Forecast Cashflow for any five companies till the next 4 data points and plot the forecasted line graph with historical data also please explain your forecasted result using prescriptive analytics.

In [ ]:

In [334]:
```python
# Importing libraries
import os
import warnings
warnings.filterwarnings('ignore')
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
# Above is a special style template for matplotlib, highly useful for visualizing time series data
%matplotlib inline
from pylab import rcParams

import statsmodels.api as sm
from numpy.random import normal, seed
from scipy.stats import norm
from statsmodels.tsa.arima_model import ARMA
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.arima_process import ArmaProcess

import math
from sklearn.metrics import mean_squared_error
import yfinance as yf
import matplotlib as plt
#Import the libraries
import math
```

In [335]:
```python
#getting the data
```

In [336]:
```python
ticker=['ONGC.NS',
'MMTC.NS',
'BEL.NS',
'LAKSHVILAS.NS','TATAPOWER.NS',
       'ADANIPOWER.BO',
       'VIKRAMTH.BO']
```

In [337]:
```python
tick=yf.download(ticker,period="5y")
```

```
[*********************100%***********************]  7 of 7 completed
```

In [338]:
```python
tick.head()
```

Out[338]:

	Adj Close							Close	
	ADANIPOWER.BO	BEL.NS	LAKSHVILAS.NS	MMTC.NS	ONGC.NS	TATAPOWER.NS	VIKRAMTH.BO	ADANIPOWER.BO	BEL.NS
Date									
2016-05-16	29.450001	90.955719	70.175209	24.267611	108.445633	59.868546	75.184097	29.450001	103.1590
2016-05-17	29.299999	90.779396	70.573441	24.203154	112.540443	59.655487	74.949303	29.299999	102.9590
2016-05-18	29.850000	92.166054	70.529182	24.074244	114.681511	59.655487	74.902336	29.850000	104.5318
2016-05-19	29.000000	89.597092	70.042473	24.170925	112.433388	58.291943	73.869202	29.000000	101.6181
2016-05-20	28.500000	88.178360	70.130966	23.687508	114.065956	59.144161	71.896851	28.500000	100.0090

5 rows × 42 columns

In [339]:
```python
#downloading only close stock of compines
tick=yf.download(ticker,period="5y").Close
```

```
[*********************100%***********************]  7 of 7 completed
```

In [340]:
```python
tick
```

Out[340]:

Date	ADANIPOWER.BO	BEL.NS	LAKSHVILAS.NS	MMTC.NS	ONGC.NS	TATAPOWER.NS	VIKRAMTH.BO
2016-05-16	29.450001	103.159088	73.598473	25.100000	135.066666	70.250000	80.050003
2016-05-17	29.299999	102.959091	74.016121	25.033333	140.166672	70.000000	79.800003
2016-05-18	29.850000	104.531815	73.969711	24.900000	142.833328	70.000000	79.750000
2016-05-19	29.000000	101.618179	73.459259	25.000000	140.033340	68.400002	78.650002
2016-05-20	28.500000	100.009087	73.552071	24.500000	142.066666	69.400002	76.550003
...	...	...	...	...	...	...	...
2021-05-07	97.050003	137.899994	NaN	46.099998	111.449997	102.699997	192.000000
2021-05-10	98.550003	144.750000	NaN	48.549999	113.900002	109.949997	193.000000
2021-05-11	99.000000	148.550003	NaN	54.349998	118.099998	109.349998	196.300003
2021-05-12	97.449997	146.850006	NaN	56.049999	115.099998	106.599998	201.449997
2021-05-14	95.849998	143.350006	NaN	52.849998	112.949997	101.500000	198.350006

1232 rows × 7 columns

In [341]:
```python
tick.head()
```

Out[341]:

Date	ADANIPOWER.BO	BEL.NS	LAKSHVILAS.NS	MMTC.NS	ONGC.NS	TATAPOWER.NS	VIKRAMTH.BO
2016-05-16	29.450001	103.159088	73.598473	25.100000	135.066666	70.250000	80.050003
2016-05-17	29.299999	102.959091	74.016121	25.033333	140.166672	70.000000	79.800003
2016-05-18	29.850000	104.531815	73.969711	24.900000	142.833328	70.000000	79.750000
2016-05-19	29.000000	101.618179	73.459259	25.000000	140.033340	68.400002	78.650002
2016-05-20	28.500000	100.009087	73.552071	24.500000	142.066666	69.400002	76.550003

In [342]:
```python
#copying dataframe
df = tick
```

In [343]:
```python
tick.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1232 entries, 2016-05-16 to 2021-05-14
Data columns (total 7 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   ADANIPOWER.BO  1232 non-null   float64
 1   BEL.NS         1232 non-null   float64
 2   LAKSHVILAS.NS  1134 non-null   float64
 3   MMTC.NS        1232 non-null   float64
 4   ONGC.NS        1232 non-null   float64
 5   TATAPOWER.NS   1232 non-null   float64
 6   VIKRAMTH.BO    1232 non-null   float64
dtypes: float64(7)
memory usage: 77.0 KB
```

In [344]:
```python
df
```

Out[344]:

