In [1]:
```python
# Import Neeeded Liberaries:
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import plotly.express as px
import plotly.graph_objects as go

from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.graphics.tsaplots import plot_pacf
from statsmodels.tsa.arima_model import ARIMA
import statsmodels.api as sm
```

In [2]:
```python
# Import Our Data
data = pd.read_csv("Thecleverprogrammer.csv")
data.head()
```

Out[2]:

	Date	Views
0	01/06/2021	7831
1	02/06/2021	7798
2	03/06/2021	7401
3	04/06/2021	7054
4	05/06/2021	7973

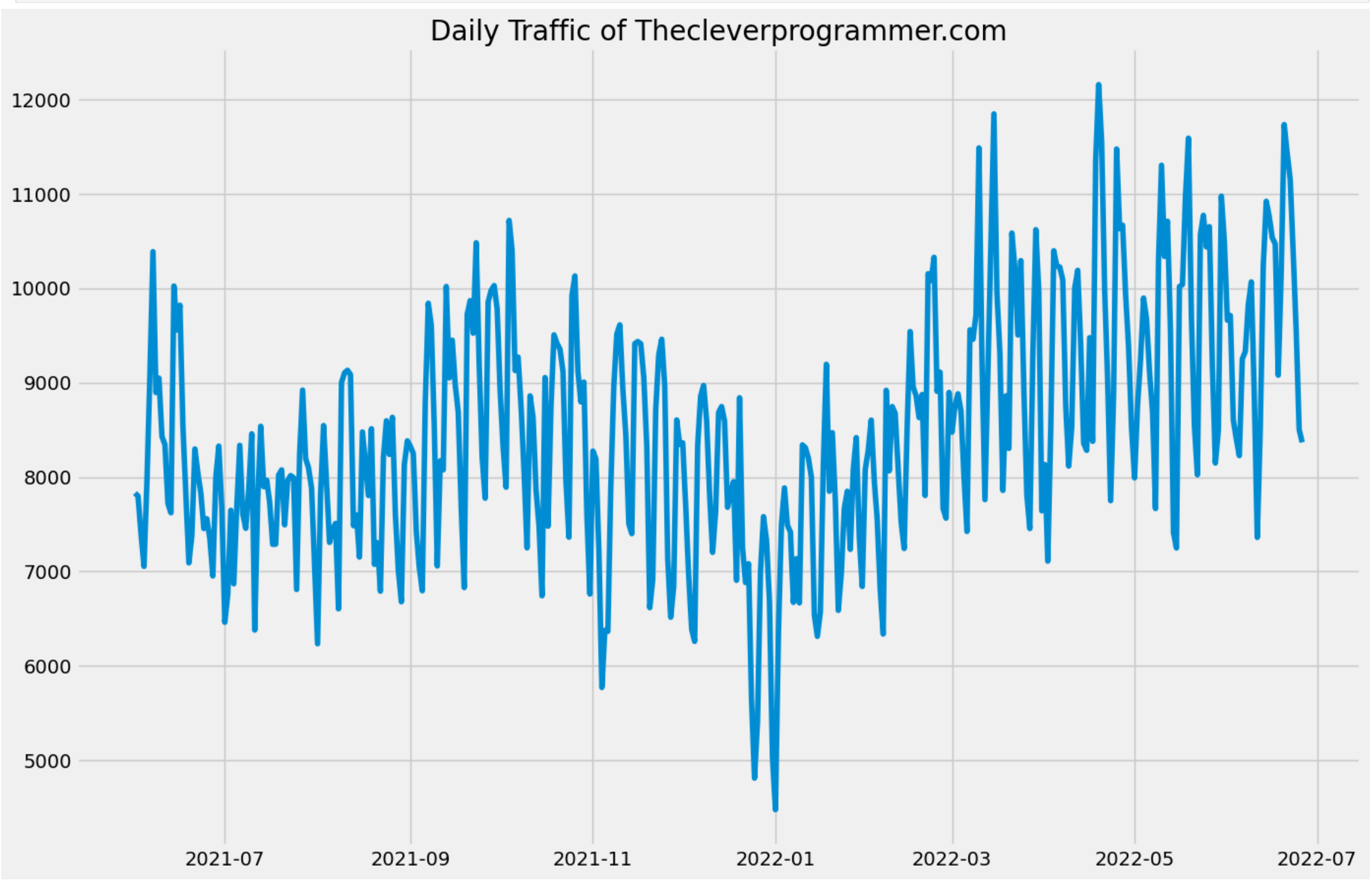
In [3]:
```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 391 entries, 0 to 390
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Date    391 non-null    object
 1   Views   391 non-null    int64
dtypes: int64(1), object(1)
memory usage: 6.2+ KB
```

