# HW-Forecasting

2015147533 유현석

## 1. SES 단순 평활 예측법

**ses = SimpleExpSmoothing(data).fit()**에서 fit 함수를 이용해서 최적의 알파 값을 찾아 예측을 해보겠습니다.

또한 그 예측된 결과를 토대로 다음 같은 값들 및 그림을 얻을 수 있었습니다.

### 1-1 SSE로 standard error 판단하기

sse함수를 이용하여 standard error를 판단할 수 있었고 그 결과는 다음과 같습니다.

```
ses.sse
```

```
14551.848333915868
```

### 1-2 forecast로 예측값 구하기

ses.forecast 함수를 이용하여 12개월치 데이터를 예측할 수 있었고 그 결과값들을 data2에 저장하여 출력하였습니다.

```
data2=ses.forecast(12)
```

```
data2
```

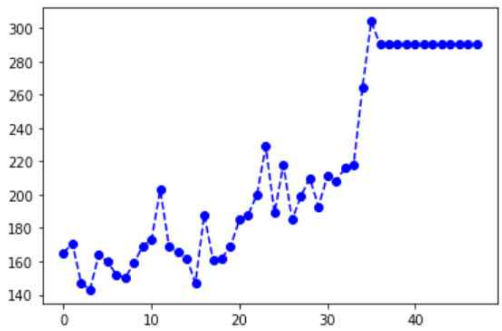
```
36    289.958166
37    289.958166
38    289.958166
39    289.958166
40    289.958166
41    289.958166
42    289.958166
43    289.958166
44    289.958166
45    289.958166
46    289.958166
47    289.958166
dtype: float64
```

### 1-3 plot 그리기

data2에 저정된 예측된 결과 값들을 기존에 값이 저장된 data1에 합쳐준 후 plot 함수를 이용하여 그래프를 그렸습니다.

```
data1.plot(style='--',marker='o',color='blue')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0xf440d09188>
```




## 2. Holt 이중 평활 예측법

**ExponentialSmoothing** 함수를 이용하여 이중 평활 예측을 하기 앞서 linregress 함수를 이용하여 추세 및 p-value를 확인하고 그 이후에 예측을 진행하도록 하겠습니다.

### 2-1 linregress 추세 및 p-value 확인하기

linregress 함수를 이용하여 확인했을 때 slope는 2.5386라는 값을 구할 수 있고 다음 그림과 같이 pvalue값이  1.17 e^(-8)이기 때문에 0.05보다 작아 유의미하다고 볼 수 있습니다. pvalue값을 만족한다고 절대적으로 볼순 없지만 그래도 수치 자체로 봤을 땐 의사결정에 유용하게 사용 될 수 있다고 생각합니다.

<linregress 확인하기>

```
linregress(x=xgo,y= data)
```

```
LinregressResult(slope=2.5386100386100385, intercept=139.06349206349205, rvalue=0.7879800756316014, pvalue=1.1694156537818794e-08, stderr=0.3401817205126318 5)
```

<pvalue 확인하기>

```
linregress(x=xgo,y= data).pvalue
```

```
1.1694156537818794e-08
```

### 2-2 params로 확인하기

내부적으로 계산되는 initial level과 slope를 확인하기 위해서 params를 이용하여 수치를 볼 수 있습니다. smoothing_level 은 0.66 smoothing_slope는 0.0으로 최적화 해준 것을 확인 할 수 있습니다. 또한 initial_level과 initial_slope 역시 각각 159, 3.55로 확인 할 수 있습니다.

```
hz= ExponentialSmoothing(data,trend='add').fit()
```

```
C:₩ProgramData₩Anaconda3₩lib₩site-packages₩statsm
er_equal
  loc = initial_p >= ub
```

```
hz.params
```

```
{'smoothing_level': 0.6608136050293513,
 'smoothing_slope': 0.0,
 'smoothing_seasonal': nan,
 'damping_slope': nan,
 'initial_level': 158.99599068312165,
 'initial_slope': 3.5492978658760395,
 'initial_seasons': array([], dtype=float64),
 'use_boxcox': False,
 'lamda': None,
 'remove_bias': False}
```

### 2-3 level 과 trend 확인하기

ExponentialSmoothing 이용한 hz의 첫 번째 level과 slope를 다음과 같이 확인 해보았습니다. 각각 164.16, 3.55로 확인 할 수 있었습니다.

```
print(hz.level[0], hz.slope[0])
```

```
164.16739527224135 3.5492978658760395
```

### 2-4 forecast로 예측 data 값 구하기

**forecast** 함수를 이용하여 data4에 예측값을 저장하였고 그 결과는 다음과 같습니다.

```
data4
```

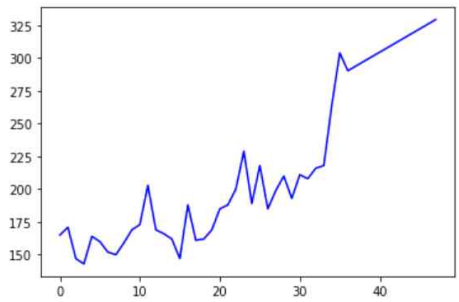
```
36    290.320147
37    293.869445
38    297.418743
39    300.968040
40    304.517338
41    308.066636
42    311.615934
43    315.165232
44    318.714530
45    322.263828
46    325.813126
47    329.362423
dtype: float64
```