Date	ADANIPOWER.BO	BEL.NS	LAKSHVILAS.NS	MMTC.NS	ONGC.NS	TATAPOWER.NS	VIKRAMTH.BO
2016-05-16	29.450001	103.159088	73.598473	25.100000	135.066666	70.250000	80.050003
2016-05-17	29.299999	102.959091	74.016121	25.033333	140.166672	70.000000	79.800003
2016-05-18	29.850000	104.531815	73.969711	24.900000	142.833328	70.000000	79.750000
2016-05-19	29.000000	101.618179	73.459259	25.000000	140.033340	68.400002	78.650002
2016-05-20	28.500000	100.009087	73.552071	24.500000	142.066666	69.400002	76.550003
...	...	...	...	...	...	...	...
2021-05-07	97.050003	137.899994	NaN	46.099998	111.449997	102.699997	192.000000
2021-05-10	98.550003	144.750000	NaN	48.549999	113.900002	109.949997	193.000000
2021-05-11	99.000000	148.550003	NaN	54.349998	118.099998	109.349998	196.300003
2021-05-12	97.449997	146.850006	NaN	56.049999	115.099998	106.599998	201.449997
2021-05-14	95.849998	143.350006	NaN	52.849998	112.949997	101.500000	198.350006

1232 rows × 7 columns

In [345]:
```python
df.columns
```

Out[345]:
```
Index(['ADANIPOWER.BO', 'BEL.NS', 'LAKSHVILAS.NS', 'MMTC.NS', 'ONGC.NS',
       'TATAPOWER.NS', 'VIKRAMTH.BO'],
      dtype='object')
```

In [346]:
```python
df.rename(columns={
                    'ADANIPOWER.BO': 'Adanipower closing Stock',
                    'BEL.NS': 'BEL Closing Stock',
                    'LAKSHVILAS.NS': 'lakshivilas closing Stock',
                    'MMTC.NS': 'MMTC losing Stock',
                    'ONGC.NS': 'Ongc closing Stock',
                    'TATAPOWER.NS': 'Tatapower closing Stock',
                    'VIKRAMTH.BO': 'vikram solar closing Stock'
                     }, inplace=True)
```

In [347]:
```python
df.head()
```

Out[347]:

Date	Adanipower closing Stock	BEL Closing Stock	lakshivilas closing Stock	MMTC losing Stock	Ongc closing Stock	Tatapower closing Stock	vikram solar closing Stock
2016-05-16	29.450001	103.159088	73.598473	25.100000	135.066666	70.250000	80.050003
2016-05-17	29.299999	102.959091	74.016121	25.033333	140.166672	70.000000	79.800003
2016-05-18	29.850000	104.531815	73.969711	24.900000	142.833328	70.000000	79.750000
2016-05-19	29.000000	101.618179	73.459259	25.000000	140.033340	68.400002	78.650002
2016-05-20	28.500000	100.009087	73.552071	24.500000	142.066666	69.400002	76.550003