In [4]:
```python
# We need to convert Date column to datetime instead of object:
data["Date"] = pd.to_datetime(data["Date"], format = "%d/%m/%Y" )
data.info()
```

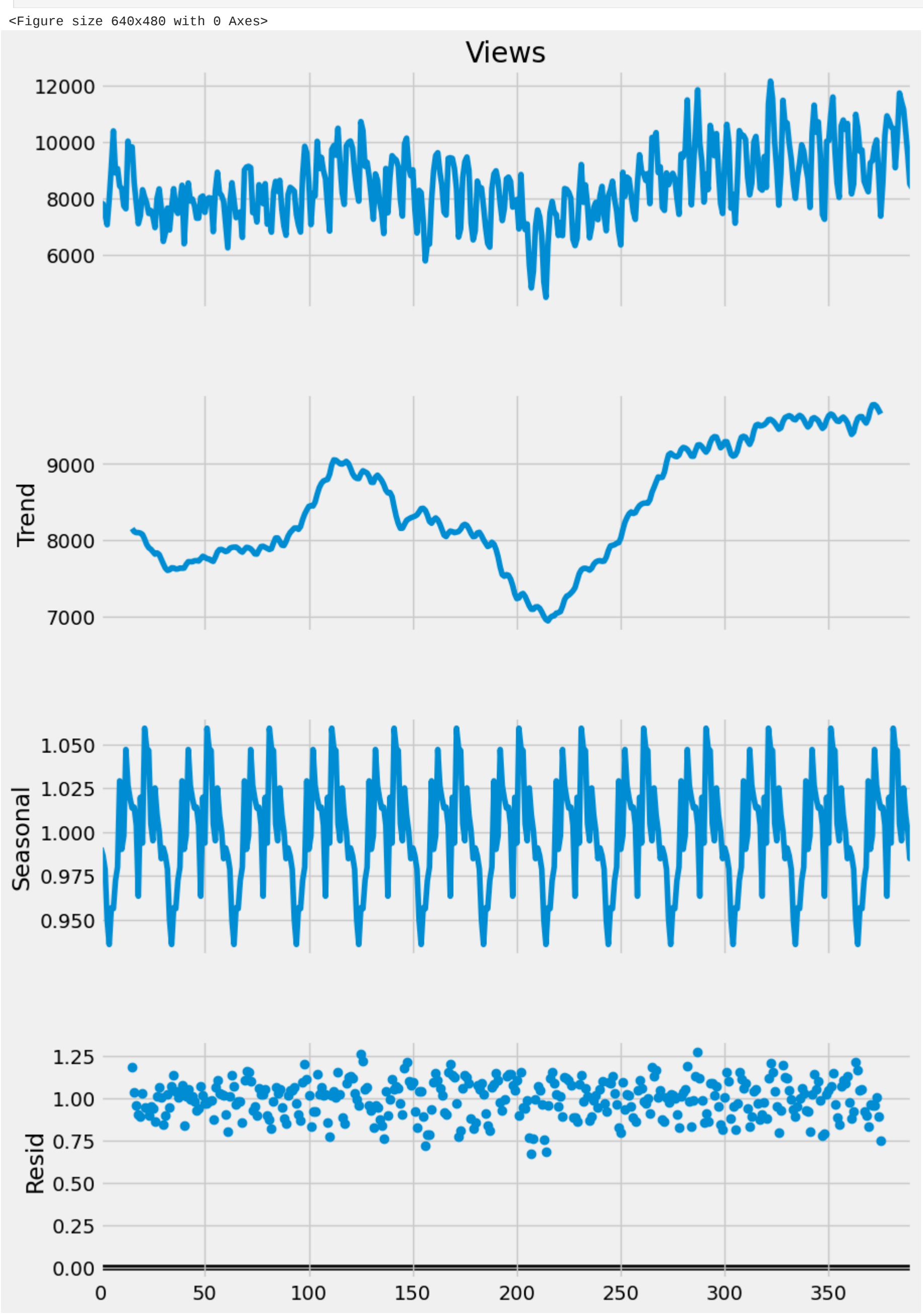
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 391 entries, 0 to 390
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Date    391 non-null    datetime64[ns]
 1   Views   391 non-null    int64
dtypes: datetime64[ns](1), int64(1)
memory usage: 6.2 KB
```

In [5]:
```python
