In [ ]:

```
import matplotlib.pyplot as plt
import pandas as pd
df = pd.read_csv('/content/drive/My Drive/SyncPC/Data Analytic/slide/TimeSeries/a10.csv',
parse_dates=['date'], index_col='date')
df
```

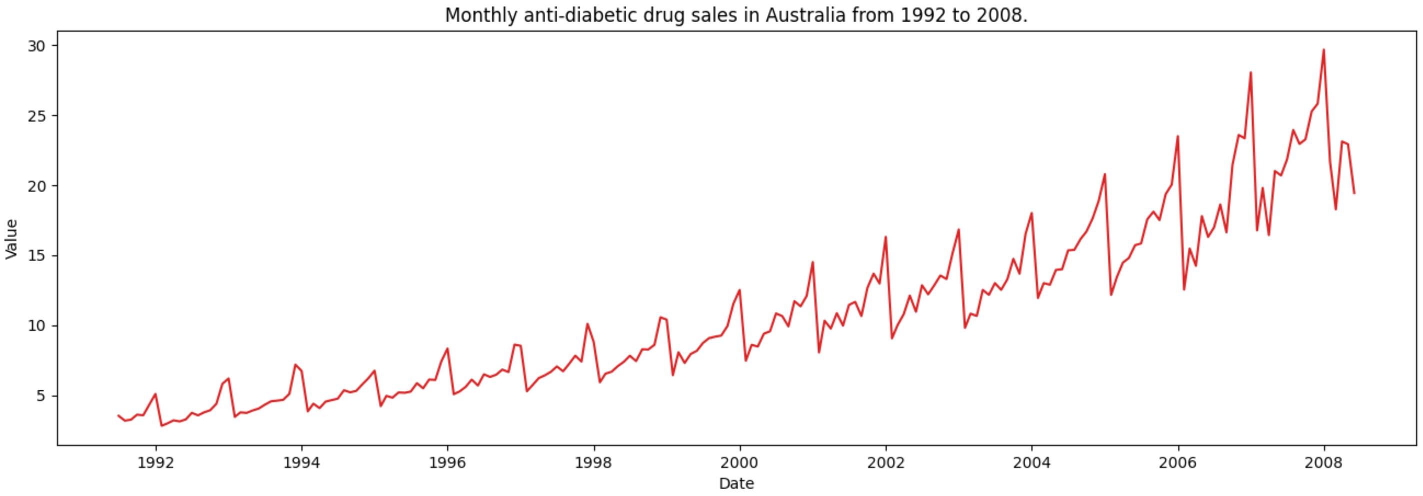
Out[ ]:

	value
date	
1991-07-01	3.526591
1991-08-01	3.180891
1991-09-01	3.252221
1991-10-01	3.611003
1991-11-01	3.565869
...	...
2008-02-01	21.654285
2008-03-01	18.264945
2008-04-01	23.107677
2008-05-01	22.912510
2008-06-01	19.431740

204 rows × 1 columns

In [ ]:

```
# Draw Plot
def plot_df(df, x, y, title="", xlabel='Date', ylabel='Value', dpi=100):
    plt.figure(figsize=(16,5), dpi=dpi)
    plt.plot(x, y, color='tab:red')
    plt.gca().set(title=title, xlabel=xlabel, ylabel=ylabel)
    plt.show()

plot_df(df, x=df.index, y=df.value, title='Monthly anti-diabetic drug sales in Australia
from 1992 to 2008.')
```



Draw Monthly anti-diabetic drug sales in Australia from 1992 to 1993.