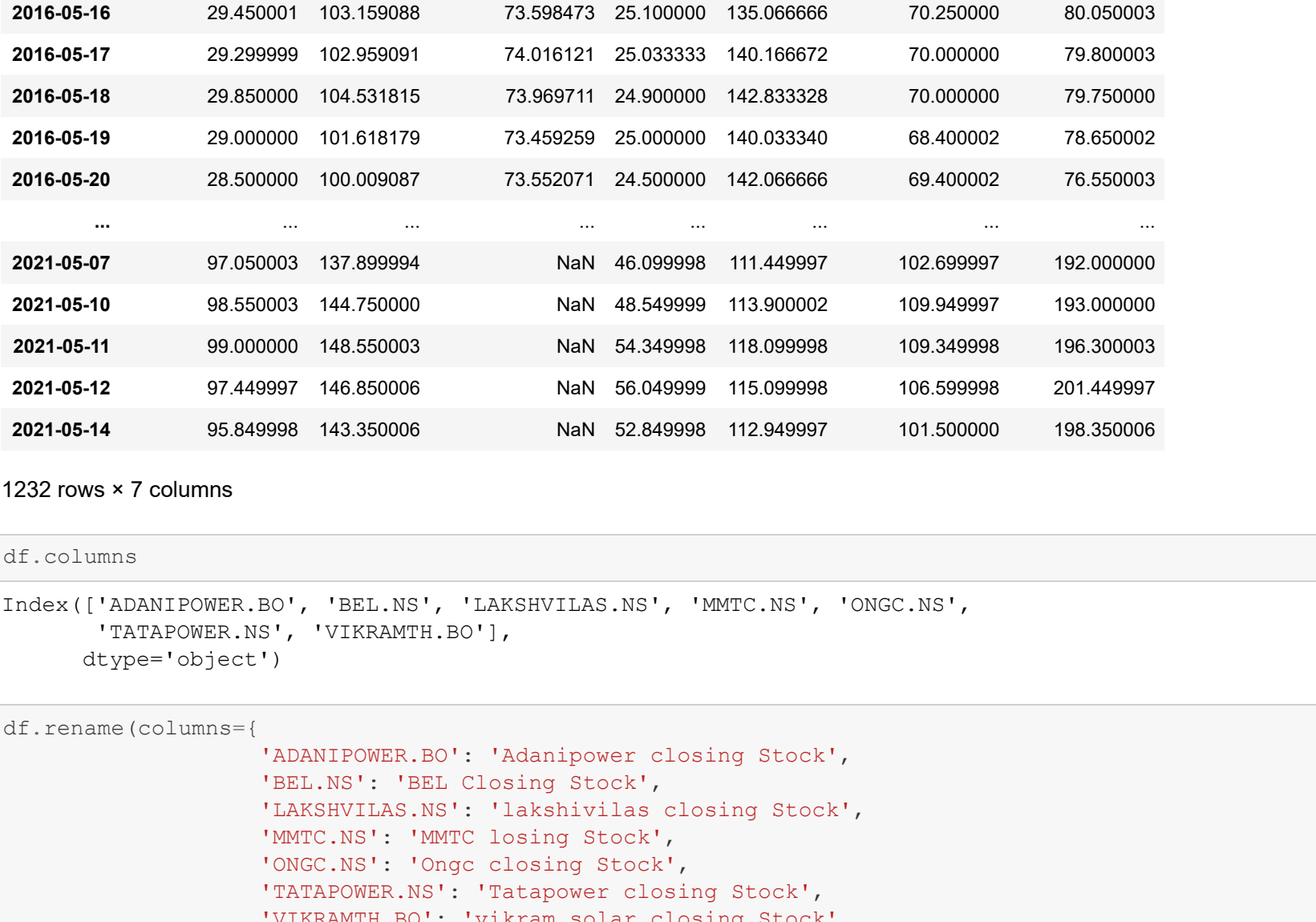
In [348]:
```python
df.columns
```

Out[348]:
```
Index(['Adanipower closing Stock', 'BEL Closing Stock',
       'lakshivilas closing Stock', 'MMTC losing Stock', 'Ongc closing Stock',
       'Tatapower closing Stock', 'vikram solar closing Stock'],
      dtype='object')
```

In [ ]:

In [349]:
```python
#plotting the line plot
import matplotlib.pyplot as plt
plt.figure(figsize=(20,15))
plt.plot(df)
plt.legend(df.columns)
```

Out[349]:
```
<matplotlib.legend.Legend at 0x1d2649d8b20>
```



In [350]:
```python
#seems 2020 stocks prize reduced due to covid-19
```

Window functions;- Window functions are used to identify sub periods, calculates sub-metrics of sub-periods.

Rolling - Same size and sliding

Expanding - Contains all prior values

In [354]:
```python
#checking for null values
df.isnull().sum()
```

Out[354]:
```
Adanipower closing Stock       0
BEL Closing Stock              0
lakshivilas closing Stock     98
MMTC losing Stock              0
Ongc closing Stock             0
Tatapower closing Stock        0
vikram solar closing Stock     0
dtype: int64
```

In [356]:
```python
df = df.dropna()
```

In [357]:
```python
df.shape
```

Out[357]:
```
(1134, 7)
```

In [358]:
```python
df.isnull().sum()
```

Out[358]:
```
Adanipower closing Stock      0
BEL Closing Stock             0
lakshivilas closing Stock     0
MMTC losing Stock             0
Ongc closing Stock            0
Tatapower closing Stock       0
vikram solar closing Stock    0
dtype: int64
```

decompose:- One use of a creating a smoothed trend line in this fashion is to perform seasonal decomposition, to break out the original data into components for the trend, cyclical deviations from this trend and then whatever residuals are left over.

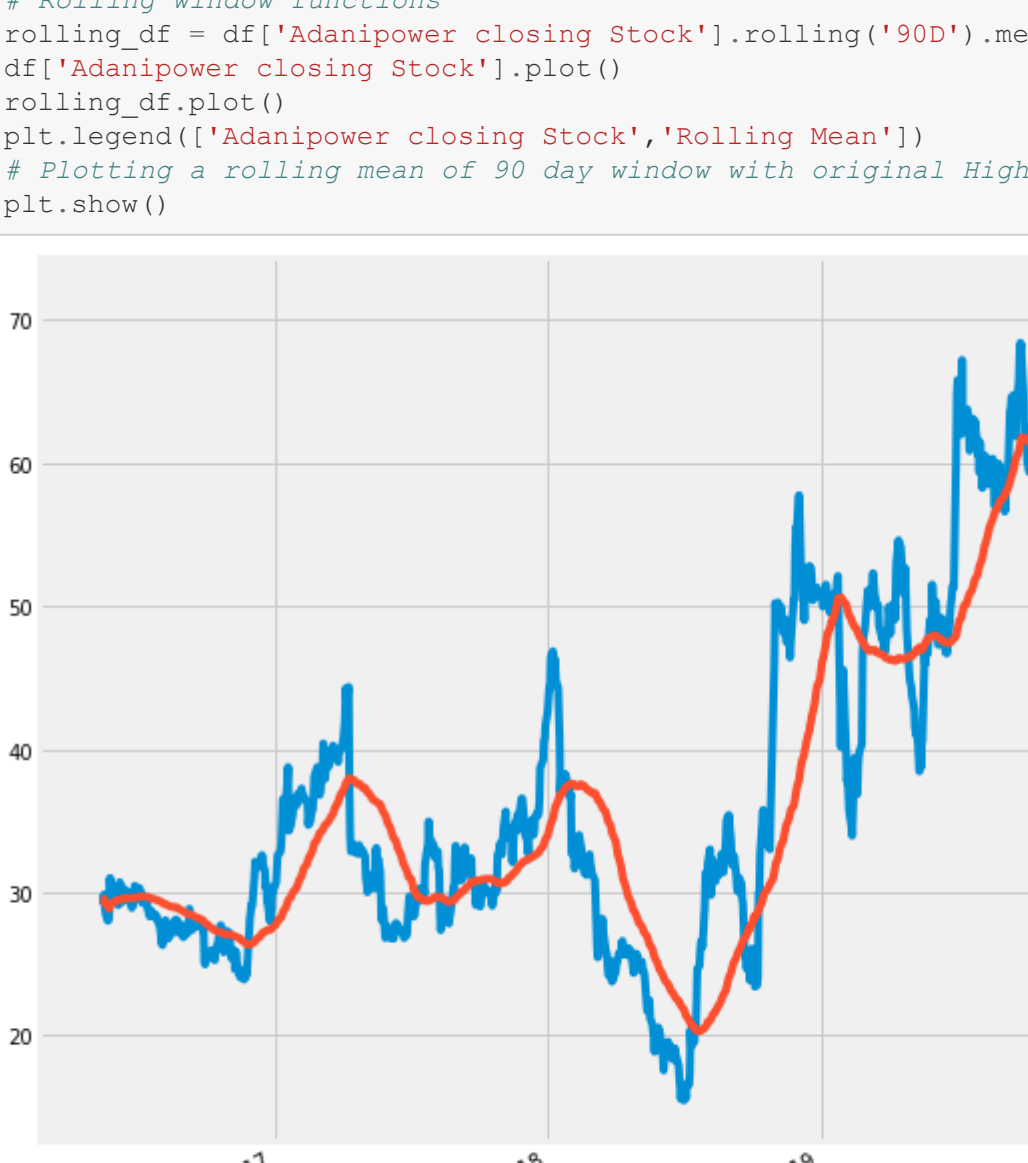
In [381]:
```python
def decompose(feture):
    # Now, for decomposition...
    rcParams['figure.figsize'] = 11, 9
    decomposed = sm.tsa.seasonal_decompose(feture,freq=360) # The frequncy is annual
    figure = decomposed.plot()
    plt.show()
```