# Now lets take a look at the daily website traffic:
plt.style.use("fivethirtyeight")
plt.figure(figsize = (15,10))
plt.plot(data["Date"], data["Views"])
plt.title("Daily Traffic of Thecleverprogrammer.com")
plt.show()
```



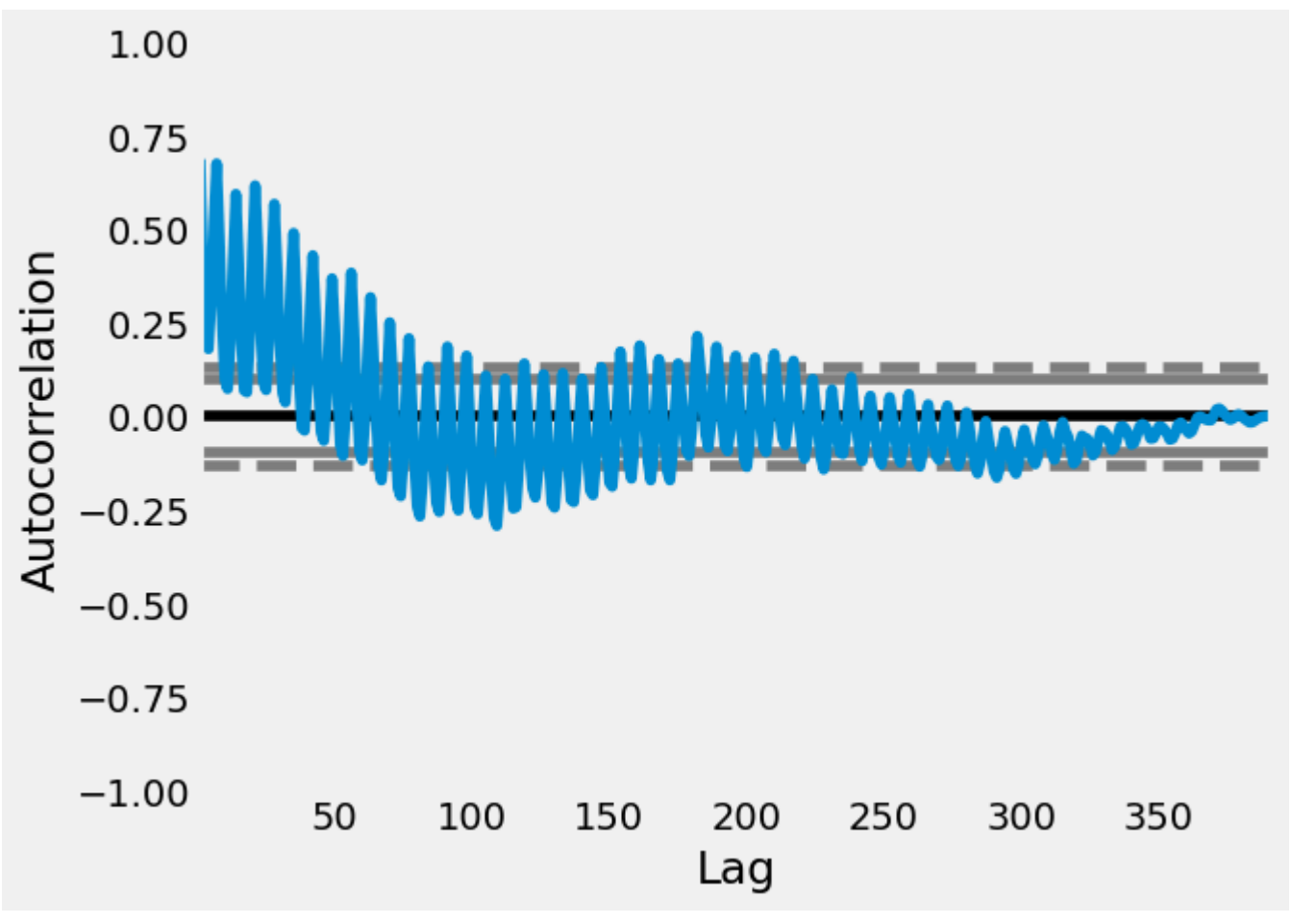
In [7]:
```python
# Lets see if our data is seasonal or stationary:

result = seasonal_decompose(data["Views"], model="multiplicative", period = 30)
fig = plt.figure()
fig = result.plot()
fig.set_size_inches(10,15)
```