## 2-5 plot 구하기

위에 1번과 마찬가지로 data4에 저정된 예측된 결과값들을 기존에 값이 저장된 data3에 합쳐준 후 plot 함수를 이용하여 그래프를 그렸습니다.

```
data3.plot(style='-',color='blue')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0xf44248b148>
```




## 2-6 Deviations from mean 구하기

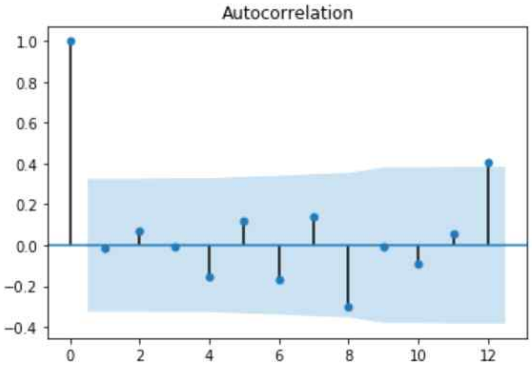
**.resid** 함수를 이용하여 Deviations from mean 값들을 다음과 같이 구해주었습니다.

```
In [50]: df['resid']

Out[50]: 0      2.454711
         1      3.283307
         2    -26.435645
         3    -16.515909
         4     11.848731
         5     -3.530370
         6    -12.746751
         7     -9.872822
         8      2.101975
         9      7.163663
         10     2.880519
         11    27.427735
         12   -28.246183
         13   -16.130019
         14   -13.020381
         15   -22.965634
         16    29.661072
         17   -20.488666
         18    -9.498775
         19     0.228847
         20    12.528324
         21     3.700139
         22     9.705739
         23    28.742757
         24   -33.800146
         25    13.986153
         26   -31.805385
         27    -0.337252
         28     7.336311
         29   -18.060921
         30     8.324683
         31    -3.725678
         32     3.187003
         33    -0.468310
         34    42.291858
         35    50.795525
```

## 2-7 자기 상관 그래프 그리기

**plot_acf** 함수를 이용하여 자기 상관 그래프를 아래와 같이 그려줬습니다. 기간은 12개월로 설정해주었습니다.




### 2-8 자기 상관 계수 확인하기

**acf** 함수를 이용하여 자기 상관 계수를 다음과 같이 확인할 수 있었습니다. upper와 lower을 각각 0.3333, -0.3333으로 설정하였을 때 12개월 단위로 했을 때 0.4089로 upper를 초과하게 되어 12개월 단위의 Seasonality가 존재함을 알 수 있습니다.

```
acf(x=hz.resid,nlags=12)
```

```
C:₩ProgramData₩Anaconda3₩lib₩site-packages₩statsmodels₩tsa₩stattools.py:572:
future version of statsmodels. To suppress this warning, explicitly set fft=
  FutureWarning
```

```
array([ 1.        , -0.01736206,  0.06753744, -0.00440462, -0.15176792,
        0.1199185 , -0.16848861,  0.13962339, -0.30455665, -0.00583066,
       -0.09231976,  0.0524205 ,  0.40895093])
```

# 3. Holt-winters 삼중 평활 예측법

### 3-1 MA(moving Average) smoothed 구하기

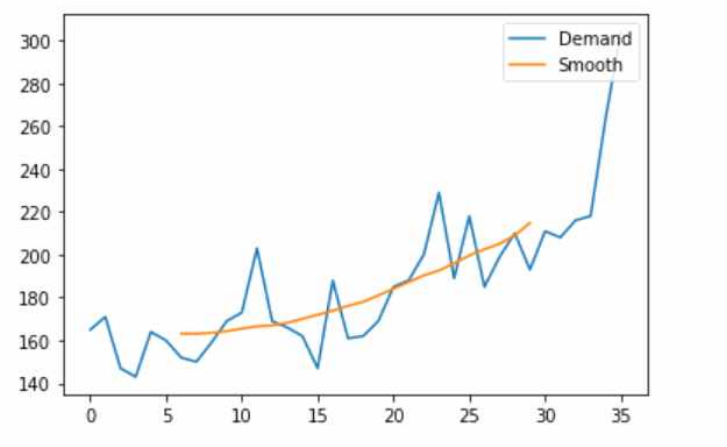
2개의 12개월 치 데이터들의 평균의 평균 값들을 다음과 같이 구해주었습니다. 그리고 그 데이터들을 토대로 Demand와 Smooth 그래프를 plot을 이용하여 그림을 그릴 수 있습니다.