In [382]:
```python
dfcol= df.columns
```

In [384]:
```python
decompose(df[dfcol[0]])
decompose(df[dfcol[1]])
decompose(df[dfcol[2]])
decompose(df[dfcol[3]])
decompose(df[dfcol[4]])
decompose(df[dfcol[5]])
decompose(df[dfcol[6]])
```

In [385]:
```python
#in 2020 adani power and vikram solar stockes incresed, while reaming are in decrsed trend
```

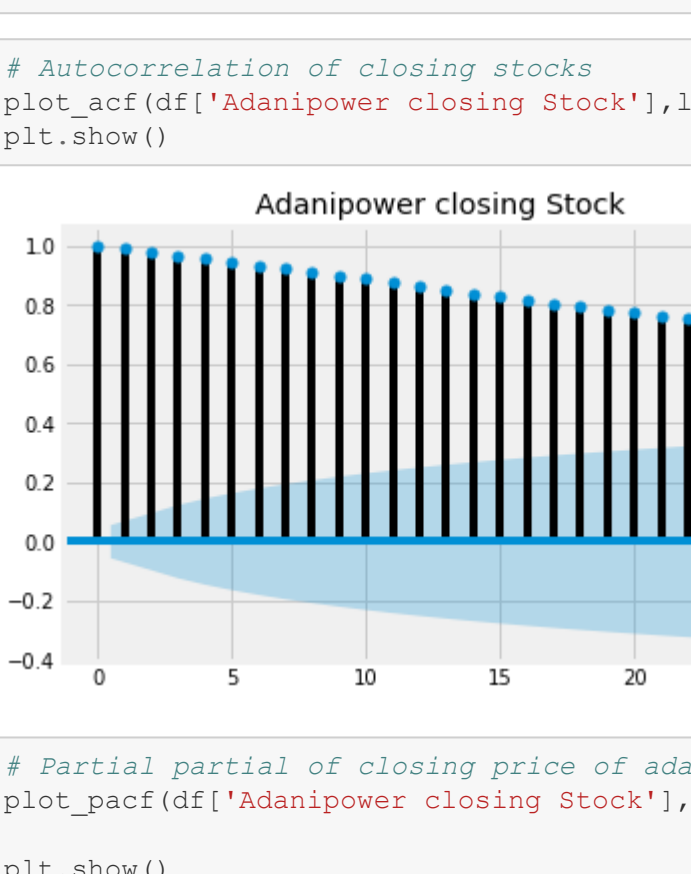
In [390]:
```python
# Rolling window functions
rolling_df = df['Adanipower closing Stock'].rolling('90D').mean()
df['Adanipower closing Stock'].plot()
rolling_df.plot()
plt.legend(['Adanipower closing Stock','Rolling Mean'])
# Plotting a rolling mean of 90 day window with original High attribute of Adanipower  stocks
plt.show()
```



Autocorrelation - The autocorrelation function (ACF) measures how a series is correlated with itself at different lags. Partial Autocorrelation - The partial autocorrelation function can be interpreted as a regression of the series against its past lags. The terms can be interpreted the same way as a standard linear regression, that is the contribution of a change in that particular lag while holding others constant.

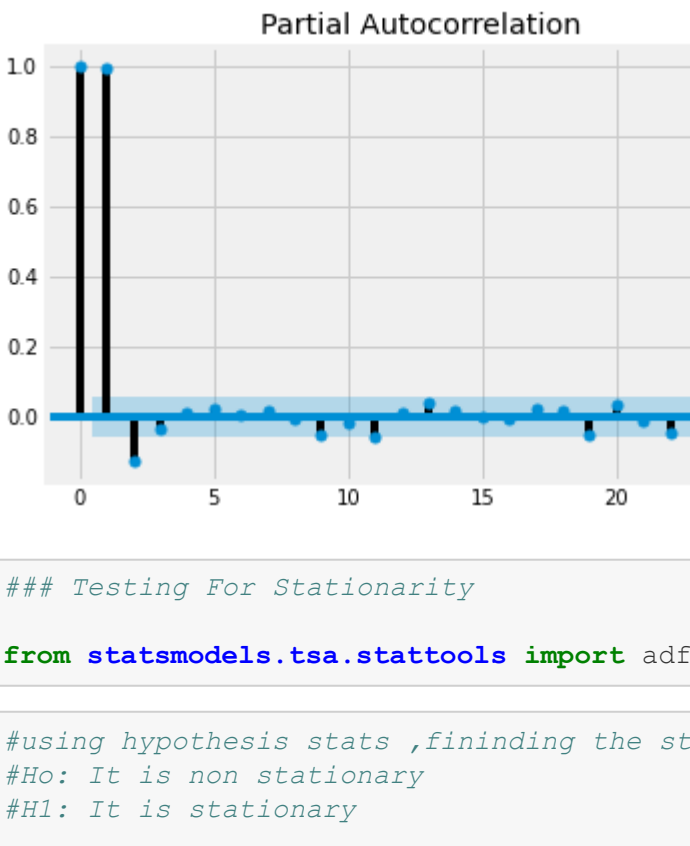
In [131]:
```python
import matplotlib.pyplot as plt
```