In [ ]:

```
df['year'] = [d.year for d in df.index]
df
```

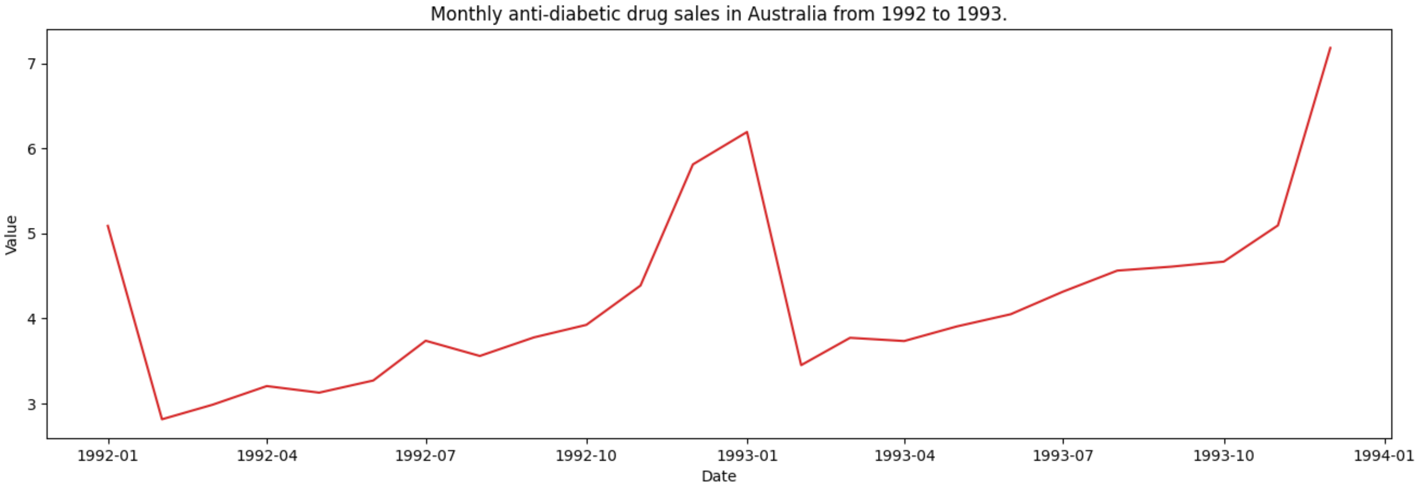
Out[ ]:

date	value	year
1991-07-01	3.526591	1991
1991-08-01	3.180891	1991
1991-09-01	3.252221	1991
1991-10-01	3.611003	1991
1991-11-01	3.565869	1991
...	...	...
2008-02-01	21.654285	2008
2008-03-01	18.264945	2008
2008-04-01	23.107677	2008
2008-05-01	22.912510	2008
2008-06-01	19.431740	2008

204 rows × 2 columns

In [ ]:

```
df_1992_1993 = df[(df.year == 1992) | (df.year == 1993)]
plot_df(df_1992_1993, x=df_1992_1993.index, y=df_1992_1993.value, title='Monthly anti-diabetic drug sales in Australia from 1992 to 1993.')
```