```
<Figure size 640x480 with 0 Axes>
```



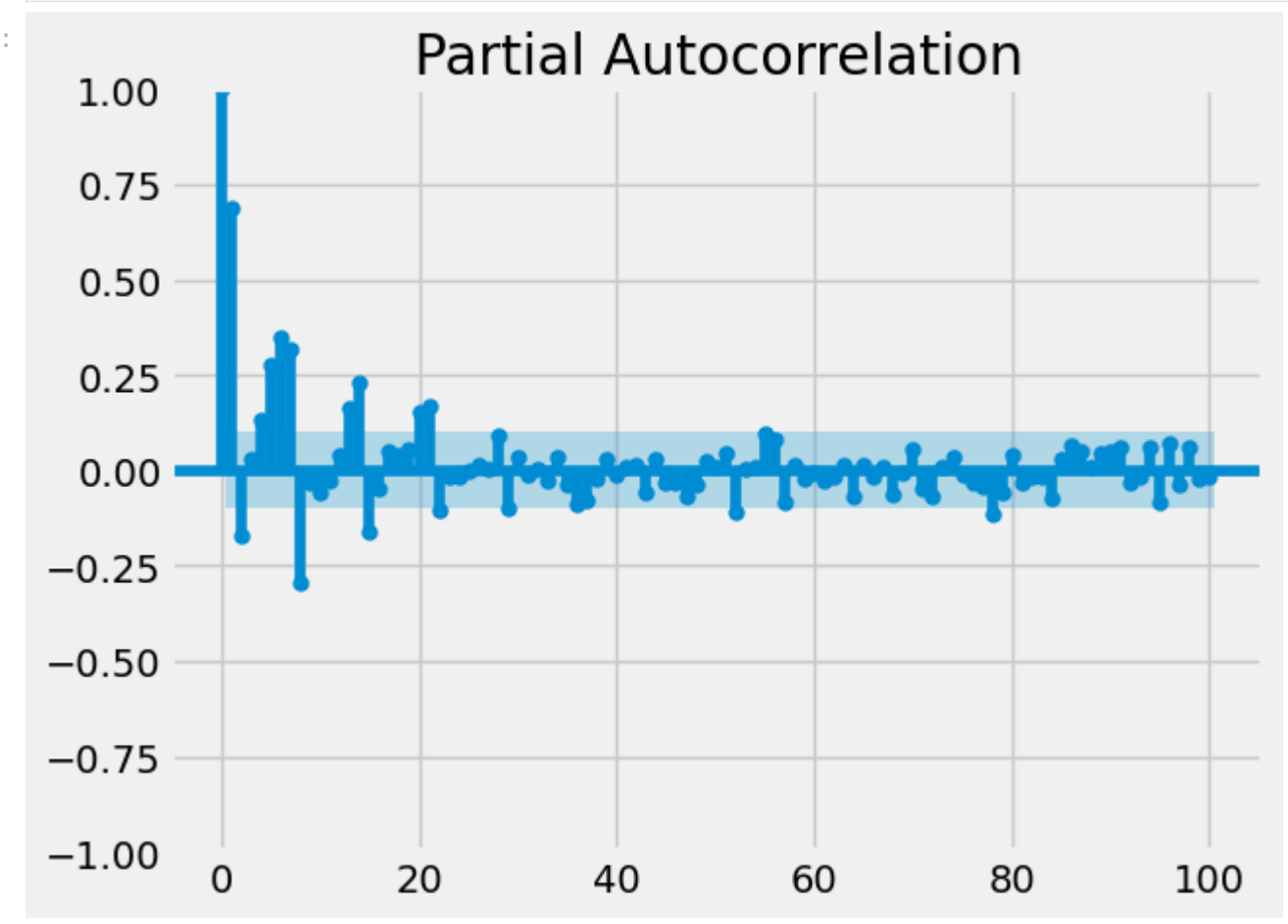
In [8]:
```python
# As we confirmed that our data is seasonal we will use SARIMA model to predict our traffic:
# As we will use SARIMA model, we should get the q,d,p values.
# Our data is not stationary so d=1
# we will find q and p using autocorrelation and partial autocorrelation

