Stationary_Test_Kospi_10

February 23, 2021

[36]:
```python
from dateutil.parser import parse
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
```

[37]:
```python
# FinanceDataReader로 데이터를 불러옵니다
# 예측할 종목은 한양증권(001750) 입니다

import FinanceDataReader as fdr

Samsung_Electronics = '005930'
SK_hynix = '000660'
LG_Chem = '051910'
NAVER_Corporation = '035420'
Samsung_Biologics = '207940'
Hyundai_Motor_Company = '005380'
Samsung_SDI = '006400'
Kakao = '035720'
Celltrion = '068270'
Kia_Corporation = '000270'
```

[38]:
```python
df_Samsung_Electronics = fdr.DataReader(Samsung_Electronics , '2020-02-22',␣
 ↪'2021-02-22')
df_SK_hynix = fdr.DataReader(SK_hynix , '2020-02-22', '2021-02-22')
df_LG_Chem = fdr.DataReader(LG_Chem , '2020-02-22', '2021-02-22')
df_NAVER_Corporation = fdr.DataReader(NAVER_Corporation , '2020-02-22',␣
 ↪'2021-02-22')
df_Samsung_Biologics = fdr.DataReader(Samsung_Biologics , '2020-02-22',␣
 ↪'2021-02-22')
df_Hyundai_Motor_Company = fdr.DataReader(Hyundai_Motor_Company , '2020-02-22',␣
 ↪'2021-02-22')
df_Samsung_SDI = fdr.DataReader(Samsung_SDI , '2020-02-22', '2021-02-22')
df_Kakao = fdr.DataReader(Kakao , '2020-02-22', '2021-02-22')
df_Celltrion = fdr.DataReader(Celltrion , '2020-02-22', '2021-02-22')
df_Kia_Corporation = fdr.DataReader(Kia_Corporation , '2020-02-22', '2021-02-22')
```

[39]:
```python
df_SE = df_Samsung_Electronics

df_SE = df_SE.loc[:, ['Close']]

df_SE['Close']
```

[39]:
```
Date
2020-02-24    56800
2020-02-25    57900
2020-02-26    56500
2020-02-27    55900
2020-02-28    54200
              ...
2021-02-16    84900
2021-02-17    83200
2021-02-18    82100
2021-02-19    82600
2021-02-22    82200
Name: Close, Length: 247, dtype: int64
```

[40]:
```python
df_all_dates = pd.DataFrame(index=pd.date_range(start='2020-02-22',
                                                end='2021-02-22'))


df_all_dates
```

[40]:
```
Empty DataFrame
Columns: []
Index: [2020-02-22 00:00:00, 2020-02-23 00:00:00, 2020-02-24 00:00:00,
2020-02-25 00:00:00, 2020-02-26 00:00:00, 2020-02-27 00:00:00, 2020-02-28
00:00:00, 2020-02-29 00:00:00, 2020-03-01 00:00:00, 2020-03-02 00:00:00,
2020-03-03 00:00:00, 2020-03-04 00:00:00, 2020-03-05 00:00:00, 2020-03-06
00:00:00, 2020-03-07 00:00:00, 2020-03-08 00:00:00, 2020-03-09 00:00:00,
2020-03-10 00:00:00, 2020-03-11 00:00:00, 2020-03-12 00:00:00, 2020-03-13
00:00:00, 2020-03-14 00:00:00, 2020-03-15 00:00:00, 2020-03-16 00:00:00,
2020-03-17 00:00:00, 2020-03-18 00:00:00, 2020-03-19 00:00:00, 2020-03-20
00:00:00, 2020-03-21 00:00:00, 2020-03-22 00:00:00, 2020-03-23 00:00:00,
2020-03-24 00:00:00, 2020-03-25 00:00:00, 2020-03-26 00:00:00, 2020-03-27
00:00:00, 2020-03-28 00:00:00, 2020-03-29 00:00:00, 2020-03-30 00:00:00,
2020-03-31 00:00:00, 2020-04-01 00:00:00, 2020-04-02 00:00:00, 2020-04-03
00:00:00, 2020-04-04 00:00:00, 2020-04-05 00:00:00, 2020-04-06 00:00:00,
2020-04-07 00:00:00, 2020-04-08 00:00:00, 2020-04-09 00:00:00, 2020-04-10
00:00:00, 2020-04-11 00:00:00, 2020-04-12 00:00:00, 2020-04-13 00:00:00,
2020-04-14 00:00:00, 2020-04-15 00:00:00, 2020-04-16 00:00:00, 2020-04-17
00:00:00, 2020-04-18 00:00:00, 2020-04-19 00:00:00, 2020-04-20 00:00:00,
2020-04-21 00:00:00, 2020-04-22 00:00:00, 2020-04-23 00:00:00, 2020-04-24
00:00:00, 2020-04-25 00:00:00, 2020-04-26 00:00:00, 2020-04-27 00:00:00,
2020-04-28 00:00:00, 2020-04-29 00:00:00, 2020-04-30 00:00:00, 2020-05-01
```

```
00:00:00, 2020-05-02 00:00:00, 2020-05-03 00:00:00, 2020-05-04 00:00:00,
2020-05-05 00:00:00, 2020-05-06 00:00:00, 2020-05-07 00:00:00, 2020-05-08
00:00:00, 2020-05-09 00:00:00, 2020-05-10 00:00:00, 2020-05-11 00:00:00,
2020-05-12 00:00:00, 2020-05-13 00:00:00, 2020-05-14 00:00:00, 2020-05-15
00:00:00, 2020-05-16 00:00:00, 2020-05-17 00:00:00, 2020-05-18 00:00:00,
2020-05-19 00:00:00, 2020-05-20 00:00:00, 2020-05-21 00:00:00, 2020-05-22
00:00:00, 2020-05-23 00:00:00, 2020-05-24 00:00:00, 2020-05-25 00:00:00,
2020-05-26 00:00:00, 2020-05-27 00:00:00, 2020-05-28 00:00:00, 2020-05-29
00:00:00, 2020-05-30 00:00:00, 2020-05-31 00:00:00, ...]

[367 rows x 0 columns]
```