In [132]:
```python
# Autocorrelation of closing stocks
plot_acf(df['Adanipower closing Stock'],lags=25,title="Adanipower closing Stock")
plt.show()
```



In [133]:
```python
# Partial partial of closing price of adanipower stocks
plot_pacf(df['Adanipower closing Stock'],lags=25)

plt.show()
```



In [134]:
```python
### Testing For Stationarity

from statsmodels.tsa.stattools import adfuller
```

In [391]:
```python
#using hypothesis stats ,fininding the stationary
#Ho: It is non stationary
#H1: It is stationary

def adfuller_test(stocks):
    result=adfuller(stocks)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations Used']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) )
    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data has n
o unit root and is stationary")
    else:
        print("weak evidence against null hypothesis, time series has a unit root, indicating it is no
n-stationary ")
```

In [392]:
```python
adfuller_test(df['Adanipower closing Stock'])
```

```
ADF Test Statistic : -2.0981541615087265
p-value : 0.24525746994299158
#Lags Used : 1
Number of Observations Used : 1132
weak evidence against null hypothesis, time series has a unit root, indicating it is non-stationary
```

In [138]:
```python
adfuller_test(df['Tatapower closing Stock'])
```

```
ADF Test Statistic : -1.0956589901406053
p-value : 0.7167999413137178
#Lags Used : 14
Number of Observations Used : 1217
weak evidence against null hypothesis, time series has a unit root, indicating it is non-stationary
```

In [ ]:
```python
#to convert stationary
```

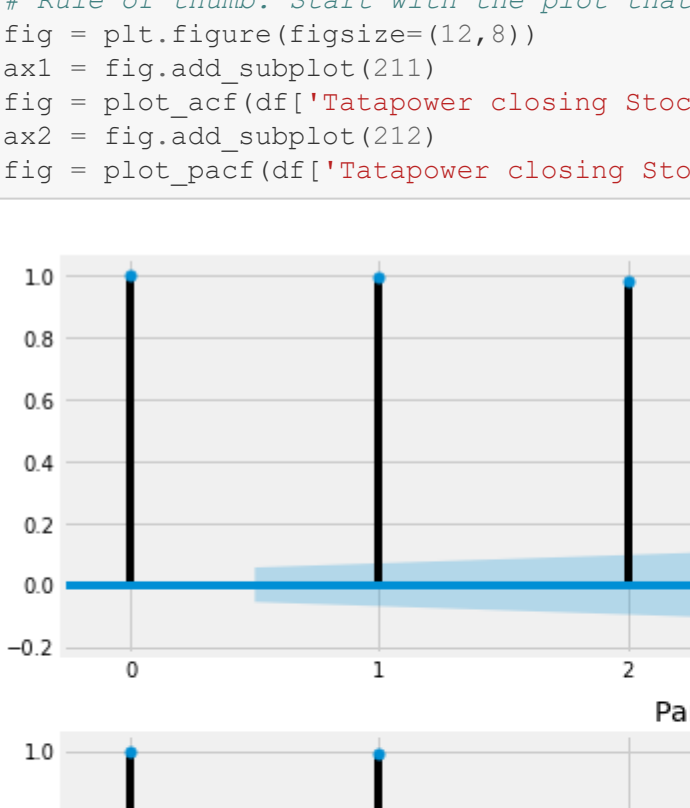
In [139]:
```python
df['Seasonal First Difference']=df['Tatapower closing Stock']-df['Tatapower closing Stock'].shift(1)
```