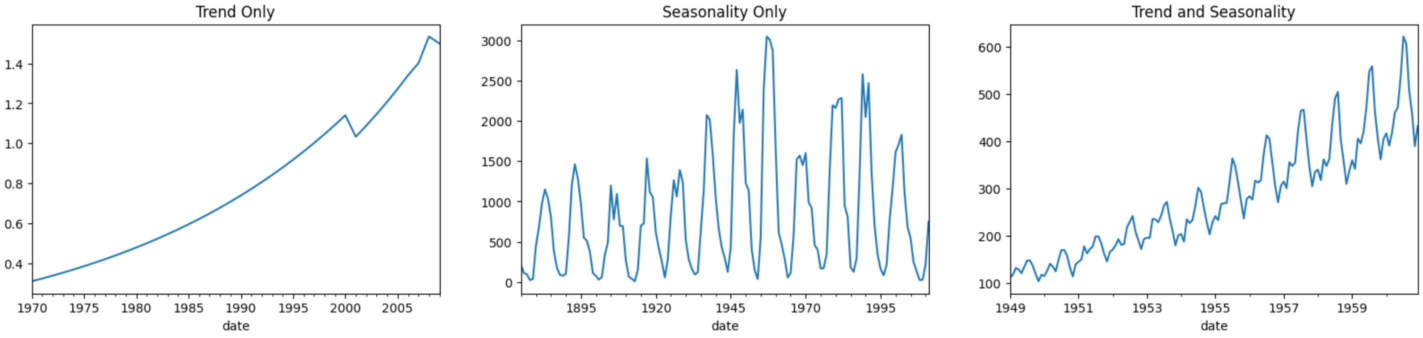


In [ ]:

```
fig, axes = plt.subplots(1,3, figsize=(20,4), dpi=100)
pd.read_csv('https://raw.githubusercontent.com/selva86/datasets/master/guinearice.csv', parse_dates=['date'], index_col='date').plot(title='Trend Only', legend=False, ax=axes[0])

pd.read_csv('https://raw.githubusercontent.com/selva86/datasets/master/sunspotarea.csv', parse_dates=['date'], index_col='date').plot(title='Seasonality Only', legend=False, ax=axes[1])

pd.read_csv('https://raw.githubusercontent.com/selva86/datasets/master/AirPassengers.csv', parse_dates=['date'], index_col='date').plot(title='Trend and Seasonality', legend=False, ax=axes[2])
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7017aa2f28>
```


Trend Only

Seasonality Only

Trend and Seasonality

Decompose a series into trend, seasonal and error

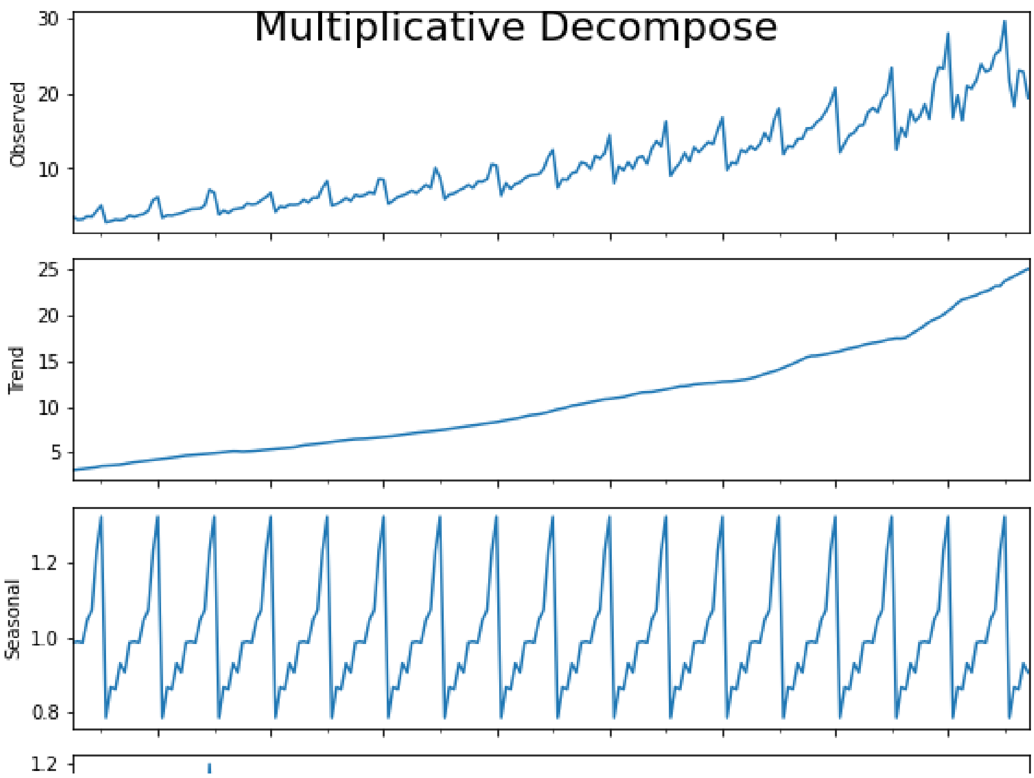
Additive time series: Value = Base Level + Trend + Seasonality + Error

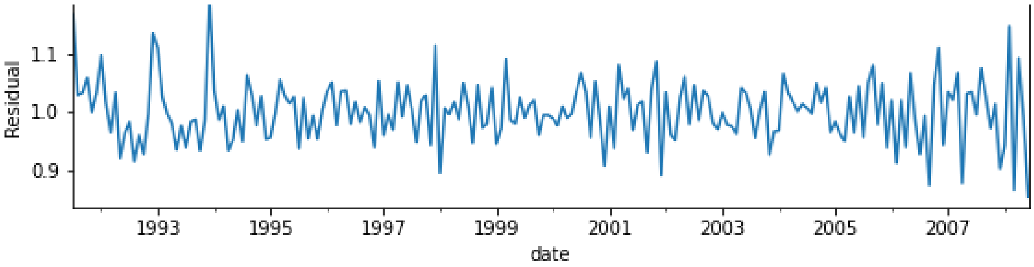
Multiplicative Time Series: Value = Base Level x Trend x Seasonality x Error