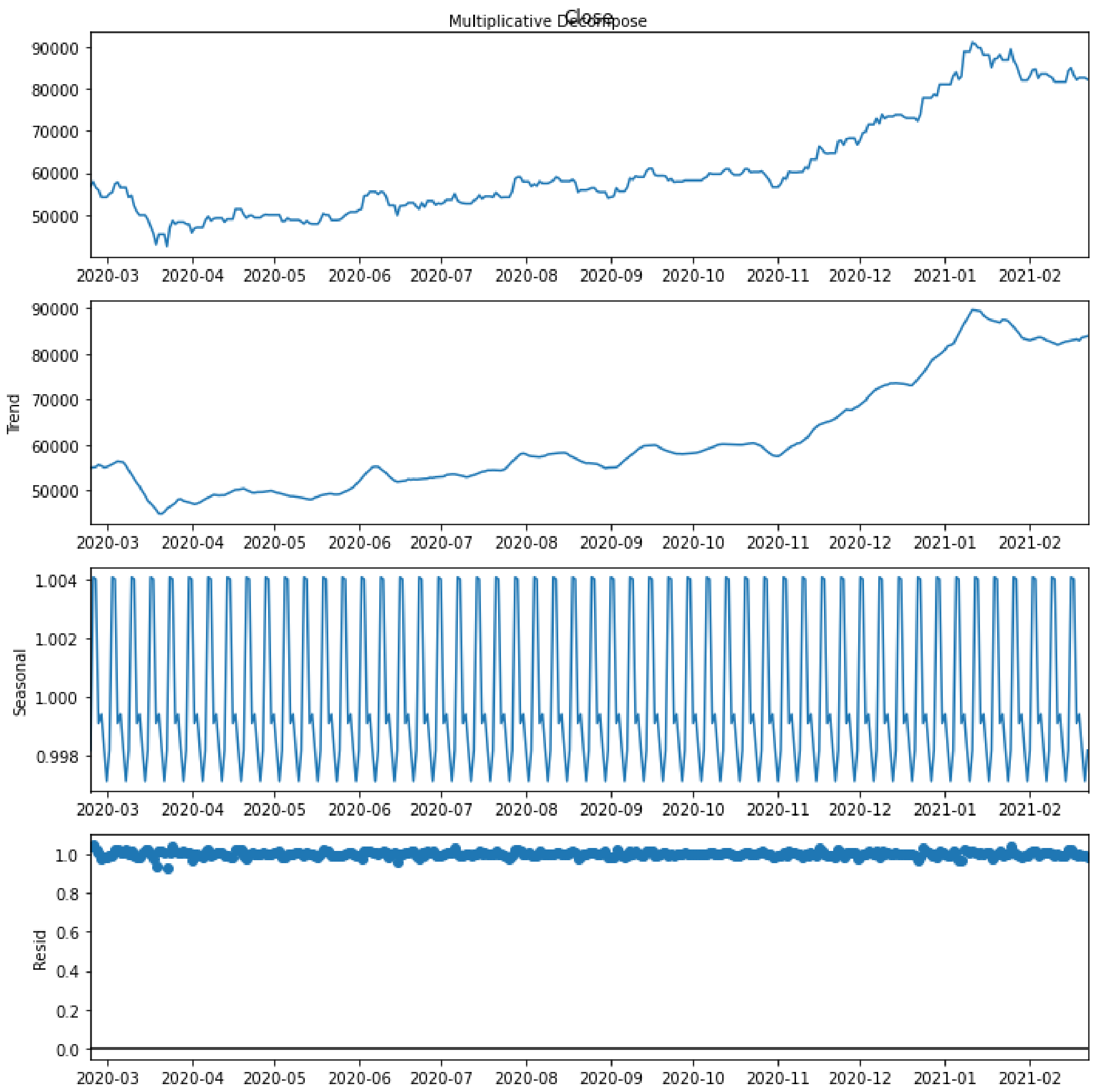
[41]:
```
# 가능한 모든 날짜로 DataFrame을 만들고 가격을 결합합니다

df_SE = df_all_dates.join(df_SE['Close'], how='left').fillna(method='ffill').
 ↪asfreq('D')
df_SE = df_SE.dropna()

df_SE
```

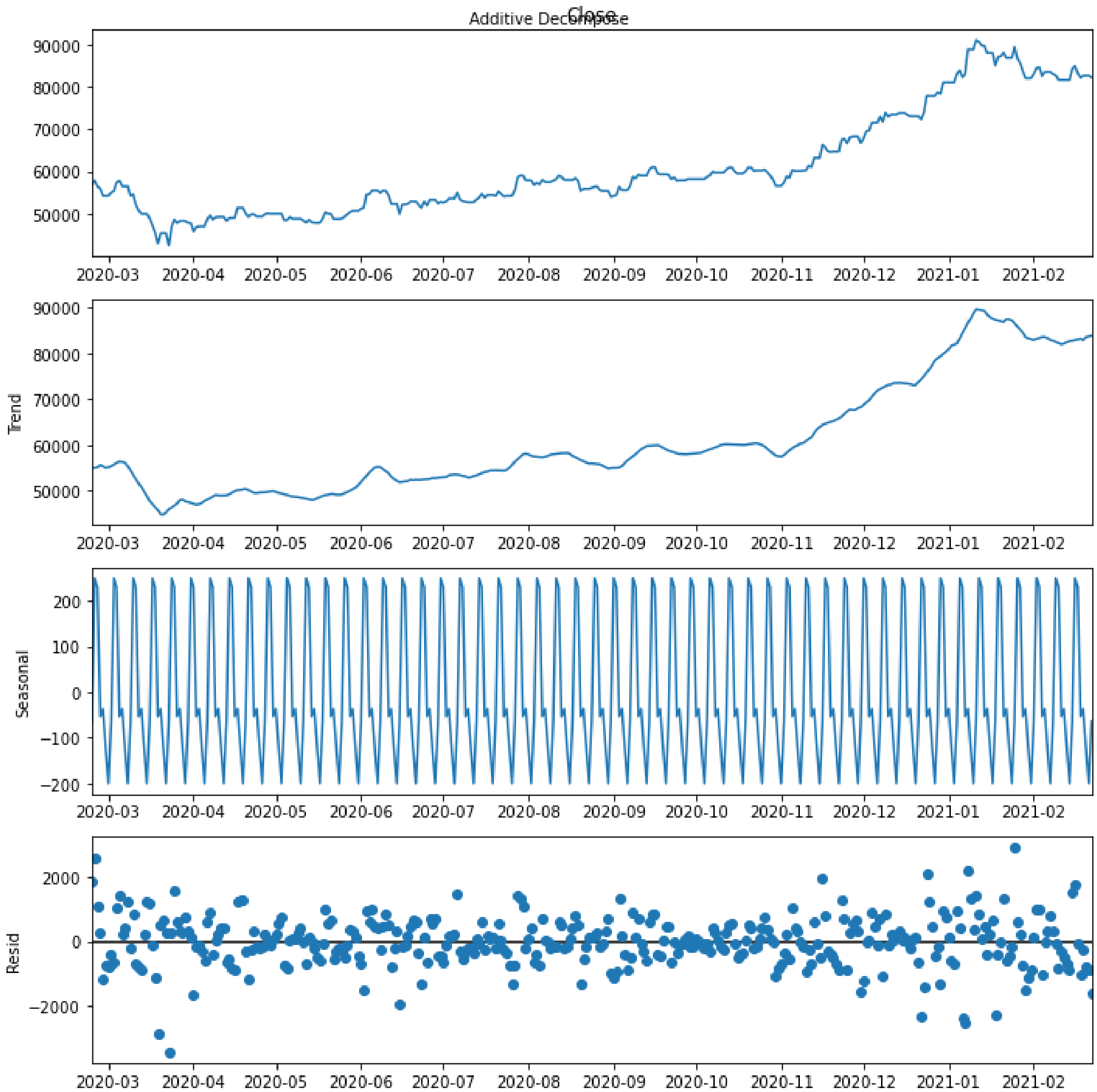
[41]:
```
              Close
2020-02-24  56800.0
2020-02-25  57900.0
2020-02-26  56500.0
2020-02-27  55900.0
2020-02-28  54200.0
...             ...
2021-02-18  82100.0
2021-02-19  82600.0
2021-02-20  82600.0
2021-02-21  82600.0
2021-02-22  82200.0

[365 rows x 1 columns]
```