# Autocorrelation:
pd.plotting.autocorrelation_plot(data["Views"])
```

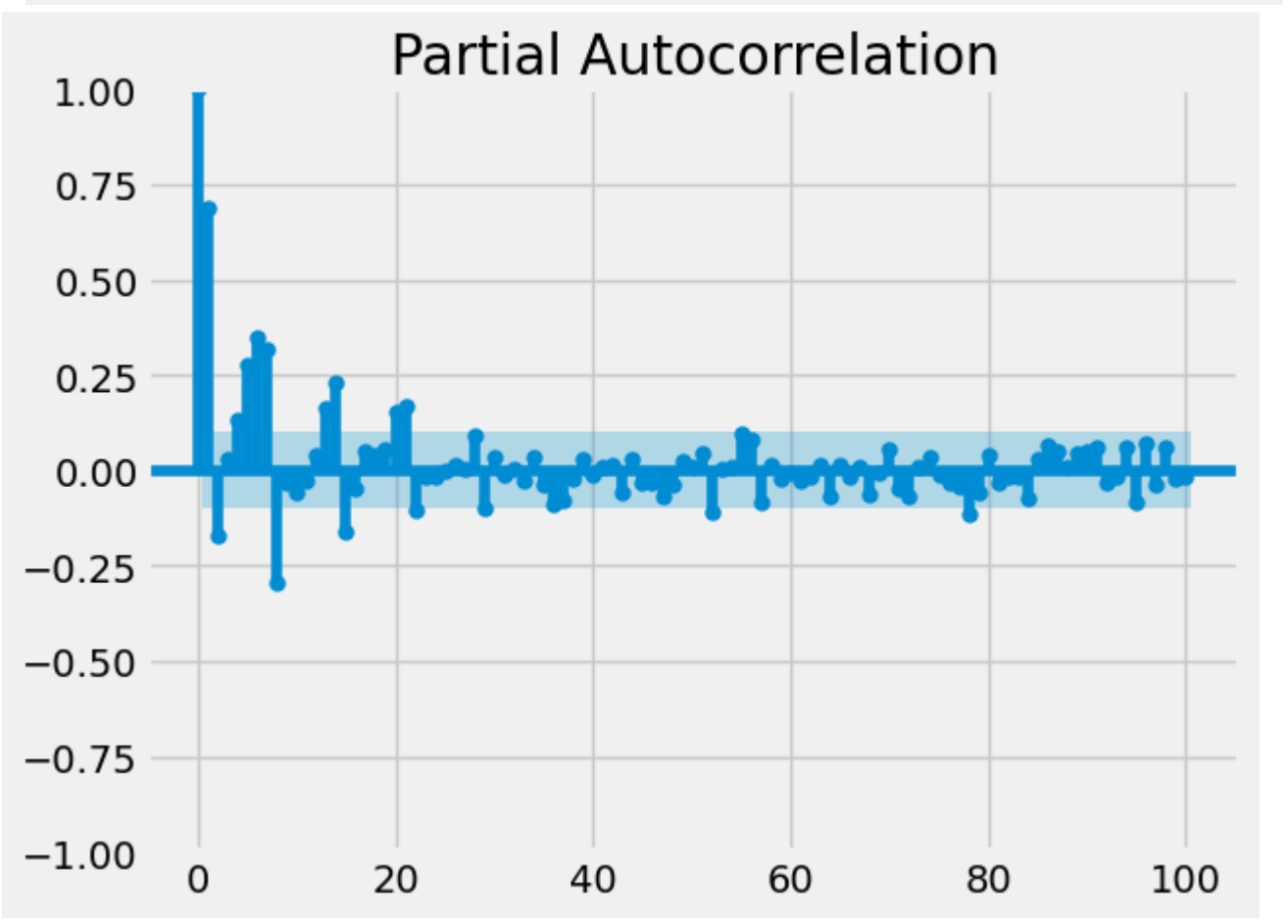
Out[8]: `<Axes: xlabel='Lag', ylabel='Autocorrelation'>`



In [9]:
```python
# partial autocorrelation:
plot_pacf(data["Views"], lags = 100)
```

Out[9]:





In [10]:
```python
# Now lets build our model SARIMA:

p,d,q = 5,1,2
model = sm.tsa.statespace.SARIMAX(data["Views"], order = (p,d,q), seasonal_order = (p,d,q,12))
model = model.fit()
print(model.summary())
```