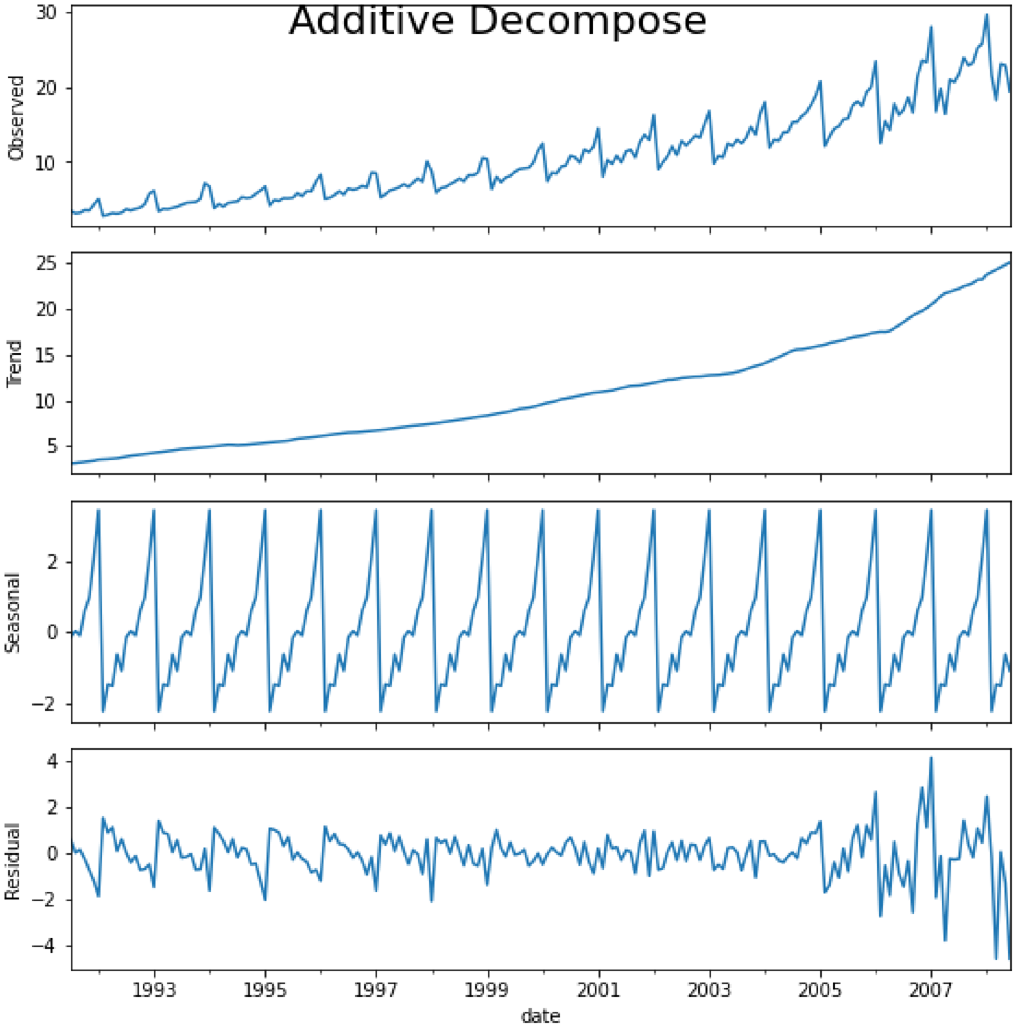
In [ ]:

```
from statsmodels.tsa.seasonal import seasonal_decompose
from dateutil.parser import parse

# Import Data
df = pd.read_csv('/content/drive/My Drive/SyncPC/Data Analytic/slide/TimeSeries/a10.csv', parse_dates=['date'], index_col='date')

# Multiplicative Decomposition
result_mul = seasonal_decompose(df['value'], model='multiplicative', extrapolate_trend='freq')

# Additive Decomposition
result_add = seasonal_decompose(df['value'], model='additive', extrapolate_trend='freq')

# Plot
plt.rcParams.update({'figure.figsize': (8,8)})
result_mul.plot().suptitle('Multiplicative Decompose', fontsize=22)
result_add.plot().suptitle('Additive Decompose', fontsize=22)
plt.show()
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning: pandas.util.testing is deprecated. Use the functions in the public API at pandas.testing instead.
  import pandas.util.testing as tm
```