1 시계열 데이터를 구성요소로 분해합니다

[43]:
```
from statsmodels.tsa.seasonal import seasonal_decompose
from dateutil.parser import parse


# 곱하기(승법적) 분해
result_mul = seasonal_decompose(df_SE['Close'], model='multiplicative',␣
 ↪extrapolate_trend='freq')

# 더하기(가법적) 분해
```

```
result_add = seasonal_decompose(df_SE['Close'], model='additive',␣
 ↪extrapolate_trend='freq')

# Plot
plt.rcParams.update({'figure.figsize': (10,10)})
result_mul.plot().suptitle('Multiplicative Decompose', fontsize=10)
result_add.plot().suptitle('Additive Decompose', fontsize=10)
plt.show()
```





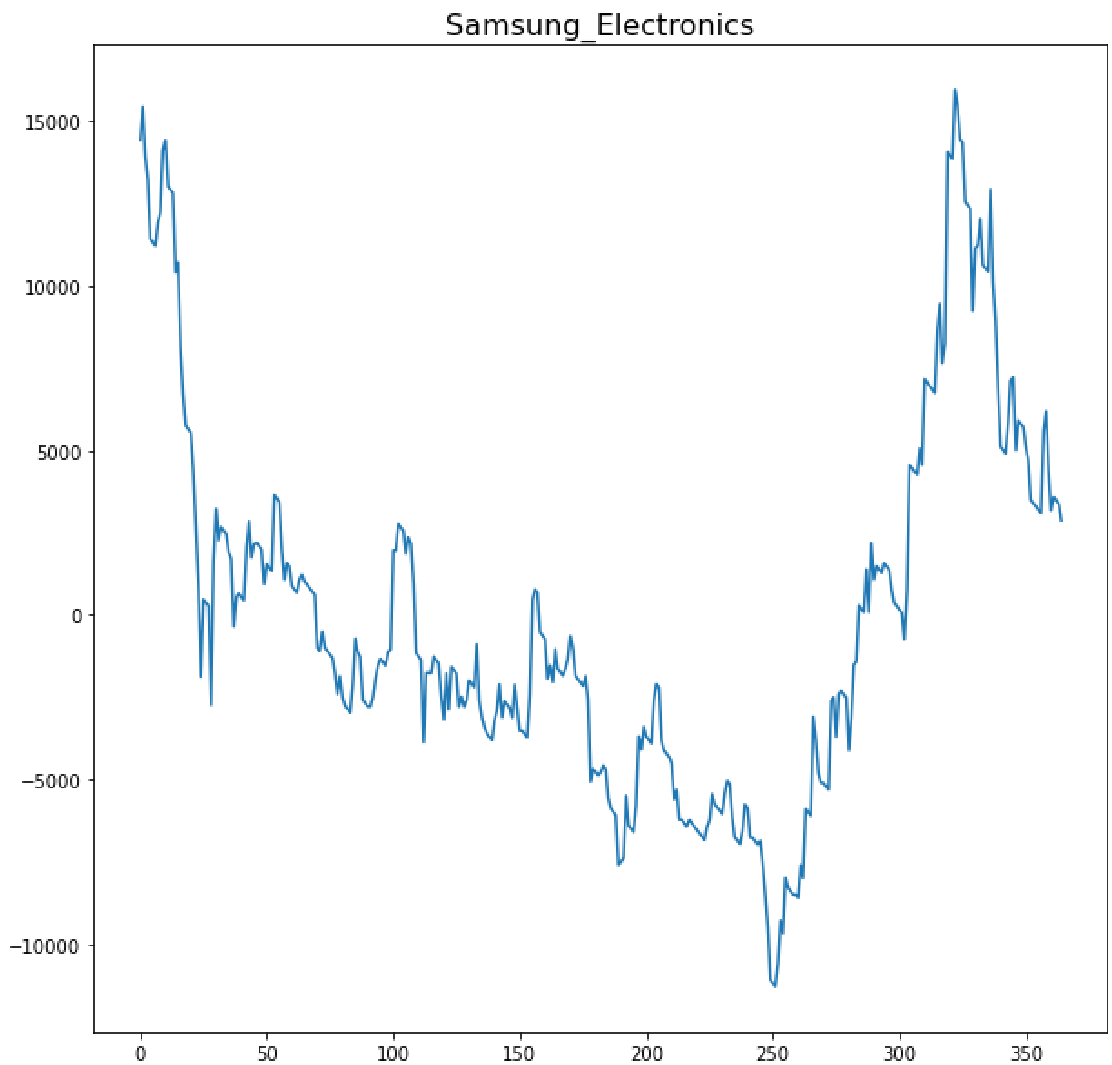
[44]:
```python
# 실제 값(관측 값) = Product of (Seasonal * Trend * Resid)
df_SE_reconstructed = pd.concat([result_mul.seasonal, result_mul.trend,␣
 ↪result_mul.resid, result_mul.observed], axis=1)
df_SE_reconstructed.columns = ['seas', 'trend', 'resid', 'actual_values']
df_SE_reconstructed.head()
```