In [140]:
```python
adfuller_test(df['Seasonal First Difference'].dropna())
```

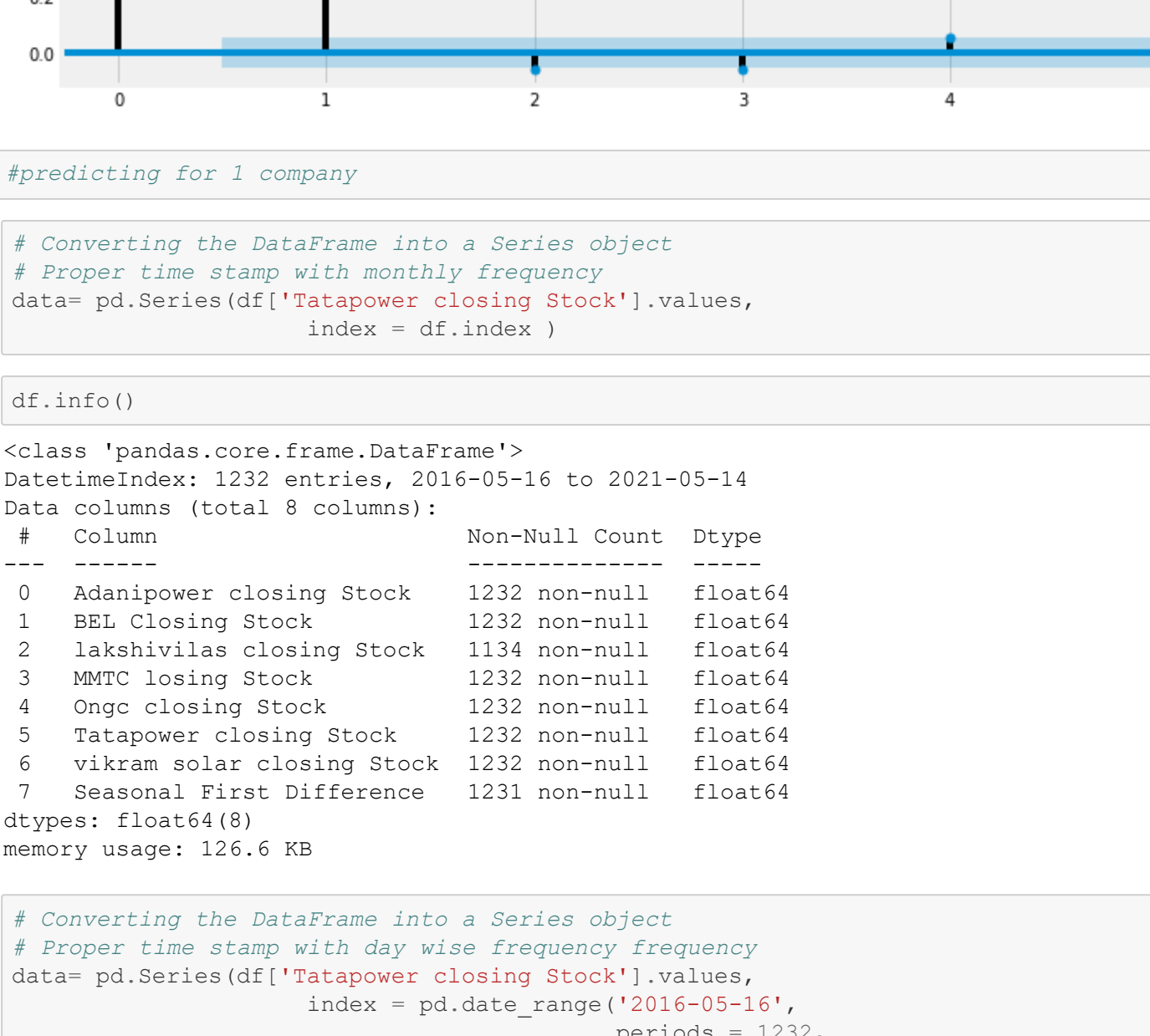
```
ADF Test Statistic : -9.992604027060011
p-value : 1.9779338728186803e-17
#Lags Used : 13
Number of Observations Used : 1217
strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data has no unit root an
d is stationary
```

In [141]:
```python
df['Seasonal First Difference'].plot()
```

Out[141]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1d2602b6820>
```



In [143]:
```python
# ACF and PACF plots
# Rule of thumb: Start with the plot that shows the least number of significant lags
fig = plt.figure(figsize=(12,8))
ax1 = fig.add_subplot(211)
fig = plot_acf(df['Tatapower closing Stock'], lags=5, ax=ax1)
ax2 = fig.add_subplot(212)
fig = plot_pacf(df['Tatapower closing Stock'], lags=5, ax=ax2)
```



In [ ]:
```python
#predicting for 1 company
```

In [219]:
```python
# Converting the DataFrame into a Series object
# Proper time stamp with monthly frequency
data= pd.Series(df['Tatapower closing Stock'].values,
                index = df.index )
```

In [223]:
```python
df.info()
```

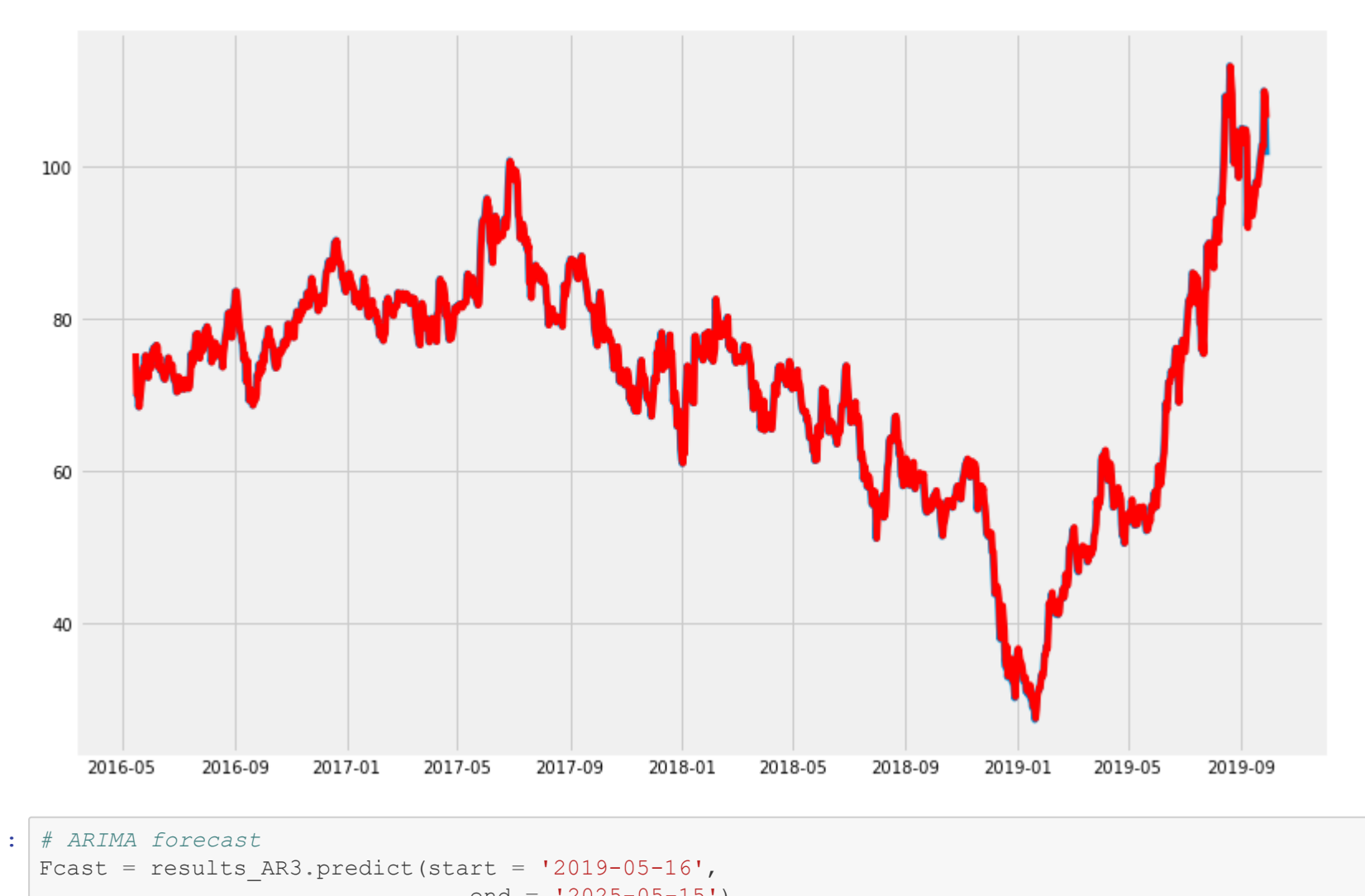
```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1232 entries, 2016-05-16 to 2021-05-14
Data columns (total 8 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   Adanipower closing Stock    1232 non-null   float64
 1   BEL Closing Stock           1232 non-null   float64
 2   lakshivilas closing Stock   1134 non-null   float64
 3   MMTC losing Stock           1232 non-null   float64
 4   Ongc closing Stock          1232 non-null   float64
 5   Tatapower closing Stock     1232 non-null   float64
 6   vikram solar closing Stock  1232 non-null   float64
 7   Seasonal First Difference   1231 non-null   float64
dtypes: float64(8)
memory usage: 126.6 KB
```

In [277]:
```python
# Converting the DataFrame into a Series object
# Proper time stamp with day wise frequency frequency
data= pd.Series(df['Tatapower closing Stock'].values,
                index = pd.date_range('2016-05-16',
                                      periods = 1232,
                                      freq = 'D'))
```

In [313]:
```
# ARIMA model setup
model = ARIMA(data, order=(2, 0, 0))
results_AR3 = model.fit()
print(results_AR3.summary())
plt.figure(figsize=(12,8))
plt.plot(data)
plt.plot(results_AR3.fittedvalues, color='red')
```

