R09546042_TSA_HW_02

October 10, 2021

[57]:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.linear_model import LinearRegression
from statsmodels.tsa.holtwinters import ExponentialSmoothing as HWES
```

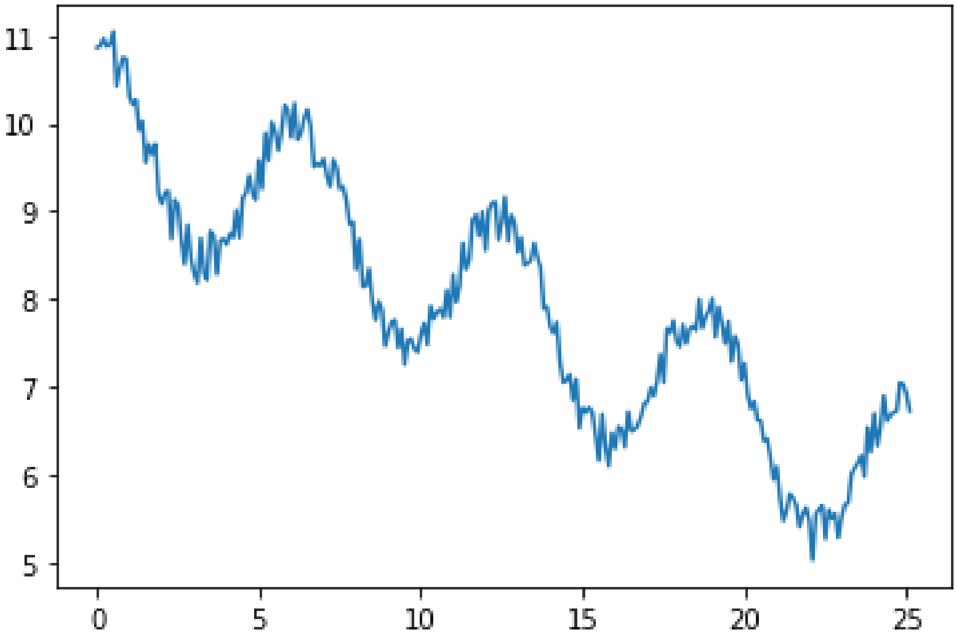
1 Q1

1.1 a.

use cos() for waving time-series, -x/6 for dwindling trend, and normal distribution for noise.

[58]:

```
x = np.arange(0,8*np.pi,0.1)    # start,stop,step
y = 10+np.cos(x)-x/6

y_noise = []

for i in y:
    n = 0.3
    noise = np.random.uniform(-n,n)
    y_noise.append(i + noise)

plt.plot(x,y_noise)
plt.show()
```

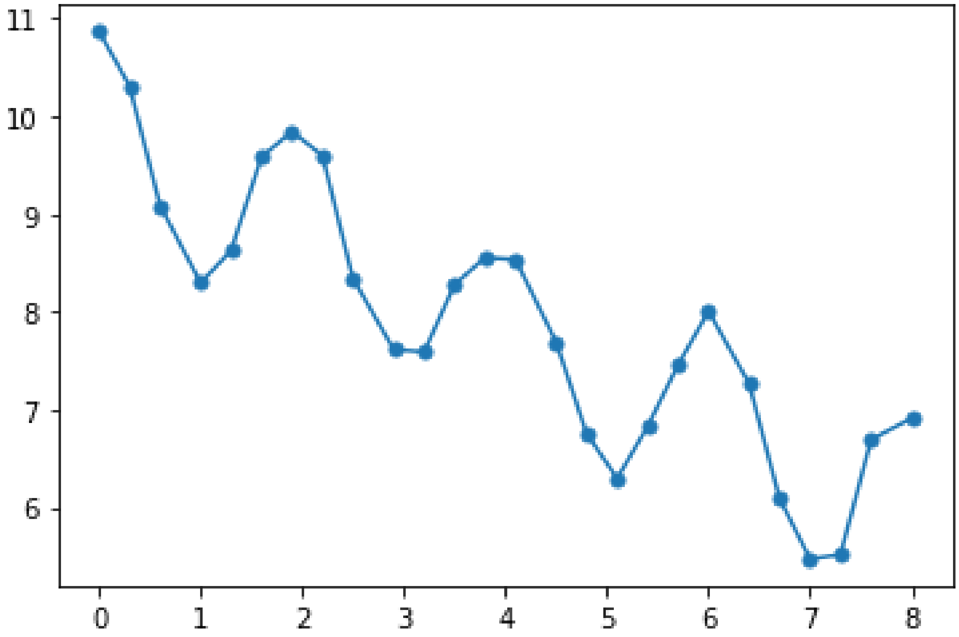


1.2 b

period is 3.14, close to 3.