[44]:

	seas	trend	resid	actual_values
2020-02-24	0.998161	54996.938776	1.034687	56800.0
2020-02-25	1.004070	55085.204082	1.046838	57900.0
2020-02-26	1.004005	55173.469388	1.019958	56500.0
2020-02-27	0.999077	55671.428571	1.005033	55900.0
2020-02-28	0.999394	55414.285714	0.978681	54200.0

[51]:
```
# 최적화 직선을 제거해서 트렌드를 제거한다.

from scipy import signal

detrended = signal.detrend(df_SE.Close.values)
plt.plot(detrended)
plt.title('Samsung_Electronics detrended by subtracting the least squares fit',␣
 ↪fontsize=16)
```

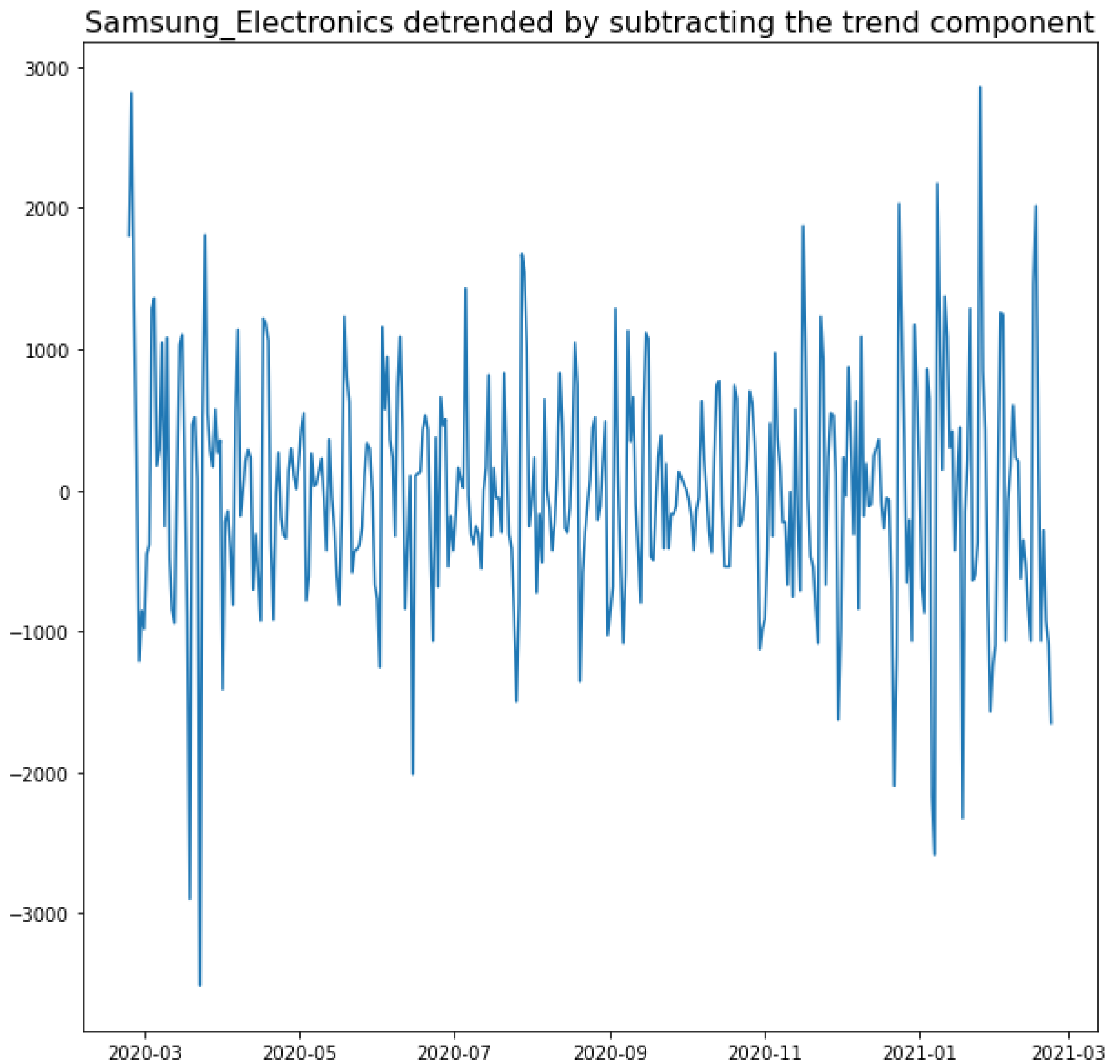
[51]: Text(0.5, 1.0, 'Samsung_Electronics')



[53]:
```
from statsmodels.tsa.seasonal import seasonal_decompose
```

```
result_mul = seasonal_decompose(df_SE['Close'], model='multiplicative',␣
 ↪extrapolate_trend='freq')
detrended = df_SE.Close.values - result_mul.trend
plt.plot(detrended)
plt.title('Samsung_Electronics detrended by subtracting the trend component',␣
 ↪fontsize=16)
```