```
                              ARMA Model Results
==============================================================================
Dep. Variable:                      y   No. Observations:                 1232
Model:                     ARMA(2, 0)   Log Likelihood               -2277.196
Method:                       css-mle   S.D. of innovations              1.533
Date:                Sat, 15 May 2021   AIC                           4562.391
Time:                        14:39:47   BIC                           4582.857
Sample:                    05-16-2016   HQIC                          4570.090
                         - 09-29-2019
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         75.4528      8.440      8.940      0.000      58.911      91.995
ar.L1.y        1.0111      0.029     35.365      0.000       0.955       1.067
ar.L2.y       -0.0157      0.029     -0.547      0.584      -0.072       0.040
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            1.0046           +0.0000j            1.0046            0.0000
AR.2           63.5439           +0.0000j           63.5439            0.0000
-----------------------------------------------------------------------------
```

Out[313]: [<matplotlib.lines.Line2D at 0x1d26b7c5af0>]



In [393]:
```
# ARIMA forecast
Fcast = results_AR3.predict(start = '2019-05-16',
                            end = '2025-05-15')
```

In [394]:
```
Fcast
```

Out[394]:
```
2019-05-16    54.034907
2019-05-17    55.210240
2019-05-18    55.343897
2019-05-19    53.976519
2019-05-20    53.845995
                ...
2025-05-11    75.454840
2025-05-12    75.454831
2025-05-13    75.454821
2025-05-14    75.454812
2025-05-15    75.454803
Freq: D, Length: 2192, dtype: float64
```

In [395]:
```
# Forecast comparison
plt.figure(figsize = (12, 8))
plt.plot(df['Tatapower closing Stock'], linewidth = 2, label = "original")
plt.plot(Fcast, color='red', linewidth = 2,
         label = "Predicted by ARIMA 2, 0, 0")
#plt.plot(Fcast202, color='blue', linewidth = 2,label = "ARIMA 2 0 2")
plt.legend()
```

Out[395]: <matplotlib.legend.Legend at 0x1d27a2dba60>

In [364]:
```
# Split data into train / test sets
train = df.iloc[:len(df)-12]
test = df.iloc[len(df)-12:] # set one year(12 months) for testing

