                   1
With                    1
hari                    1
Graffiti                1
Halldórsson             1
Name: count, Length: 7375, dtype: int64
Number of value_counts in 'Access_Origin': Access_Origin
all-agents      41480
all-access      25559
spider          13486
mobile-web      12773
desktop         10987
                ...
1974.jpg            1
BUILDING            1
Dean                1
Krispies.jpg        1
Auf                 1
Name: count, Length: 4085, dtype: int64
```

In [54]:

```
eng_lan=train[train['Language_Domain']=='English'].reset_index(drop=True)
eng_lan1= eng_lan.select_dtypes(include='float64')
eng_lan1
```

Out[54]:

	2015-07-01	2015-07-02	2015-07-03	2015-07-04	2015-07-05	2015-07-06	2015-07-07	2015-07-08	2015-07-09	2015-07-10	...	2016-12-22	2016-12-23	2016-12-24	2016-12-25	2016-12-26	2016-12-27	2016-12-28
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	13.0	16.0	14.0	6.0	5.0	3.0	10.0
1	64.0	77.0	72.0	72.0	67.0	77.0	109.0	99.0	71.0	96.0	...	90.0	104.0	94.0	82.0	114.0	67.0	90.0
2	129.0	149.0	113.0	113.0	121.0	150.0	161.0	162.0	116.0	167.0	...	200.0	156.0	126.0	98.0	97.0	107.0	118.0
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN

4 rows × 550 columns

In [55]:

```
eng_lan.isnull().sum()
```

Out[55]:

	0
Specific_Name	0
Language_Domain	0
Access_Type	0
Access_Origin	0
2015-07-01	2
...	...
2016-12-27	1
2016-12-28	1
2016-12-29	1
2016-12-30	1

2016-12-31	0

554 rows × 1 columns

dtype: int64

In [56]:

```
eng_lan1.isnull().sum()
```

Out[56]:

	0
2015-07-01	2
2015-07-02	2
2015-07-03	2
2015-07-04	2
2015-07-05	2
...	...
2016-12-27	1
2016-12-28	1
2016-12-29	1
2016-12-30	1
2016-12-31	1

550 rows × 1 columns

dtype: int64

In [57]:

```
eng_lan1.interpolate(method='linear', inplace=True)
eng_lan1
```

Out[57]:

	2015-07-01	2015-07-02	2015-07-03	2015-07-04	2015-07-05	2015-07-06	2015-07-07	2015-07-08	2015-07-09	2015-07-10	...	2016-12-22	2016-12-23	2016-12-24	2016-12-25	2016-12-26	2016-12-27	2016-12-28
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	13.0	16.0	14.0	6.0	5.0	3.0	10.0
1	64.0	77.0	72.0	72.0	67.0	77.0	109.0	99.0	71.0	96.0	...	90.0	104.0	94.0	82.0	114.0	67.0	90.0
2	129.0	149.0	113.0	113.0	121.0	150.0	161.0	162.0	116.0	167.0	...	200.0	156.0	126.0	98.0	97.0	107.0	118.0
3	129.0	149.0	113.0	113.0	121.0	150.0	161.0	162.0	116.0	167.0	...	200.0	156.0	126.0	98.0	97.0	107.0	118.0

4 rows × 550 columns

In [58]:

```
eng_lan1=eng_lan1.T
eng_lan1
```

Out[58]:

	0	1	2	3
2015-07-01	NaN	64.0	129.0	129.0
2015-07-02	NaN	77.0	149.0	149.0
2015-07-03	NaN	72.0	113.0	113.0

	0	1	2	3
2015-07-04	NaN	72.0	113.0	113.0
2015-07-05	NaN	67.0	121.0	121.0
...	...	...	...	...
2016-12-27	3.0	67.0	107.0	107.0
2016-12-28	10.0	90.0	118.0	118.0
2016-12-29	1.0	101.0	148.0	148.0
2016-12-30	5.0	89.0	111.0	111.0
2016-12-31	8.0	77.0	84.0	84.0

550 rows × 4 columns

In [59]:

```
# Load Exog_Campaign_eng.csv
exog_data = pd.read_csv('Exog_Campaign_eng.csv')
```

In [60]:

```
exog_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 550 entries, 0 to 549
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Exog    550 non-null    int64
dtypes: int64(1)
memory usage: 4.4 KB
```

In [61]:

```
eng_lan1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 550 entries, 2015-07-01 to 2016-12-31
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   0       24 non-null     float64
 1   1       550 non-null    float64
 2   2       550 non-null    float64
 3   3       550 non-null    float64
dtypes: float64(4)
memory usage: 37.6+ KB
```

In [62]:

```
eng_lan1['common']=range(0,0+len(eng_lan1))
```

In [63]:

```
eng_lan1.reset_index(inplace=True)
eng_lan1
```

Out[63]:

	index	0	1	2	3	common
0	2015-07-01	NaN	64.0	129.0	129.0	0
1	2015-07-02	NaN	77.0	149.0	149.0	1
2	2015-07-03	NaN	72.0	113.0	113.0	2
3	2015-07-04	NaN	72.0	113.0	113.0	3
4	2015-07-05	NaN	67.0	121.0	121.0	4

...	index	0	1	2	3	common
545	2016-12-27	3.0	67.0	107.0	107.0	545
546	2016-12-28	10.0	90.0	118.0	118.0	546
547	2016-12-29	1.0	101.0	148.0	148.0	547
548	2016-12-30	5.0	89.0	111.0	111.0	548
549	2016-12-31	8.0	77.0	84.0	84.0	549

550 rows × 6 columns

In [65]:

```
exog_data['common']=range(0,0+len(exog_data))
exog_data
```

Out[65]:

	Exog	common
0	0	0
1	0	1
2	0	2
3	0	3
4	0	4
...	...	...
545	1	545
546	1	546
547	1	547
548	0	548
549	0	549

550 rows × 2 columns

In [66]:

```
ts_df=pd.merge(eng_lan1,exog_data,on="common")
ts_df
```

Out[66]:

	index	0	1	2	3	common	Exog
0	2015-07-01	NaN	64.0	129.0	129.0	0	0
1	2015-07-02	NaN	77.0	149.0	149.0	1	0
2	2015-07-03	NaN	72.0	113.0	113.0	2	0
3	2015-07-04	NaN	72.0	113.0	113.0	3	0
4	2015-07-05	NaN	67.0	121.0	121.0	4	0
...	...	...	...	...	...	...	...
545	2016-12-27	3.0	67.0	107.0	107.0	545	1
546	2016-12-28	10.0	90.0	118.0	118.0	546	1
547	2016-12-29	1.0	101.0	148.0	148.0	547	1
548	2016-12-30	5.0	89.0	111.0	111.0	548	0
549	2016-12-31	8.0	77.0	84.0	84.0	549	0

550 rows × 7 columns

In [67]:

```
ts_df=ts_df.drop("common", axis=1)
ts_df
```

Out[67]:

	index	0	1	2	3	Exog
0	2015-07-01	NaN	64.0	129.0	129.0	0
1	2015-07-02	NaN	77.0	149.0	149.0	0
2	2015-07-03	NaN	72.0	113.0	113.0	0
3	2015-07-04	NaN	72.0	113.0	113.0	0
4	2015-07-05	NaN	67.0	121.0	121.0	0
...	...	...	...	...	...	...
545	2016-12-27	3.0	67.0	107.0	107.0	1
546	2016-12-28	10.0	90.0	118.0	118.0	1
547	2016-12-29	1.0	101.0	148.0	148.0	1
548	2016-12-30	5.0	89.0	111.0	111.0	0
549	2016-12-31	8.0	77.0	84.0	84.0	0

550 rows × 6 columns

In [68]:

```
#ts_df["Exog"]=ts_df["Exog"].shift(4,axis=0)
ts_df.isnull().sum()
```

Out[68]:

	0
index	0
0	526
1	0
2	0
3	0
Exog	0

dtype: int64

In [69]:

```
ts_df.interpolate(method='linear', inplace=True)
```

In [70]:

```
ts_df['date'] = ts_df['index']
ts_df = ts_df.drop('index', axis=1)
ts_df['date'] = pd.to_datetime(ts_df['date'])
ts_df.set_index('date', inplace=True)

ts_df
```

Out[70]:

	0	1	2	3	Exog
date					
2015-07-01	NaN	64.0	129.0	129.0	0

	0	1	2	3	Exog
date					
2015-07-02	NaN	77.0	149.0	149.0	0
2015-07-03	NaN	72.0	113.0	113.0	0
2015-07-04	NaN	72.0	113.0	113.0	0
2015-07-05	NaN	67.0	121.0	121.0	0
...	...	...	...	...	...
2016-12-27	3.0	67.0	107.0	107.0	1
2016-12-28	10.0	90.0	118.0	118.0	1
2016-12-29	1.0	101.0	148.0	148.0	1
2016-12-30	5.0	89.0	111.0	111.0	0
2016-12-31	8.0	77.0	84.0	84.0	0

550 rows × 5 columns

In [71]:

```
ts_df = ts_df.fillna(method='bfill')
```

In [72]:

```
ts_df.dtypes
```

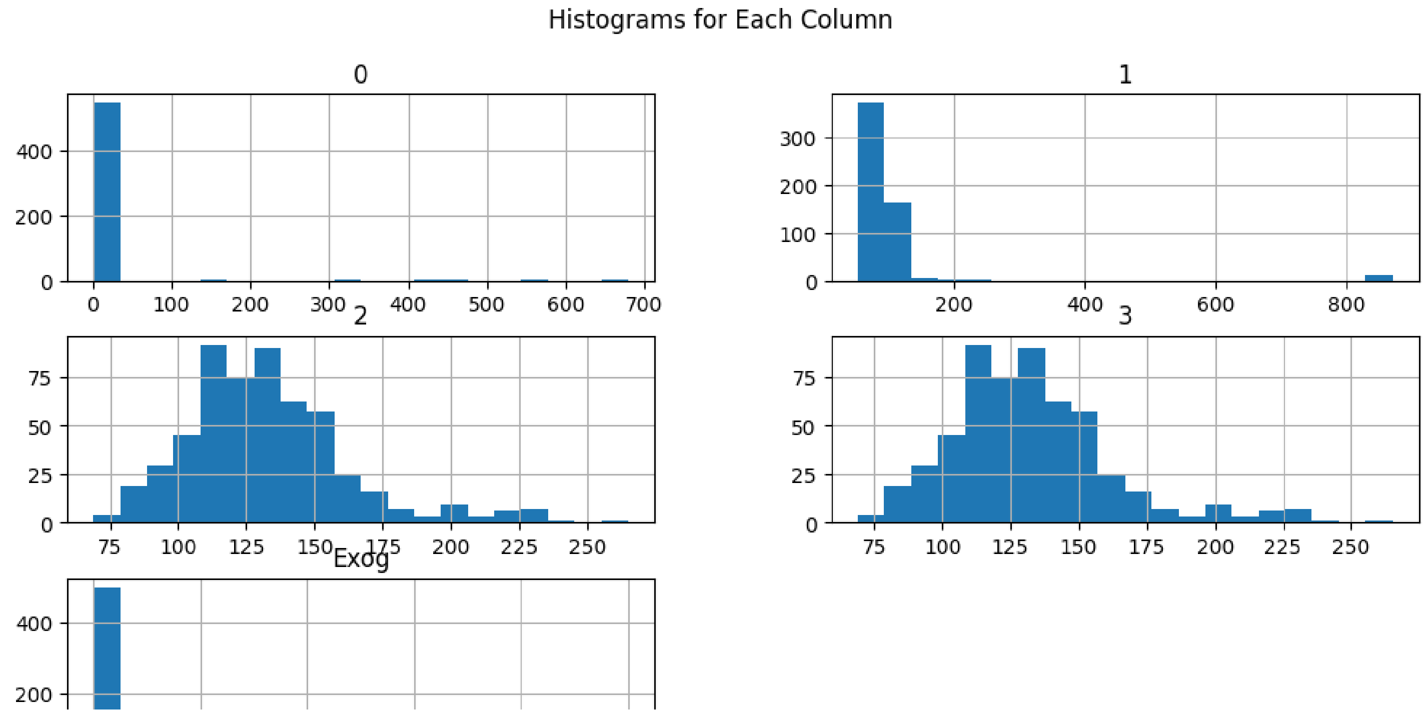
Out[72]:

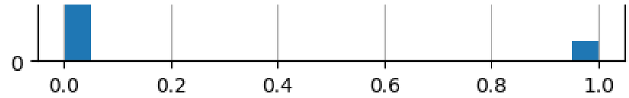
	0
0	float64
1	float64
2	float64
3	float64
Exog	int64

dtype: object

In [73]:

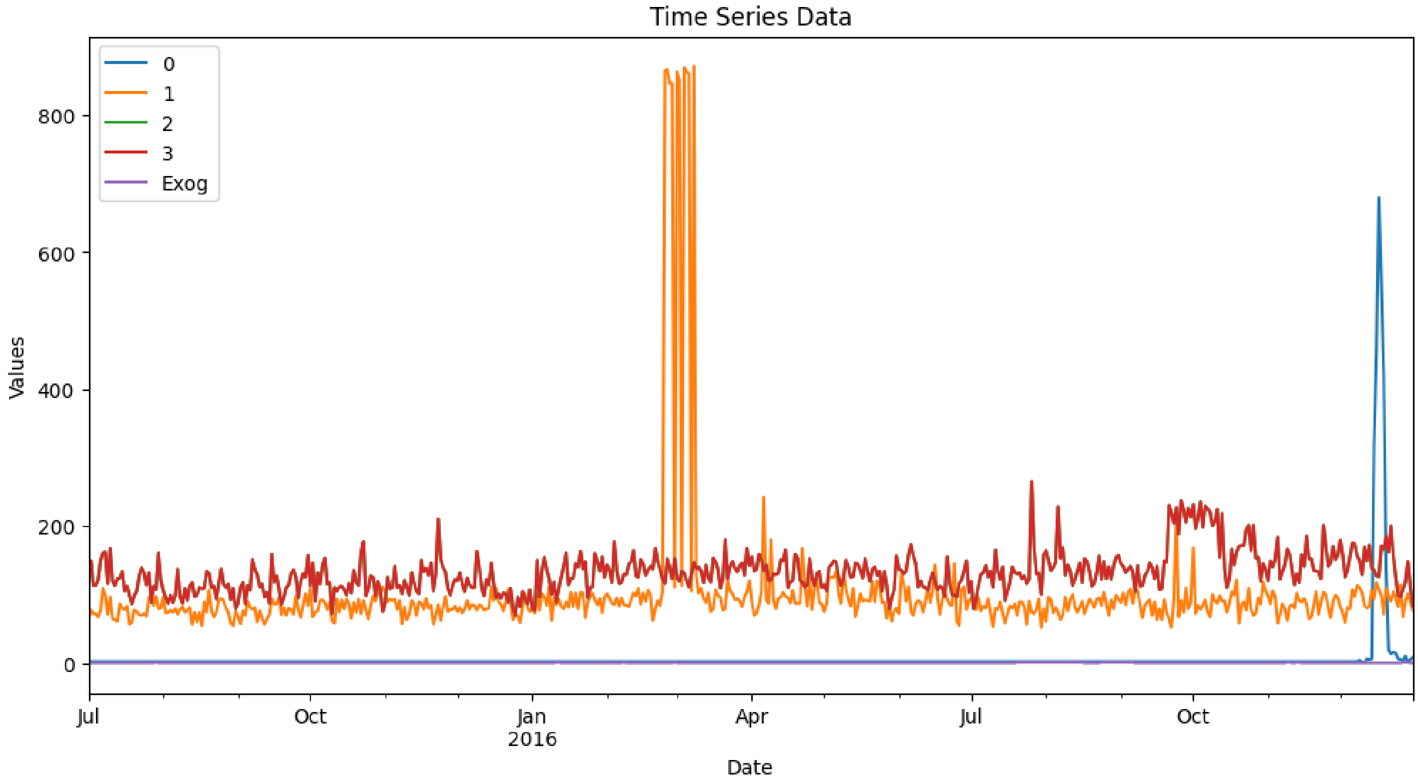
```
ts_df.hist(figsize=(12, 6), bins=20)
plt.suptitle('Histograms for Each Column')
plt.show()
```





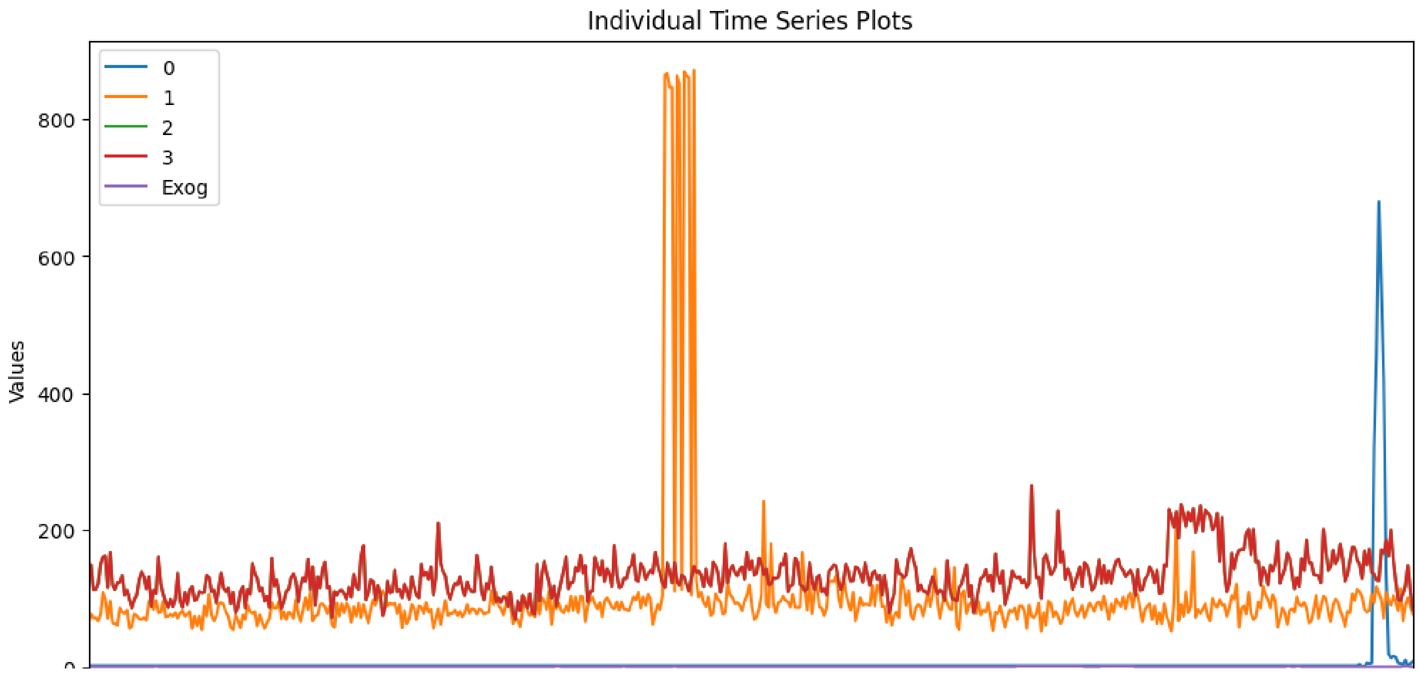
In [74]:

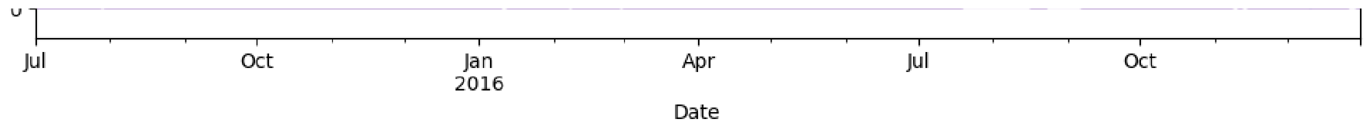
```
ts_df.plot(figsize=(12, 6))
plt.title('Time Series Data')
plt.xlabel('Date')
plt.ylabel('Values')
plt.show()
```



In [75]:

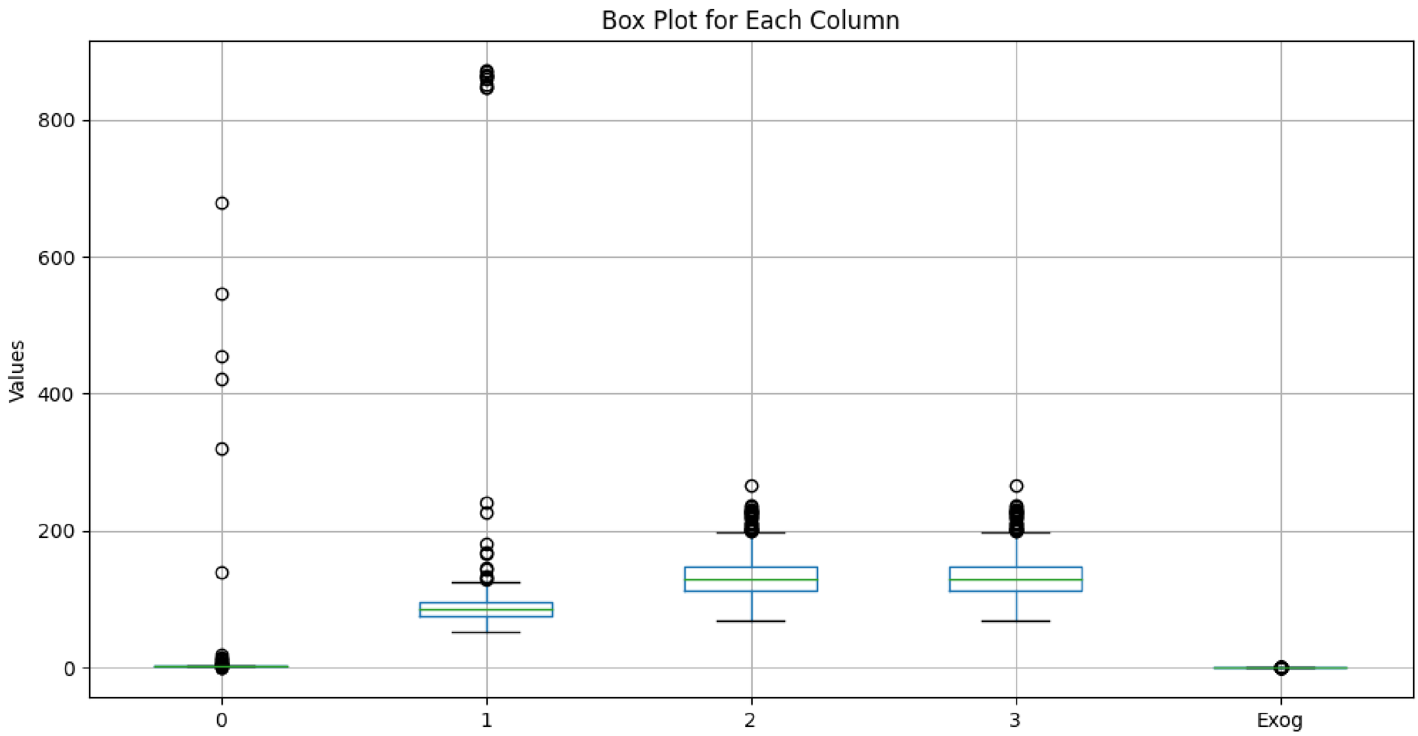
```
for column in ts_df.columns:
    ts_df[column].plot(figsize=(12, 6), label=column)