In [ ]:

```
# Extract the Components ----
# Actual Values = Product of (Seasonal * Trend * Resid)
df_reconstructed = pd.concat([result_mul.seasonal, result_mul.trend, result_mul.resid, r
esult_mul.observed], axis=1)
df_reconstructed.columns = ['seas', 'trend', 'resid', 'actual_values']
df_reconstructed.head()
```

Out[ ]:

	seas	trend	resid	actual_values
date				
1991-07-01	0.987845	3.060085	1.166629	3.526591
1991-08-01	0.990481	3.124765	1.027745	3.180891
1991-09-01	0.987476	3.189445	1.032615	3.252221
1991-10-01	1.048329	3.254125	1.058513	3.611003
1991-11-01	1.074527	3.318805	0.999923	3.565869

In [ ]:

```
df.head()
```

Out[ ]:

date	value
1991-07-01	3.526591
1991-08-01	3.180891
1991-09-01	3.252221
1991-10-01	3.611003
1991-11-01	3.565869

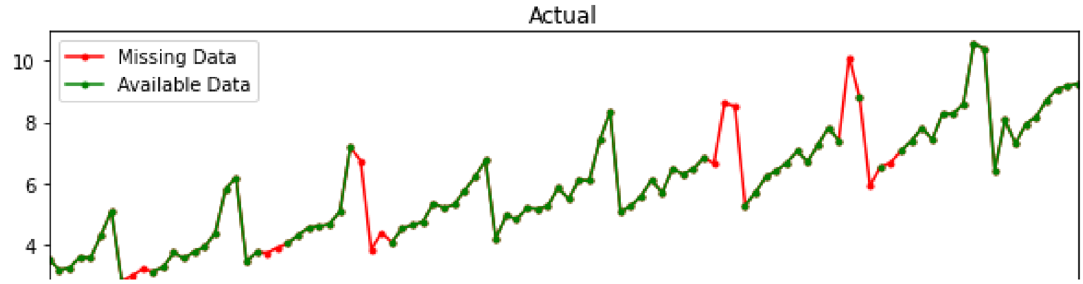
Filling missing values

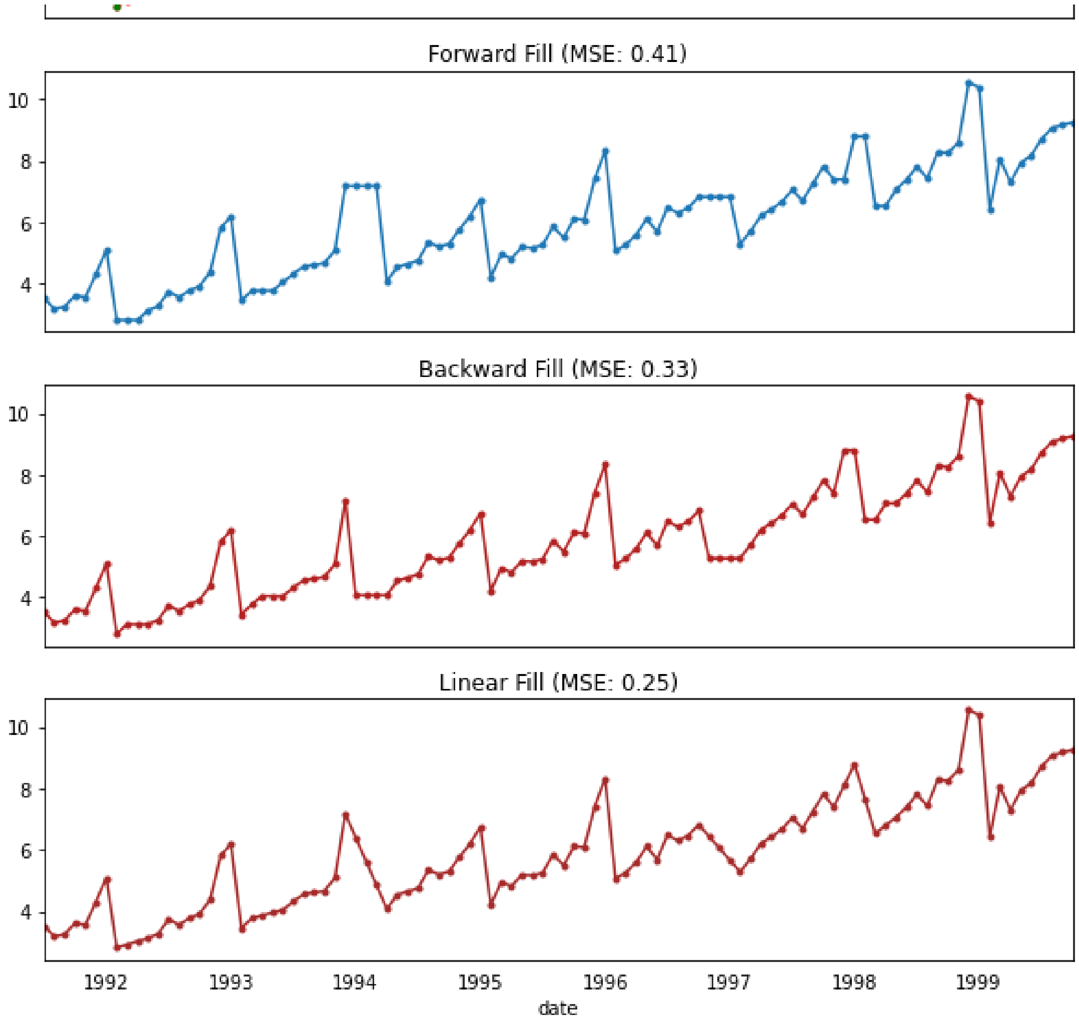
In [ ]:

```
# # Generate dataset
import numpy as np
from scipy.interpolate import interp1d
from sklearn.metrics import mean_squared_error
df_orig = pd.read_csv('/content/drive/My Drive/SyncPC/Data Analytic/slide/TimeSeries/a10.csv', parse_dates=['date'], index_col='date').head(100)
df = pd.read_csv('/content/drive/My Drive/SyncPC/Data Analytic/slide/TimeSeries/a10_missing.csv', parse_dates=['date'], index_col='date').head(100)

fig, axes = plt.subplots(4, 1, sharex=True, figsize=(10, 12))
plt.rcParams.update({'xtick.bottom' : False})


## 1. Actual -------------------------------
df_orig.plot(title='Actual', ax=axes[0], label='Actual', color='red', style=".-")
df.plot(title='Actual', ax=axes[0], label='Actual', color='green', style=".-")
axes[0].legend(["Missing Data", "Available Data"])

## 2. Forward Fill --------------------------
df_ffill = df.ffill()
error = np.round(mean_squared_error(df_orig['value'], df_ffill['value']), 2)
df_ffill['value'].plot(title='Forward Fill (MSE: ' + str(error) +")", ax=axes[1], label='Forward Fill', style=".-")

## 3. Backward Fill -------------------------
df_bfill = df.bfill()
error = np.round(mean_squared_error(df_orig['value'], df_bfill['value']), 2)
df_bfill['value'].plot(title="Backward Fill (MSE: " + str(error) +")", ax=axes[2], label='Back Fill', color='firebrick', style=".-")

## 4. Linear Interpolation ------------------
df['rownum'] = np.arange(df.shape[0])
df_nona = df.dropna(subset = ['value'])
f = interp1d(df_nona['rownum'], df_nona['value'])
df['linear_fill'] = f(df['rownum'])
error = np.round(mean_squared_error(df_orig['value'], df['linear_fill']), 2)
df['linear_fill'].plot(title="Linear Fill (MSE: " + str(error) +")", ax=axes[3], label='Cubic Fill', color='brown', style=".-")
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fad81452160>
```

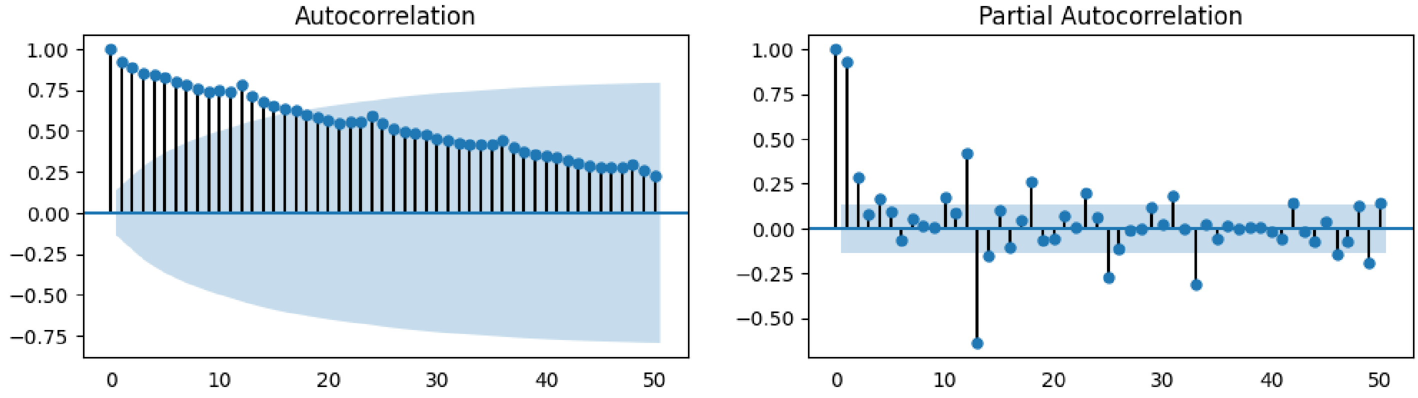




Autocorrelation and Partial autocorrelation

In [ ]:

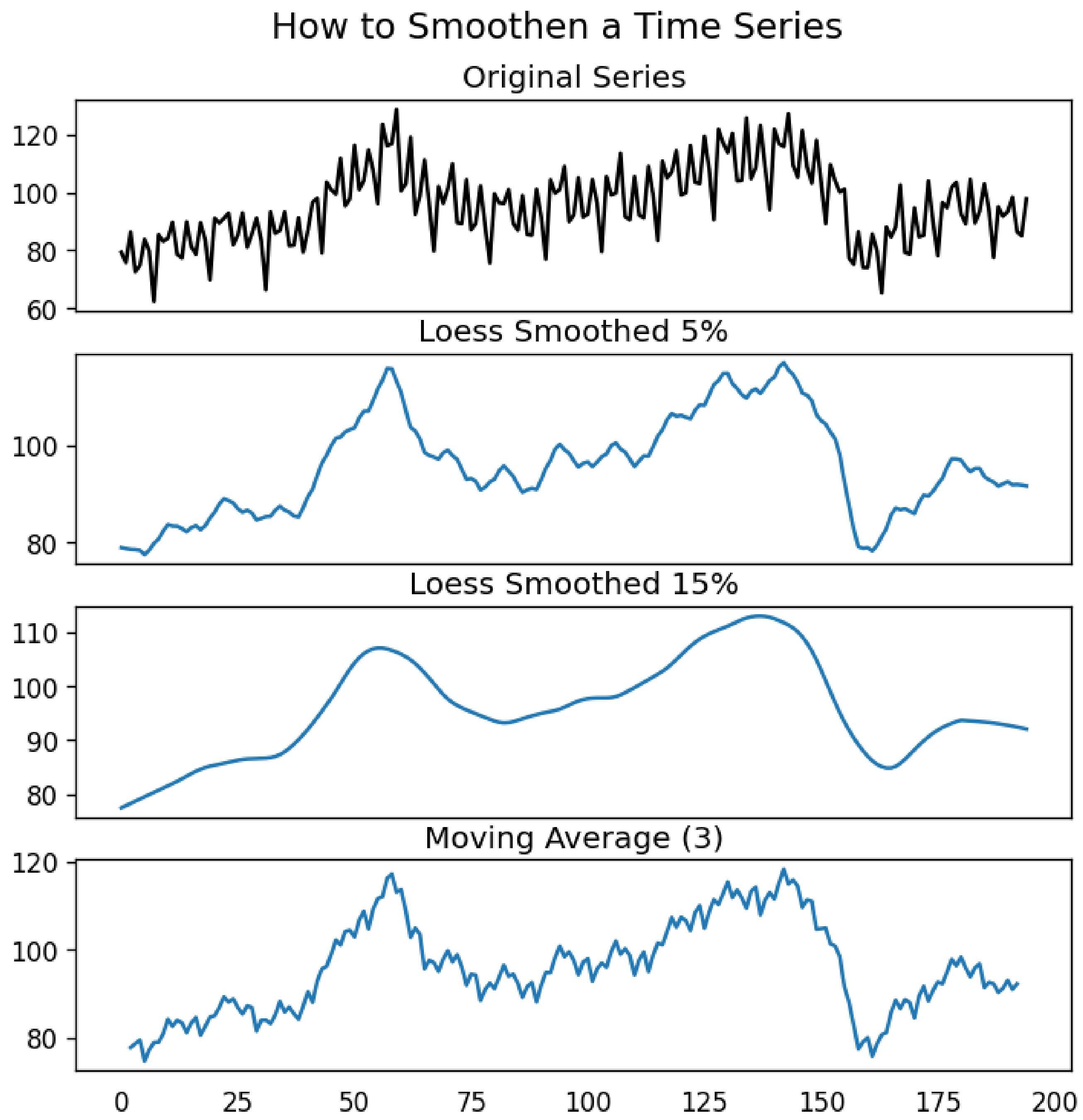
```
from statsmodels.tsa.stattools import acf, pacf
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
%matplotlib inline

df = pd.read_csv('/content/drive/My Drive/SyncPC/Data Analytic/slide/TimeSeries/a10.csv')

# Calculate ACF and PACF upto 50 lags
# acf_50 = acf(df.value, nlags=50)
# pacf_50 = pacf(df.value, nlags=50)

# Draw Plot
fig, axes = plt.subplots(1,2,figsize=(12,3), dpi= 100)
plot_acf(df.value.tolist(), lags=50,ax=axes[0])
plot_pacf(df.value.tolist(), lags=50, ax=axes[1]);
```



Smoothing

In [ ]:

```
from statsmodels.nonparametric.smoothers_lowess import lowess
plt.rcParams.update({'xtick.bottom' : False, 'axes.titlepad':5})

# Import
df_orig = pd.read_csv('/content/drive/My Drive/SyncPC/Data Analytic/slide/TimeSeries/elec
equip.csv' )

# 1. Moving Average
df_ma = df_orig.value.rolling(5, center=True).mean()

# 2. Loess Smoothing (5% and 15%)
df_loess_5 = pd.DataFrame(lowess(df_orig.value, np.arange(len(df_orig.value)), frac=0.05
)[:, 1], index=df_orig.index, columns=['value'])
df_loess_15 = pd.DataFrame(lowess(df_orig.value, np.arange(len(df_orig.value)), frac=0.1
5)[:, 1], index=df_orig.index, columns=['value'])

# Plot
fig, axes = plt.subplots(4,1, figsize=(7, 7), sharex=True, dpi=120)
df_orig['value'].plot(ax=axes[0], color='k', title='Original Series')
df_loess_5['value'].plot(ax=axes[1], title='Loess Smoothed 5%')
df_loess_15['value'].plot(ax=axes[2], title='Loess Smoothed 15%')
df_ma.plot(ax=axes[3], title='Moving Average (3)')
fig.suptitle('How to Smoothen a Time Series', y=0.95, fontsize=14)
plt.show()
```


How to Smoothen a Time Series
Original Series
Loess Smoothed 5%
Loess Smoothed 15%
Moving Average (3)