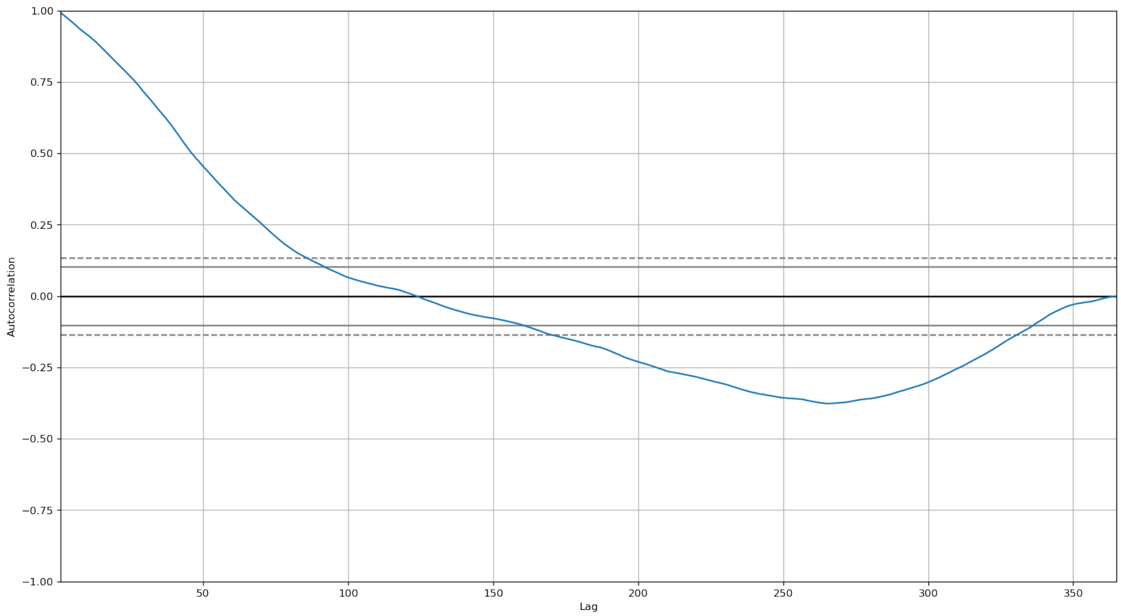
[53]: Text(0.5, 1.0, 'Samsung_Electronics detrended by subtracting the trend component')



2 계절성 확인하기

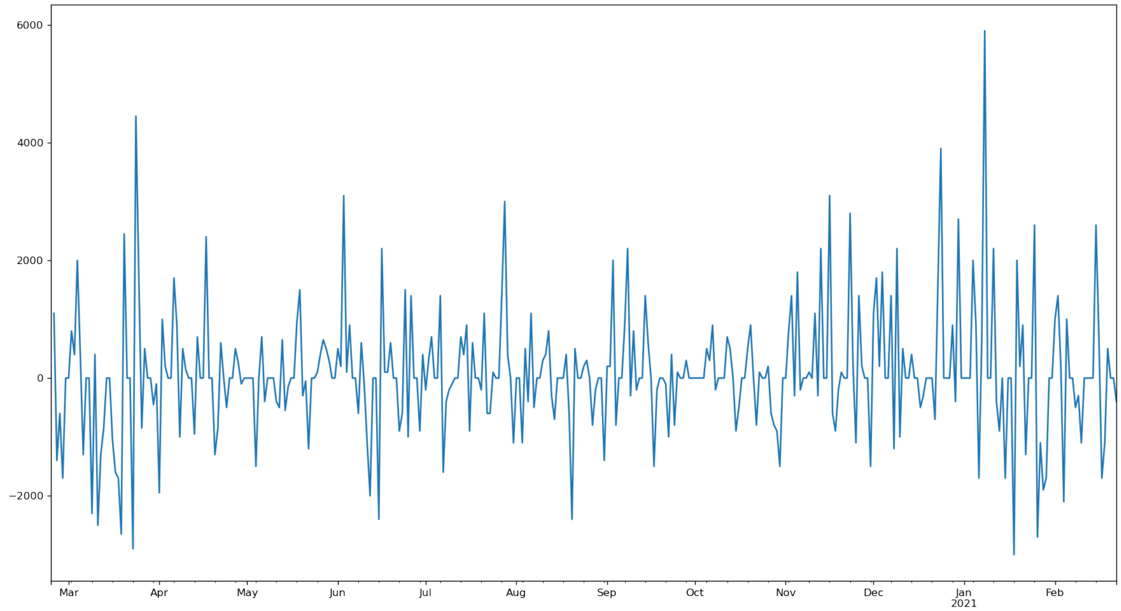
[59]:
```
from pandas.plotting import autocorrelation_plot

# Draw Plot
plt.rcParams.update({'figure.figsize':(18,10), 'figure.dpi':120})
autocorrelation_plot(df_SE.Close.tolist())
```