plt.title('Individual Time Series Plots')
plt.xlabel('Date')
plt.ylabel('Values')
plt.legend()
plt.show()
```





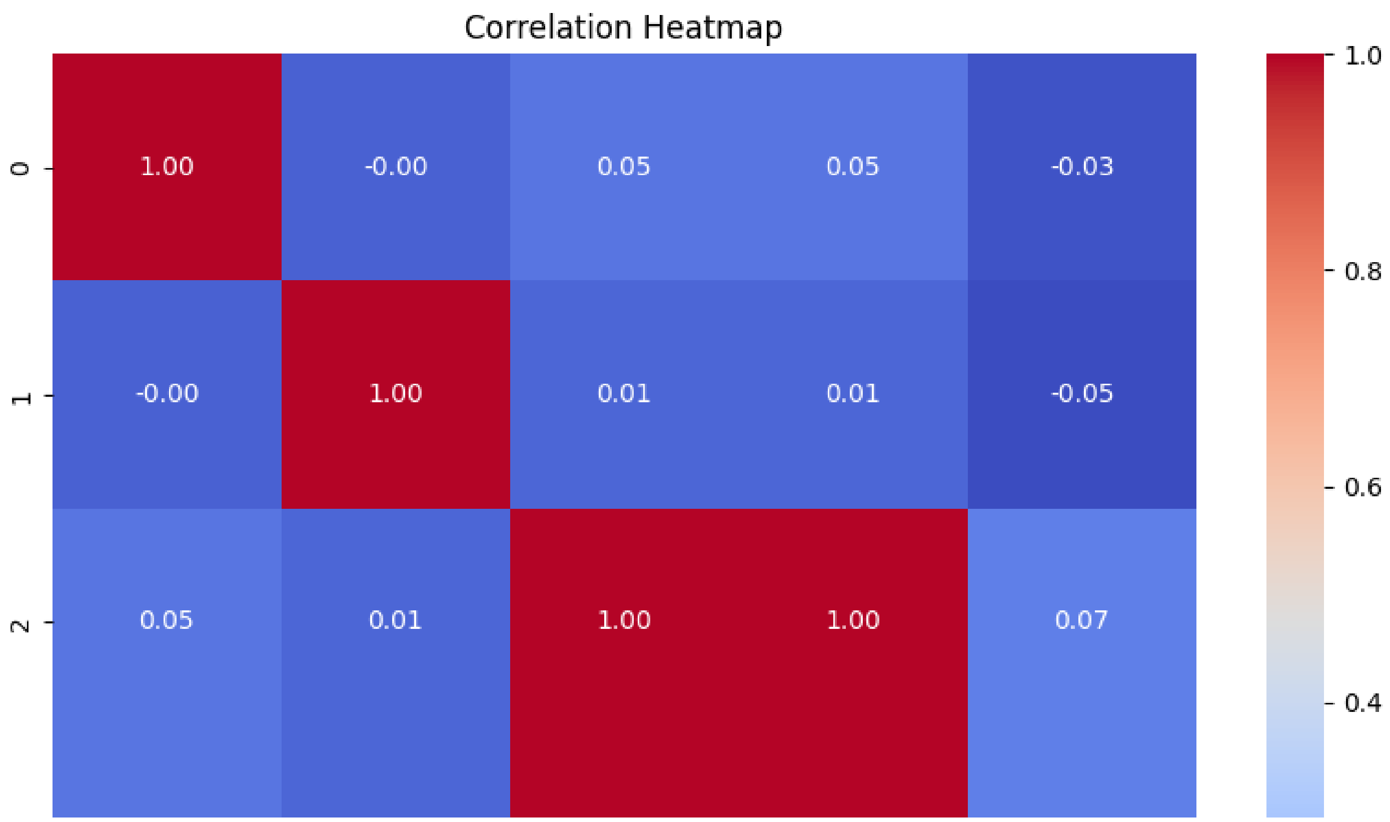
In [76]:

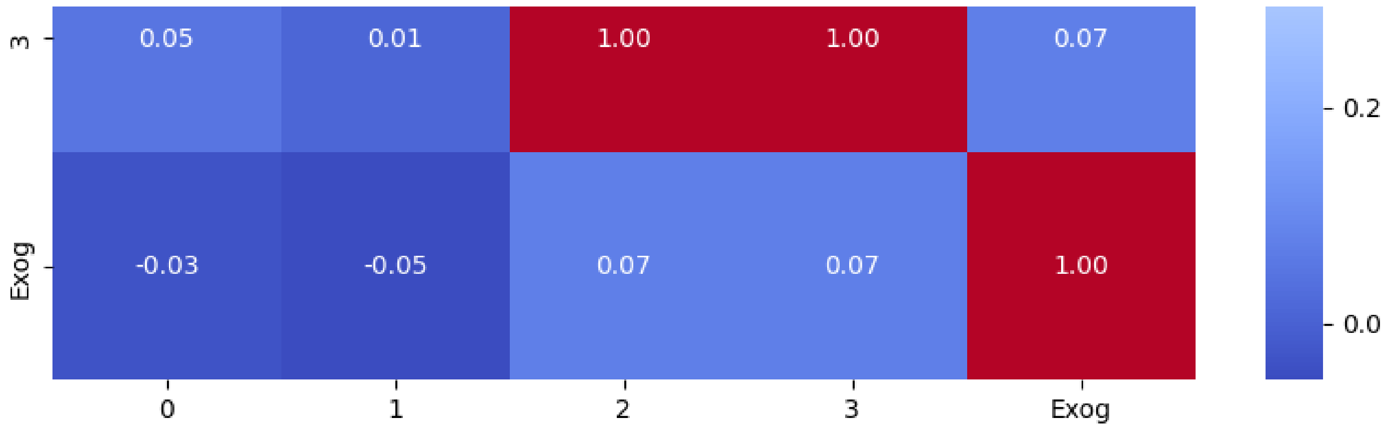
```
ts_df.boxplot(figsize=(12, 6))
plt.title('Box Plot for Each Column')
plt.ylabel('Values')
plt.show()
```



In [77]:

```
corr_matrix = ts_df.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Heatmap')
plt.show()
```





In [78]:

```
ts_df.isnull().sum()
```

Out[78]:

	0
0	0
1	0
2	0
3	0
Exog	0

dtype: int64

In [79]:

```
ts_df.columns,ts_df.dtypes
```

Out[79]:

```
(Index([0, 1, 2, 3, 'Exog'], dtype='object'),
 0        float64
 1        float64
 2        float64
 3        float64
 Exog       int64
 dtype: object)
```

In [80]:

```
ts_df.columns = ['a', 'b', 'c', 'd','Exog']
ts_df
```

Out[80]:

	a	b	c	d	Exog
date					
2015-07-01	2.0	64.0	129.0	129.0	0
2015-07-02	2.0	77.0	149.0	149.0	0
2015-07-03	2.0	72.0	113.0	113.0	0
2015-07-04	2.0	72.0	113.0	113.0	0
2015-07-05	2.0	67.0	121.0	121.0	0
...	...	...	...	...	...
2016-12-27	3.0	67.0	107.0	107.0	1
2016-12-28	10.0	90.0	118.0	118.0	1

date	a	b	c	d	Exog
2016-12-29	1.0	101.0	148.0	148.0	1
2016-12-30	5.0	89.0	111.0	111.0	0
2016-12-31	8.0	77.0	84.0	84.0	0

550 rows × 5 columns

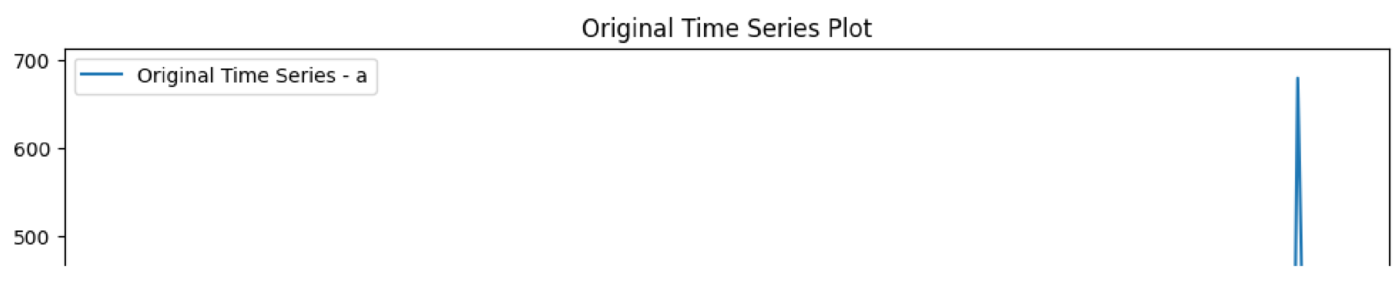
ARIMA Model

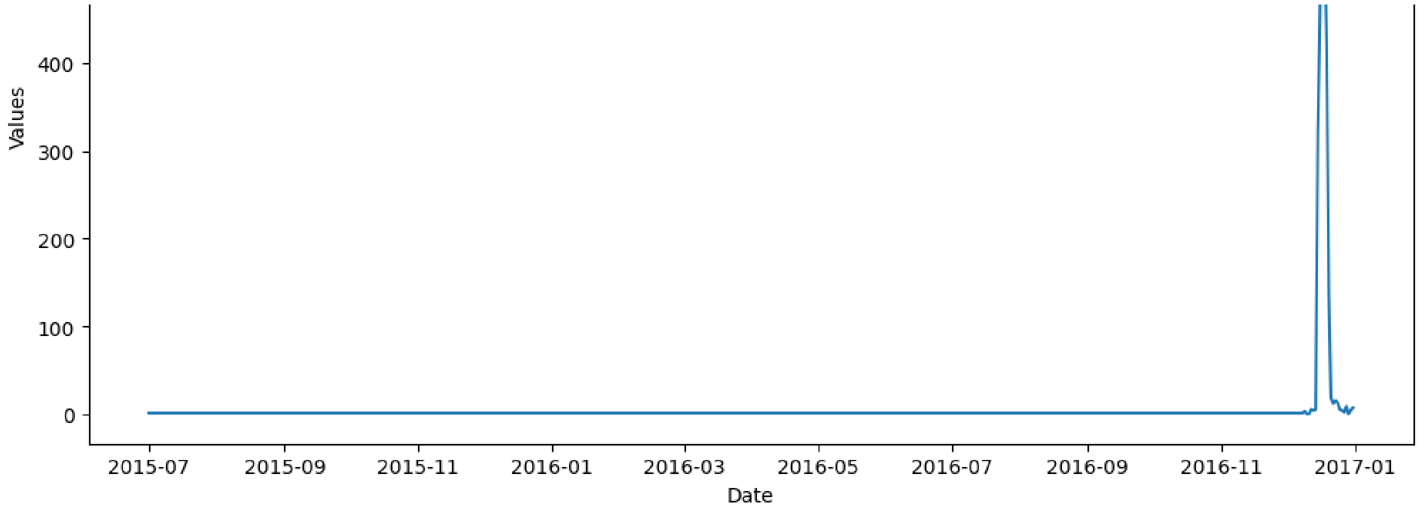
In [81]:

```
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_squared_error
```

In [82]:

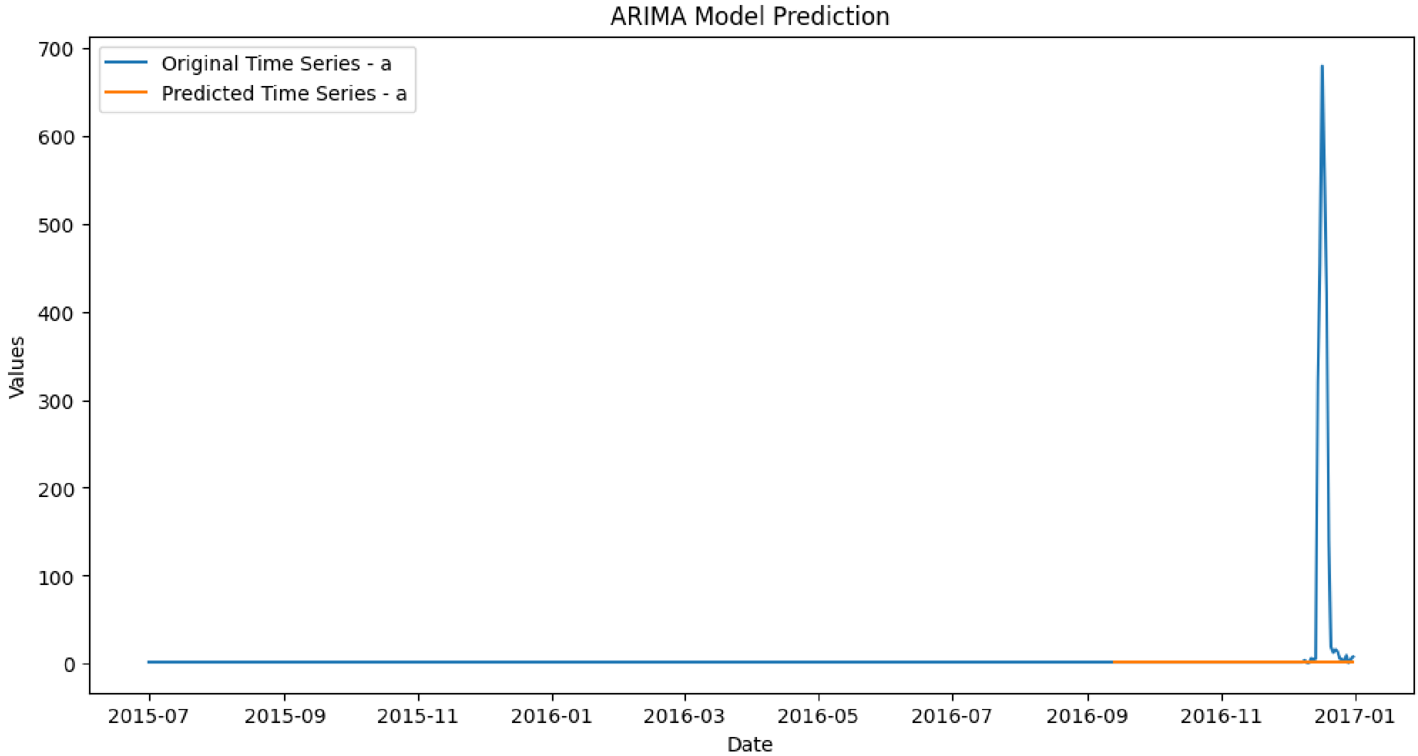
```
# Choose a specific column to model, for example, 'a'
column_to_model = 'a'

# Extract the time series data from the selected column
ts = ts_df[column_to_model]

# Check for missing values and handle them if needed
ts = ts.fillna(method='ffill')

# Plot the original time series
plt.figure(figsize=(12, 6))
plt.plot(ts.index, ts.values, label=f'Original Time Series - {column_to_model}')
plt.title('Original Time Series Plot')
plt.xlabel('Date')
plt.ylabel('Values')
plt.legend()
plt.show()

# Split the data into training and testing sets
train_size = int(len(ts) * 0.8)
train, test = ts[:train_size], ts[train_size:]

# Fit an ARIMA model
order = (1, 1, 1)  # Replace with appropriate values based on model tuning
model = ARIMA(train, order=order)
result = model.fit()

# Make predictions on the testing set
predictions = result.predict(start=len(train), end=len(train) + len(test) - 1, typ='leve
ls')

# Evaluate the model using Mean Squared Error (MSE)
mse = mean_squared_error(test, predictions)
print(f'Mean Squared Error: {mse}')

# Plot the original and predicted time series
plt.figure(figsize=(12, 6))
plt.plot(ts.index, ts.values, label=f'Original Time Series - {column_to_model}')
plt.plot(test.index, predictions, label=f'Predicted Time Series - {column_to_model}')
plt.title('ARIMA Model Prediction')
plt.xlabel('Date')
plt.ylabel('Values')
plt.legend()
plt.show()
```






```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: ConvergenceWarning
: Maximum Likelihood optimization failed to converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
```

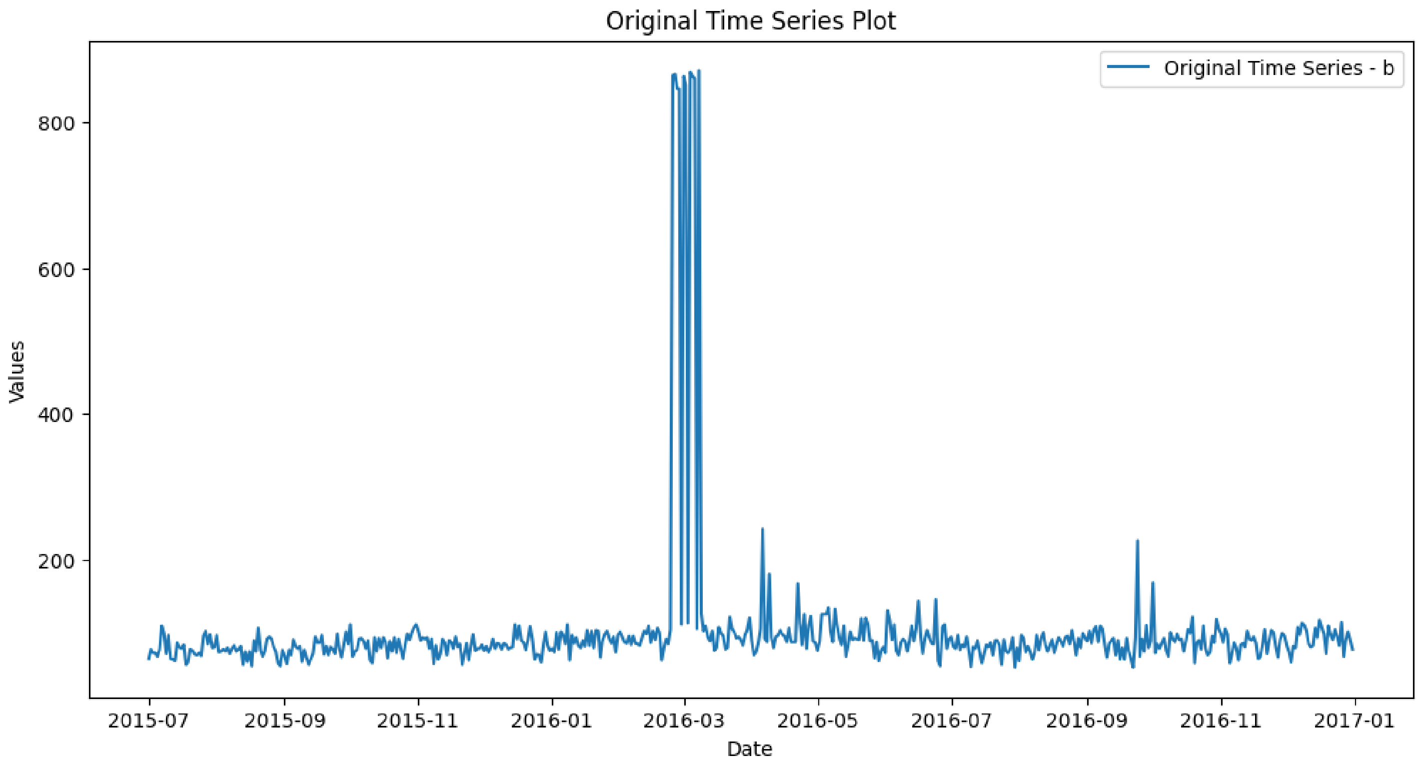
```
Mean Squared Error: 11421.09090909091
```



In [83]:

```
# Choose a specific column to model, for example, 'b'
column_to_model = 'b'

# Extract the time series data from the selected column
ts = ts_df[column_to_model]

# Check for missing values and handle them if needed
ts = ts.fillna(method='ffill')

# Plot the original time series
plt.figure(figsize=(12, 6))
```

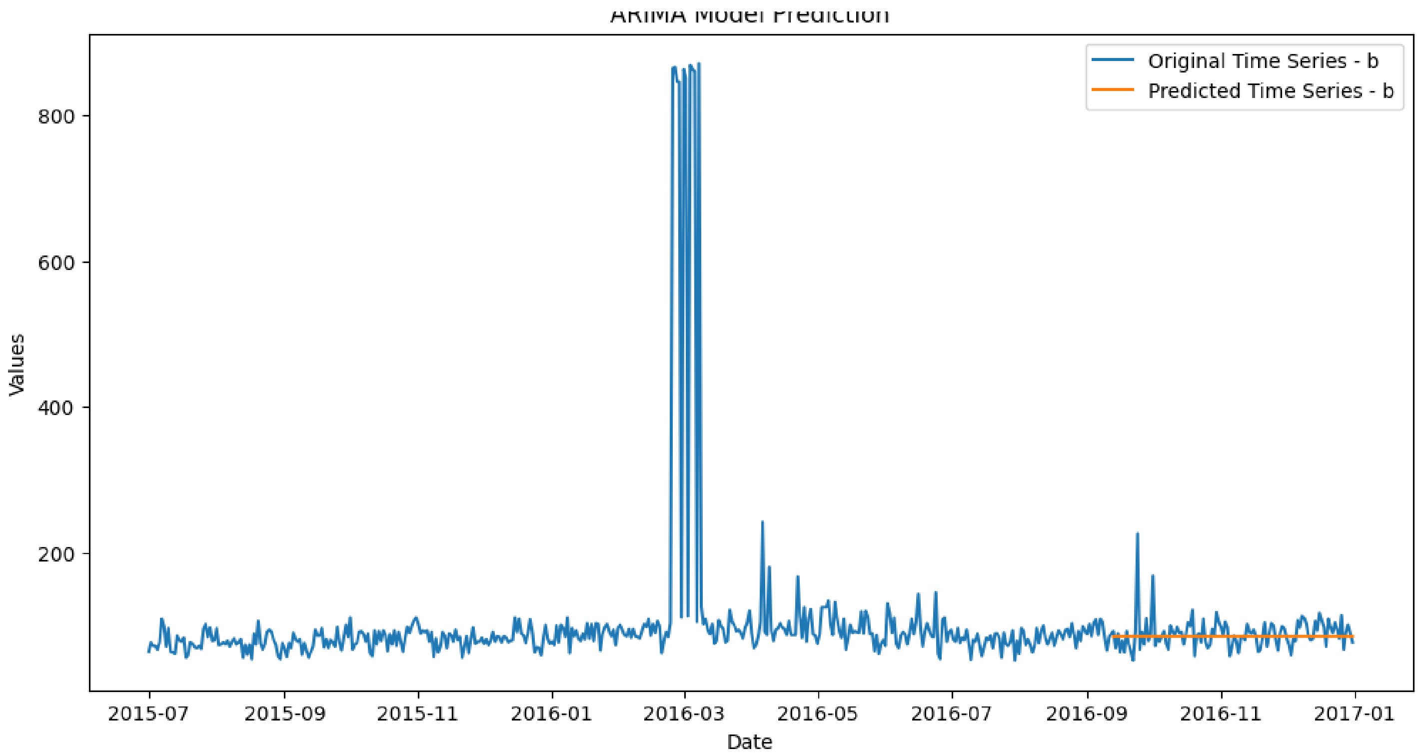
```
plt.plot(ts.index, ts.values, label=f'Original Time Series - {column_to_model}')
plt.title('Original Time Series Plot')
plt.xlabel('Date')
plt.ylabel('Values')
plt.legend()
plt.show()

# Split the data into training and testing sets
train_size = int(len(ts) * 0.8)
train, test = ts[:train_size], ts[train_size:]

# Fit an ARIMA model
order = (1, 1, 1)  # Replace with appropriate values based on model tuning
model = ARIMA(train, order=order)
result = model.fit()

# Make predictions on the testing set
predictions = result.predict(start=len(train), end=len(train) + len(test) - 1, typ='leve
ls')

# Evaluate the model using Mean Squared Error (MSE)
mse = mean_squared_error(test, predictions)
print(f'Mean Squared Error: {mse}')

# Plot the original and predicted time series
plt.figure(figsize=(12, 6))
plt.plot(ts.index, ts.values, label=f'Original Time Series - {column_to_model}')
plt.plot(test.index, predictions, label=f'Predicted Time Series - {column_to_model}')
plt.title('ARIMA Model Prediction')
plt.xlabel('Date')
plt.ylabel('Values')
plt.legend()
plt.show()
```




```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
```

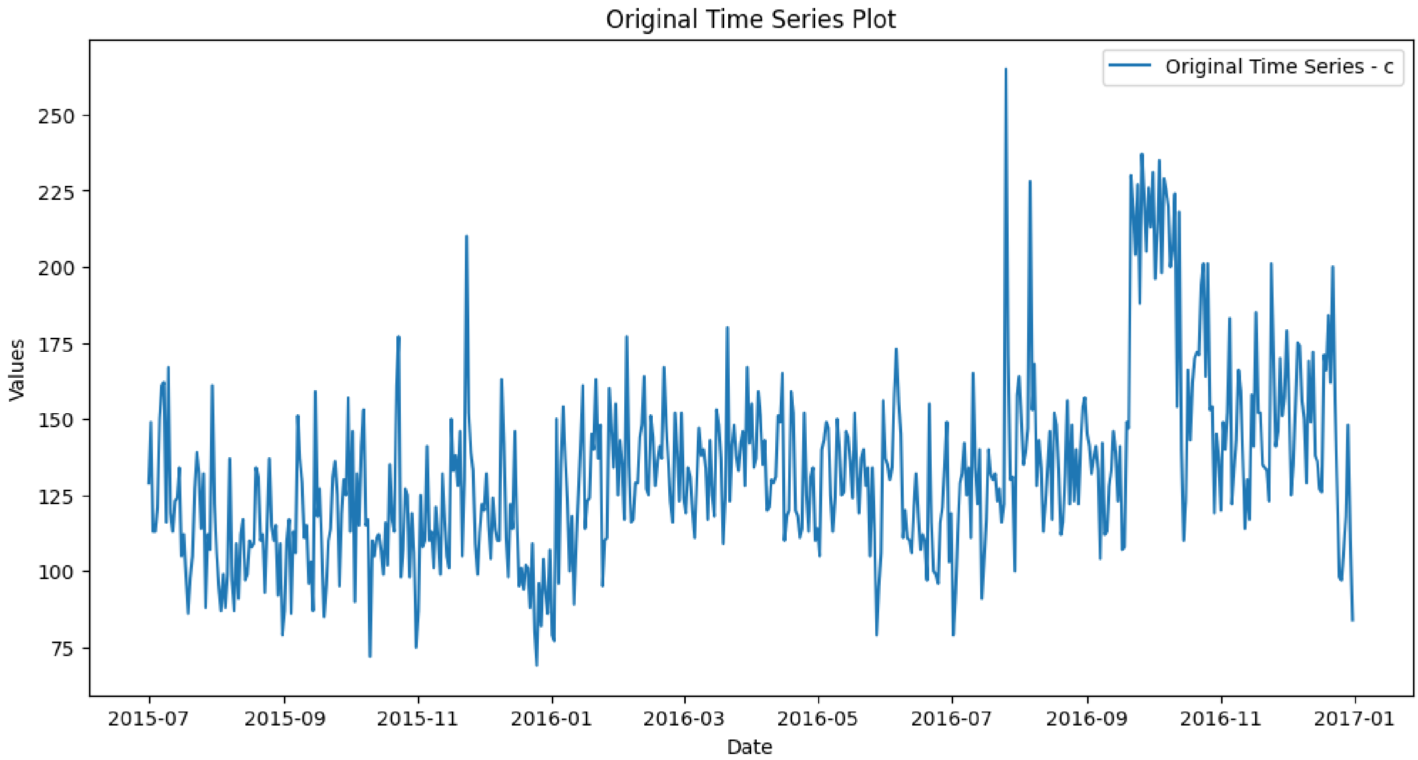
```
Mean Squared Error: 465.8498134754531
```

ARIMA Model Prediction



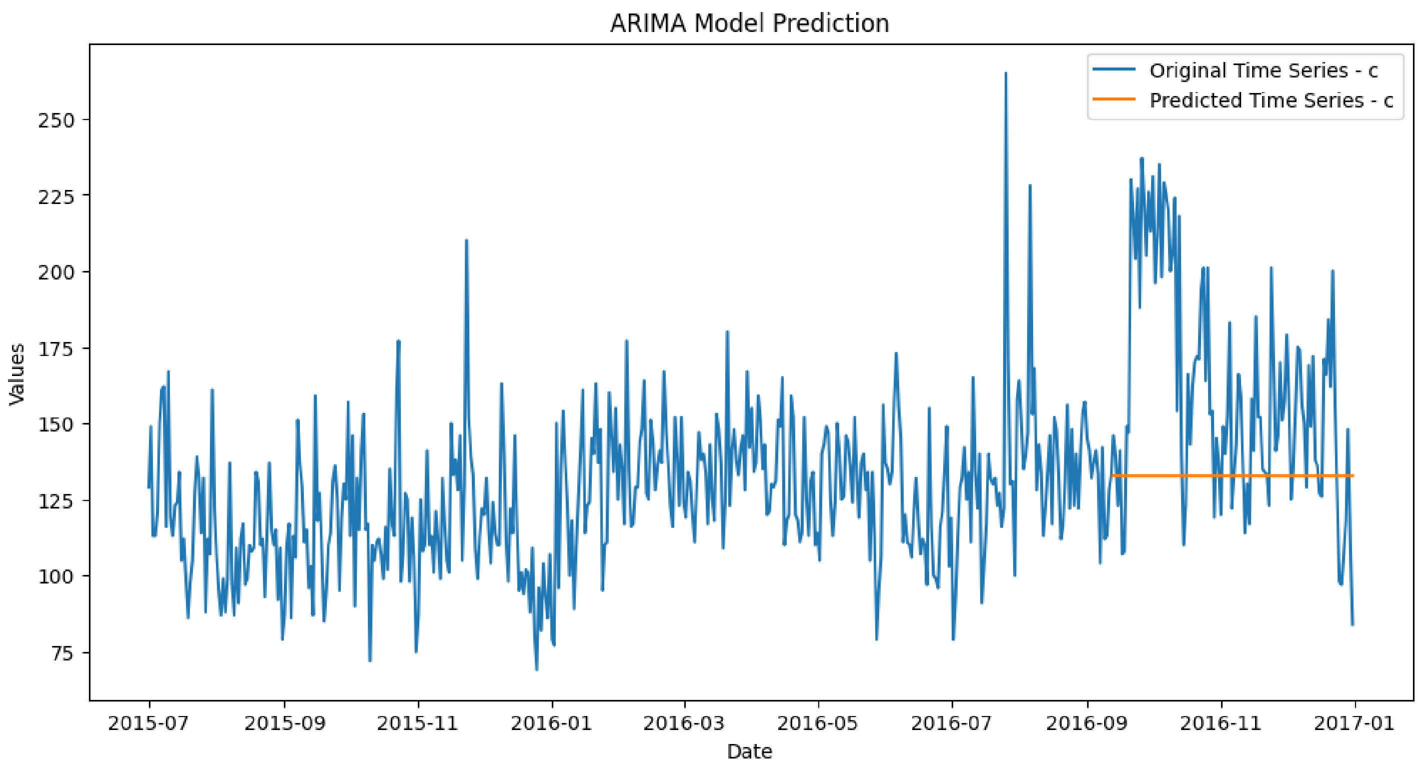
In [84]:

```
# Choose a specific column to model, for example, 'c'
column_to_model = 'c'

# Extract the time series data from the selected column
ts = ts_df[column_to_model]

# Check for missing values and handle them if needed
ts = ts.fillna(method='ffill')

# Plot the original time series
plt.figure(figsize=(12, 6))
plt.plot(ts.index, ts.values, label=f'Original Time Series - {column_to_model}')
plt.title('Original Time Series Plot')
plt.xlabel('Date')
plt.ylabel('Values')
plt.legend()
plt.show()

# Split the data into training and testing sets
train_size = int(len(ts) * 0.8)
train, test = ts[:train_size], ts[train_size:]

# Fit an ARIMA model
order = (1, 1, 1)  # Replace with appropriate values based on model tuning
model = ARIMA(train, order=order)
result = model.fit()

# Make predictions on the testing set
predictions = result.predict(start=len(train), end=len(train) + len(test) - 1, typ='leve
ls')

# Evaluate the model using Mean Squared Error (MSE)
mse = mean_squared_error(test, predictions)
print(f'Mean Squared Error: {mse}')

# Plot the original and predicted time series
plt.figure(figsize=(12, 6))
plt.plot(ts.index, ts.values, label=f'Original Time Series - {column_to_model}')
plt.plot(test.index, predictions, label=f'Predicted Time Series - {column_to_model}')
plt.title('ARIMA Model Prediction')
plt.xlabel('Date')
plt.ylabel('Values')
plt.legend()
plt.show()
```




```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
```

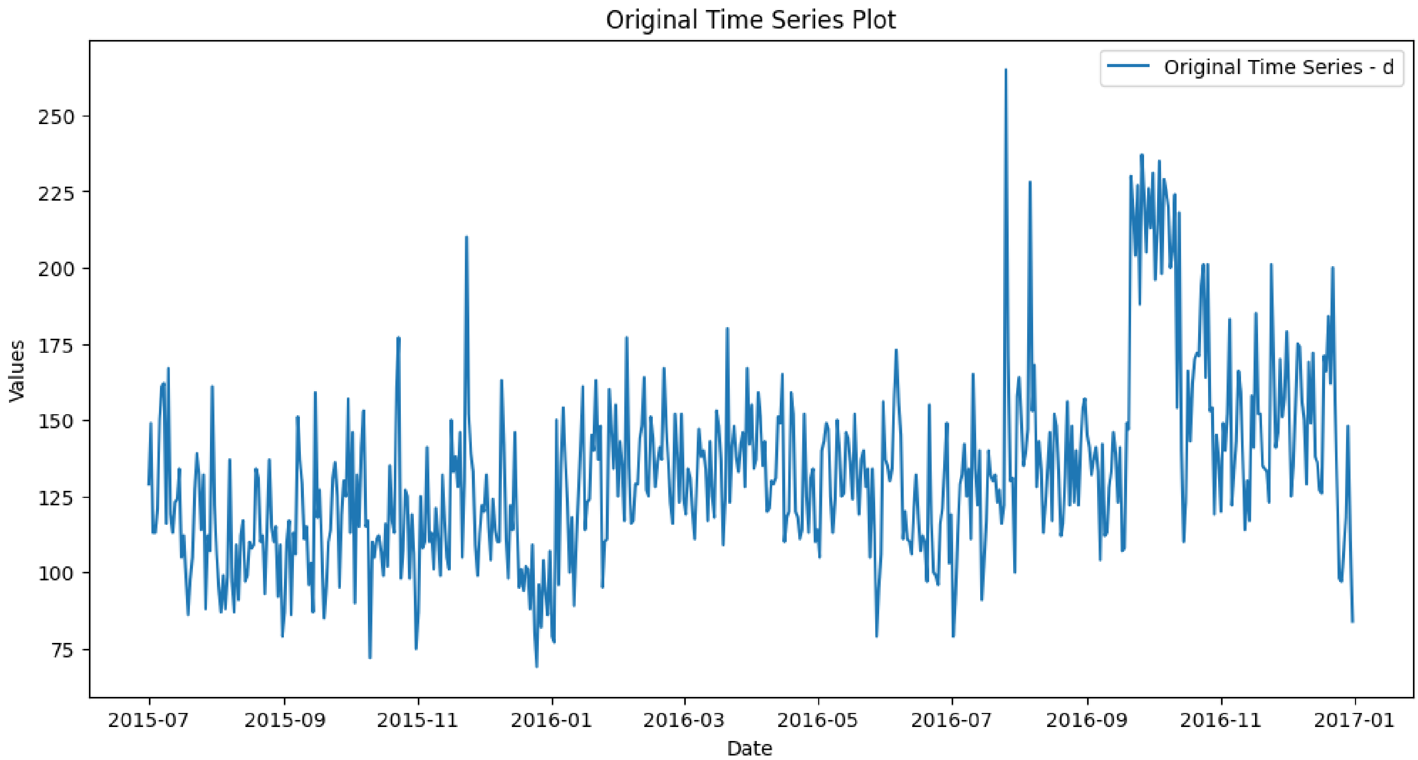
```
Mean Squared Error: 2101.7014796767253
```



In [85]:

```
# Choose a specific column to model, for example, 'd'
column_to_model = 'd'

# Extract the time series data from the selected column
ts = ts_df[column_to_model]
```

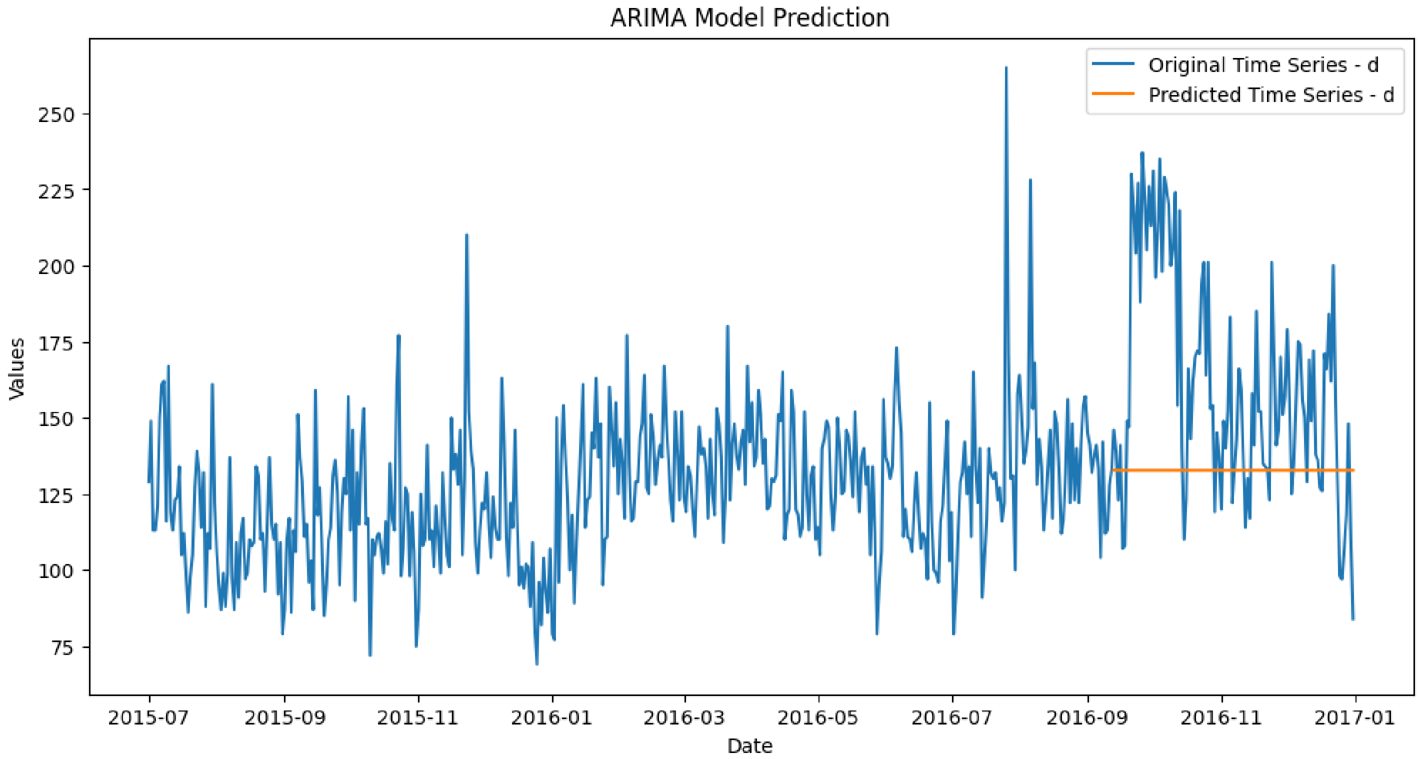
```
# Check for missing values and handle them if needed
ts = ts.fillna(method='ffill')

# Plot the original time series
plt.figure(figsize=(12, 6))
plt.plot(ts.index, ts.values, label=f'Original Time Series - {column_to_model}')
plt.title('Original Time Series Plot')
plt.xlabel('Date')
plt.ylabel('Values')
plt.legend()
plt.show()

# Split the data into training and testing sets
train_size = int(len(ts) * 0.8)
train, test = ts[:train_size], ts[train_size:]

# Fit an ARIMA model
order = (1, 1, 1)  # Replace with appropriate values based on model tuning
model = ARIMA(train, order=order)
result = model.fit()

# Make predictions on the testing set
predictions = result.predict(start=len(train), end=len(train) + len(test) - 1, typ='leve
ls')

# Evaluate the model using Mean Squared Error (MSE)
mse = mean_squared_error(test, predictions)
print(f'Mean Squared Error: {mse}')

# Plot the original and predicted time series
plt.figure(figsize=(12, 6))
plt.plot(ts.index, ts.values, label=f'Original Time Series - {column_to_model}')
plt.plot(test.index, predictions, label=f'Predicted Time Series - {column_to_model}')
plt.title('ARIMA Model Prediction')
plt.xlabel('Date')
plt.ylabel('Values')
plt.legend()
plt.show()
```




```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
```

```
Mean Squared Error: 2101.7014796767253
```



Dickey-Fuller test

In [86]:

```
from statsmodels.tsa.stattools import adfuller

# Choose a specific column for the test, for example, 'a'
column_for_test = 'a'

# Extract the time series data from the selected column
ts = ts_df[column_for_test]

# Define a function to perform the Dickey-Fuller test and print the results
def adf_test(timeseries):
    result = adfuller(timeseries, autolag='AIC')
    print('ADF Statistic:', result[0])
    print('p-value:', result[1])
    print('Critical Values:', result[4])

    if result[1] <= 0.05:
        print("Reject the null hypothesis. The data is stationary.")
    else:
        print("Fail to reject the null hypothesis. The data is non-stationary.")

# Perform the Dickey-Fuller test
adf_test(ts)
```

```
ADF Statistic: 46.93090929226497
p-value: 1.0
Critical Values: {'1%': -3.4427485933555886, '5%': -2.8670087381529723, '10%': -2.5696826
41509434}
Fail to reject the null hypothesis. The data is non-stationary.
```

In [87]:

```
from statsmodels.tsa.stattools import adfuller

# Choose a specific column for the test, for example, 'b'
```

```python
column_for_test = 'b'

# Extract the time series data from the selected column
ts = ts_df[column_for_test]

# Define a function to perform the Dickey-Fuller test and print the results
def adf_test(timeseries):
    result = adfuller(timeseries, autolag='AIC')
    print('ADF Statistic:', result[0])
    print('p-value:', result[1])
    print('Critical Values:', result[4])

    if result[1] <= 0.05:
        print("Reject the null hypothesis. The data is stationary.")
    else:
        print("Fail to reject the null hypothesis. The data is non-stationary.")

# Perform the Dickey-Fuller test
adf_test(ts)
```

```
ADF Statistic: -6.118188627806909
p-value: 8.987823533504544e-08
Critical Values: {'1%': -3.4425405682241816, '5%': -2.8669171671779816, '10%': -2.5696338
432333636}
Reject the null hypothesis. The data is stationary.
```

In [88]:

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

# Choose a specific column for the test, for example, 'c'
column_for_test = 'c'

# Extract the time series data from the selected column
ts = ts_df[column_for_test]

# Define a function to perform the Dickey-Fuller test and print the results
def adf_test(timeseries):
    result = adfuller(timeseries, autolag='AIC')
    print('ADF Statistic:', result[0])
    print('p-value:', result[1])
    print('Critical Values:', result[4])

    if result[1] <= 0.05:
        print("Reject the null hypothesis. The data is stationary.")
    else:
        print("Fail to reject the null hypothesis. The data is non-stationary.")

# Perform the Dickey-Fuller test
adf_test(ts)
```

```
ADF Statistic: -2.950284297402434
p-value: 0.03980533248386065
Critical Values: {'1%': -3.4425861905056556, '5%': -2.8669372502674824, '10%': -2.5696445
454608505}
Reject the null hypothesis. The data is stationary.
```

In [89]:

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

# Choose a specific column for the test, for example, 'd'
column_for_test = 'd'

# Extract the time series data from the selected column
ts = ts_df[column_for_test]

# Define a function to perform the Dickey-Fuller test and print the results
def adf_test(timeseries):
    result = adfuller(timeseries, autolag='AIC')
    print('ADF Statistic:', result[0])
    print('p-value:', result[1])
```

```
    print('Critical Values:', result[4])

    if result[1] <= 0.05:
        print("Reject the null hypothesis. The data is stationary.")
    else:
        print("Fail to reject the null hypothesis. The data is non-stationary.")

# Perform the Dickey-Fuller test
adf_test(ts)
```

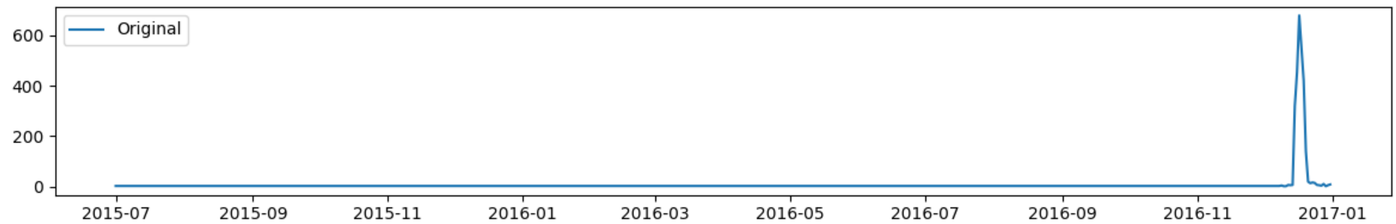
```
ADF Statistic: -2.950284297402434
p-value: 0.03980533248386065
Critical Values: {'1%': -3.4425861905056556, '5%': -2.8669372502674824, '10%': -2.5696445
454608505}
Reject the null hypothesis. The data is stationary.
```

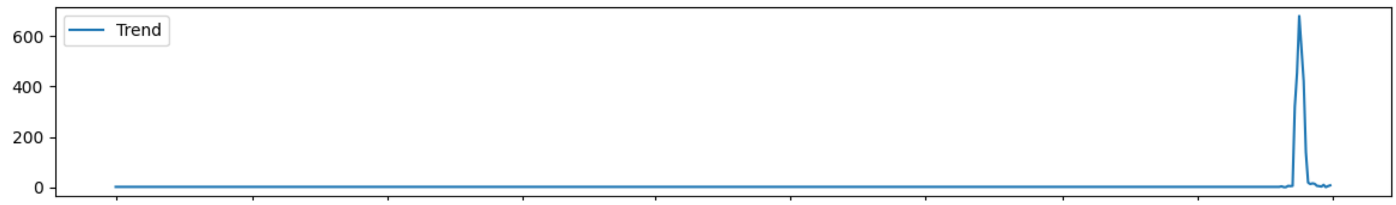
Trying different methods for stationarity.

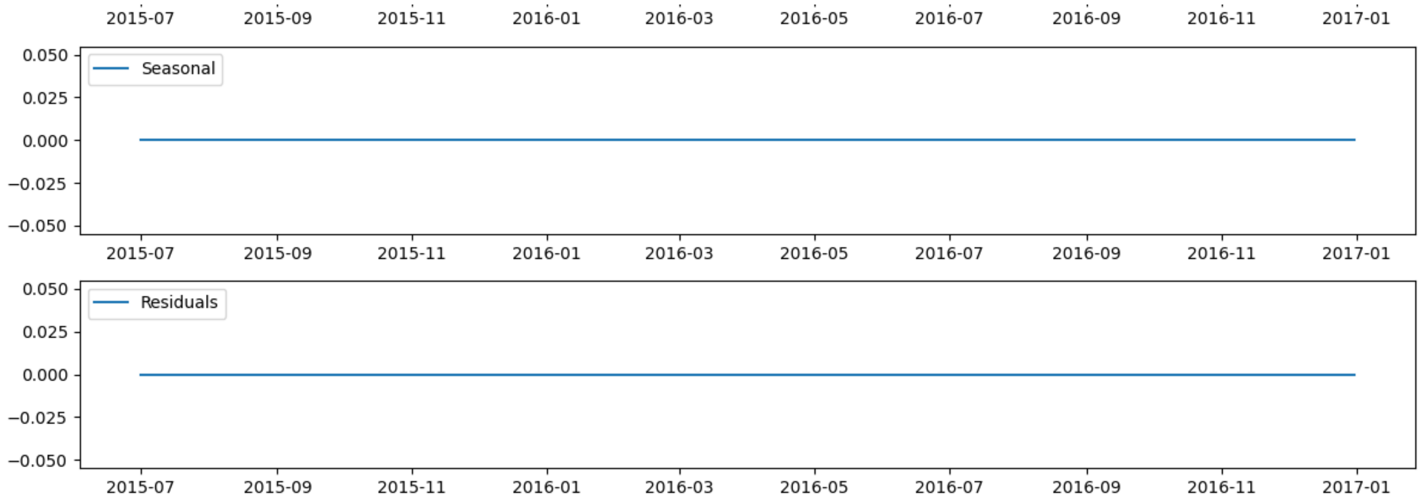
Decomposition of series.

In [90]:

```
import matplotlib.pyplot as plt
from statsmodels.tsa.seasonal import seasonal_decompose

# Choose a specific column for decomposition, for example, 'a'
column_for_decomposition = 'a'

# Extract the time series data from the selected column
ts = ts_df[column_for_decomposition]

# Decompose the time series
result = seasonal_decompose(ts, model='additive', period=1)

# Plot the decomposed components
plt.figure(figsize=(12, 8))
plt.subplot(4, 1, 1)
plt.plot(ts, label='Original')
plt.legend(loc='upper left')

plt.subplot(4, 1, 2)
plt.plot(result.trend, label='Trend')
plt.legend(loc='upper left')

plt.subplot(4, 1, 3)
plt.plot(result.seasonal, label='Seasonal')
plt.legend(loc='upper left')

plt.subplot(4, 1, 4)
plt.plot(result.resid, label='Residuals')
plt.legend(loc='upper left')

plt.tight_layout()
plt.show()
```

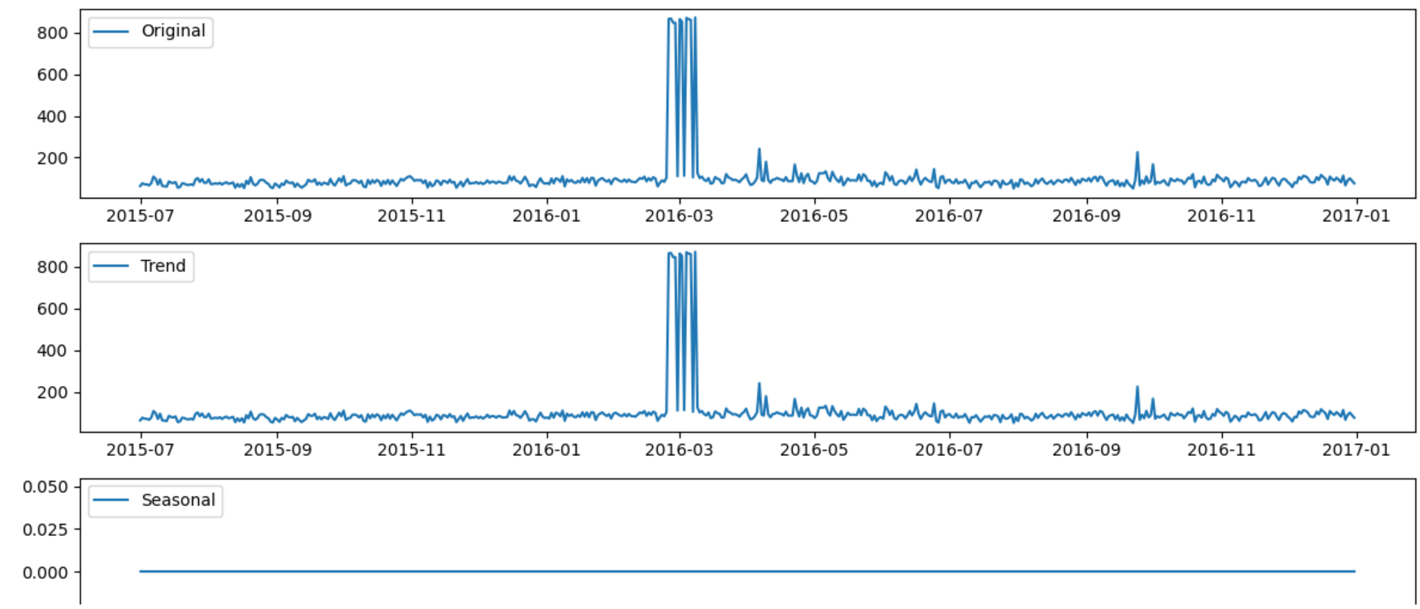


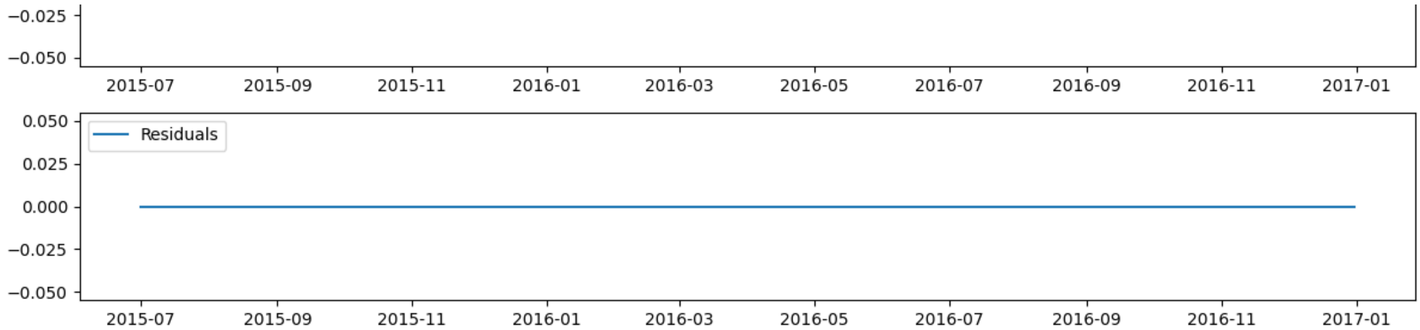




In [91]:

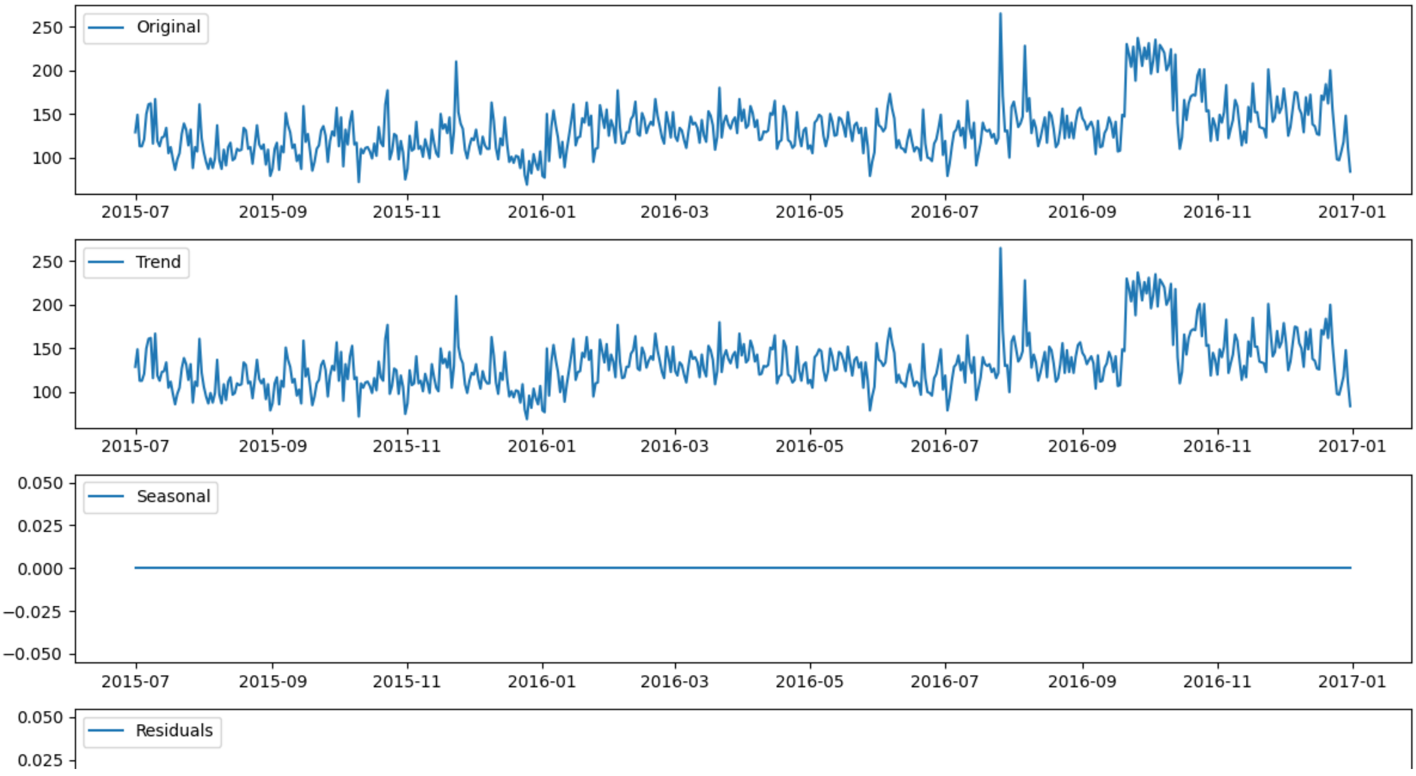
```
import matplotlib.pyplot as plt
from statsmodels.tsa.seasonal import seasonal_decompose

# Choose a specific column for decomposition, for example, 'b'
column_for_decomposition = 'b'

# Extract the time series data from the selected column
ts = ts_df[column_for_decomposition]

# Decompose the time series
result = seasonal_decompose(ts, model='additive', period=1)

# Plot the decomposed components
plt.figure(figsize=(12, 8))
plt.subplot(4, 1, 1)
plt.plot(ts, label='Original')
plt.legend(loc='upper left')

plt.subplot(4, 1, 2)
plt.plot(result.trend, label='Trend')
plt.legend(loc='upper left')

plt.subplot(4, 1, 3)
plt.plot(result.seasonal, label='Seasonal')
plt.legend(loc='upper left')

plt.subplot(4, 1, 4)
plt.plot(result.resid, label='Residuals')
plt.legend(loc='upper left')

plt.tight_layout()
plt.show()
```

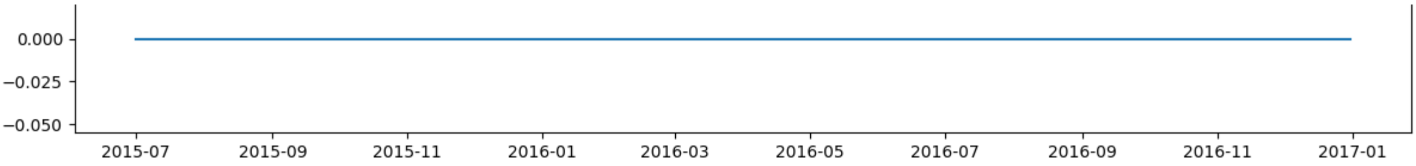




In [92]:

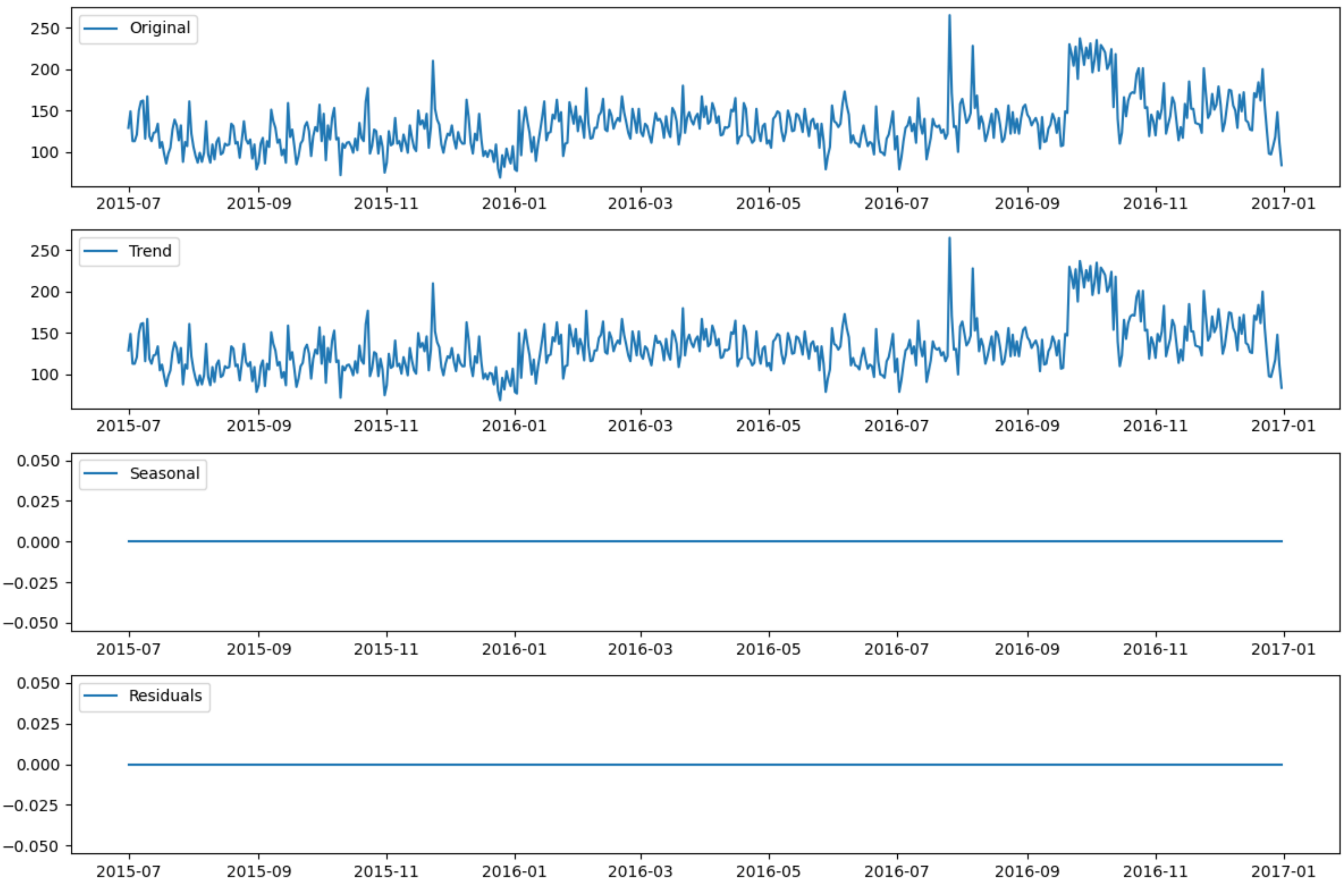
```
import matplotlib.pyplot as plt
from statsmodels.tsa.seasonal import seasonal_decompose

# Choose a specific column for decomposition, for example, 'c'
column_for_decomposition = 'c'

# Extract the time series data from the selected column
ts = ts_df[column_for_decomposition]

# Decompose the time series
result = seasonal_decompose(ts, model='additive', period=1)

# Plot the decomposed components
plt.figure(figsize=(12, 8))
plt.subplot(4, 1, 1)
plt.plot(ts, label='Original')
plt.legend(loc='upper left')

plt.subplot(4, 1, 2)
plt.plot(result.trend, label='Trend')
plt.legend(loc='upper left')

plt.subplot(4, 1, 3)
plt.plot(result.seasonal, label='Seasonal')
plt.legend(loc='upper left')

plt.subplot(4, 1, 4)
plt.plot(result.resid, label='Residuals')
plt.legend(loc='upper left')

plt.tight_layout()
plt.show()
```





In [93]:

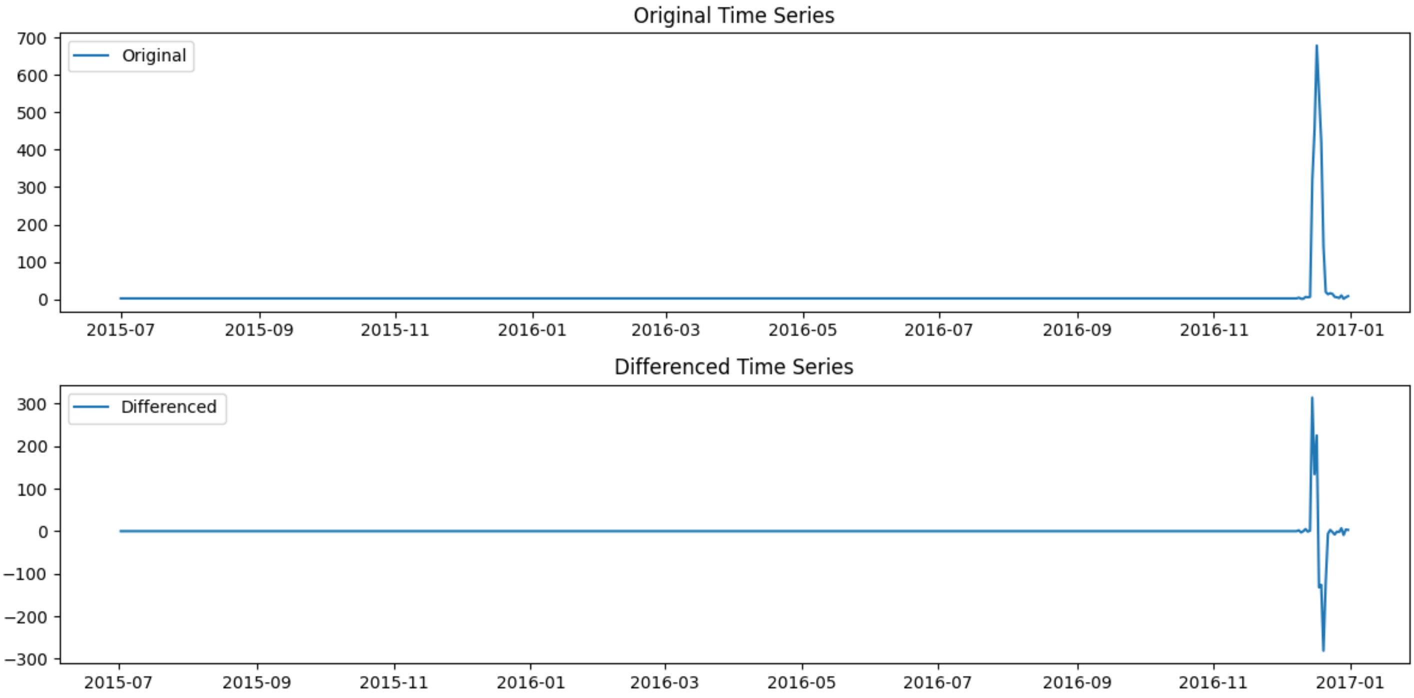
```
import matplotlib.pyplot as plt
from statsmodels.tsa.seasonal import seasonal_decompose