# Fit a SARIMAX(0, 1, 1)x(2, 1, 1, 12) on the training set
from statsmodels.tsa.statespace.sarimax import SARIMAX

model = SARIMAX(train['Tatapower closing Stock'],
                order = (0, 1, 1),
                seasonal_order =(2, 1, 1, 12))

result = model.fit()
result.summary()
```

Out[364]:

SARIMAX Results

Dep. Variable:	Tatapower closing Stock	**No. Observations:**	1122
Model:	SARIMAX(0, 1, 1)x(2, 1, 1, 12)	**Log Likelihood**	-1956.086
Date:	Sat, 15 May 2021	**AIC**	3922.171
Time:	14:57:26	**BIC**	3947.227
Sample:	0	**HQIC**	3931.646
	- 1122		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ma.L1	0.0326	0.028	1.165	0.244	-0.022	0.088
ar.S.L12	0.0426	0.027	1.567	0.117	-0.011	0.096
ar.S.L24	-0.0107	0.031	-0.343	0.732	-0.072	0.050
ma.S.L12	-0.9997	0.937	-1.067	0.286	-2.837	0.837
sigma2	1.8996	1.769	1.074	0.283	-1.568	5.367

Ljung-Box (Q):	43.87	**Jarque-Bera (JB):**	392.19
Prob(Q):	0.31	**Prob(JB):**	0.00
Heteroskedasticity (H):	1.36	**Skew:**	0.39
Prob(H) (two-sided):	0.00	**Kurtosis:**	5.80

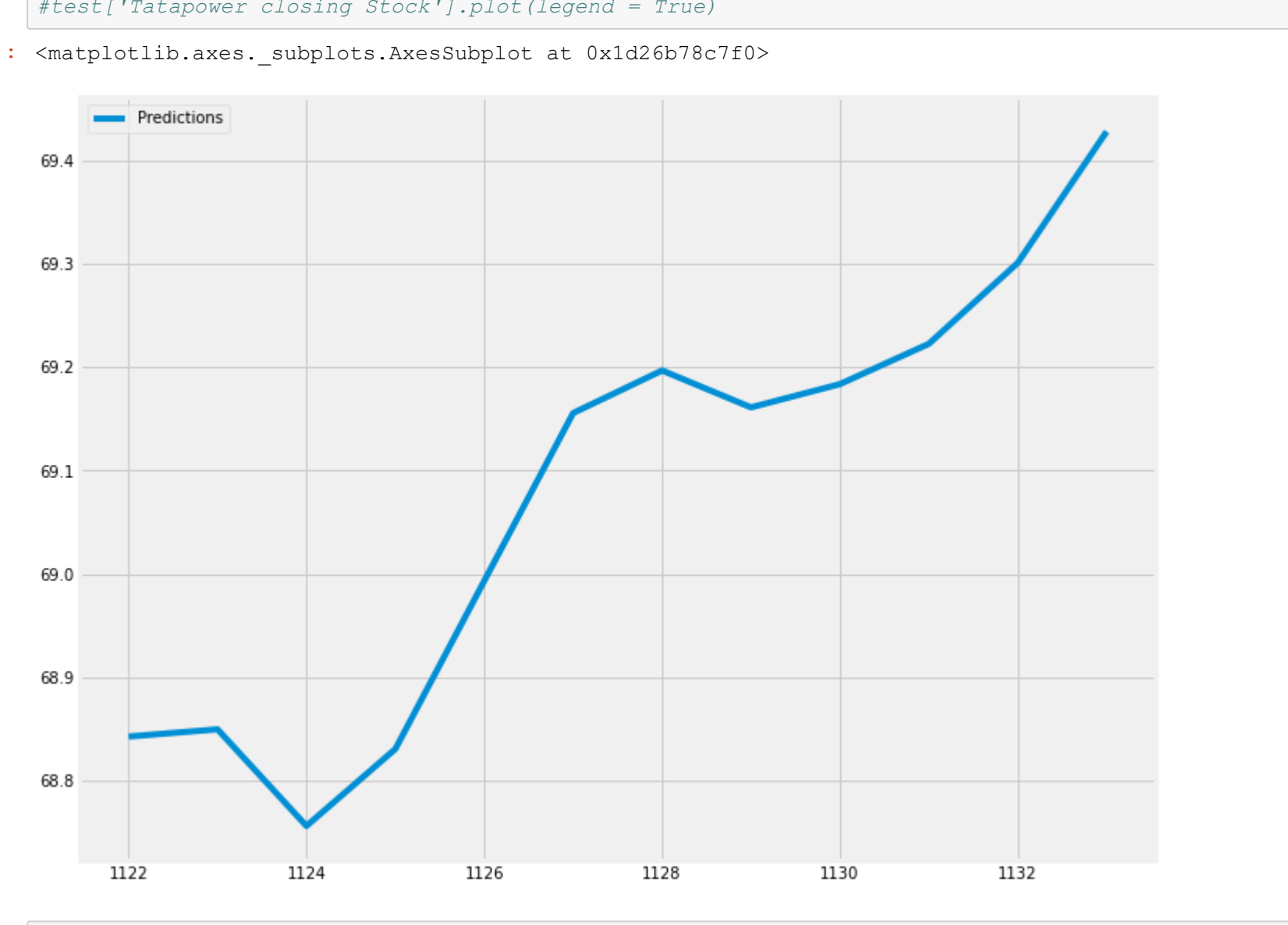
Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

In [367]:
```
start = len(train)
end = len(train) + len(test) - 1

# Predictions for one-year against the test set
predictions = result.predict(start, end,
                             typ = 'levels').rename("Predictions")

# plot predictions and actual values
predictions.plot(legend = True)
#test['Tatapower closing Stock'].plot(legend = True)
```

Out[367]: <matplotlib.axes._subplots.AxesSubplot at 0x1d26b78c7f0>



In [368]:
```
# Load specific evaluation tools
from sklearn.metrics import mean_squared_error
from statsmodels.tools.eval_measures import rmse

# Calculate root mean squared error
rmse(test["Tatapower closing Stock"], predictions)

# Calculate mean squared error
mean_squared_error(test["Tatapower closing Stock"], predictions)
```

Out[368]: 16.750669612713747

by Hrasha