[59]: <AxesSubplot:xlabel='Lag', ylabel='Autocorrelation'>



[60]:
```
df_SE.Close.diff().plot()
```

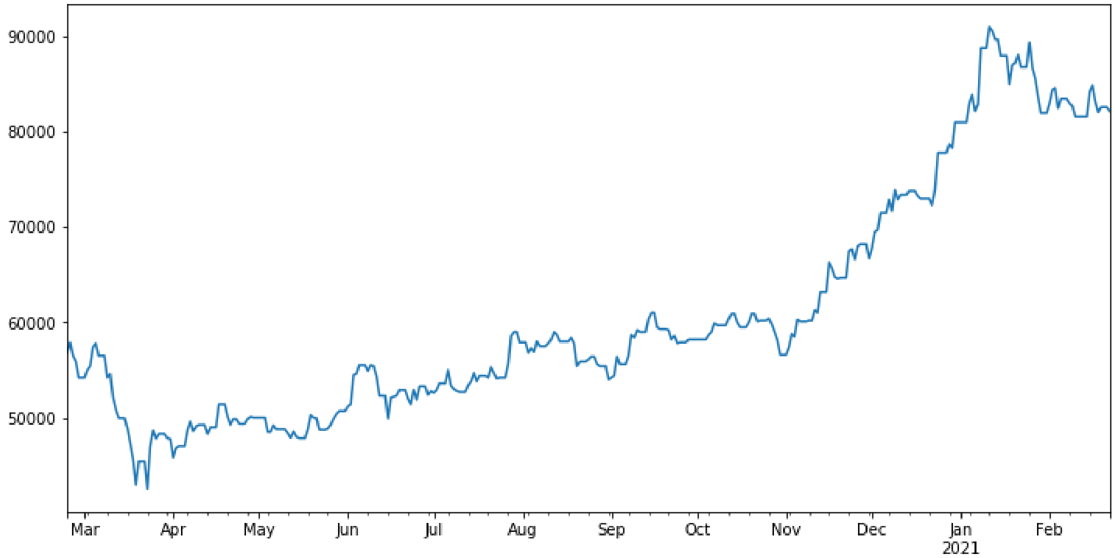
[60]: <AxesSubplot:>



3 시계열 데이터에서 계절성 제거하기

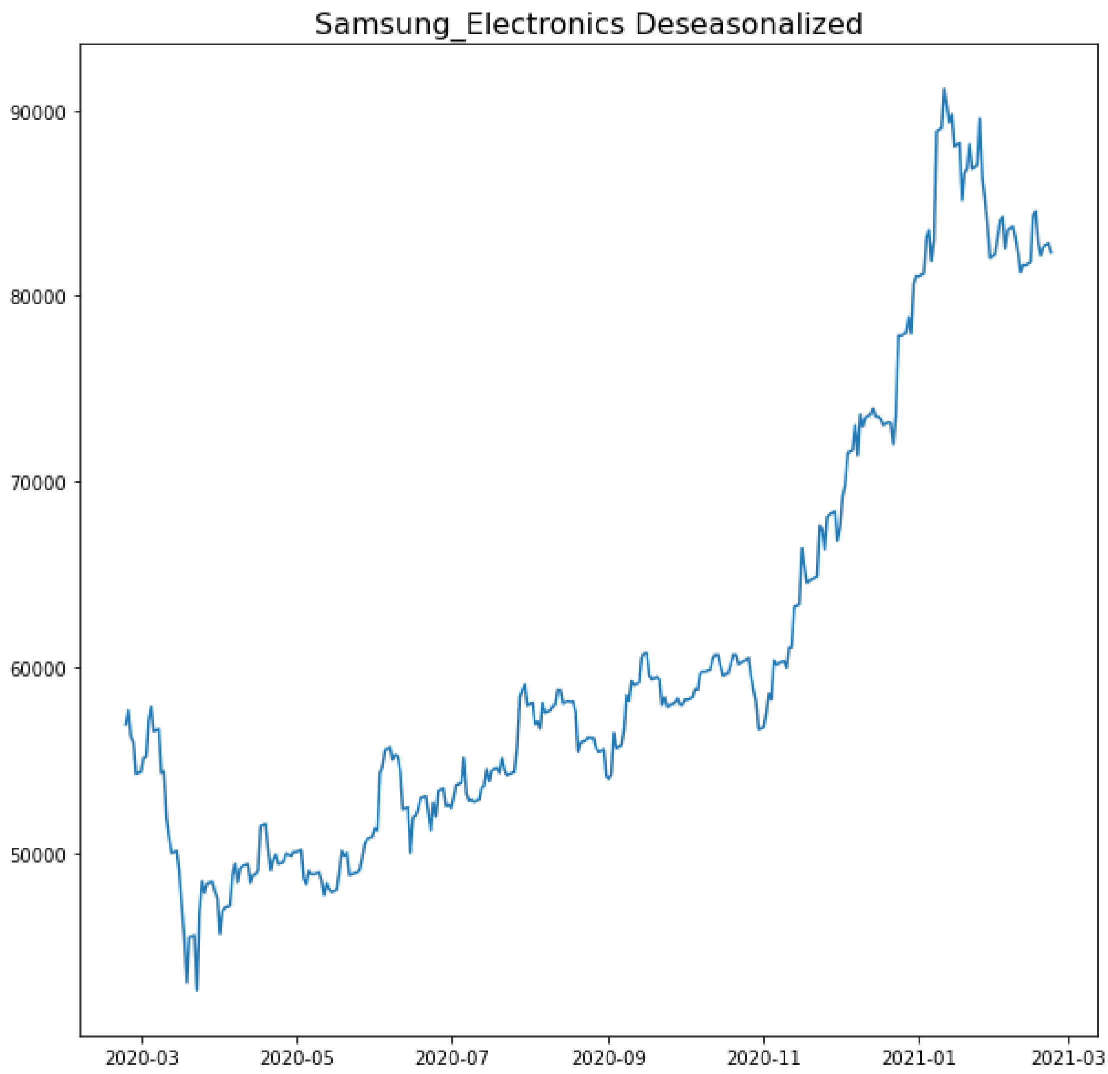
[56]:
```
df_SE['Close'].plot(figsize = (12,6))
```

[56]: <AxesSubplot:>



[57]:
```python
# 시계열 데이터를 분해한다.
result_mul = seasonal_decompose(df_SE['Close'], model='multiplicative',␣
↪extrapolate_trend='freq')

# 계절성 구성요소로 시계열 데이터를 나눈다.
deseasonalized = df_SE.Close.values / result_mul.seasonal

# 도표로 보여준다.

plt.plot(deseasonalized)
plt.title('Samsung_Electronics Deseasonalized', fontsize=16)
plt.plot()
```