# Choose a specific column for decomposition, for example, 'd'
column_for_decomposition = 'd'

# Extract the time series data from the selected column
ts = ts_df[column_for_decomposition]

# Decompose the time series
result = seasonal_decompose(ts, model='additive', period=1)

# Plot the decomposed components
plt.figure(figsize=(12, 8))
plt.subplot(4, 1, 1)
plt.plot(ts, label='Original')
plt.legend(loc='upper left')

plt.subplot(4, 1, 2)
plt.plot(result.trend, label='Trend')
plt.legend(loc='upper left')

plt.subplot(4, 1, 3)
plt.plot(result.seasonal, label='Seasonal')
plt.legend(loc='upper left')

plt.subplot(4, 1, 4)
plt.plot(result.resid, label='Residuals')
plt.legend(loc='upper left')

plt.tight_layout()
plt.show()
```



Differencing the series.

In [94]:

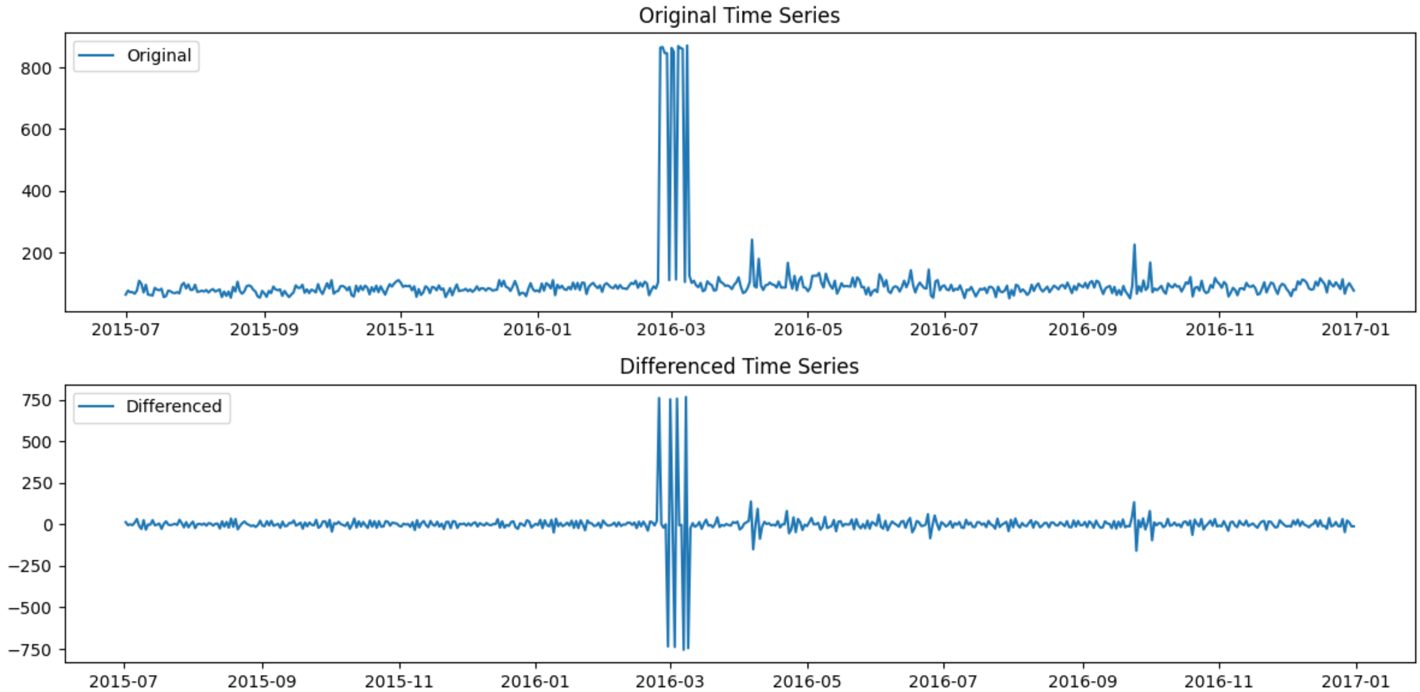
```
# Choose a specific column for differencing, for example, 'a'
column_for_differencing = 'a'

# Extract the time series data from the selected column
ts = ts_df[column_for_differencing]

# Perform differencing
ts_diff = ts.diff().dropna()

# Plot the original and differenced time series
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
plt.plot(ts, label='Original')
plt.legend(loc='upper left')
plt.title('Original Time Series')

plt.subplot(2, 1, 2)
plt.plot(ts_diff, label='Differenced')
plt.legend(loc='upper left')
plt.title('Differenced Time Series')

plt.tight_layout()
plt.show()
```



In [95]:

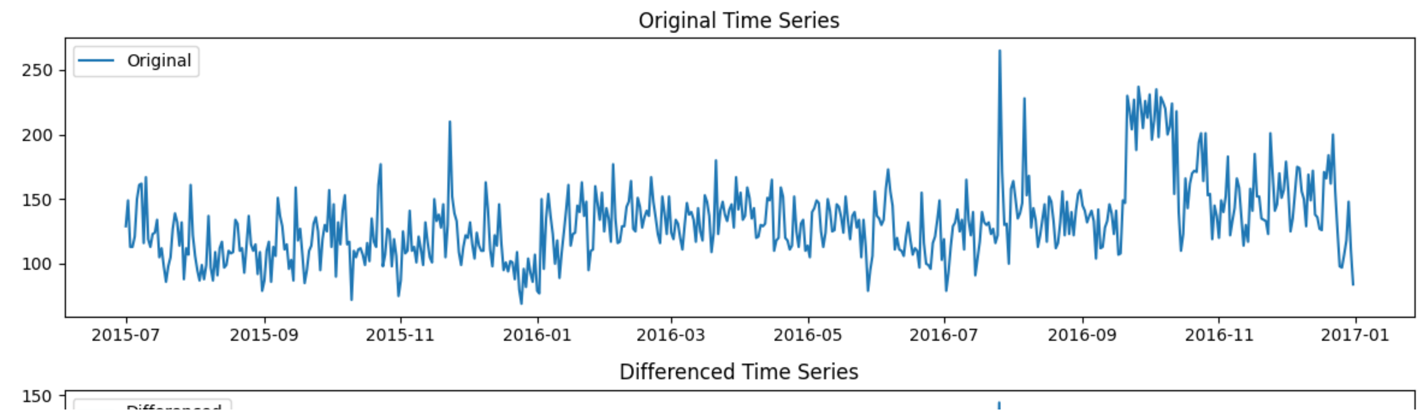
```
# Choose a specific column for differencing, for example, 'b'
column_for_differencing = 'b'

# Extract the time series data from the selected column
ts = ts_df[column_for_differencing]

# Perform differencing
ts_diff = ts.diff().dropna()

# Plot the original and differenced time series
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
plt.plot(ts, label='Original')
plt.legend(loc='upper left')
plt.title('Original Time Series')
```

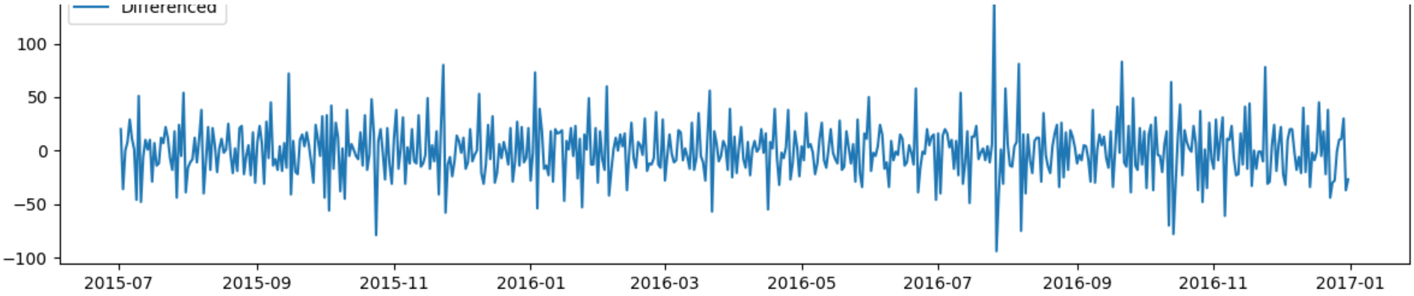
```
plt.subplot(2, 1, 2)
plt.plot(ts_diff, label='Differenced')
plt.legend(loc='upper left')
plt.title('Differenced Time Series')

plt.tight_layout()
plt.show()
```



In [96]:

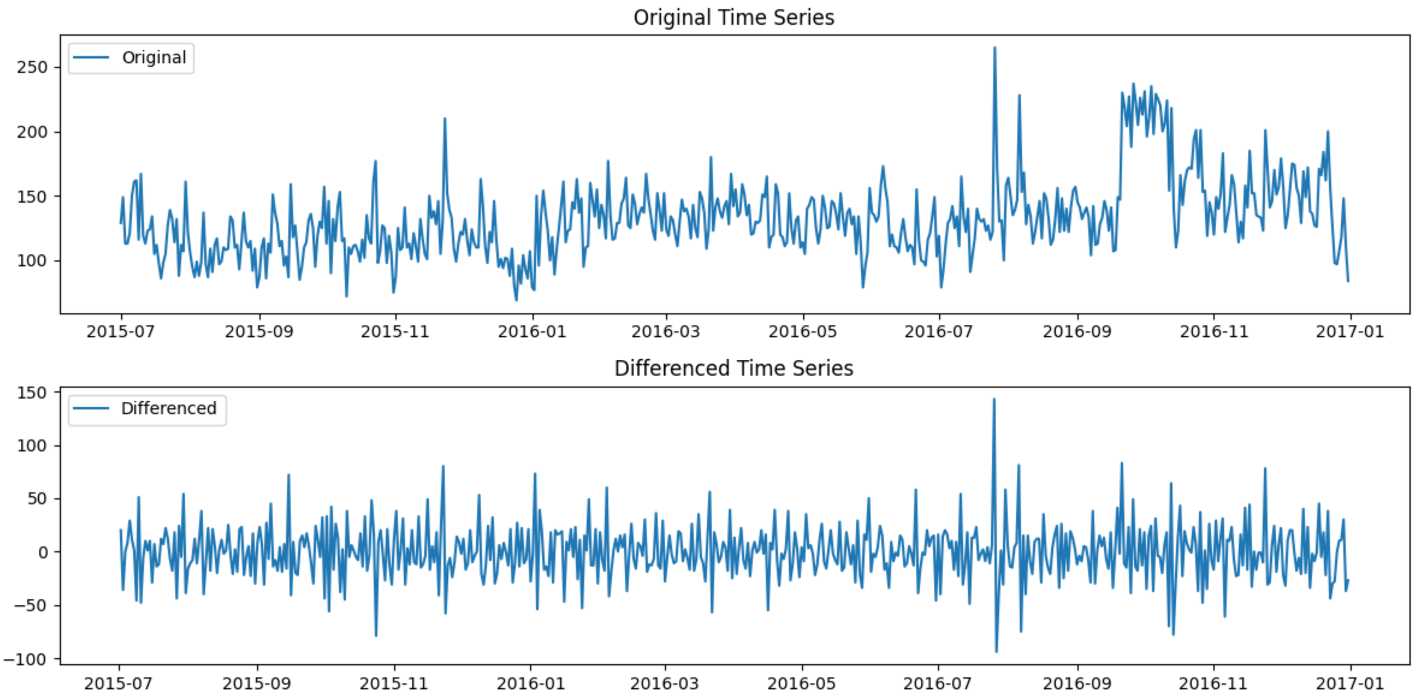
```
# Choose a specific column for differencing, for example, 'c'
column_for_differencing = 'c'

# Extract the time series data from the selected column
ts = ts_df[column_for_differencing]

# Perform differencing
ts_diff = ts.diff().dropna()

# Plot the original and differenced time series
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
plt.plot(ts, label='Original')
plt.legend(loc='upper left')
plt.title('Original Time Series')

plt.subplot(2, 1, 2)
plt.plot(ts_diff, label='Differenced')
plt.legend(loc='upper left')
plt.title('Differenced Time Series')

plt.tight_layout()
plt.show()
```





In [97]:

```
# Choose a specific column for differencing, for example, 'd'
column_for_differencing = 'd'

# Extract the time series data from the selected column
ts = ts_df[column_for_differencing]

# Perform differencing
ts_diff = ts.diff().dropna()

# Plot the original and differenced time series
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
plt.plot(ts, label='Original')
plt.legend(loc='upper left')
plt.title('Original Time Series')

plt.subplot(2, 1, 2)
plt.plot(ts_diff, label='Differenced')
plt.legend(loc='upper left')
plt.title('Differenced Time Series')

plt.tight_layout()
plt.show()
```

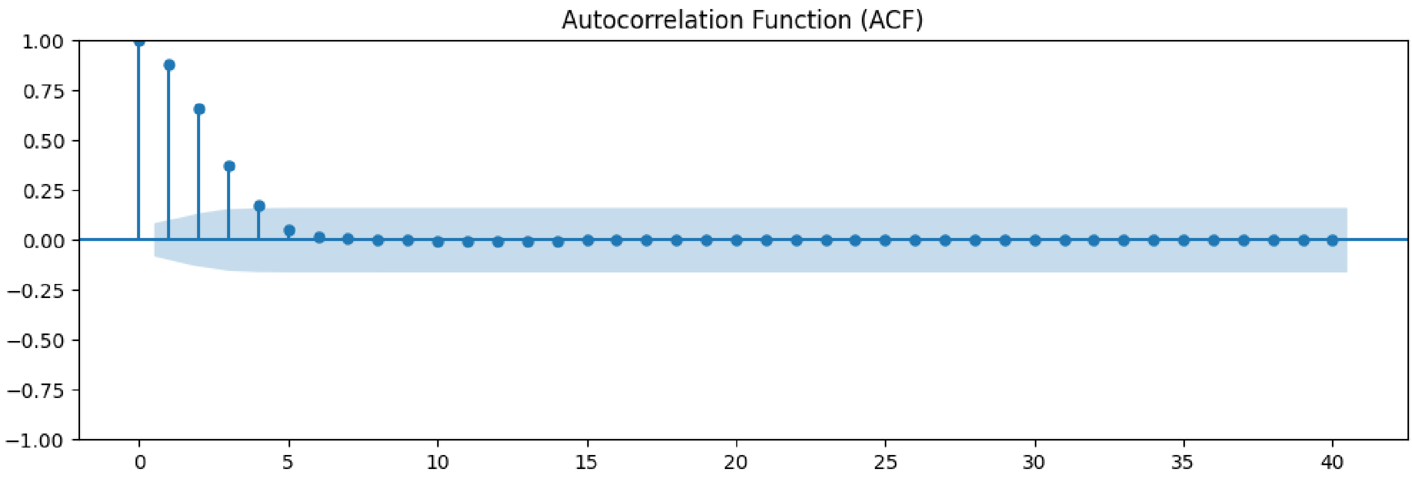


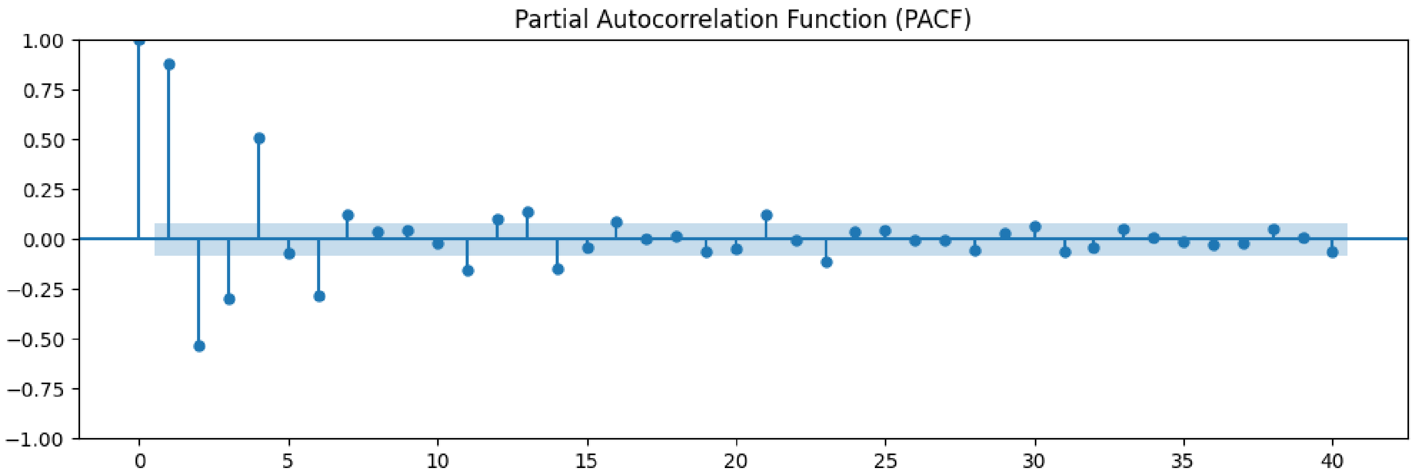
Plotting the ACF and PACF plots

In [98]:

```
import statsmodels.api as sm
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

# Choose a specific column for analysis, for example, 'a'
column_for_analysis = 'a'
```

```
# Extract the time series data from the selected column
ts = ts_df[column_for_analysis]