<smooth 데이터>

```
real_smooth
```

```
[163.16666666666669,
 163.125,
 163.54166666666666,
 164.33333333333331,
 165.5,
 166.54166666666669,
 167.0,
 168.20833333333331,
 170.08333333333331,
 171.95833333333331,
 173.875,
 176.08333333333331,
 178.0,
 181.0,
 184.125,
 187.25,
 190.33333333333331,
 192.58333333333331,
 195.95833333333331,
 199.625,
 202.54166666666669,
 205.08333333333334,
 209.0,
 214.79166666666666]
```

<plot을 이용한 그래프>




### 3-2 initial seasonal factors x 3 만들기

해당 월에 해당하는 Seasonal Factor Estimate의 평균 값들을 구하여 12개월치의 Initial Seasonal Factors을 만들었습니다 그 후 3년 치의 데이터가 필요하기 때문에 복사하여 3년 치의 데이터를 다음의 그림과 같이 구하였습니다.

```
In [52]: ini_season

Out[52]: [0.9882333992444625,
          1.0394595142086607,
          0.9329332917124478,
          0.9125977559105471,
          1.0430106047420362,
          0.9064424515366746,
          0.9208375893769152,
          0.9266209436718105,
          0.9884907528722957,
          1.0162014531884298,
          1.0480526558060541,
          1.2040049076184032,
          0.9882333992444625,
          1.0394595142086607,
          0.9329332917124478,
          0.9125977559105471,
          1.0430106047420362,
          0.9064424515366746,
          0.9208375893769152,
          0.9266209436718105,
          0.9884907528722957,
          1.0162014531884298,
          1.0480526558060541,
          1.2040049076184032]
```

### 3-3 Deseasonalized Data 만들기

각각에 해당하는 demand 값들을 위에서 구한 initial seasonal factors로 나누어 주면 구하고자 하는 3년치의 Deseasonalized Data를 아래의 그림과 같이 구할 수 있습니다.

```
deseason
```

```
[166.96460585743006,
 164.50857167840934,
 157.5675359705235,
 156.69554201053379,
 157.2371357054049,
 176.5142395181901,
 165.0671103716029,
 161.87849090223762,
 160.85127709893854,
 166.30560748535268,
 165.06804218433683,
 168.60396391701315,
 171.01223266609503,
 159.69837952406988,
 173.64585596751567,
 161.07863409474453,
 180.2474482476593,
 177.6174535151788,
 175.92678868552414,
 182.38309974985438,
 187.15400165599766,
 185.00268761684202,
 190.83010657148768,
 190.19856028076853,
 191.2503667094199,
 209.72437792920022,
 198.2992799629037,
 218.0588311894841,
 201.34023474472582,
 212.92030141881682,
 229.13921242373823,
 224.47150738443617,
 218.51494247402968,
 214.5243930876147,
 251.89574067436374,
 252.49066517621674]
```

### 3-4 params로 확인하기

위에 2번과 같이 내부적으로 계산되는 initial level과 slope를 확인하기 위해서 params를 이용하여 수치를 볼 수 있습니다. smoothing_level 은 0.21 smoothing_slope는 0.21으로 최적화 해준 것을 확인 할 수 있습니다. 또한 initial_level과 initial_slope 역시 각각 174.33, 1.18로 확인 할 수 있습니다.

```
Seas = ExponentialSmoothing(data5, trend= 'add',seasonal= 'mul',seasonal_periods=12).fit()
```

```
C:₩ProgramData₩Anaconda3₩lib₩site-packages₩statsmodels₩tsa₩holtwinters.py:725: RuntimeWarning:
equal
  loc = initial_p <= lb
C:₩ProgramData₩Anaconda3₩lib₩site-packages₩statsmodels₩tsa₩holtwinters.py:731: RuntimeWarning:
er_equal
  loc = initial_p >= ub
C:₩ProgramData₩Anaconda3₩lib₩site-packages₩statsmodels₩tsa₩holtwinters.py:744: ConvergenceWarn
e. Check mle_retvals.
  ConvergenceWarning)
```

```
Seas.params
```

```
{'smoothing_level': 0.2105406658923509,
 'smoothing_slope': 0.2105406580853047,
 'smoothing_seasonal': 0.10525226536070365,
 'damping_slope': nan,
 'initial_level': 174.33328798480414,
 'initial_slope': 1.1805499975620573,
 'initial_seasons': array([0.93748581, 0.9888166 , 0.86284025, 0.84989079, 0.9566672 ,
        0.86890229, 0.87826401, 0.86694039, 0.9072826 , 0.91781651,
        1.01915537, 1.16255999]),
 'use_boxcox': False,
 'lamda': None,
 'remove_bias': False}
```

### 3-5 예측하기

**forecast** 함수를 이용하여 data6에 예측값을 저장하였고 그 결과는 다음과 같습니다.

```
data6=Seas.forecast(12)
```

```
data6
```

```
36    244.055576
37    262.051610
38    233.706163
39    233.747217
40    268.743830
41    247.750277
42    254.536769
43    255.838929
44    272.341234
45    280.019878
46    314.037083
47    362.469505
dtype: float64
```

### 3-6 plot 그리기

위에 1번과 2번과 마찬가지로 data6에 저정된 예측된 결과값들을 기존에 값이 저장된 data5에 합쳐준 후 plot 함수를 이용하여 그래프를 그렸습니다.