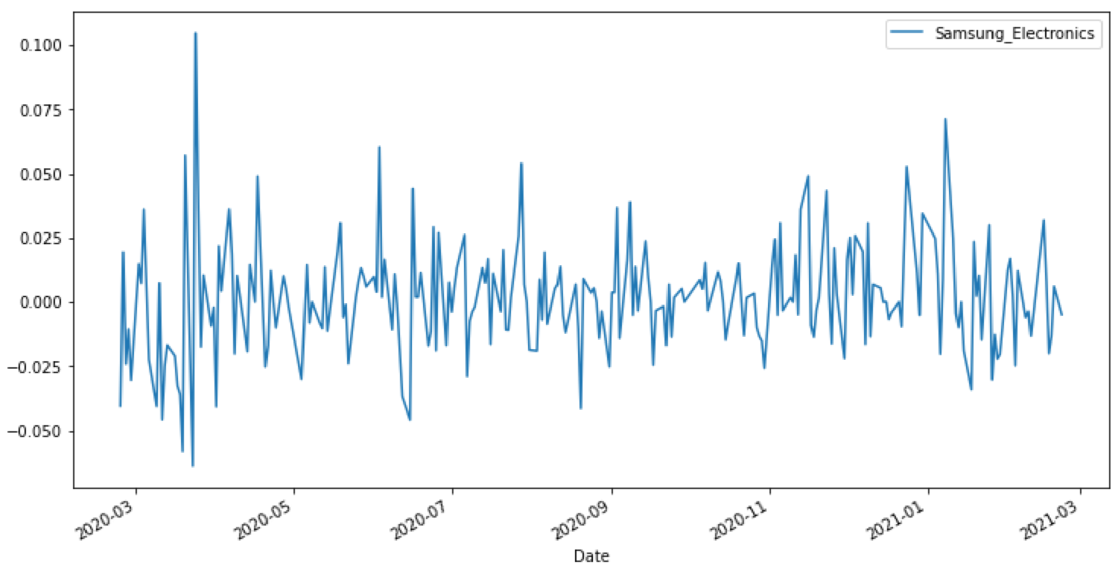
[57]: []



```
[46]: df = pd.merge(df_Samsung_Electronics['Change'], df_SK_hynix['Change'],␣
      ↪left_index=True, right_index=True, how='left').rename(columns ={'Change_x':
      ↪'Samsung_Electronics', 'Change_y':'SK_hynix'})
      df = pd.merge(df, df_LG_Chem['Change'], left_index = True, right_index=True,␣
      ↪how='left').rename(columns = {'Change':'LG_Chem'})
      df = pd.merge(df, df_NAVER_Corporation['Change'], left_index = True,␣
      ↪right_index=True, how='left').rename(columns = {'Change':'NAVER_Corporation'})
      df = pd.merge(df, df_Samsung_Biologics['Change'], left_index = True,␣
      ↪right_index=True, how='left').rename(columns = {'Change':'Samsung_Biologics'})
      df = pd.merge(df, df_Hyundai_Motor_Company['Change'], left_index = True,␣
      ↪right_index=True, how='left').rename(columns = {'Change':
      ↪'Hyundai_Motor_Company'})
      df = pd.merge(df, df_Samsung_SDI['Change'], left_index = True, right_index=True,␣
      ↪how='left').rename(columns = {'Change':'Samsung_SDI'})
      df = pd.merge(df, df_Kakao['Change'], left_index = True, right_index=True,␣
      ↪how='left').rename(columns = {'Change':'Kakao'})
      df = pd.merge(df, df_Celltrion['Change'], left_index = True, right_index=True,␣
      ↪how='left').rename(columns = {'Change':'Celltrion'})
      df = pd.merge(df, df_Kia_Corporation['Change'], left_index = True,␣
      ↪right_index=True, how='left').rename(columns = {'Change':'Kia_Corporation'})
```

```
[47]: columns = ['Samsung_Electronics']
      df[columns].plot(figsize=(12.2,6.4))
```

```
[47]: <AxesSubplot:xlabel='Date'>
```



[48]:

```python
n_obs = 20
df_train, df_test = df[0:-n_obs], df[-n_obs:]

from statsmodels.tsa.stattools import adfuller


def adf_test(df):
    result = adfuller(df.values)
    print('ADF Statistics: %f' % result[0])
    print('p-value: %f' % result[1])
    print('Critical values:')
    for key, value in result[4].items():
        print('\t%s: %.3f' % (key, value))

print('ADF Test: Samsung_Electronics Time series')
adf_test(df_train['Samsung_Electronics'])

print('\n\nADF Test: SK_hynix Time series')
adf_test(df_train['SK_hynix'])

print('\n\nADF Test: LG_Chem Time series')
adf_test(df_train['LG_Chem'])

print('\n\nADF Test: NAVER_Corporation Time series')
adf_test(df_train['NAVER_Corporation'])

print('\n\nADF Test: Samsung_Biologics Time series')
adf_test(df_train['Samsung_Biologics'])

print('\n\nADF Test: Hyundai_Motor_Company Time series')
adf_test(df_train['Hyundai_Motor_Company'])

print('\n\nADF Test: Samsung_SDI Time series')
adf_test(df_train['Samsung_SDI'])

print('\n\nADF Test: Kakao Time series')
adf_test(df_train['Kakao'])

print('\n\nADF Test: Celltrion Time series')
adf_test(df_train['Celltrion'])

print('\n\nADF Test: Kia_Corporation Time series')
adf_test(df_train['Kia_Corporation'])
```

```
ADF Test: Samsung_Electronics Time series
ADF Statistics: -15.546327
p-value: 0.000000
Critical values:
```

```
        1%: -3.460
        5%: -2.874
        10%: -2.574


ADF Test: SK_hynix Time series
ADF Statistics: -8.176693
p-value: 0.000000
Critical values:
        1%: -3.460
        5%: -2.875
        10%: -2.574


ADF Test: LG_Chem Time series
ADF Statistics: -15.725376
p-value: 0.000000
Critical values:
        1%: -3.460
        5%: -2.874
        10%: -2.574


ADF Test: NAVER_Corporation Time series
ADF Statistics: -17.600145
p-value: 0.000000
Critical values:
        1%: -3.460
        5%: -2.874
        10%: -2.574


ADF Test: Samsung_Biologics Time series
ADF Statistics: -15.489672
p-value: 0.000000
Critical values:
        1%: -3.460
        5%: -2.874
        10%: -2.574


ADF Test: Hyundai_Motor_Company Time series
ADF Statistics: -12.574456
p-value: 0.000000
Critical values:
        1%: -3.460
        5%: -2.874
        10%: -2.574
```

```
ADF Test: Samsung_SDI Time series
ADF Statistics: -8.194634
p-value: 0.000000
Critical values:
        1%: -3.460
        5%: -2.875
        10%: -2.574


ADF Test: Kakao Time series
ADF Statistics: -9.060084
p-value: 0.000000
Critical values:
        1%: -3.460
        5%: -2.874
        10%: -2.574


ADF Test: Celltrion Time series
ADF Statistics: -12.100141
p-value: 0.000000
Critical values:
        1%: -3.460
        5%: -2.874
        10%: -2.574


ADF Test: Kia_Corporation Time series
ADF Statistics: -8.712487
p-value: 0.000000
Critical values:
        1%: -3.460
        5%: -2.874
        10%: -2.574
```