```
[59]: df_original_series = pd.DataFrame({
          'period': x,
          'demand': y_noise
      })

      # include average interger.
      df_interger_series = df_original_series[df_original_series['period'] % 1 == 0]
      df_interger_series['period'] = df_interger_series['period'].div(np.pi).round(1)
      plt.plot(df_interger_series['period'],df_interger_series['demand'],␣
       ↪marker='o',markersize=5)

      # deseasonalize
      series_deseasonalization = df_interger_series.loc[:, 'demand'].rolling(3).
       ↪mean().dropna()
      series_deseasonalization=series_deseasonalization.
       ↪drop([series_deseasonalization.index[22],series_deseasonalization.index[23]])

      # deseasonalize dataframe
      df_deseasonalization = pd.DataFrame({
          'Quater': sum([[1,2,3]*8,[1,2]],[]),
          'Period': df_interger_series['period'] ,
          'Demand': df_interger_series['demand'] ,
          'Deseasonalized_Demand': series_deseasonalization
      })
```

```
<ipython-input-59-ab869889bd6c>:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_interger_series['period'] =
df_interger_series['period'].div(np.pi).round(1)
```



translate x axis to # of pi

1.2.1 c

[60]:
```
#build up regressing model
reg = LinearRegression().fit(np.asarray(df_deseasonalization.loc[20:230,␣
 ↪'Period']).reshape(-1, 1),
                            df_deseasonalization.loc[:,␣
 ↪'Deseasonalized_Demand'].dropna())
#predict nan value
values = pd.Series(reg.predict(np.asarray(df_deseasonalization.loc[:,␣
 ↪'Period']).reshape(-1,1)))

df_deseasonalization.loc[0,['Deseasonalized_Demand']] = values[0]
df_deseasonalization.loc[10,['Deseasonalized_Demand']] = values[10]
df_deseasonalization.loc[240,['Deseasonalized_Demand']] = values[24]
df_deseasonalization.loc[250,['Deseasonalized_Demand']] = values[25]

# calculate seansonality factor
df_deseasonalization.loc[:, 'Seasonality'] = (df_deseasonalization.loc[:,␣
 ↪'Demand'] / df_deseasonalization.loc[:, 'Deseasonalized_Demand'])

df_Seasonality_bar= pd.DataFrame({
    'Quater': sum([[1,2,3]*8,[1,2]],[]),
    'Period': df_interger_series['period'] ,
    'Demand': df_interger_series['demand'] ,
    'Deseasonalized_Demand': series_deseasonalization
})

df_seasonality = df_deseasonalization.groupby(['Quater'], as_index=False).mean()
df_seasonality.loc[:, 'Seasonality_bar'] = df_seasonality.loc[:, 'Seasonality']
df_seasonality = df_seasonality[['Quater','Seasonality_bar']]

df_deseasonalization = pd.merge(df_deseasonalization,df_seasonality).
 ↪sort_values('Period')
```

[61]:

```
df_deseasonalization
```

[61]:

```
    Quater  Period     Demand  Deseasonalized_Demand  Seasonality  \
0        1     0.0  10.867433              10.022446     1.084310
9        2     0.3  10.307763               8.407980     1.225950
18       3     0.6   9.086565              10.087254     0.900797
1        1     1.0   8.297640               9.230656     0.898922
10       2     1.3   8.627718               8.670641     0.995050
19       3     1.6   9.585892               8.837084     1.084735
2        1     1.9   9.843057               9.352223     1.052483
11       2     2.2   9.598873               9.675941     0.992035
20       3     2.5   8.340692               9.260874     0.900638
3        1     2.9   7.615876               8.518480     0.894042
12       2     3.2   7.591517               7.849361     0.967151
21       3     3.5   8.279429               7.828941     1.057541
4        1     3.8   8.556947               8.142631     1.050882
13       2     4.1   8.536198               8.457525     1.009302
22       3     4.5   7.692330               8.261825     0.931069
5        1     4.8   6.759370               7.662633     0.882121
14       2     5.1   6.298355               6.916685     0.910603
23       3     5.4   6.830760               6.629495     1.030359
6        1     5.7   7.458044               6.862386     1.086800
15       2     6.0   8.007461               7.432088     1.077417
24       3     6.4   7.266487               7.577331     0.958977
7        1     6.7   6.091357               7.121768     0.855315
```