```
C:\Users\hytham2022\anaconda3\lib\site-packages\statsmodels\tsa\statespace\sarimax.py:978: UserWarning: Non-invertible starting MA parameters foun
d. Using zeros as starting parameters.
  warn('Non-invertible starting MA parameters found.'
C:\Users\hytham2022\anaconda3\lib\site-packages\statsmodels\base\model.py:607: ConvergenceWarning: Maximum Likelihood optimization failed to conver
ge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
                                     SARIMAX Results
==========================================================================================
Dep. Variable:                                Views   No. Observations:                391
Model:             SARIMAX(5, 1, 2)x(5, 1, 2, 12)   Log Likelihood               -3098.830
Date:                              Thu, 18 Apr 2024   AIC                           6227.660
Time:                                      20:53:25   BIC                           6286.683
Sample:                                           0   HQIC                          6251.085
                                              - 391
Covariance Type:                                opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.7747      0.130      5.976      0.000       0.521       1.029
ar.L2         -0.7883      0.136     -5.799      0.000      -1.055      -0.522
ar.L3         -0.1346      0.172     -0.782      0.434      -0.472       0.203
ar.L4         -0.2057      0.153     -1.344      0.179      -0.505       0.094
ar.L5         -0.1355      0.137     -0.992      0.321      -0.403       0.132
ma.L1         -1.1859      0.084    -14.057      0.000      -1.351      -1.021
ma.L2          0.9019      0.072     12.527      0.000       0.761       1.043
ar.S.L12      -0.2351      4.354     -0.054      0.957      -8.769       8.299
ar.S.L24       0.0621      0.542      0.115      0.909      -1.001       1.125
ar.S.L36      -0.1748      0.297     -0.588      0.556      -0.757       0.407
ar.S.L48      -0.2074      0.858     -0.242      0.809      -1.888       1.474
ar.S.L60       0.0115      0.888      0.013      0.990      -1.728       1.751
ma.S.L12      -0.6998      4.355     -0.161      0.872      -9.235       7.835
ma.S.L24      -0.1189      3.571     -0.033      0.973      -7.117       6.879
sigma2      1.257e+06    1.7e-05   7.39e+10      0.000    1.26e+06    1.26e+06
===================================================================================
Ljung-Box (L1) (Q):                   0.02   Jarque-Bera (JB):                 1.12
Prob(Q):                              0.90   Prob(JB):                         0.57
Heteroskedasticity (H):               1.06   Skew:                             0.13
Prob(H) (two-sided):                  0.76   Kurtosis:                         2.99
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
[2] Covariance matrix is singular or near-singular, with condition number 2.72e+27. Standard errors may be unstable.
```

In [11]:
```python
# Lets predict the upcoming 50 days of traffic:
predictions = model.predict(len(data), len(data)+50)
print(predictions)
```

```
391     9881.755634
392    10788.545815
393    10723.396655
394     9854.099484
395     8798.029464
396     8259.835914
397     8939.196672
398     9714.369114
399    10295.288296
400    10591.685924
401     9877.300350
402     9329.093264
403     9005.973829
404     9064.958141
405    10495.216695
406    10997.610314
407    10914.724320
408    10081.863120
409     9413.067634
410     8609.592770
411     9166.192866
412    10346.612552
413    10640.452779
414    10819.166572
415    10264.626031
416     9399.331790
417     8968.693521
418     9153.516869
419     9927.139866
420    10276.011646
421    10748.584615
422     9903.665388
423     9501.354336
424     8982.879396
425     8846.918600
426    10178.429295
427    10895.992374
428    10926.192056
429    10367.926406
430     9402.269446
431     8667.510953
432     8710.540332
433    10091.048074
434    10564.548935
435    10889.959214
436    10439.021199
437     9302.844995
438     9151.422690
439     9365.257904
440    10324.000955
441    11190.458112
Name: predicted_mean, dtype: float64
```

In [12]:
```python
# Lets visualize the predictions:
data["Views"].plot(legend = True, label="Training Data", figsize=(15,10))
predictions.plot(legend= True, label="Predictions")
```

Out[12]: `<Axes: >`