[49]:
```
from statsmodels.tsa.stattools import kpss


def kpss_test(df):
    statistic, p_value, n_lags, critical_values = kpss(df.values)

    print(f'KPSS Statistic: {statistic}')
    print(f'p-value: {p_value}')
    print(f'num lags: {n_lags}')
    print('Critial Values:')
```

```python
    for key, value in critical_values.items():
        print(f'{key} : {value}')

print('KPSS Test: Samsung_Electronics Time series')
kpss_test(df_train['Samsung_Electronics'])

print('\n\nKPSS Test: SK_hynix Time series')
kpss_test(df_train['SK_hynix'])

print('\n\nKPSS Test: LG_Chem Time series')
kpss_test(df_train['LG_Chem'])

print('\n\nKPSS Test: NAVER_Corporation Time series')
kpss_test(df_train['NAVER_Corporation'])

print('\n\nKPSS Test: Samsung_Biologics Time series')
kpss_test(df_train['Samsung_Biologics'])

print('\n\nKPSS Test: Hyundai_Motor_Company Time series')
kpss_test(df_train['Hyundai_Motor_Company'])

print('\n\nKPSS Test: Samsung_SDI Time series')
kpss_test(df_train['Samsung_SDI'])

print('\n\nKPSS Test: Kakao Time series')
kpss_test(df_train['Kakao'])

print('\n\nKPSS Test: Celltrion Time series')
kpss_test(df_train['Celltrion'])

print('\n\nKPSS Test: Kia_Corporation Time series')
kpss_test(df_train['Kia_Corporation'])
```

```
KPSS Test: Samsung_Electronics Time series
KPSS Statistic: 0.5193574582566969
p-value: 0.03730687877101421
num lags: 15
Critial Values:
10% : 0.347
5% : 0.463
2.5% : 0.574
1% : 0.739


KPSS Test: SK_hynix Time series
KPSS Statistic: 0.5653878160335788
p-value: 0.026939681073518282
num lags: 15
```

```
Critial Values:
10% : 0.347
5% : 0.463
2.5% : 0.574
1% : 0.739


KPSS Test: LG_Chem Time series
KPSS Statistic: 0.12437754401303307
p-value: 0.1
num lags: 15
Critial Values:
10% : 0.347
5% : 0.463
2.5% : 0.574
1% : 0.739


KPSS Test: NAVER_Corporation Time series
KPSS Statistic: 0.14844396042715371
p-value: 0.1
num lags: 15
Critial Values:
10% : 0.347
5% : 0.463
2.5% : 0.574
1% : 0.739


KPSS Test: Samsung_Biologics Time series
KPSS Statistic: 0.15508263246185267
p-value: 0.1
num lags: 15
Critial Values:
10% : 0.347
5% : 0.463
2.5% : 0.574
1% : 0.739


KPSS Test: Hyundai_Motor_Company Time series
KPSS Statistic: 0.2693733832072474
p-value: 0.1
num lags: 15
Critial Values:
10% : 0.347
5% : 0.463
2.5% : 0.574
```

```
1% : 0.739


KPSS Test: Samsung_SDI Time series
KPSS Statistic: 0.20493978086912173
p-value: 0.1
num lags: 15
Critial Values:
10% : 0.347
5% : 0.463
2.5% : 0.574
1% : 0.739


KPSS Test: Kakao Time series
KPSS Statistic: 0.1164330285144876
p-value: 0.1
num lags: 15
Critial Values:
10% : 0.347
5% : 0.463
2.5% : 0.574
1% : 0.739


KPSS Test: Celltrion Time series
KPSS Statistic: 0.138232335539308
p-value: 0.1
num lags: 15
Critial Values:
10% : 0.347
5% : 0.463
2.5% : 0.574
1% : 0.739


KPSS Test: Kia_Corporation Time series
KPSS Statistic: 0.40171412015048596
p-value: 0.07641632752134225
num lags: 15
Critial Values:
10% : 0.347
5% : 0.463
2.5% : 0.574
1% : 0.739
```

```
C:\ProgramData\Anaconda3\envs\muiiya\lib\site-
packages\statsmodels\tsa\stattools.py:1850: FutureWarning: The behavior of using
```

```
nlags=None will change in release 0.13.Currently nlags=None is the same as
nlags="legacy", and so a sample-size lag length is used. After the next release,
the default will change to be the same as nlags="auto" which uses an automatic
lag length selection method. To silence this warning, either use "auto" or
"legacy"
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\envs\muiiya\lib\site-
packages\statsmodels\tsa\stattools.py:1885: InterpolationWarning: The test
statistic is outside of the range of p-values available in the
look-up table. The actual p-value is greater than the p-value returned.

  warnings.warn(
C:\ProgramData\Anaconda3\envs\muiiya\lib\site-
packages\statsmodels\tsa\stattools.py:1885: InterpolationWarning: The test
statistic is outside of the range of p-values available in the
look-up table. The actual p-value is greater than the p-value returned.

  warnings.warn(
C:\ProgramData\Anaconda3\envs\muiiya\lib\site-
packages\statsmodels\tsa\stattools.py:1885: InterpolationWarning: The test
statistic is outside of the range of p-values available in the
look-up table. The actual p-value is greater than the p-value returned.

  warnings.warn(
C:\ProgramData\Anaconda3\envs\muiiya\lib\site-
packages\statsmodels\tsa\stattools.py:1885: InterpolationWarning: The test
statistic is outside of the range of p-values available in the
look-up table. The actual p-value is greater than the p-value returned.

  warnings.warn(
C:\ProgramData\Anaconda3\envs\muiiya\lib\site-
packages\statsmodels\tsa\stattools.py:1885: InterpolationWarning: The test
statistic is outside of the range of p-values available in the
look-up table. The actual p-value is greater than the p-value returned.

  warnings.warn(
C:\ProgramData\Anaconda3\envs\muiiya\lib\site-
packages\statsmodels\tsa\stattools.py:1885: InterpolationWarning: The test
statistic is outside of the range of p-values available in the
look-up table. The actual p-value is greater than the p-value returned.

  warnings.warn(
C:\ProgramData\Anaconda3\envs\muiiya\lib\site-
packages\statsmodels\tsa\stattools.py:1885: InterpolationWarning: The test
statistic is outside of the range of p-values available in the
look-up table. The actual p-value is greater than the p-value returned.

  warnings.warn(
```

```
[ ]:
```

```
[ ]:
```