```
16        2      7.0    5.474414                   6.277419     0.872080
25        3      7.3    5.519303                   5.695025     0.969145
8         1      7.6    6.696175                   6.188090     1.082107
17        2      8.0    6.914511                   5.986282     1.155059

    Seasonality_bar
0           0.987442
9           1.022739
18          0.979158
1           0.987442
10          1.022739
19          0.979158
2           0.987442
11          1.022739
20          0.979158
3           0.987442
12          1.022739
21          0.979158
4           0.987442
13          1.022739
22          0.979158
5           0.987442
14          1.022739
23          0.979158
6           0.987442
15          1.022739
24          0.979158
7           0.987442
16          1.022739
25          0.979158
8           0.987442
17          1.022739
```

1.2.2 d

[62]:

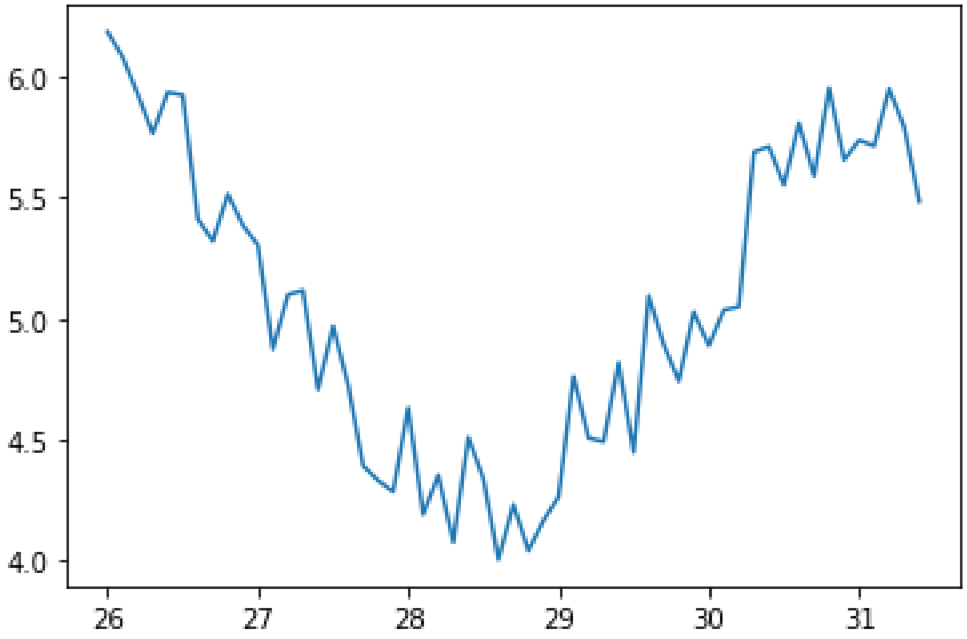
```
df_deseasonalization.loc[:, 'Forecast'] =  (reg.predict(np.
 ↪asarray(df_deseasonalization.loc[:, 'Period']).reshape(-1,1)) *␣
 ↪df_deseasonalization.loc[:, 'Seasonality_bar'])
df_deseasonalization.loc[:, 'Error'] =  (df_deseasonalization.loc[:,␣
 ↪'Demand']-df_deseasonalization.loc[:, 'Forecast'])
df_deseasonalization.loc[:, 'Error_Squre'] =  (df_deseasonalization.loc[:,␣
 ↪'Error']*df_deseasonalization.loc[:, 'Error'])
MSE = df_deseasonalization['Error_Squre'].sum()/len(df_deseasonalization)
MAPE = ((abs(df_deseasonalization.loc[:, 'Error']) / abs(df_deseasonalization.
 ↪loc[:, 'Demand'])).sum())*100/len(df_deseasonalization)

print("MSE:",MSE.round(3))
print("MAPE:",MAPE.round(3))
```

```
MSE: 0.523
MAPE: 8.641
```

1.2.3 e

[63]:

```
x = np.arange(26.0,10*np.pi,0.1)
y = 10+np.cos(x)-x/6

y_noise = []

for i in y:
    n = 0.3
    noise = np.random.uniform(-n,n)
    y_noise.append(i + noise)

plt.plot(x,y_noise)
plt.show()
```



[64]:

```
#true model
df_true_model = pd.DataFrame({
    'Period': x,
    'Demand': y_noise,
    'Forecast':y,
})
```

```
df_true_model.loc[:, 'Error'] =  (df_true_model.loc[:, 'Demand']-df_true_model.
 ↪loc[:, 'Forecast'])
df_true_model.loc[:, 'Error_Squre'] =  (df_true_model.loc[:,␣
 ↪'Error']*df_true_model.loc[:, 'Error'])
MSE = df_true_model['Error_Squre'].sum()/len(df_true_model)
MAPE = ((abs(df_true_model.loc[:, 'Error']) / abs(df_true_model.loc[:,␣
 ↪'Demand'])).sum())*100/len(df_true_model)

print("MSE:",MSE.round(3))
print("MAPE:",MAPE.round(3))
```

```
MSE: 0.035
MAPE: 3.28
```

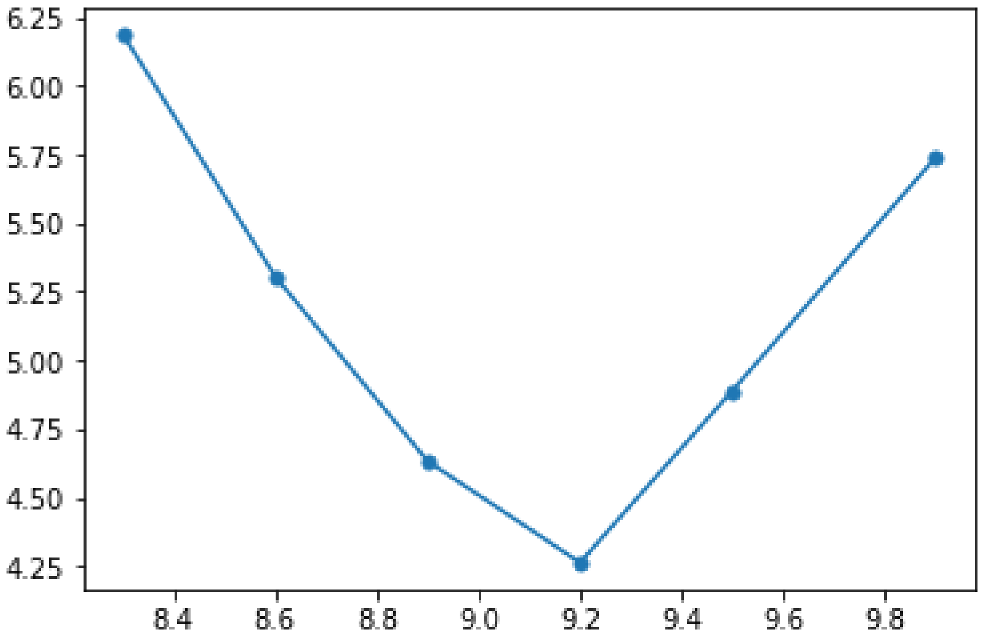
[66]:

```
#time-series model
df_time_series_model = pd.DataFrame({
    'Period': x,
    'Demand': y_noise
})


df_interger_series = df_time_series_model[df_time_series_model['Period'].
 ↪round(1) % 1.0 == 0]
df_interger_series['Period'] = df_interger_series['Period'].div(np.pi).round(1)
plt.plot(df_interger_series['Period'],df_interger_series['Demand'],␣
 ↪marker='o',markersize=5)

df_time_series_deseasonalization= pd.DataFrame({   })
df_time_series_deseasonalization = pd.
 ↪merge(df_interger_series,df_deseasonalization, how="outer").
 ↪sort_values('Period')
df_time_series_deseasonalization.loc[:,'Quater']=sum([[1,2,3]*10,[1,2]],[])

df_time_series_deseasonalization=df_time_series_deseasonalization.
 ↪drop(['Seasonality_bar'], axis=1)
df_time_series_deseasonalization = pd.
 ↪merge(df_time_series_deseasonalization,df_seasonality,how = "outer").
 ↪sort_values('Period')

df_time_series_deseasonalization.loc[:, 'Forecast'] =  (reg.predict(np.
 ↪asarray(df_time_series_deseasonalization.loc[:, 'Period']).reshape(-1,1)) *␣
 ↪df_time_series_deseasonalization.loc[:, 'Seasonality_bar'])
df_time_series_deseasonalization.loc[:, 'Error'] = ␣
 ↪(df_time_series_deseasonalization.loc[:,␣
 ↪'Demand']-df_time_series_deseasonalization.loc[:, 'Forecast'])
```

```
df_time_series_deseasonalization.loc[:, 'Error_Squre'] = ␣
 ↪(df_time_series_deseasonalization.loc[:,␣
 ↪'Error']*df_time_series_deseasonalization.loc[:, 'Error'])
MSE = df_time_series_deseasonalization['Error_Squre'].sum()/
 ↪len(df_time_series_deseasonalization)
MAPE = ((abs(df_time_series_deseasonalization.loc[:, 'Error']) /␣
 ↪abs(df_time_series_deseasonalization.loc[:, 'Demand'])).sum())*100/
 ↪len(df_deseasonalization)

print("MSE:",MSE.round(3))
print("MAPE:",MAPE.round(3))
```

```
MSE: 0.513
MAPE: 11.534
```

```
<ipython-input-66-45637702f052>:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_interger_series['Period'] =
df_interger_series['Period'].div(np.pi).round(1)
```

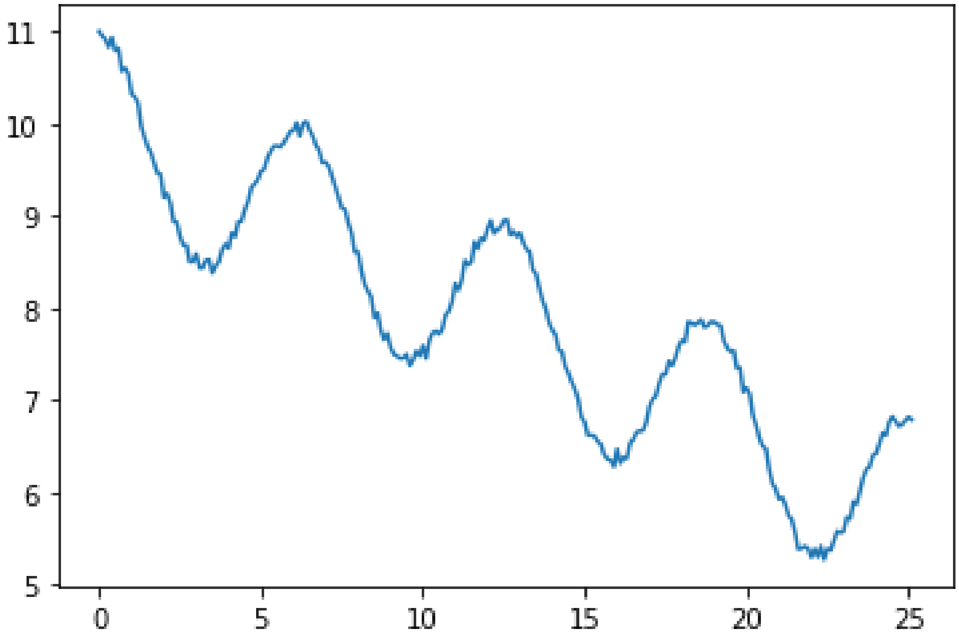


1.2.4 f

change noise factor to 0.1 only, 3 time less than previous data

[67]:

```
x = np.arange(0,8*np.pi,0.1)    # start,stop,step
y = 10+np.cos(x)-x/6

y_noise = []

for i in y:
    n = 0.1
    noise = np.random.uniform(-n,n)
    y_noise.append(i + noise)

plt.plot(x,y_noise)
plt.show()
```

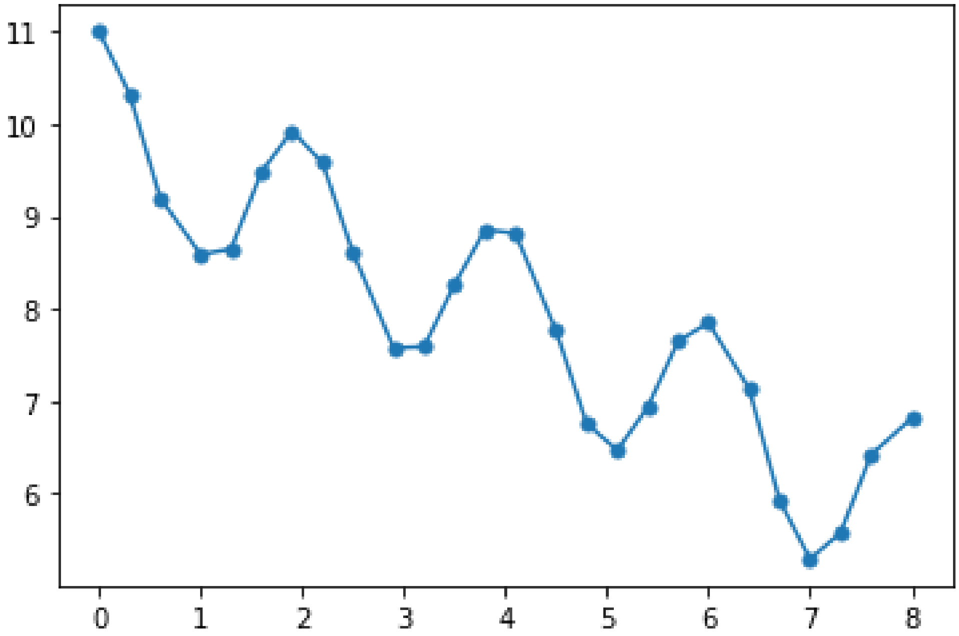


[68]:

```
df_original_series = pd.DataFrame({
    'period': x,
    'demand': y_noise
})

# include average interger.
df_interger_series = df_original_series[df_original_series['period'] % 1 == 0]
df_interger_series['period'] = df_interger_series['period'].div(np.pi).round(1)
plt.plot(df_interger_series['period'],df_interger_series['demand'],␣
 ↪marker='o',markersize=5)

# deseasonalize
```

```
series_deseasonalization = df_interger_series.loc[:, 'demand'].rolling(3).
↪mean().dropna()
series_deseasonalization=series_deseasonalization.
↪drop([series_deseasonalization.index[22],series_deseasonalization.index[23]])

# deseasonalize dataframe
df_deseasonalization = pd.DataFrame({
    'Quater': sum([[1,2,3]*8,[1,2]],[]),
    'Period': df_interger_series['period'] ,
    'Demand': df_interger_series['demand'] ,
    'Deseasonalized_Demand': series_deseasonalization
})
```

```
<ipython-input-68-ab869889bd6c>:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_interger_series['period'] =
df_interger_series['period'].div(np.pi).round(1)
```



[69]:
```
#build up regressing model
reg = LinearRegression().fit(np.asarray(df_deseasonalization.loc[20:230,␣
↪'Period']).reshape(-1, 1),
                            df_deseasonalization.loc[:,␣
 ↪'Deseasonalized_Demand'].dropna())
#predict nan value
values = pd.Series(reg.predict(np.asarray(df_deseasonalization.loc[:,␣
 ↪'Period']).reshape(-1,1)))

df_deseasonalization.loc[0,['Deseasonalized_Demand']] = values[0]
df_deseasonalization.loc[10,['Deseasonalized_Demand']] = values[10]
df_deseasonalization.loc[240,['Deseasonalized_Demand']] = values[24]
df_deseasonalization.loc[250,['Deseasonalized_Demand']] = values[25]

# calculate seansonality factor
df_deseasonalization.loc[:, 'Seasonality'] = (df_deseasonalization.loc[:,␣
 ↪'Demand'] / df_deseasonalization.loc[:, 'Deseasonalized_Demand'])

df_Seasonality_bar= pd.DataFrame({
    'Quater': sum([[1,2,3]*8,[1,2]],[]),
    'Period': df_interger_series['period'] ,
    'Demand': df_interger_series['demand'] ,
    'Deseasonalized_Demand': series_deseasonalization
})

df_seasonality = df_deseasonalization.groupby(['Quater'], as_index=False).mean()
df_seasonality.loc[:, 'Seasonality_bar'] = df_seasonality.loc[:, 'Seasonality']
df_seasonality = df_seasonality[['Quater','Seasonality_bar']]

df_deseasonalization = pd.merge(df_deseasonalization,df_seasonality).
 ↪sort_values('Period')
```

[70]:

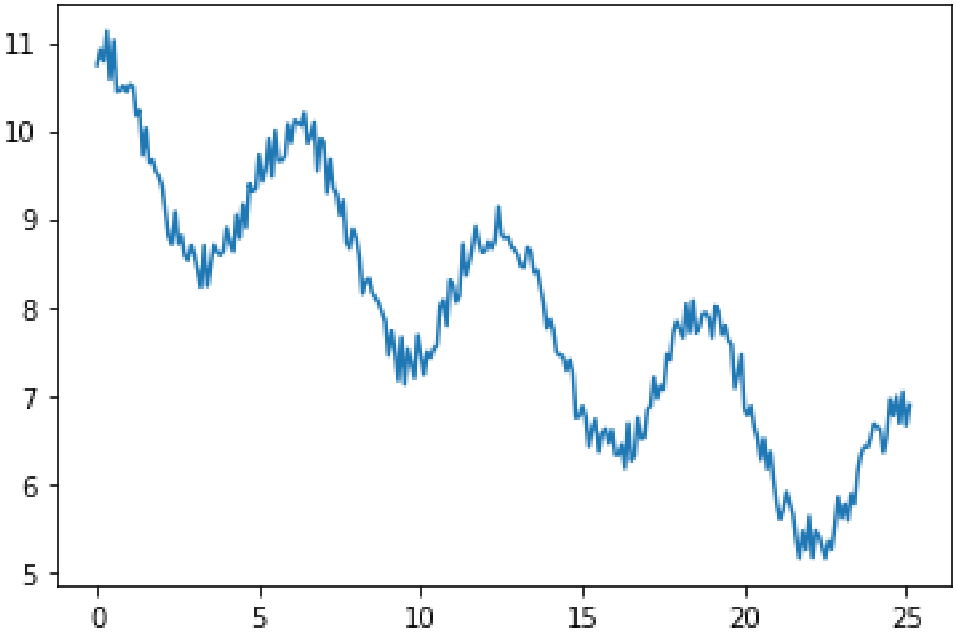
```
df_deseasonalization.loc[:, 'Forecast'] =  (reg.predict(np.
 ↪asarray(df_deseasonalization.loc[:, 'Period']).reshape(-1,1)) *␣
 ↪df_deseasonalization.loc[:, 'Seasonality_bar'])
df_deseasonalization.loc[:, 'Error'] =  (df_deseasonalization.loc[:,␣
 ↪'Demand']-df_deseasonalization.loc[:, 'Forecast'])
df_deseasonalization.loc[:, 'Error_Squre'] =  (df_deseasonalization.loc[:,␣
 ↪'Error']*df_deseasonalization.loc[:, 'Error'])
MSE = df_deseasonalization['Error_Squre'].sum()/len(df_deseasonalization)
MAPE = ((abs(df_deseasonalization.loc[:, 'Error']) / abs(df_deseasonalization.
 ↪loc[:, 'Demand'])).sum())*100/len(df_deseasonalization)

print("MSE:",MSE.round(3))
print("MAPE:",MAPE.round(3))
```

```
MSE: 0.515
MAPE: 8.447
```

in this simulation, we have better performance in both MSE and MAPE

2 Q2

```
[71]: x = np.arange(0,8*np.pi,0.1)    # start,stop,step
      y = 10+np.cos(x)-x/6

      y_noise = []

      for i in y:
          n = 0.3
          noise = np.random.uniform(-n,n)
          y_noise.append(i + noise)

      plt.plot(x,y_noise)
      plt.show()
```



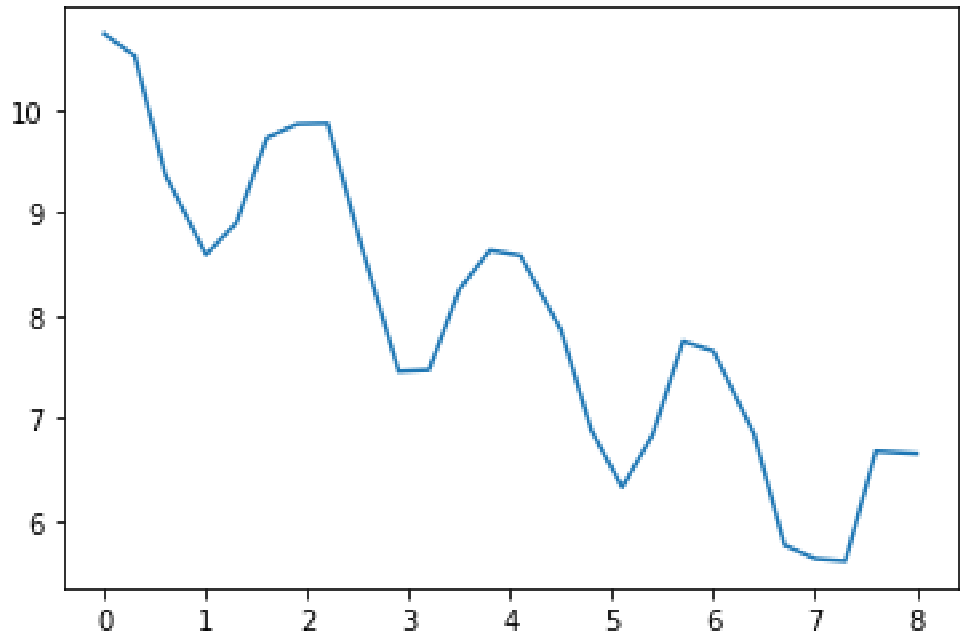
```
[72]: df_original_series = pd.DataFrame({
          'Period': x,
          'Demand': y_noise
      })

      # include average interger.
      df_interger_series = df_original_series[df_original_series['Period'] % 1 == 0]
      df_interger_series['Period'] = df_interger_series['Period'].div(np.pi).round(1)
      plt.plot(df_interger_series['Period'],df_interger_series['Demand'])
```

```
<ipython-input-72-bbe5da21f942>:8: SettingWithCopyWarning:
```

```
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_interger_series['Period'] =
df_interger_series['Period'].div(np.pi).round(1)
```

[72]:

```
[<matplotlib.lines.Line2D at 0x1ee3dc94100>]
```



[73]:

```python
# train the data with Holt-Winters algorithms with statsmodels module.
HWES_model = HWES(df_interger_series.loc[:, 'Demand'], seasonal_periods=8,␣
 ↪trend='add', seasonal='mul')
HWES_fit_report = HWES_model.fit()
print(HWES_fit_report.summary())
```

```
                       ExponentialSmoothing Model Results
================================================================================
Dep. Variable:                       Demand   No. Observations:                   26
Model:                 ExponentialSmoothing   SSE                             11.221
Optimized:                             True   AIC                              2.152
Trend:                             Additive   BIC                             17.249
Seasonal:                    Multiplicative   AICC                            40.334
Seasonal Periods:                         8   Date:                 Sun, 10 Oct 2021
Box-Cox:                              False   Time:                         03:13:49
Box-Cox Coeff.:                        None
```

```
================================================================================
=
                          coeff                 code              optimized
--------------------------------------------------------------------------------
-
smoothing_level               1.0000000                alpha
True
smoothing_trend              3.5816e-14                 beta
True
smoothing_seasonal           1.4901e-08                gamma
True
initial_level                 8.1211206                  l.0
True
initial_trend                -0.1204870                  b.0
True
initial_seasons.0             1.3426997                  s.0
True
initial_seasons.1             1.3198900                  s.1
True
initial_seasons.2             1.3150706                  s.2
True
initial_seasons.3             1.3339039                  s.3
True
initial_seasons.4             1.3653020                  s.4
True
initial_seasons.5             1.3973825                  s.5
True
initial_seasons.6             1.3933601                  s.6
True
initial_seasons.7             1.3725840                  s.7
True
--------------------------------------------------------------------------------
-

C:\Users\TerryYang\anaconda3\envs\TENSORFLOW\lib\site-
packages\statsmodels\tsa\base\tsa_model.py:578: ValueWarning: An unsupported
index was provided and will be ignored when e.g. forecasting.
  warnings.warn('An unsupported index was provided and will be'
C:\Users\TerryYang\anaconda3\envs\TENSORFLOW\lib\site-
packages\statsmodels\tsa\holtwinters\model.py:427: FutureWarning: After 0.13
initialization must be handled at model creation
  warnings.warn(
```