# Plot ACF and PACF
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 8))

# ACF plot
plot_acf(ts, lags=40, ax=ax1)
ax1.set_title('Autocorrelation Function (ACF)')

# PACF plot
plot_pacf(ts, lags=40, ax=ax2)
ax2.set_title('Partial Autocorrelation Function (PACF)')

plt.show()
```

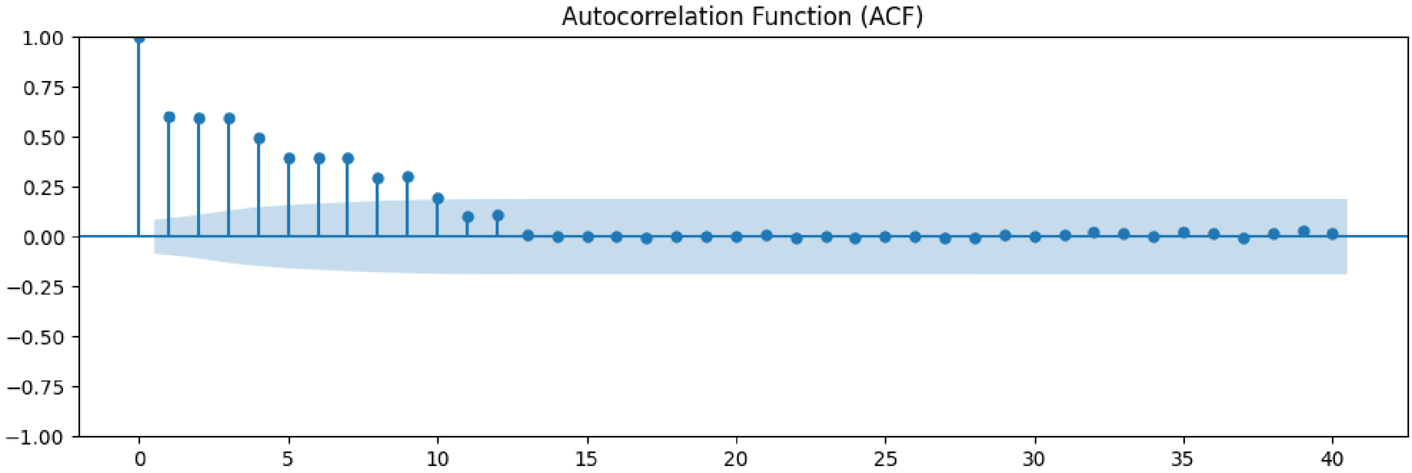


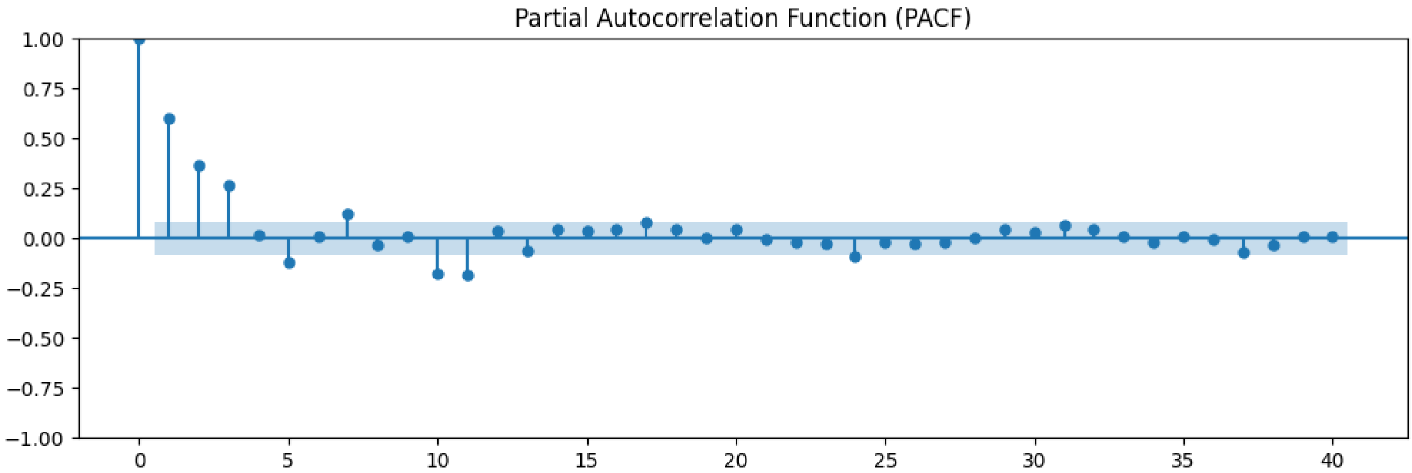


In [99]:

```
import statsmodels.api as sm
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

# Choose a specific column for analysis, for example, 'b'
column_for_analysis = 'b'

# Extract the time series data from the selected column
ts = ts_df[column_for_analysis]

# Plot ACF and PACF
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 8))

# ACF plot
plot_acf(ts, lags=40, ax=ax1)
ax1.set_title('Autocorrelation Function (ACF)')

# PACF plot
plot_pacf(ts, lags=40, ax=ax2)
ax2.set_title('Partial Autocorrelation Function (PACF)')
```

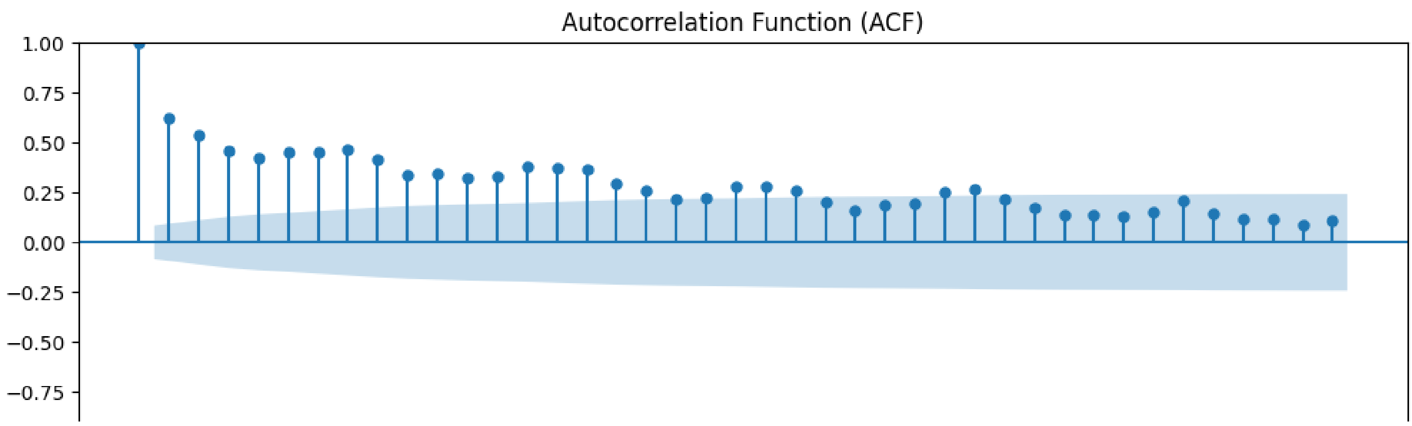
```
plt.show()
```

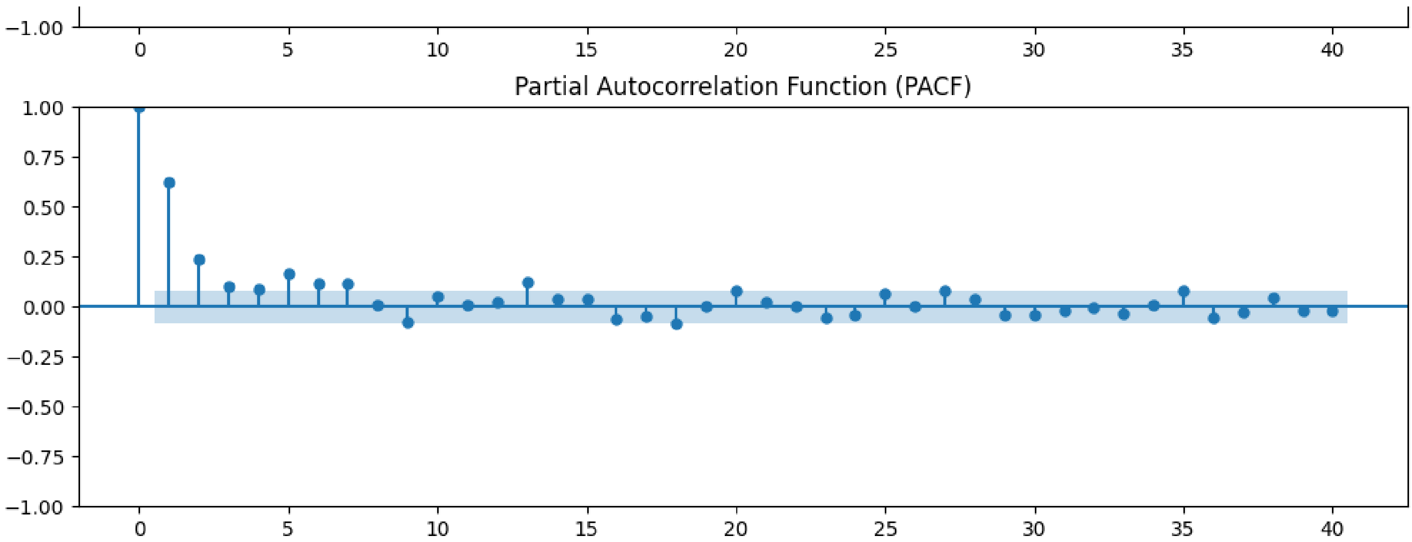




In [100]:

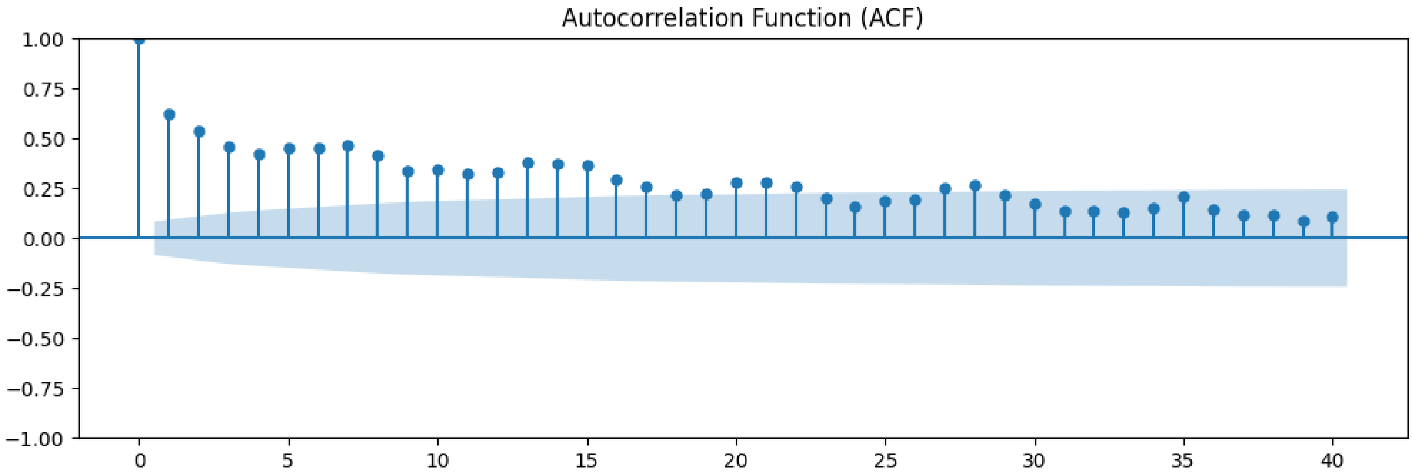
```
import statsmodels.api as sm
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

# Choose a specific column for analysis, for example, 'c'
column_for_analysis = 'c'

# Extract the time series data from the selected column
ts = ts_df[column_for_analysis]

# Plot ACF and PACF
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 8))

# ACF plot
plot_acf(ts, lags=40, ax=ax1)
ax1.set_title('Autocorrelation Function (ACF)')

# PACF plot
plot_pacf(ts, lags=40, ax=ax2)
ax2.set_title('Partial Autocorrelation Function (PACF)')

plt.show()
```

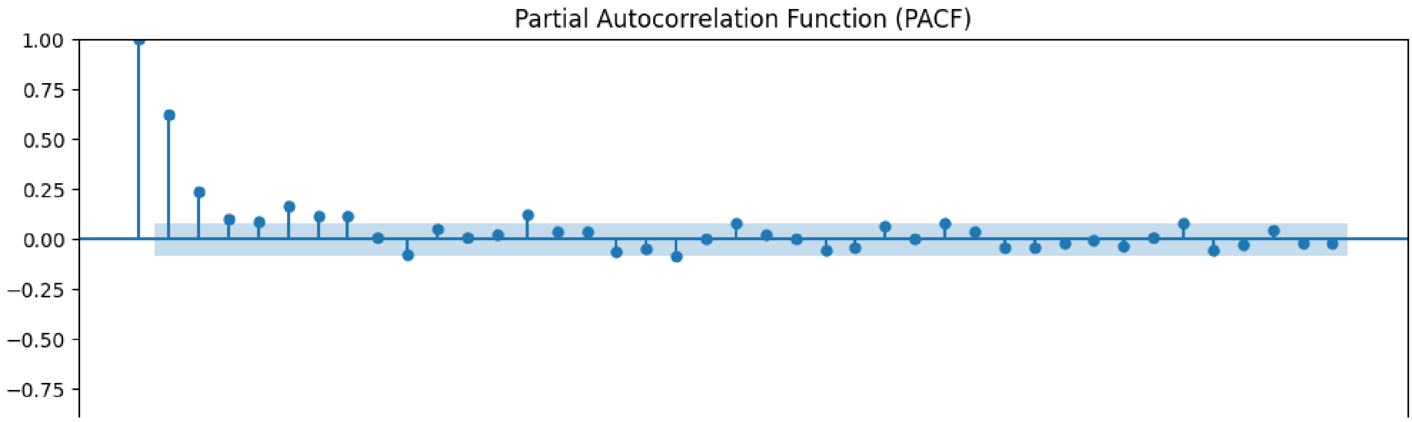


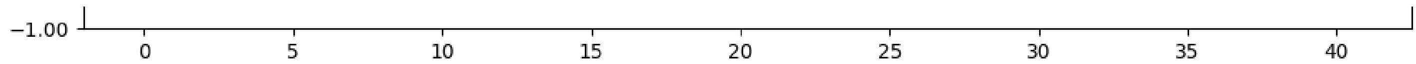


In [101]:

```python
import statsmodels.api as sm
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

# Choose a specific column for analysis, for example, 'd'
column_for_analysis = 'd'

# Extract the time series data from the selected column
ts = ts_df[column_for_analysis]

# Plot ACF and PACF
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 8))

# ACF plot
plot_acf(ts, lags=40, ax=ax1)
ax1.set_title('Autocorrelation Function (ACF)')

# PACF plot
plot_pacf(ts, lags=40, ax=ax2)
ax2.set_title('Partial Autocorrelation Function (PACF)')

plt.show()
```







ARIMA Model

In [102]:

```
from statsmodels.tsa.arima.model import ARIMA

# Choose a specific column for modeling, for example, 'a'
column_for_modeling = 'a'

n_forecast_steps = 30
# Extract the time series data from the selected column
ts = ts_df[column_for_modeling]

# Fit ARIMA model
order = (1, 1, 1)  # Replace with appropriate values based on model tuning
model_arima = ARIMA(ts, order=order)
result_arima = model_arima.fit()

# Make predictions
forecast_arima = result_arima.predict(start=len(ts), end=len(ts) + n_forecast_steps - 1,
typ='levels')
forecast_arima
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
```

Out[102]:

	predicted_mean
2017-01-01	9.371121
2017-01-02	10.033466
2017-01-03	10.353423
2017-01-04	10.507984
2017-01-05	10.582648
2017-01-06	10.618715
2017-01-07	10.636138
2017-01-08	10.644555
2017-01-09	10.648621
2017-01-10	10.650585
2017-01-11	10.651533
2017-01-12	10.651992
2017-01-13	10.652213
2017-01-14	10.652320
2017-01-15	10.652372
2017-01-16	10.652397
2017-01-17	10.652409
2017-01-18	10.652415
2017-01-19	10.652417

	predicted_mean
2017-01-20	10.652419
2017-01-21	10.652419
2017-01-22	10.652420
2017-01-23	10.652420
2017-01-24	10.652420
2017-01-25	10.652420
2017-01-26	10.652420
2017-01-27	10.652420
2017-01-28	10.652420
2017-01-29	10.652420
2017-01-30	10.652420

dtype: float64

In [103]:

```
from statsmodels.tsa.arima.model import ARIMA

# Choose a specific column for modeling, for example, 'b'
column_for_modeling = 'b'

n_forecast_steps = 30
# Extract the time series data from the selected column
ts = ts_df[column_for_modeling]

# Fit ARIMA model
order = (1, 1, 1)  # Replace with appropriate values based on model tuning
model_arima = ARIMA(ts, order=order)
result_arima = model_arima.fit()

# Make predictions
forecast_arima = result_arima.predict(start=len(ts), end=len(ts) + n_forecast_steps - 1,
typ='levels')
forecast_arima
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
```

Out[103]:

	predicted_mean
2017-01-01	86.691001
2017-01-02	85.165118
2017-01-03	85.405374
2017-01-04	85.367545
2017-01-05	85.373501
2017-01-06	85.372563
2017-01-07	85.372711
2017-01-08	85.372688
2017-01-09	85.372691
2017-01-10	85.372691

	predicted_mean
2017-01-11	85.372691
2017-01-12	85.372691
2017-01-13	85.372691
2017-01-14	85.372691
2017-01-15	85.372691
2017-01-16	85.372691
2017-01-17	85.372691
2017-01-18	85.372691
2017-01-19	85.372691
2017-01-20	85.372691
2017-01-21	85.372691
2017-01-22	85.372691
2017-01-23	85.372691
2017-01-24	85.372691
2017-01-25	85.372691
2017-01-26	85.372691
2017-01-27	85.372691
2017-01-28	85.372691
2017-01-29	85.372691
2017-01-30	85.372691

dtype: float64

In [104]:

```
from statsmodels.tsa.arima.model import ARIMA

# Choose a specific column for modeling, for example, 'c'
column_for_modeling = 'c'

n_forecast_steps = 30
# Extract the time series data from the selected column
ts = ts_df[column_for_modeling]

# Fit ARIMA model
order = (1, 1, 1)  # Replace with appropriate values based on model tuning
model_arima = ARIMA(ts, order=order)
result_arima = model_arima.fit()

# Make predictions
forecast_arima = result_arima.predict(start=len(ts), end=len(ts) + n_forecast_steps - 1,
typ='levels')
forecast_arima
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
```

Out[104]:

	predicted_mean
2017-01-01	110.945646

2017-01-02	predicted_mean 120.281565
2017-01-03	120.278760
2017-01-04	120.822493
2017-01-05	120.970524
2017-01-06	121.010825
2017-01-07	121.021797
2017-01-08	121.024785
2017-01-09	121.025598
2017-01-10	121.025819
2017-01-11	121.025879
2017-01-12	121.025896
2017-01-13	121.025900
2017-01-14	121.025902
2017-01-15	121.025902
2017-01-16	121.025902
2017-01-17	121.025902
2017-01-18	121.025902
2017-01-19	121.025902
2017-01-20	121.025902
2017-01-21	121.025902
2017-01-22	121.025902
2017-01-23	121.025902
2017-01-24	121.025902
2017-01-25	121.025902
2017-01-26	121.025902
2017-01-27	121.025902
2017-01-28	121.025902
2017-01-29	121.025902
2017-01-30	121.025902

dtype: float64

In [105]:

```
from statsmodels.tsa.arima.model import ARIMA

# Choose a specific column for modeling, for example, 'd'
column_for_modeling = 'd'

n_forecast_steps = 30
# Extract the time series data from the selected column
ts = ts_df[column_for_modeling]

# Fit ARIMA model
order = (1, 1, 1)  # Replace with appropriate values based on model tuning
model_arima = ARIMA(ts, order=order)
result_arima = model_arima.fit()

# Make predictions
forecast_arima = result_arima.predict(start=len(ts), end=len(ts) + n_forecast_steps - 1,
typ='levels')
forecast_arima
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
```

```
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
```

Out[105]:

	predicted_mean
2017-01-01	110.945646
2017-01-02	118.281565
2017-01-03	120.278760
2017-01-04	120.822493
2017-01-05	120.970524
2017-01-06	121.010825
2017-01-07	121.021797
2017-01-08	121.024785
2017-01-09	121.025598
2017-01-10	121.025819
2017-01-11	121.025879
2017-01-12	121.025896
2017-01-13	121.025900
2017-01-14	121.025902
2017-01-15	121.025902
2017-01-16	121.025902
2017-01-17	121.025902
2017-01-18	121.025902
2017-01-19	121.025902
2017-01-20	121.025902
2017-01-21	121.025902
2017-01-22	121.025902
2017-01-23	121.025902
2017-01-24	121.025902
2017-01-25	121.025902
2017-01-26	121.025902
2017-01-27	121.025902
2017-01-28	121.025902
2017-01-29	121.025902
2017-01-30	121.025902

dtype: float64

SARIMAX Model with Exogenous Variable:

In [106]:

```
from statsmodels.tsa.statespace.sarimax import SARIMAX
```

```
# Choose a specific column for modeling, for example, 'a'
column_for_modeling = 'a'

# Extract the time series data from the selected column
ts = ts_df[column_for_modeling]

# Choose an exogenous variable, for example, 'Exog'
exog_variable = ts_df['Exog']

# Fit SARIMAX model with exogenous variable
order = (1, 1, 1)  # Replace with appropriate values based on model tuning
model_sarimax = SARIMAX(ts, exog=exog_variable, order=order)
result_sarimax = model_sarimax.fit()

# Make predictions
forecast_sarimax = result_sarimax.get_forecast(steps=n_forecast_steps, exog=exog_variable
[-n_forecast_steps:])
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
```

Facebook Prophet:

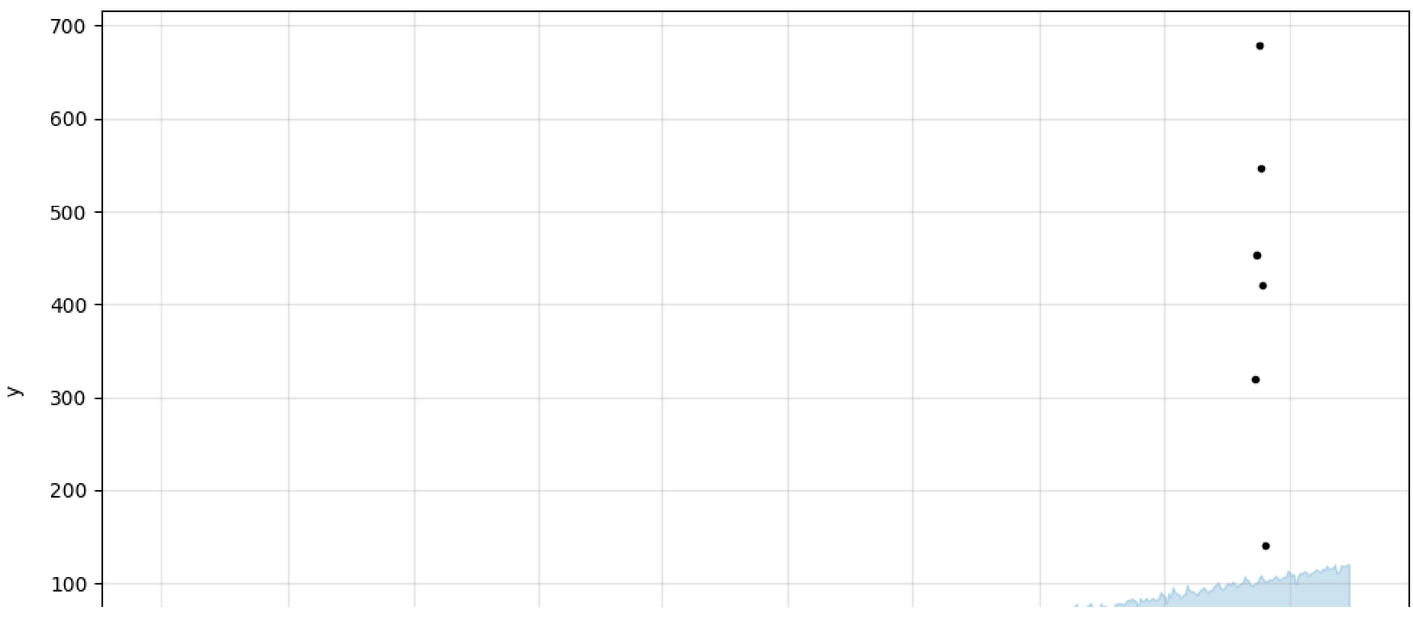
In [111]:

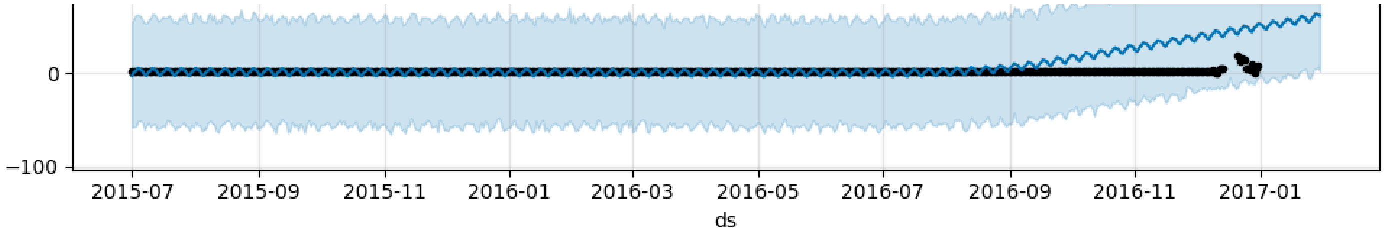
```
!pip install prophet
from prophet import Prophet # Import the Prophet class from the prophet library

# Choose a specific column for modeling, for example, 'a'
column_for_modeling = 'a'

# Assuming '2015-07-01' is the first date column in your DataFrame, we create a new 'ds'
column:
ts_df['ds'] = ts_df.index  # Assuming ts_df has a DatetimeIndex. If not, adjust according
ly.
# If ts_df index is not a DatetimeIndex, you may need to convert it:
# ts_df['ds'] = pd.to_datetime(ts_df.index)

# Extract the time series data and exogenous variable from the selected columns
ts = ts_df[['ds', column_for_modeling, 'Exog']].rename(columns={'ds': 'ds', column_for_m
odeling: 'y', 'Exog': 'exog_variable'})

# Fit Prophet model with exogenous variable
model_prophet = Prophet()
model_prophet.add_regressor('exog_variable')  # Add the exogenous variable

# Fit the model
model_prophet.fit(ts)

# Create a dataframe for future dates and exogenous variable
future_prophet = model_prophet.make_future_dataframe(periods=n_forecast_steps)
future_prophet['exog_variable'] = ts_df['Exog'].values[-1]  # Use the last known value f
or the exogenous variable

# Make predictions
forecast_prophet = model_prophet.predict(future_prophet)

# Plot the forecast
fig = model_prophet.plot(forecast_prophet)
```

```
Requirement already satisfied: prophet in /usr/local/lib/python3.10/dist-packages (1.1.6)
Requirement already satisfied: cmdstanpy>=1.0.4 in /usr/local/lib/python3.10/dist-package
s (from prophet) (1.2.4)
Requirement already satisfied: numpy>=1.15.4 in /usr/local/lib/python3.10/dist-packages (
from prophet) (1.26.4)
```

```
Requirement already satisfied: matplotlib>=2.0.0 in /usr/local/lib/python3.10/dist-packag
es (from prophet) (3.7.1)
Requirement already satisfied: pandas>=1.0.4 in /usr/local/lib/python3.10/dist-packages (
from prophet) (2.2.2)
Requirement already satisfied: holidays<1,>=0.25 in /usr/local/lib/python3.10/dist-packag
es (from prophet) (0.57)
Requirement already satisfied: tqdm>=4.36.1 in /usr/local/lib/python3.10/dist-packages (f
rom prophet) (4.66.5)
Requirement already satisfied: importlib-resources in /usr/local/lib/python3.10/dist-pack
ages (from prophet) (6.4.5)
Requirement already satisfied: stanio<2.0.0,>=0.4.0 in /usr/local/lib/python3.10/dist-pac
kages (from cmdstanpy>=1.0.4->prophet) (0.5.1)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.10/dist-packages
(from holidays<1,>=0.25->prophet) (2.8.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-package
s (from matplotlib>=2.0.0->prophet) (1.3.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (f
rom matplotlib>=2.0.0->prophet) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packag
es (from matplotlib>=2.0.0->prophet) (4.54.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packag
es (from matplotlib>=2.0.0->prophet) (1.4.7)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages
(from matplotlib>=2.0.0->prophet) (24.1)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (
from matplotlib>=2.0.0->prophet) (10.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-package
s (from matplotlib>=2.0.0->prophet) (3.1.4)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (f
rom pandas>=1.0.4->prophet) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages
(from pandas>=1.0.4->prophet) (2024.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from
python-dateutil->holidays<1,>=0.25->prophet) (1.16.0)
```

```
INFO:prophet:Disabling yearly seasonality. Run prophet with yearly_seasonality=True to ov
erride this.
INFO:prophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to over
ride this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpff74hmeo/wi65supt.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpff74hmeo/88igyrmi.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.10/dist-packages/prophet/stan_mode
l/prophet_model.bin', 'random', 'seed=9395', 'data', 'file=/tmp/tmpff74hmeo/wi65supt.json
', 'init=/tmp/tmpff74hmeo/88igyrmi.json', 'output', 'file=/tmp/tmpff74hmeo/prophet_modelx
f_0p83e/prophet_model-20241009082126.csv', 'method=optimize', 'algorithm=lbfgs', 'iter=10
000']
08:21:26 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
08:21:26 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
```





Finding a way(grid search / etc) to find the best params for at least 1 modeling approach.

In [112]:

```
import itertools
import numpy as np
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_squared_error

# Choose a specific column for modeling, for example, 'a'
column_for_modeling = 'a'

# Extract the time series data from the selected column
ts = ts_df[column_for_modeling]

# Define the range of values for p, d, and q
p_values = range(0, 3)  # Adjust the range based on your requirements
d_values = range(0, 2)  # Adjust the range based on your requirements
q_values = range(0, 3)  # Adjust the range based on your requirements

# Generate all possible combinations of p, d, and q
param_combinations = list(itertools.product(p_values, d_values, q_values))

# Initialize variables to store the best parameters and corresponding MSE
best_params = None
best_mse = np.inf

# Perform grid search
for params in param_combinations:
    try:
        # Fit ARIMA model with current parameters
        model = ARIMA(ts, order=params)
        result = model.fit()

        # Make predictions
        predictions = result.predict(start=len(ts), end=len(ts) + n_forecast_steps - 1,
typ='levels')

        # Calculate Mean Squared Error (MSE)
        mse = mean_squared_error(ts[-n_forecast_steps:], predictions)

        # Update best parameters if the current MSE is lower
        if mse < best_mse:
            best_mse = mse
            best_params = params

    except Exception as e:
        # Handle exceptions if the model fails to converge
        print(f"Error for parameters {params}: {e}")

# Display the best parameters
print(f"Best Parameters: {best_params}")
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
```

```
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
```

```
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
```

```
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
```

```
Best Parameters: (1, 1, 2)
```

In [113]:

```
# Choose a specific column for modeling, for example, 'b'
column_for_modeling = 'b'

# Extract the time series data from the selected column
ts = ts_df[column_for_modeling]

# Define the range of values for p, d, and q
p_values = range(0, 3)  # Adjust the range based on your requirements
d_values = range(0, 2)  # Adjust the range based on your requirements
q_values = range(0, 3)  # Adjust the range based on your requirements

# Generate all possible combinations of p, d, and q
param_combinations = list(itertools.product(p_values, d_values, q_values))

# Initialize variables to store the best parameters and corresponding MSE
best_params = None
best_mse = np.inf

# Perform grid search
for params in param_combinations:
    try:
        # Fit ARIMA model with current parameters
        model = ARIMA(ts, order=params)
        result = model.fit()

        # Make predictions
        predictions = result.predict(start=len(ts), end=len(ts) + n_forecast_steps - 1,
typ='levels')

        # Calculate Mean Squared Error (MSE)
        mse = mean_squared_error(ts[-n_forecast_steps:], predictions)

        # Update best parameters if the current MSE is lower
        if mse < best_mse:
            best_mse = mse
            best_params = params

    except Exception as e:
        # Handle exceptions if the model fails to converge
        print(f"Error for parameters {params}: {e}")

# Display the best parameters
print(f"Best Parameters: {best_params}")
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
```

```
Best Parameters: (1, 0, 1)
```

In [114]:

```python
# Choose a specific column for modeling, for example, 'c'
column_for_modeling = 'c'

# Extract the time series data from the selected column
ts = ts_df[column_for_modeling]

# Define the range of values for p, d, and q
p_values = range(0, 3)  # Adjust the range based on your requirements
d_values = range(0, 2)  # Adjust the range based on your requirements
q_values = range(0, 3)  # Adjust the range based on your requirements

# Generate all possible combinations of p, d, and q
param_combinations = list(itertools.product(p_values, d_values, q_values))

# Initialize variables to store the best parameters and corresponding MSE
best_params = None
best_mse = np.inf

# Perform grid search
for params in param_combinations:
    try:
        # Fit ARIMA model with current parameters
        model = ARIMA(ts, order=params)
        result = model.fit()

        # Make predictions
        predictions = result.predict(start=len(ts), end=len(ts) + n_forecast_steps - 1,
typ='levels')

        # Calculate Mean Squared Error (MSE)
        mse = mean_squared_error(ts[-n_forecast_steps:], predictions)

        # Update best parameters if the current MSE is lower
        if mse < best_mse:
            best_mse = mse
            best_params = params

    except Exception as e:
        # Handle exceptions if the model fails to converge
        print(f"Error for parameters {params}: {e}")

# Display the best parameters
print(f"Best Parameters: {best_params}")
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
```

```
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
```

```
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
```

```
  self._init_dates(dates, freq)
```

```
Best Parameters: (0, 0, 0)
```

In [115]:

```python
# Choose a specific column for modeling, for example, 'd'
column_for_modeling = 'd'

# Extract the time series data from the selected column
ts = ts_df[column_for_modeling]

# Define the range of values for p, d, and q
p_values = range(0, 3)  # Adjust the range based on your requirements
d_values = range(0, 2)  # Adjust the range based on your requirements
q_values = range(0, 3)  # Adjust the range based on your requirements

# Generate all possible combinations of p, d, and q
param_combinations = list(itertools.product(p_values, d_values, q_values))

# Initialize variables to store the best parameters and corresponding MSE
best_params = None
best_mse = np.inf

# Perform grid search
for params in param_combinations:
    try:
        # Fit ARIMA model with current parameters
        model = ARIMA(ts, order=params)
        result = model.fit()

        # Make predictions
        predictions = result.predict(start=len(ts), end=len(ts) + n_forecast_steps - 1,
typ='levels')

        # Calculate Mean Squared Error (MSE)
        mse = mean_squared_error(ts[-n_forecast_steps:], predictions)

        # Update best parameters if the current MSE is lower
        if mse < best_mse:
            best_mse = mse
            best_params = params

    except Exception as e:
        # Handle exceptions if the model fails to converge
        print(f"Error for parameters {params}: {e}")

# Display the best parameters
print(f"Best Parameters: {best_params}")
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
```

```
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
```

```
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarni
ng: No frequency information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
```

```
Best Parameters: (0, 0, 0)
```

In [ ]: