예측애널리틱스 Homework #3

고려대학교
KOREA UNIVERSITY

대학	고려대학교 공과대학
학과	산업경영공학부
학번	2017170819
이름	박상민

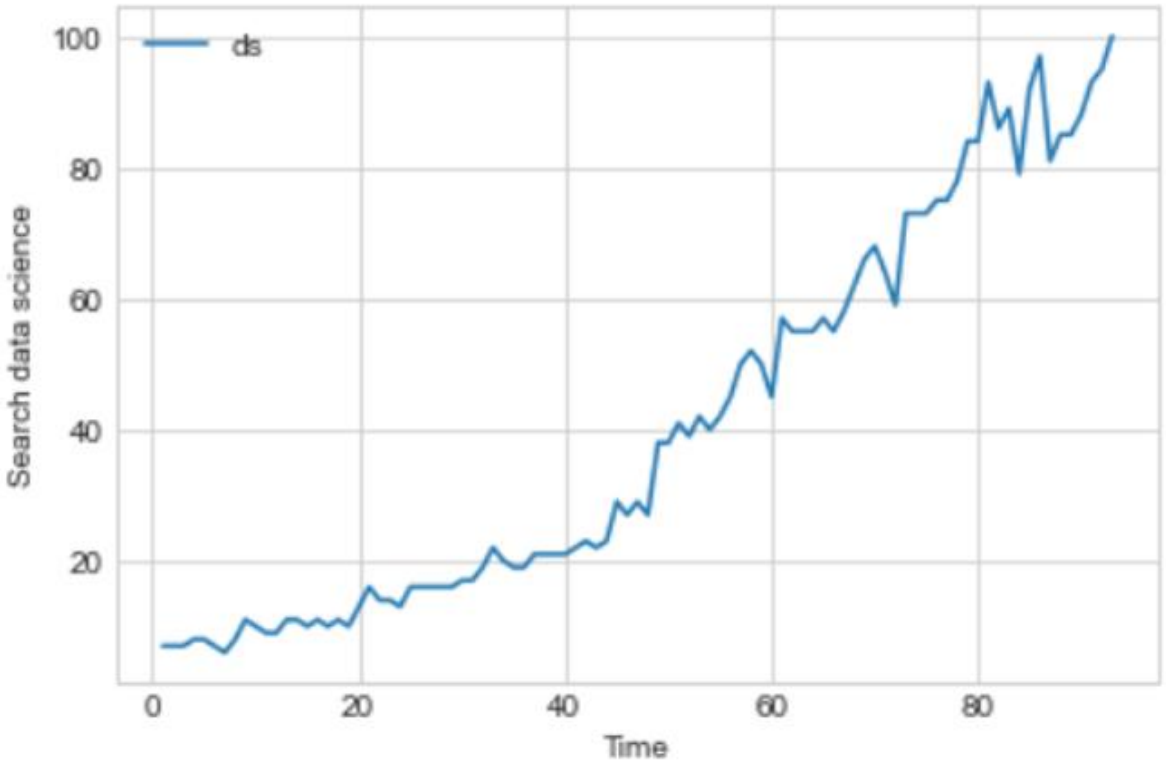
구글 트렌드를 이용해 트렌드가 존재하는 데이터를 찾았다. 'data science' 검색량 데이터는

2014 년 경부터 조금씩 증가하는 추세를 보이고 있다. 그래서 2014 년 1 월부터 2020 년 9 월까지의 데이터를 사용하였다.

2014 년 1 월을 1 번째 index 로 설정하였고, 2020 년 9 월은 93 번째 index 이다.

time	ds
1	7
2	7
3	7
4	8
5	8
...	...
89	85
90	88
91	93
92	95
93	100

데이터의 형태는 다음과 같다. 이중 80%를 training data 로 사용했고, 20%를 test data 로 사용했다.



데이터를 그래프화해보면, 증가 추세임을 알 수 있다.

1-1. Simple exponential smoothing, Double exponential smoothing 방법 적용하고 예측력 비교하기 (최소 10 시점 예측), 각 방법의 하이퍼파라미터를 다양하게 적용해 보고 가장 최적값을 사용

(1) 단순지수평활법

단순지수 평활법은 구간 평균법을 보완하기 위한 방법이며, 최근 데이터에 많은 가중치를 두고 과거로 갈수록 가중치가 줄어드는 가중 평균을 이용한다.

$$L_0 = \frac{1}{n}\sum^{n} D_i$$

먼저 L_0를 관측치들의 평균으로 정한다.

그 다음 시점 t+1 에서의 관측치와 t 시점의 예측치를 통해 t+1 시점을 예측한다.

$$L_{t+1} = \alpha D_{t+1} + (1-\alpha)L_t$$

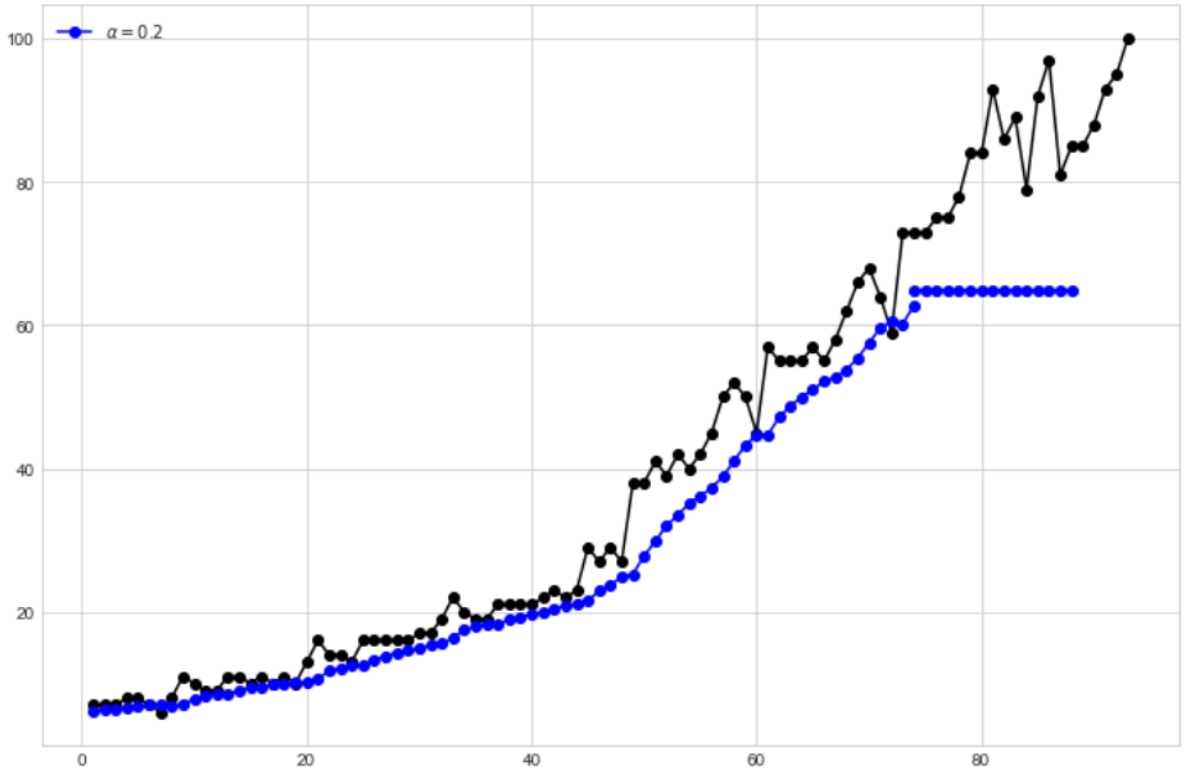
먼저 보편적인 α=0.2로 하이퍼 파라미터를 잡았다.

```
fit1 = SimpleExpSmoothing(train_data, initialization_method="estimated").fit(smoothing_level=0.2, optimized=False)
fcast1 = fit1.forecast(15).rename(r"$#alpha=0.2$")


plt.figure(figsize=(12, 8))
plt.plot(data, marker="o", color="black")
plt.plot(fit1.fittedvalues, marker="o", color="blue")
(line1,) = plt.plot(fcast1, marker="o", color="blue")


plt.legend([line1], [fcast1.name])
```

원래 초깃값은 위와 같이 정하는 것이 정석이지만, 컴퓨터 패키지는 가장 최적의 초깃값을 찾아준다. 99개의 앞부분의 데이터를 training data로 삼았기 때문에 뒷부분의 14개 시점의 데이터를 예측해보도록 하겠다.

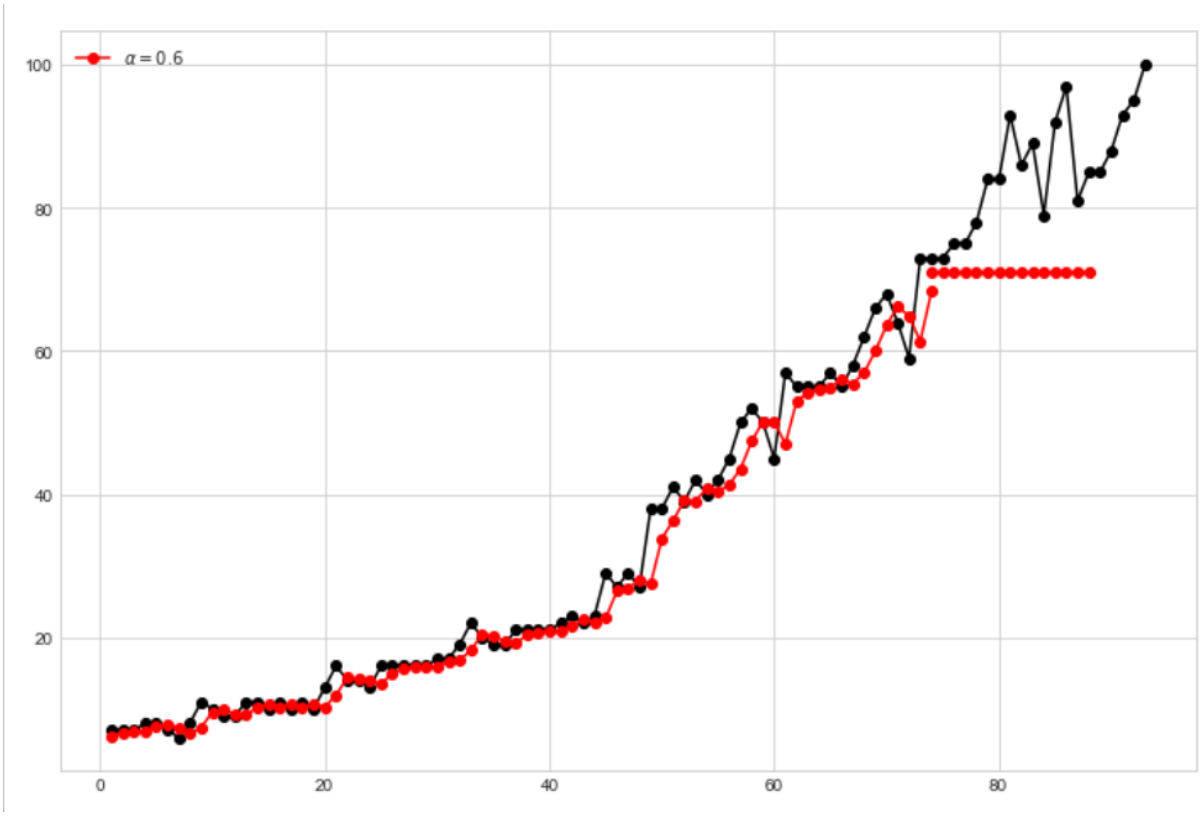


α=0.2로 설정했을 때의 모습이다. 과거 데이터에 가중치를 많이 두는 것이다. 사실 추세가 더 급하게 올라갔다면 예측정확도가 더 떨어질 수 있었다.

```
fit2 = SimpleExpSmoothing(train_data, initialization_method="estimated").fit(smoothing_level=0.6, optimized=False)
fcast2 = fit2.forecast(15).rename(r"$\alpha=0.6$")

plt.figure(figsize=(12, 8))
plt.plot(data, marker="o", color="black")
plt.plot(fit2.fittedvalues, marker="o", color="red")
(line2,) = plt.plot(fcast2, marker="o", color="red")
plt.legend([line2], [fcast2.name])
```

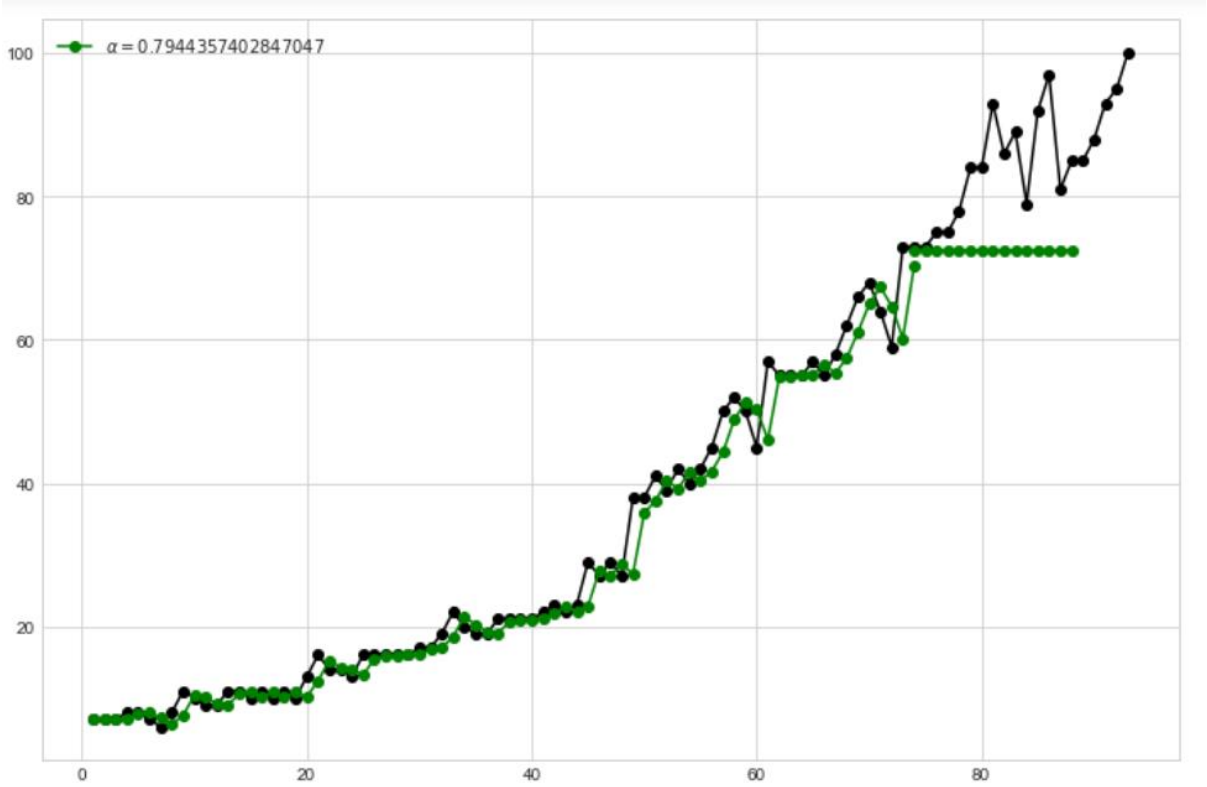
α=0.6으로 설정했을 때의 모습이다. 과거 데이터에 가중치를 적게 두는 것이다. α=0.2때보다는 예측정확도가 높아졌지만, 여전히 추세를 예측하지 못한다.



파이썬에서 가장 최적의 파라미터를 찾아주는 기능이 있기 때문에 이를 사용해보면,

```
fit3 = SimpleExpSmoothing(train_data, initialization_method="estimated").fit()
fcast3 = fit3.forecast(15).rename(r"$\alpha=%s$" % fit3.model.params["smoothing_level"])

plt.figure(figsize=(12, 8))
plt.plot(data, marker="o", color="black")
plt.plot(fit3.fittedvalues, marker="o", color="green")
(line3,) = plt.plot(fcast3, marker="o", color="green")
plt.legend([line3], [fcast3.name])
```



이와 같이 α =0.794 로 정해짐을 알 수 있다. 과거보다 최근 데이터를 중요시했다는 것이다. 다른 파라미터 값들에 비해 제일 예측 정확도가 높아보이기는 하지만, 여전이 추세를 예측하지 못한다.

(2) 이중지수 평활법

트렌드가 존재하는 시계열 데이터 예측 시 적합하다. 단순지수 평활법을 2 번 적용한 것이다. 먼저 선형회귀를 통해 L 과 B 의 초깃값을 구한다. B 는 트렌드를 잡는 부분이다. T 시점의 예측치와 growth rate, 그리고 t+1 시점의 관측치를 통해 t+1 시점의 값을 예측한다.

$$L_{t+1} = \underbrace{\alpha D_{t+1}}_{\text{Newly observed time series value}} + (1-\alpha)(L_t + \underbrace{B_t}_{\text{Growth rate}})$$

$$B_{t+1} = \beta(\underbrace{L_{t+1} - L_t}_{\text{Difference between the levels in period } t+1 \text{ and } t.}) + (1-\beta)\underbrace{B_t}_{\text{Growth rate made in time period } t}$$

B 값에는 β라는 또 다른 하이퍼 파라미터가 존재한다. 이를 통해 향후 미래의 값을 예측한다.

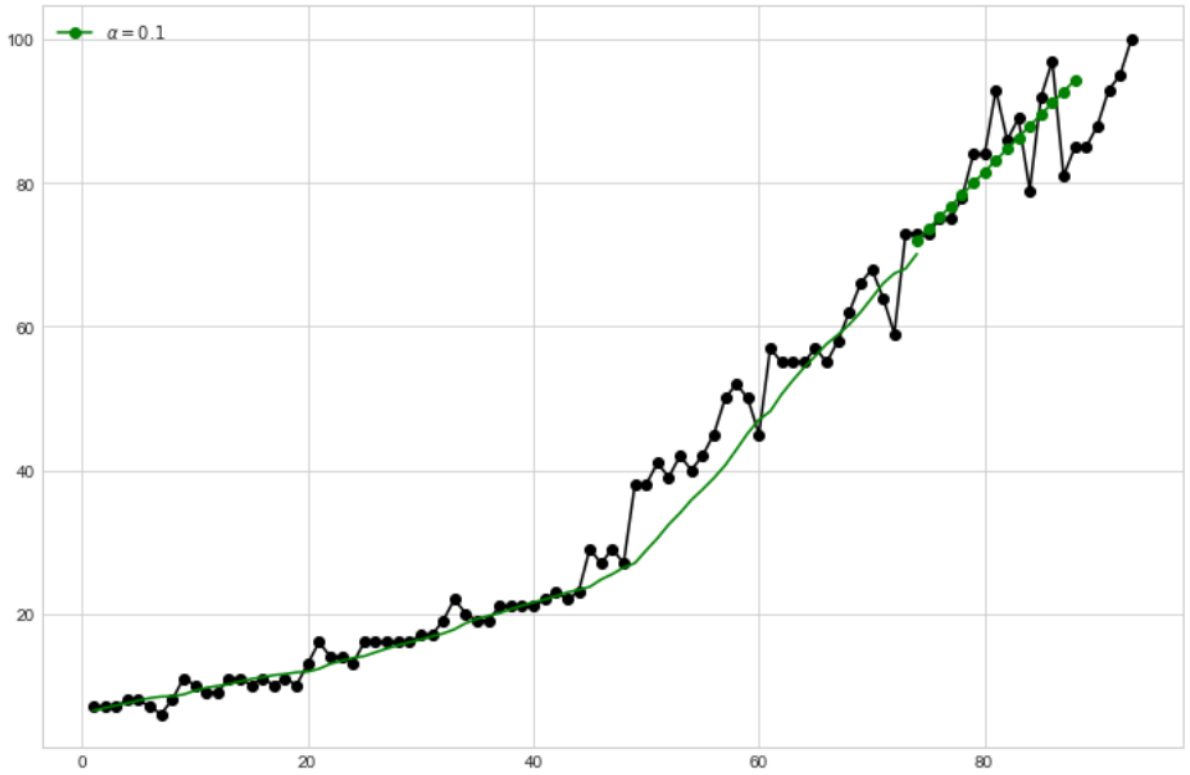
$$F_{t+1} = L_t + B_t$$
$$F_{t+2} = L_t + 2B_t$$
$$\cdots$$
$$F_{t+n} = L_t + nB_t$$

파이썬을 통해 위의 데이터를 가지고 미래 예측을 해보면 증가하는 추세를 잡고 있는 것을 볼 수 있다.

일반적으로 많이 쓰이는 파라미터인 α=0.1 , β=0.1 로 정해보았다. 이는 과거 데이터와 t 에서의 growth rate 를 중요시했다는 것이다.

```
fit6 = Holt(train_data, initialization_method="estimated").fit(smoothing_level=0.1, smoothing_trend=0.1, optimized=False)
fcast6 = fit6.forecast(15).rename(r"$₩alpha=0.8$")

plt.figure(figsize=(12, 8))
plt.plot(data, marker="o", color="black")
plt.plot(fit6.fittedvalues, color="green")
(line1,) = plt.plot(fcast6, marker="o", color="green")
plt.legend([line1], [fcast6.name])
```



증가하는 추세를 잘 확인하고 있음을 알 수 있다.

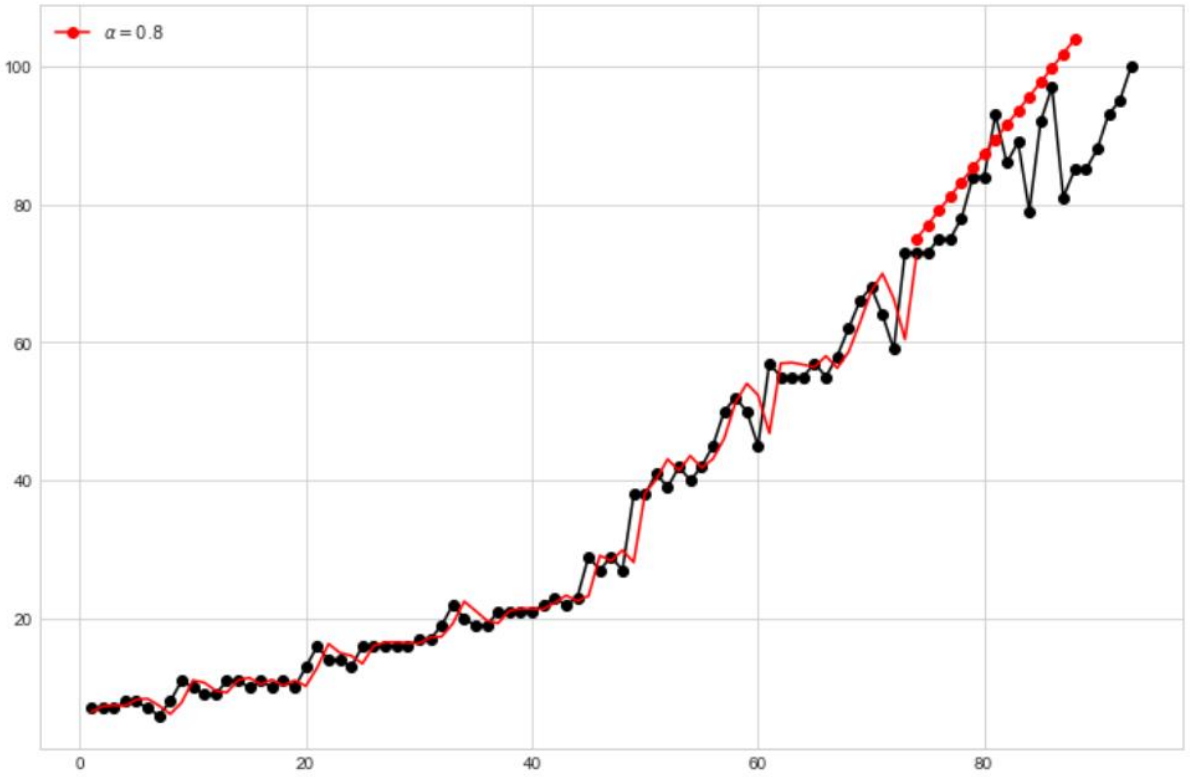
```
fit5 = Holt(train_data, initialization_method="estimated").fit(smoothing_level=0.8, smoothing_trend=0.2, optimized=False)
fcast5 = fit5.forecast(15).rename(r"$₩alpha=0.8$")

plt.figure(figsize=(12, 8))
plt.plot(data, marker="o", color="black")
plt.plot(fit5.fittedvalues, color="red")
(line1,) = plt.plot(fcast5, marker="o", color="red")
plt.legend([line1], [fcast5.name])
```

α=0.8 , β=0.2 로 정해보았다. 이는 최근 데이터에 가중치를 두고, β=0.1 일때보다 growth rate 를 덜 중요시했다는 것이다.

```
fit4 = Holt(train_data, initialization_method="estimated").fit()
fcast4 = fit4.forecast(15).rename(r"$₩alpha=%s$" % fit4.model.params["smoothing_level"])

plt.figure(figsize=(12, 8))
plt.plot(data, marker="o", color="black")
plt.plot(fit4.fittedvalues, color="blue")
(line1,) = plt.plot(fcast4, marker="o", color="blue")
plt.legend([line1], [fcast4.name])
```

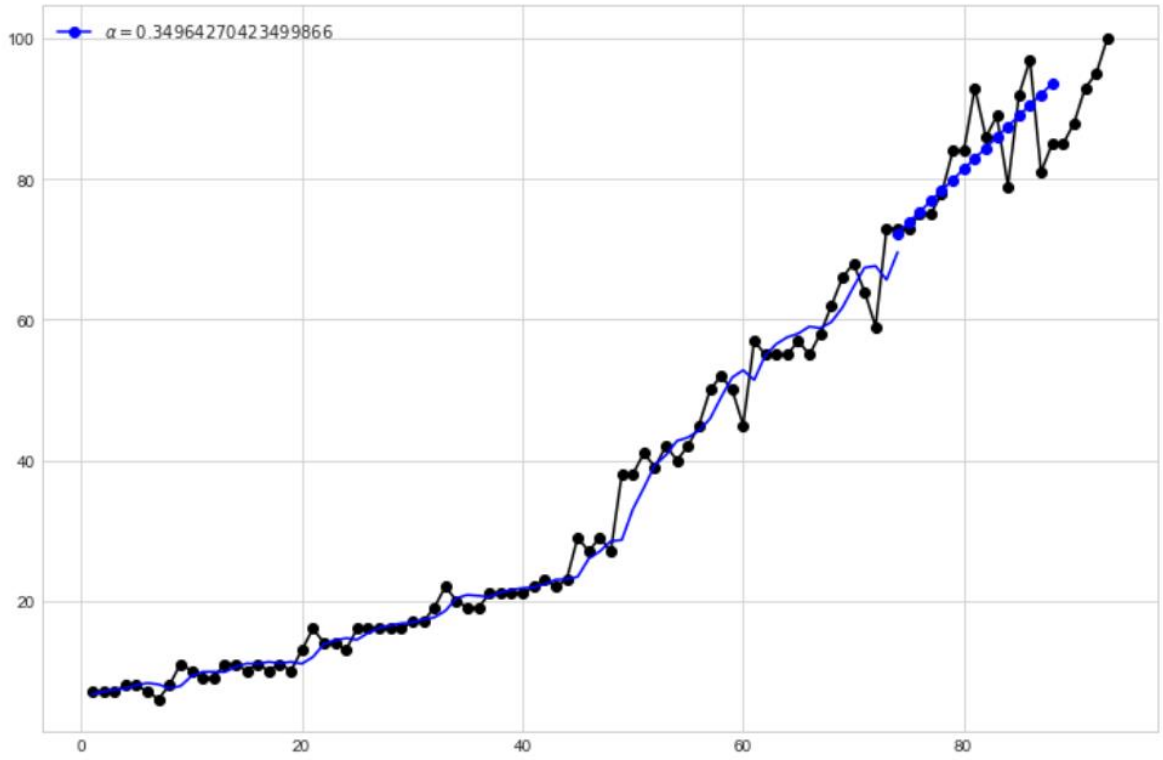


마찬가지로 급격히 증가하는 추세를 잡고 있었다.

```
fit6 = Holt(train_data, initialization_method="estimated").fit(smoothing_level=0.1, smoothing_trend=0.1, optimized=False)
fcast6 = fit6.forecast(15).rename(r"$₩alpha=0.1$")

plt.figure(figsize=(12, 8))
plt.plot(data, marker="o", color="black")
plt.plot(fit6.fittedvalues, color="green")
(line1,) = plt.plot(fcast6, marker="o", color="green")
plt.legend([line1], [fcast6.name])
```

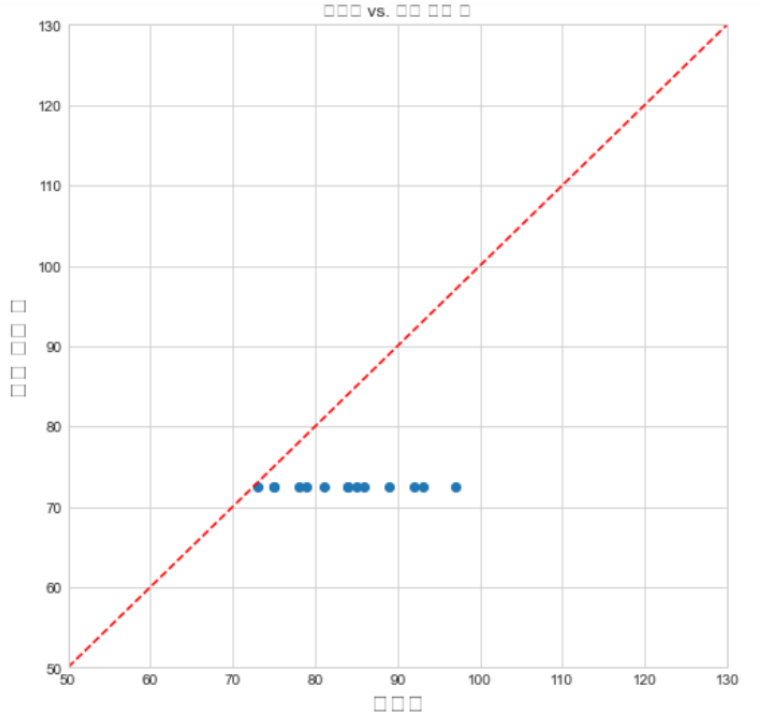
파라미터들의 최적값을 구해주는 기능을 사용했을 때, α=0.350 , β=0.124 로 정해졌다.



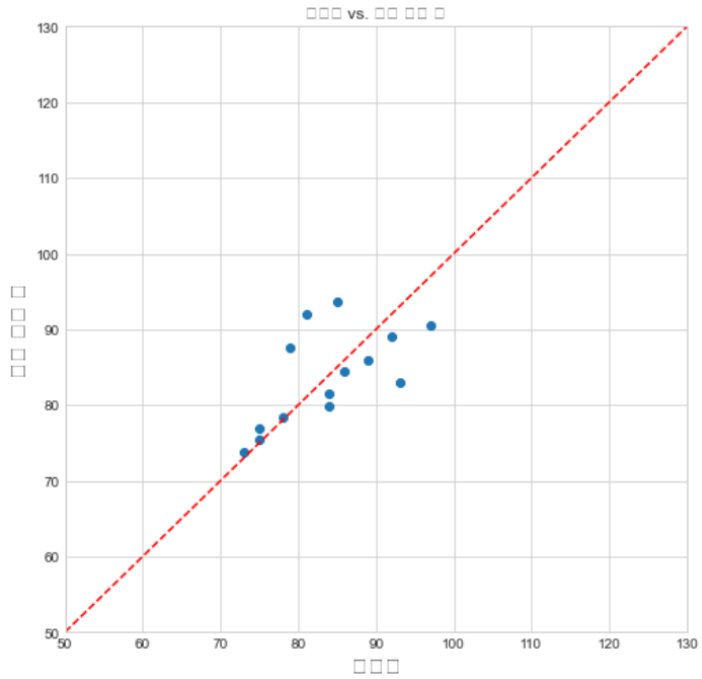
추세를 가장 정확히 예측하고 있는 것을 알 수 있다.

	Simple	Double
α	0.794436	0.349643
β	NaN	0.123964
SSE	761.292397	595.147865

최적의 파라미터들을 계산해보면 위와 같다. 이때 sum of square error 를 비교해보면 이중지수평활이 단순지수평활보다 작기 때문에 더 정교하게 fitted 되었음을 알 수 있다.



단순지수 평활법의 예측정확도를 정성적으로 확인해보면, y=x 그래프를 비교적 따르지 않기 때문에 예측정확도가 좋다고 할 수 없다.



이중지수 평활법의 예측정확도를 정성적으로 확인해보면, y=x 그래프를 비교적 따르기 때문에 예측정확도가 좋다고 할 수 있다.

<단순지수평활>

```
print('Testing MSE: {:.3f}'.format(mean_squared_error(test_data[:14], fcast3[1:])))
print('Testing RMSE: {:.3f}'.format(np.sqrt(mean_squared_error(test_data[:14], fcast3[1:]))))
print('Testing MAE: {:.3f}'.format(mean_absolute_error(test_data[:14], fcast3[1:])))
print('Testing MAPE: {:.3f}'.format(mean_absolute_percentage_error(test_data[:14], fcast3[1:])))
print('Testing R2: {:.3f}'.format(r2_score(test_data[:14], fcast3[1:])))
```

```
Testing MSE: 174.762
Testing RMSE: 13.220
Testing MAE: 11.185
Testing MAPE: 12.761
Testing R2: -2.519
```

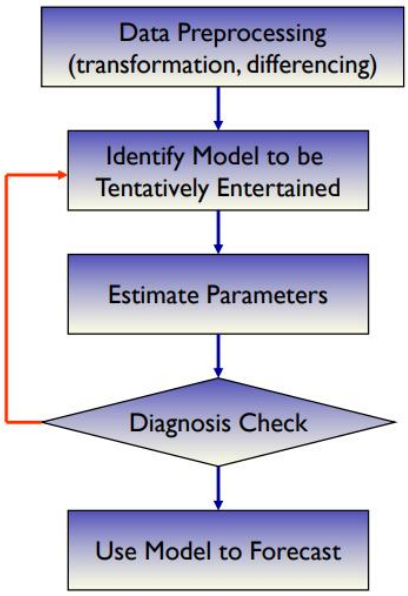
<이중지수평활>

```
print('Testing MSE: {:.3f}'.format(mean_squared_error(test_data[:14], fcast4[1:])))
print('Testing RMSE: {:.3f}'.format(np.sqrt(mean_squared_error(test_data[:14], fcast4[1:]))))
print('Testing MAE: {:.3f}'.format(mean_absolute_error(test_data[:14], fcast4[1:])))
print('Testing MAPE: {:.3f}'.format(mean_absolute_percentage_error(test_data[:14], fcast4[1:])))
print('Testing R2: {:.3f}'.format(r2_score(test_data[:14], fcast4[1:])))
```

```
Testing MSE: 32.805
Testing RMSE: 5.728
Testing MAE: 4.449
Testing MAPE: 9.148
Testing R2: 0.339
```

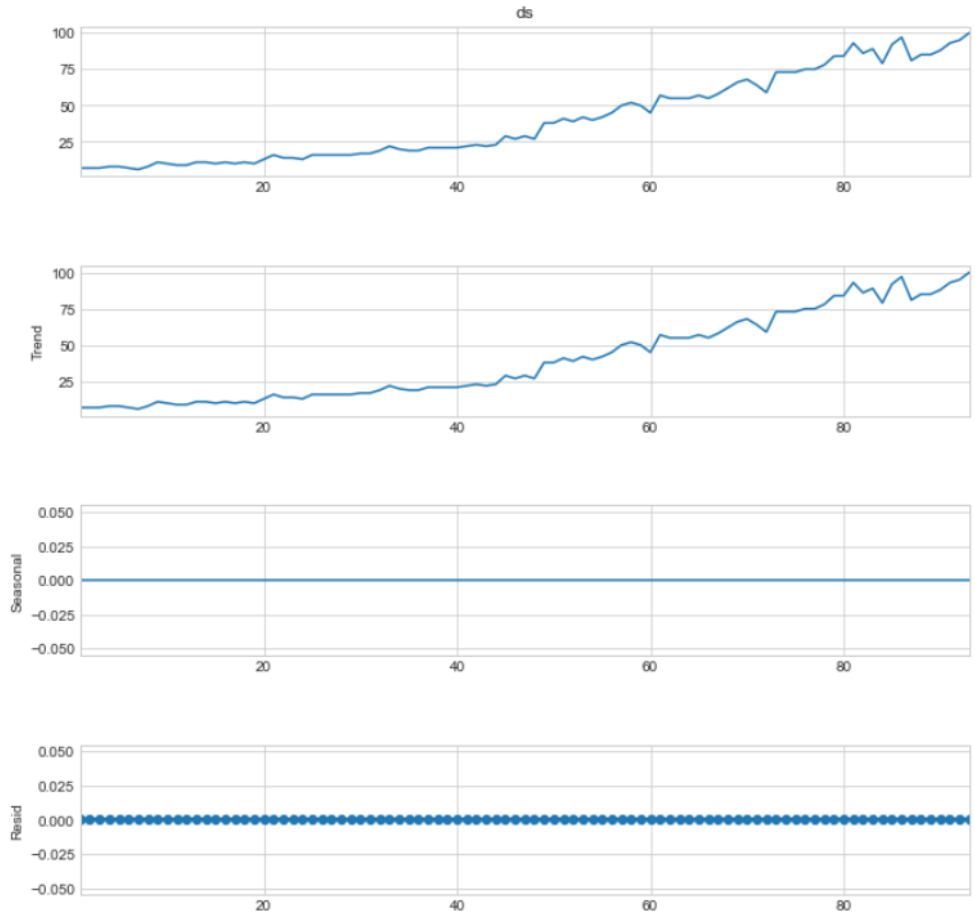
예측정확도를 비교해봐도 단순지수평활은 R2 값이 음수가 나오고, 오차들도 이중지수평활법에 비해 매우 높다. 이는 이중지수평활법의 예측 성능이 더 좋다고 할 수 있다.

1-2. ARIMA 모델링

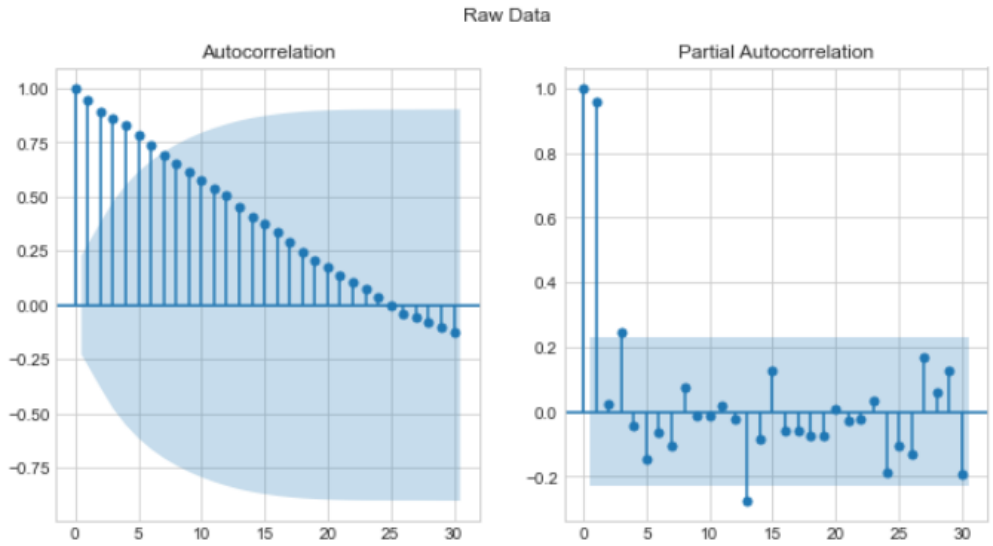


box-jenkins ARIMA modeling

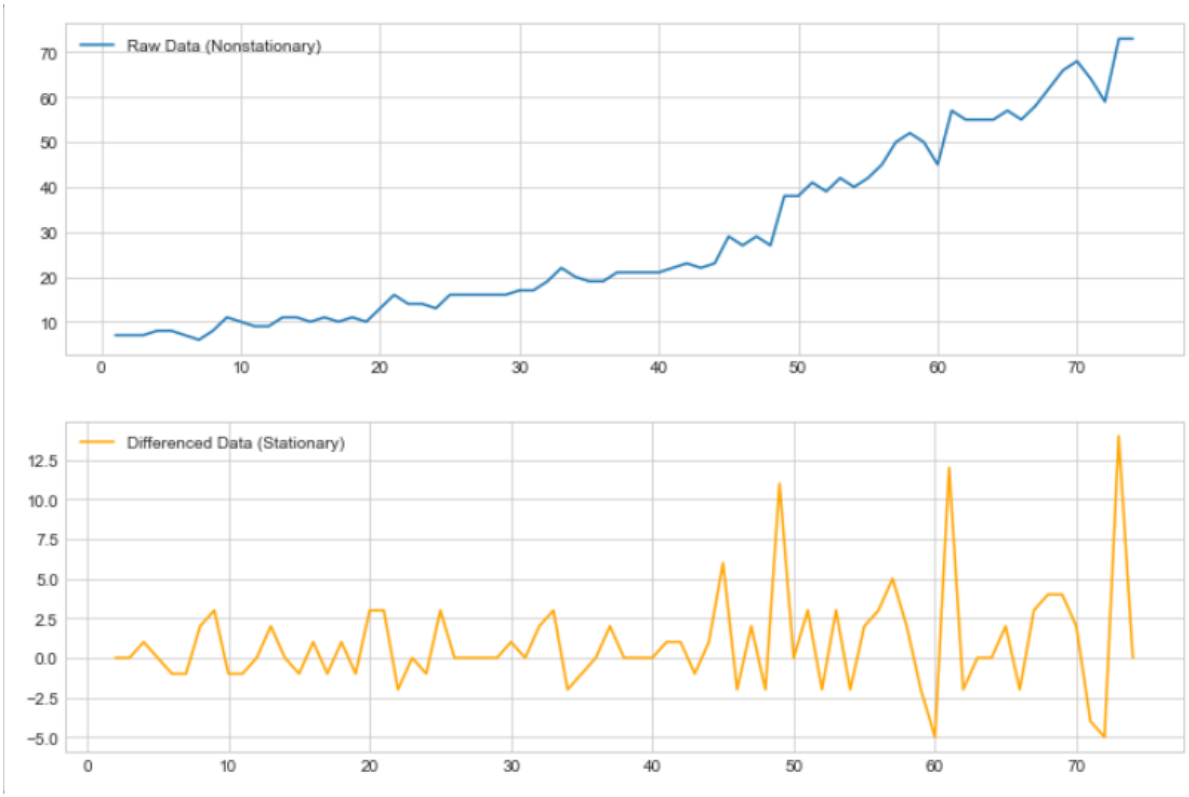
먼저 데이터 전처리를 한다. 마찬가지로 80%를 training data, 20%를 Testing data 로 했다.



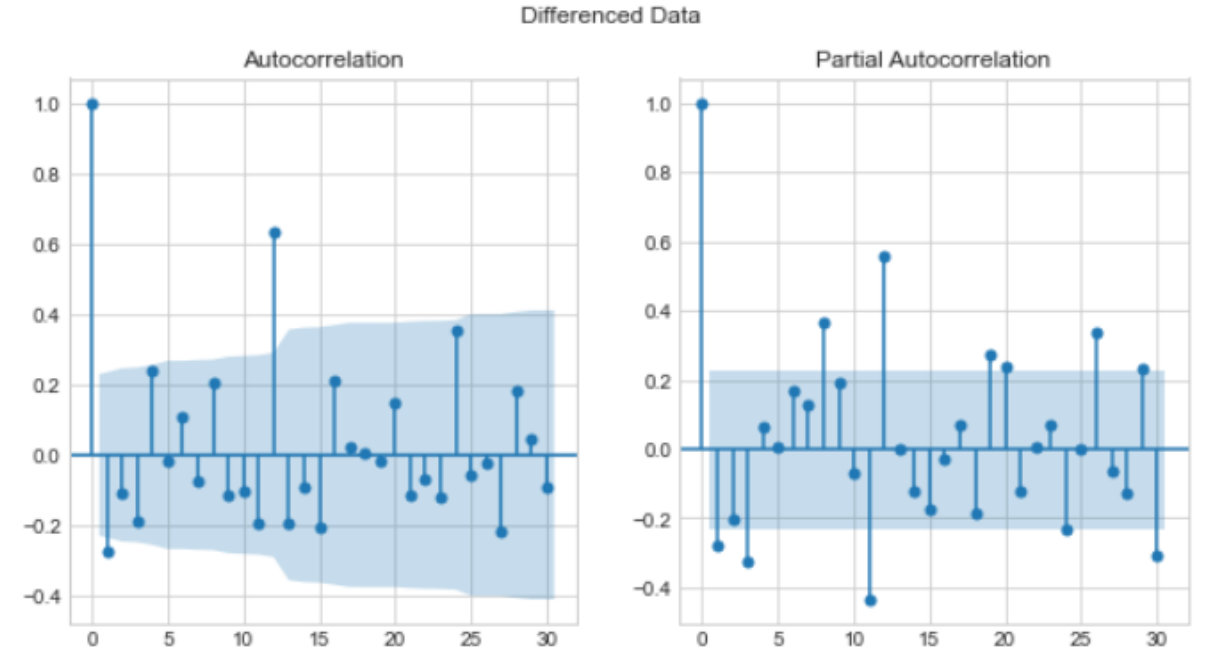
데이터를 trend, seasonal, error 로 decompose 해보면 trend 가 있는 데이터임을 알 수 있다.



ACF, PACF 를 그려보면 ACF 가 점진적으로 감소하기 때문에 nonstationary data 임을 알 수 있다.



Differencing 을 한 후 raw data 와 비교하면 꽤 많이 stationary data 가 되었음을 알 수 있다.



ACF 도 특정한 패턴이 보이지 않는다.

AIC 가 최소가 되는 최적의 (p,d,q) 파라미터 조합을 찾는다.

P 는 0~3, d 는 1~2, q 는 0~3 으로 범위를 정했다. 파이썬으로 찾아본 결과,

```
# Search optimal parameters

optimal = [(pdq[i], j) for i, j in enumerate(aic) if j == min(aic)]
optimal
```

```
[((2, 1, 2), 366.45)]
```

AIC 가 가장 작은 파라미터 조합은 (2, 1, 2)로 나타났다.

ARIMA Model Results

Dep. Variable:	D.y	**No. Observations:**	73
Model:	ARIMA(2, 1, 2)	**Log Likelihood**	-177.224
Method:	css-mle	**S.D. of innovations**	2.642
Date:	Sat, 09 Apr 2022	**AIC**	366.447
Time:	23:13:04	**BIC**	380.190
Sample:	1	**HQIC**	371.924

	coef	std err	z	P>|z|	[0.025	0.975]
const	0.8706	0.247	3.529	0.000	0.387	1.354
ar.L1.D.y	0.9251	0.065	14.187	0.000	0.797	1.053
ar.L2.D.y	-0.8433	0.081	-10.464	0.000	-1.001	-0.685
ma.L1.D.y	-1.2704	0.027	-46.739	0.000	-1.324	-1.217
ma.L2.D.y	1.0000	nan	nan	nan	nan	nan

Roots

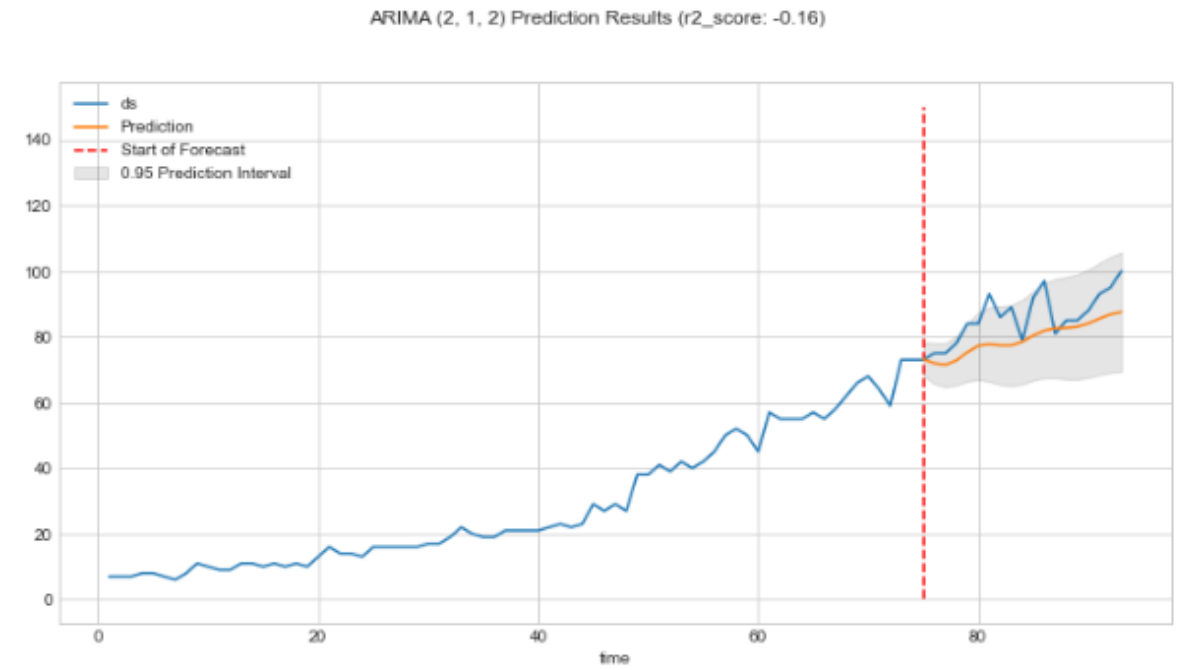
	Real	Imaginary	Modulus	Frequency
AR.1	0.5485	-0.9407j	1.0890	-0.1660
AR.2	0.5485	+0.9407j	1.0890	0.1660
MA.1	0.6352	-0.7724j	1.0000	-0.1405
MA.2	0.6352	+0.7724j	1.0000	0.1405

ARIMA 모델 결과를 보면 p-value 값도 다 충분히 작고, AIC 도 366.45 로 경우의 수 중 가장 작았다.

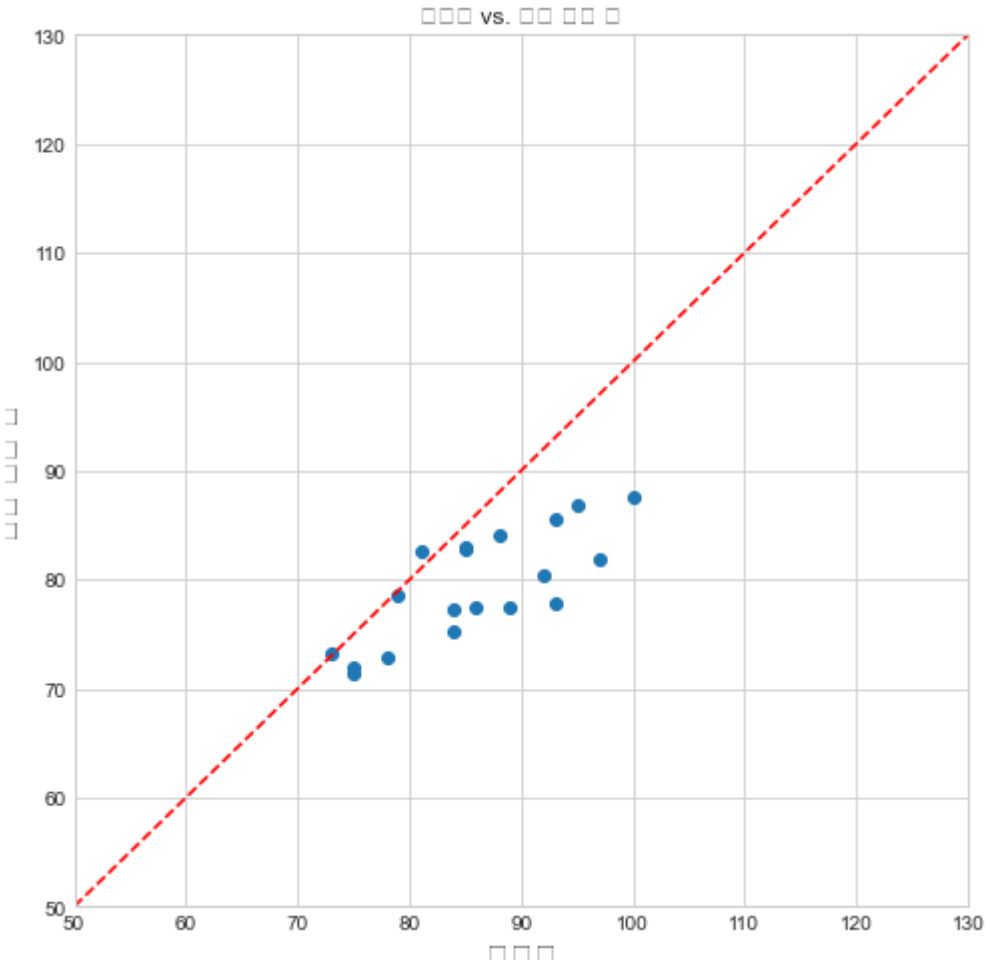
```
prediction = model_opt_fit.forecast(len(test_data))
predicted_value = prediction[0]
predicted_ub = prediction[2][:,0]
predicted_lb = prediction[2][:,1]
predict_index = list(test_data.index)
r2 = r2_score(test_data, predicted_value)
predict_index[0]
```

75

```
fig, ax = plt.subplots(figsize=(12, 6))
data.plot(ax=ax);
ax.vlines(75, 0, 150, linestyle='--', color='r', label='Start of Forecast');
ax.plot(predict_index, predicted_value, label = 'Prediction')
ax.fill_between(predict_index, predicted_lb, predicted_ub, color = 'k', alpha = 0.1,
ax.legend(loc='upper left')
plt.suptitle(f'ARIMA {optimal[0][0]} Prediction Results (r2_score: {round(r2,2)})')
plt.show()
```



이를 통해 75 번째 관측치부터 예측된 ARIMA 를 그려보면 비교적 증가 추세를 잘 따르고 있고, 95% prediction interval 안에 관측치가 대부분 포함된다. 하지만 R2 값이 음수가 나왔다. 이는 너무 많은 범위를 예측했고, test data 의 fluctuation 이 많기 때문에 정확도가 떨어졌기 때문이라고 생각한다.



예측정확도를 정성적으로 분석하면, 비교적 y=x 추세를 따라가지만, 아주 정확하지는 않다. 역시 test data 의 범위가 너무 크기 때문이라고 생각한다.

```
print('Testing MSE: {:.3f}'.format(mean_squared_error(test_data, predicted_value)))
print('Testing RMSE: {:.3f}'.format(np.sqrt(mean_squared_error(test_data, predicted_value))))
print('Testing MAE: {:.3f}'.format(mean_absolute_error(test_data, predicted_value)))
print('Testing MAPE: {:.3f}'.format(mean_absolute_percentage_error(test_data, predicted_value)))
print('Testing R2: {:.3f}'.format(r2_score(test_data, predicted_value)))
```

```
Testing MSE: 67.245
Testing RMSE: 8.200
Testing MAE: 6.731
Testing MAPE: 10.242
Testing R2: -0.158
```

ARIMA 모델의 예측 정확도는 다음과 같다. 예측이 되는 범위의 fluctuation 이 많고, 예측 범위가 길기 때문에 정확도가 낮게 나타났다.

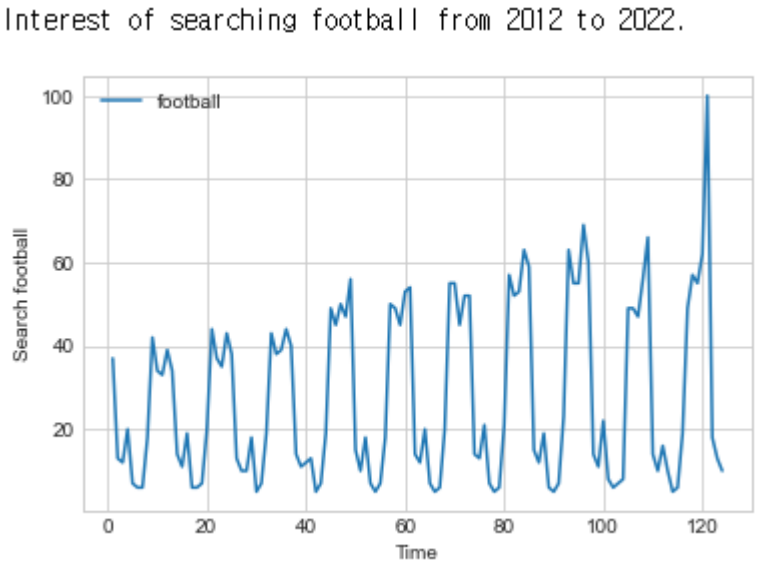
2-1. Additive Holt-Winters, Multiplicative Holt-Winters, Seasonal ARIMA 방법론 적용하고 예측력을 비교

홀트-윈터 지수평활법은 삼중지수 평활법으로, 트렌드 뿐만 아니라 seasonal variation 도 잡아낼 수 있다.

Seasonal variation 이 있는 데이터로 'nfl' 검색량을 결정했다. 미국 미식축구리그인 nfl 은 추춘제로, 겨울에 높은 검색량을 보이고 2 월 슈퍼볼 때 절정을 기록한다.

참고

검색량이 많아지는 2012 년 1 월부터 현재까지 데이터로 분석을 해보았다.



Holt-Winter 지수평활법은 Additive 와 Multiplicative 방법이 있다. Holt-Winter 지수평활법에는 트렌드를 잡아주는 부분(b)와 seasonal variation 을 잡아주는 부분(sn)이 존재한다.

마찬가지로 데이터의 앞쪽 80%를 Training data 로 사용했고, 20%를 Testing data 로 사용했다. 즉, 99 시점이 training data 이고 향후 25 시점을 test data 와 비교해보았다. 주기는 1 년이기 때문에 L=12 로 정했다.

<Additive Holt-Winter 지수평활법>

$$l_T = \alpha(y_T - sn_{T-L}) + (1-\alpha)(l_{T-1} + b_{T-1})$$
$$b_T = \gamma(l_T - l_{T-1}) + (1-\gamma)b_{T-1}$$
$$sn_T = \delta(y_T - l_T) - (1-\delta)sn$$

<Multiplicative Holt-Winter 지수평활법>

$$l_T = \alpha\left({}^{y_T}/_{sn_{T-L}}\right) + (1-\alpha)(l_{T-1} + b_{T-1})$$
$$b_T = \gamma(l_T - l_{T-1}) + (1-\gamma)b_{T-1}$$
$$sn_T = \delta\left({}^{y_T}/_{sn_{T-L}}\right) - (1-\delta)sn_T$$

L_T는 시점 T 에서의 level, b_T는 시점 T 에서의 growth rate, sn_T는 additive/multiplicative seasonal factor 이다.

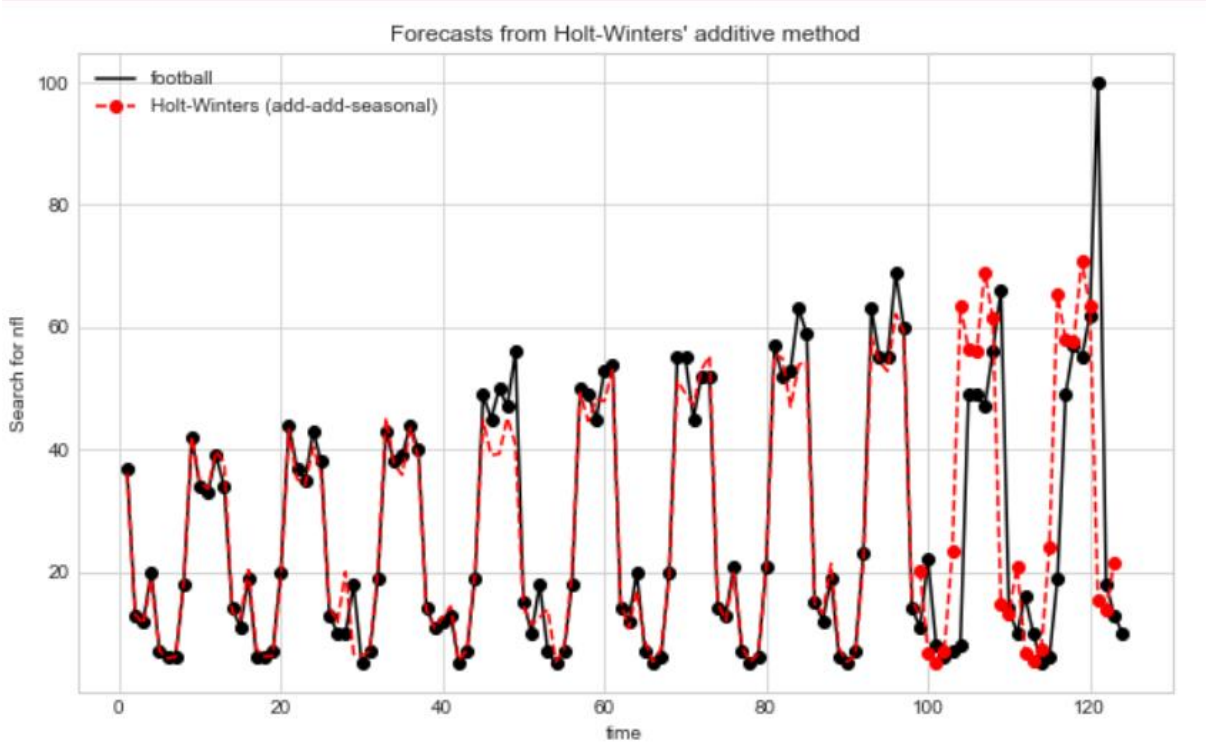
먼저 additive Holt-winter 지수평활법을 진행했다.

```
fit1 = ExponentialSmoothing(
    train_data,
    seasonal_periods=12,
    trend="add",
    seasonal="add",
    use_boxcox=True,
    initialization_method="estimated",
).fit()

ax = data.plot(
    figsize=(10, 6),
    marker="o",
    color="black",
    title="Forecasts from Holt-Winters' additive method",
)
ax.set_ylabel("Search for nfl")
ax.set_xlabel("Time")
fit1.fittedvalues.plot(ax=ax, style="--", color="red")


fit1.forecast(25).rename("Holt-Winters (add-add-seasonal)").plot(
    ax=ax, style="--", marker="o", color="red", legend=True
)

plt.show()
```

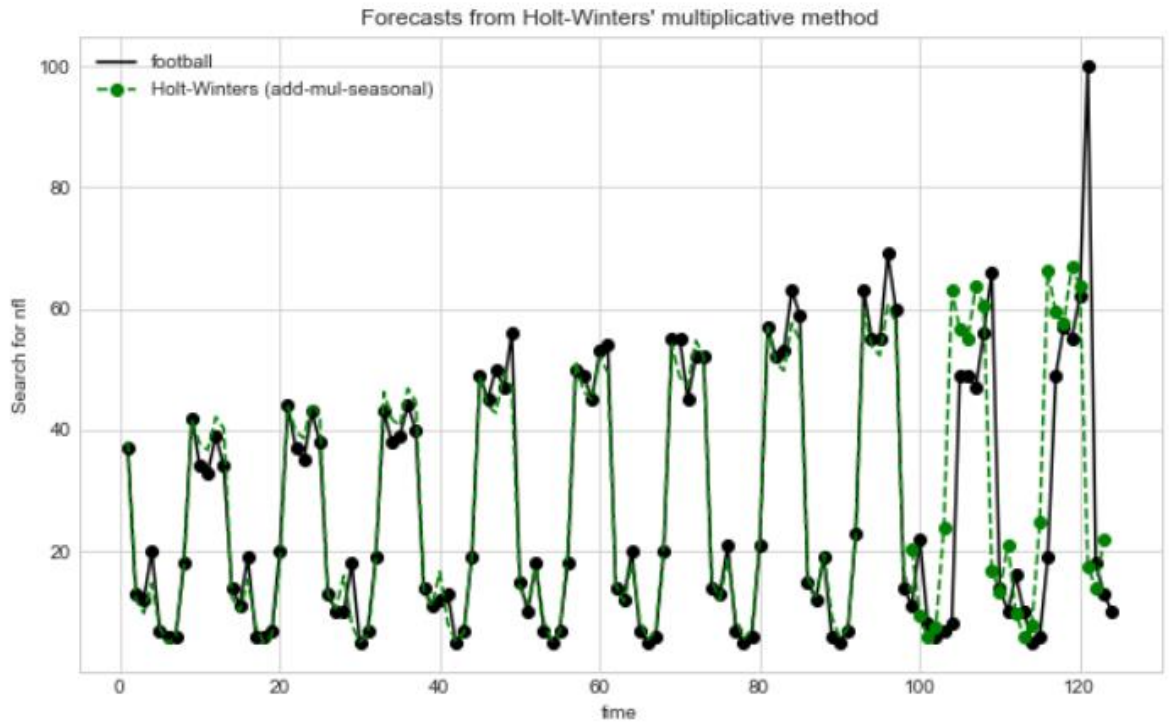


Additive Holt-Winter 지수평활법을 한 결과이다. 계절성을 잘 catch 하고 있음을 알 수 있다.

```
fit2 = ExponentialSmoothing(
    train_data,
    seasonal_periods=12,
    trend="add",
    seasonal="mul",
    use_boxcox=True,
    initialization_method="estimated",
).fit()

ax = data.plot(
    figsize=(10, 6),
    marker="o",
    color="black",
    title="Forecasts from Holt-Winters' multiplicative method",
)
ax.set_ylabel("Search for nfl")
ax.set_xlabel("Time")
fit2.fittedvalues.plot(ax=ax, style="--", color="green")

fit2.forecast(25).rename("Holt-Winters (add-mul-seasonal)").plot(
    ax=ax, style="--", marker="o", color="green", legend=True
)
plt.show()
```



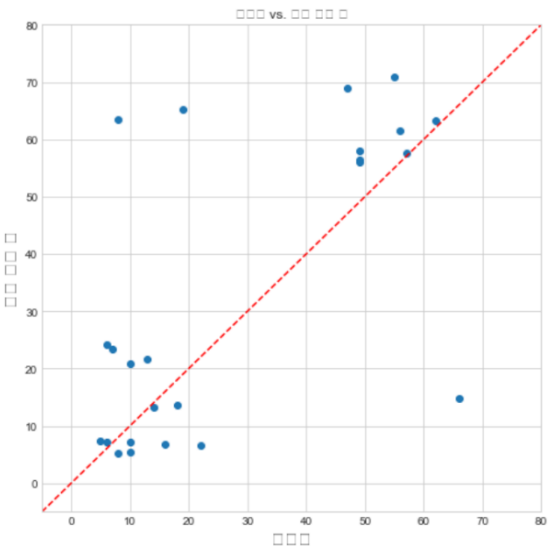
Multiplicative Holt-Winter 지수평활법을 한 결과이다. 계절성을 잘 catch 하고 있음을 알 수 있다.

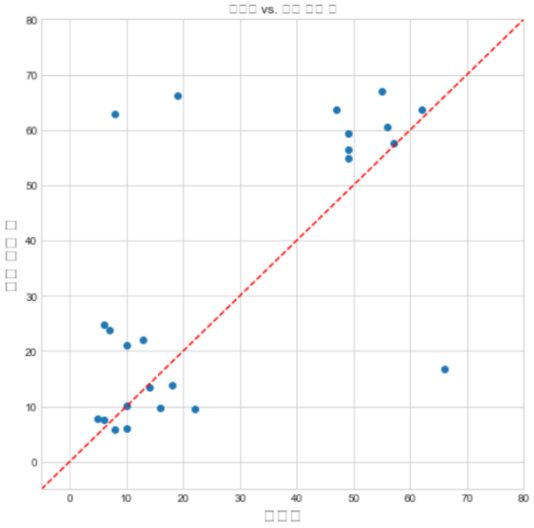
```
results = pd.DataFrame(
    index=[r"$\alpha$", r"$\beta$", r"$\phi$", r"$\gamma$", r"$l_0$", "$b_0$", "SSE"]
)
params = [
    "smoothing_level",
    "smoothing_trend",
    "damping_trend",
    "smoothing_seasonal",
    "initial_level",
    "initial_trend",
]
results["Additive"] = [fit1.params[p] for p in params] + [fit1.sse]
results["Multiplicative"] = [fit2.params[p] for p in params] + [fit2.sse]
results
```

	Additive	Multiplicative
α	1.490117e-08	1.490116e-08
β	1.468295e-08	1.186225e-08
ϕ	NaN	NaN
γ	7.301119e-01	0.000000e+00
l_0	3.552006e+00	3.528988e+00
b_0	4.154897e-03	5.460651e-03
SSE	1.173341e+03	8.542533e+02

최적의 파라미터 결과를 살펴보면 α 는 강의자료에서의 α (smoothing level), β 는 강의자료에서의 γ (smoothing trend), γ 는 강의자료에서의 δ (smoothing seasonal)를 의미한다. α, β, γ 모두 거의 0 에 가까운 값을 얻었다. SSE 값을 통해 Multiplicative 가 조금 더 정교하게 fitted 되었음을 알 수 있다.

Additive Holt-Winter 지수평활법의 예측정확도를 정성적으로 분석해보면 실제값과 출력값이 차이가 많이 나는 outlier 가 있기는 하지만, 대부분 y=x 추세를 따라가기 때문에 예측정확도가 무난하다고 정성적으로 판단할 수 있다.





Multiplicative Holt-Winter 지수평활법의 예측정확도를 정성적으로 분석하면 역시 실제값과 출력값이 차이가 많이 나는 outlier 가 있기는 하지만, 대부분 y=x 추세를 따라가기 때문에 예측정확도가 무난하다고 정성적으로 판단할 수 있다.

< Additive Holt-Winter 지수평활법의 예측정확도>

```
print('Testing MSE: {:.3f}'.format(mean_squared_error(test_data, fit1.forecast(26)[1:])))
print('Testing RMSE: {:.3f}'.format(np.sqrt(mean_squared_error(test_data, fit1.forecast(26)[1:]))))
print('Testing MAE: {:.3f}'.format(mean_absolute_error(test_data, fit1.forecast(26)[1:])))
print('Testing MAPE: {:.3f}'.format(mean_absolute_percentage_error(test_data, fit1.forecast(26)[1:])))
print('Testing R2: {:.3f}'.format(r2_score(test_data, fit1.forecast(26)[1:])))
```

```
Testing MSE: 684.349
Testing RMSE: 26.160
Testing MAE: 16.136
Testing MAPE: 181.746
Testing R2: -0.059
```

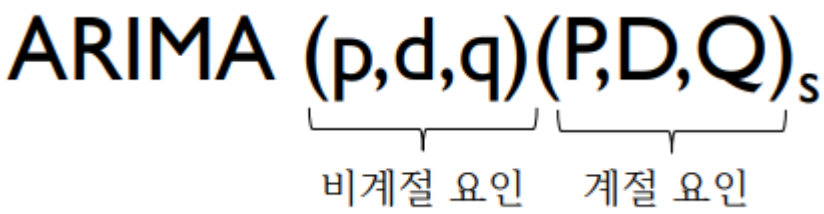
< Multiplicative Holt-Winter 지수평활법의 예측정확도>

```
print('Testing MSE: {:.3f}'.format(mean_squared_error(test_data, fit2.forecast(26)[1:])))
print('Testing RMSE: {:.3f}'.format(np.sqrt(mean_squared_error(test_data, fit2.forecast(26)[1:]))))
print('Testing MAE: {:.3f}'.format(mean_absolute_error(test_data, fit2.forecast(26)[1:])))
print('Testing MAPE: {:.3f}'.format(mean_absolute_percentage_error(test_data, fit2.forecast(26)[1:])))
print('Testing R2: {:.3f}'.format(r2_score(test_data, fit2.forecast(26)[1:])))
```

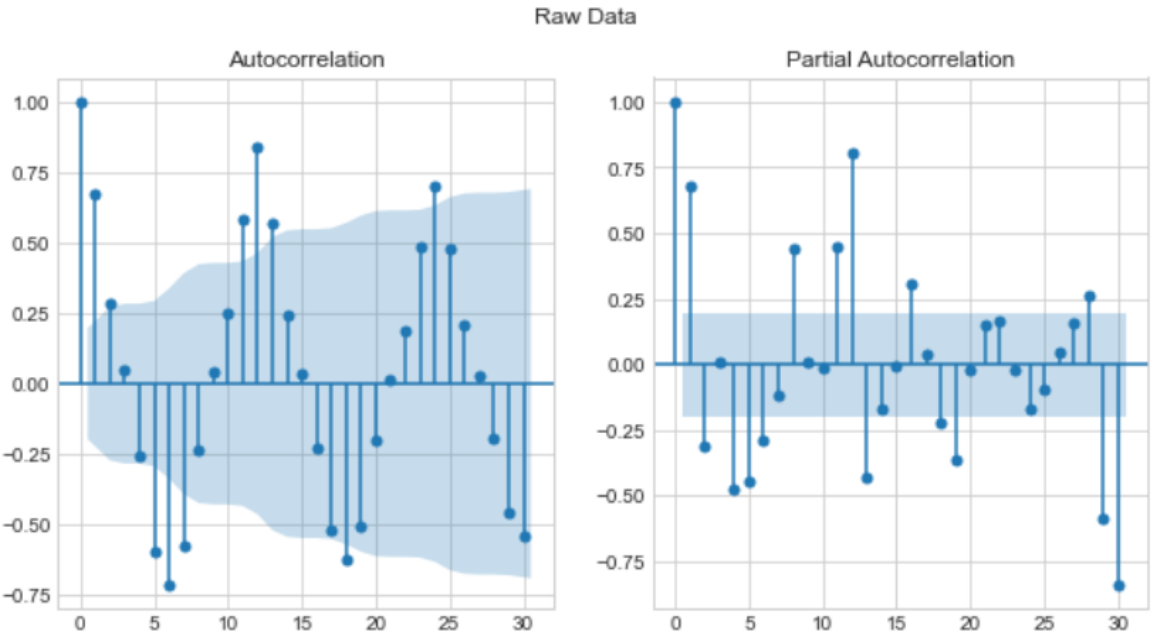
```
Testing MSE: 647.877
Testing RMSE: 25.453
Testing MAE: 15.299
Testing MAPE: 179.833
Testing R2: -0.002
```

예측정확도가 큰 편이고, R2 값도 음수이다. 이는 예측해야하는 범위가 너무 길기도 했고, test data 에서 큰 fluctuation 이 있었기 때문에 예측하기 어려웠던 점이 작용했을 것이다. 그래도 Multiplicative Holt-Winter 지수평활법이 Additive-Holt Winter 지수평활법보다 오차도 작고 R2 값이 크기 때문에 잘 예측한 모델이라는 결론은 내릴 수 있다.

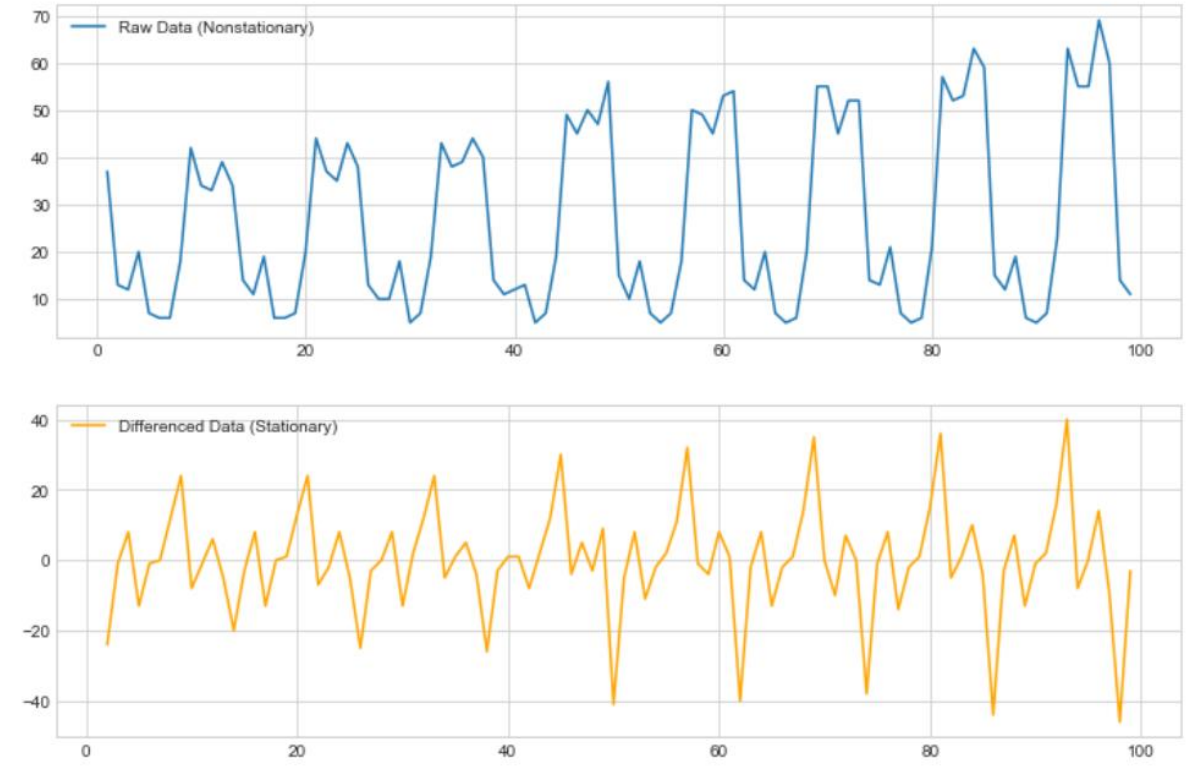
SARIMA 로 모델링해보았다. SARIMA 는 seasonal variation 을 반영한 ARIMA 모델이다.



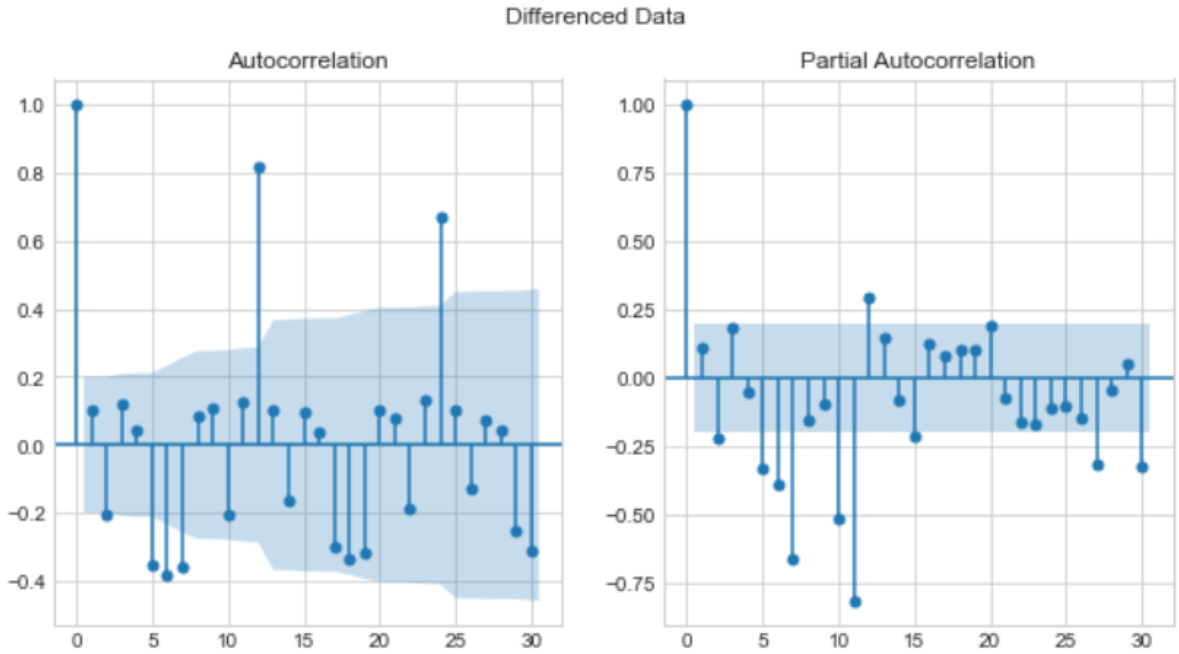
파라미터는 총 7 개이다.



ACF, PACF plot 을 보면 사인 함수 형태로 감소하기 때문에 nonstationary data 라고 할 수 있다. 따라서 differencing 을 통해 stationary 하기 만들었다.



차분한 데이터를 보면 stationary 에 가까워졌음을 알 수 있다.



ACF, PACF plot 을 분석해봐도 특정한 패턴이 없기 때문에 stationary data 라고 할 수 있다.

파이썬 기능을 통해 AIC 가 최소인 최적의 파라미터 조합을 찾았다.

```
[(((0, 1, 1), (0, 1, 0, 12)), 482.19)]
```

AIC 가 482.19 로 최소일 때의 파라미터 조합이다.

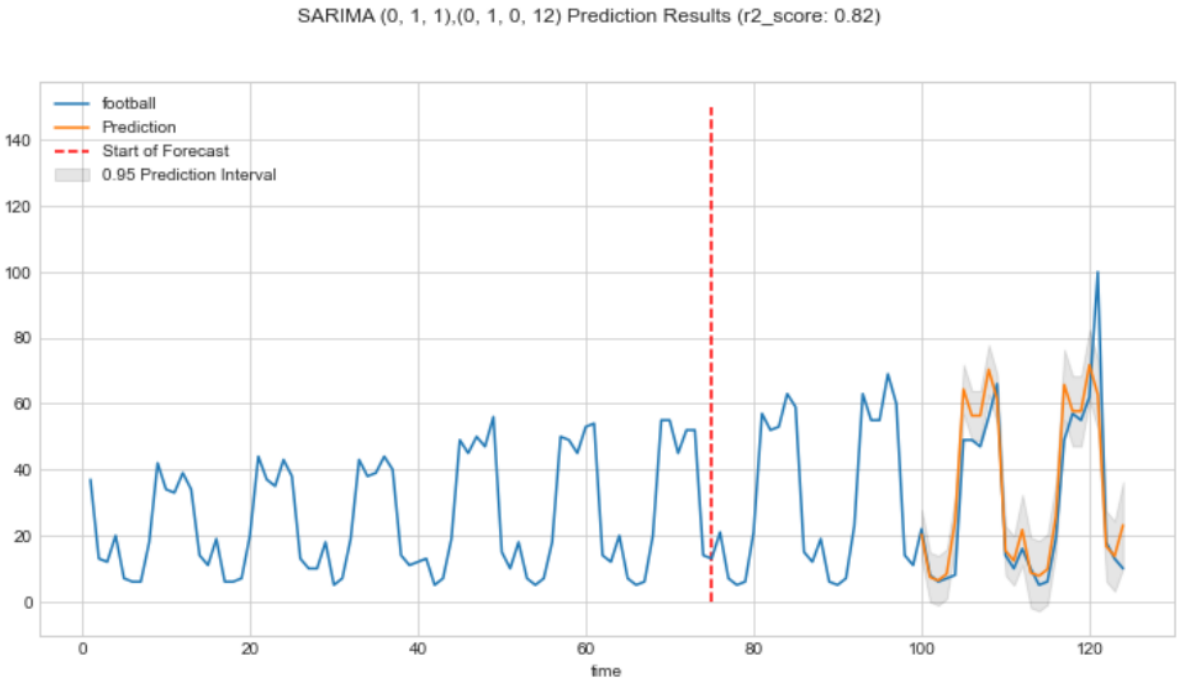
SARIMAX Results

Dep. Variable:	y	**No. Observations:**	99
Model:	SARIMAX(0, 1, 1)x(0, 1, [], 12)	**Log Likelihood**	-239.094
Date:	Sun, 10 Apr 2022	**AIC**	482.188
Time:	17:40:32	**BIC**	487.097
Sample:	0	**HQIC**	484.164
	- 99		
Covariance Type:	opg		

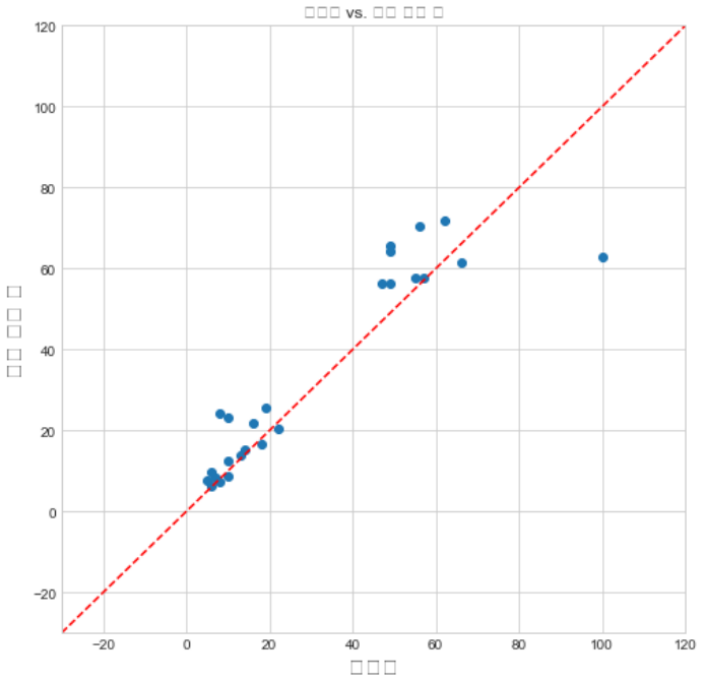
	coef	std err	z	P>|z|	[0.025	0.975]
ma.L1	-0.9990	1.310	-0.763	0.446	-3.566	1.568
sigma2	14.4616	18.073	0.800	0.424	-20.961	49.885

Ljung-Box (L1) (Q):	0.73	**Jarque-Bera (JB):**	33.35
Prob(Q):	0.39	**Prob(JB):**	0.00
Heteroskedasticity (H):	0.88	**Skew:**	0.92
Prob(H) (two-sided):	0.72	**Kurtosis:**	5.43

SARIMAX 결과표이다. AIC 값이 가장 작은 값인 482.19 로 최적 파라미터가 정해졌다.



SARIMAX 그래프를 그려보면 seasonal variation 을 잘 따라가고 있음을 알 수 있고, R2 값이 0.82 이므로 평균값 사용하는 것 대비 82%의 설명력 향상을 보여준다.



SARIMA 의 예측정확도를 정성적으로 분석해보면 점들이 y=x 추세를 대부분 따르기 때문에 예측정확도가 높다고 분석할 수 있다.

```
print('Testing MSE: {:.3f}'.format(mean_squared_error(test_data, predicted_value)))
print('Testing RMSE: {:.3f}'.format(np.sqrt(mean_squared_error(test_data, predicted_value))))
print('Testing MAE: {:.3f}'.format(mean_absolute_error(test_data, predicted_value)))
print('Testing MAPE: {:.3f}'.format(mean_absolute_percentage_error(test_data, predicted_value)))
print('Testing R2: {:.3f}'.format(r2_score(test_data, predicted_value)))
```

```
Testing MSE: 117.442
Testing RMSE: 10.837
Testing MAE: 7.119
Testing MAPE: 184.878
Testing R2: 0.818
```

예측정확도는 Holt-winter 지수평활법보다 매우 정확하고, R2 값도 상당히 높다. 따라서 SARIMA 모델이 더 정교하고 정확한 예측모델이라고 결론지었다.

2-2. Additive Holt-Winters 와 Multiplicative Holt-Winters 방법에서 Weighting parameter (alpha, gamma, delta) 변경하여 예측력 비교

최적의 파라미터 값들이 모두 작았기 때문에 살짝 크게 해서 비교해보았다.

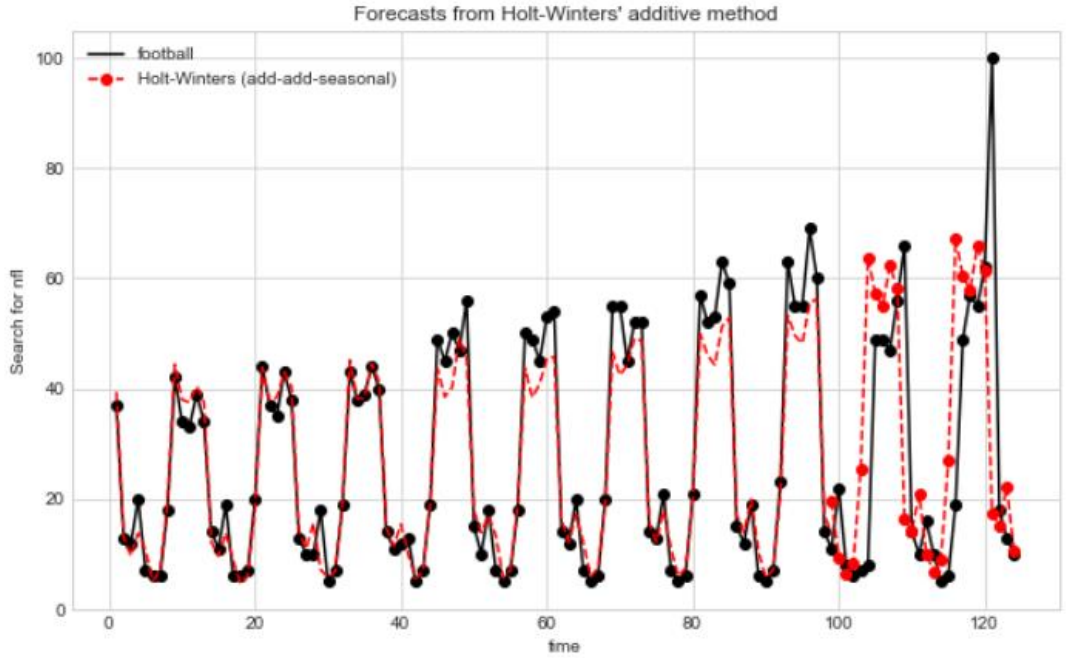
Case 1) α =0.2 γ=0.2 δ=0.2 Additive

```
fit3 = ExponentialSmoothing(
    train_data,
    seasonal_periods=12,
    trend="add",
    seasonal="add",
    use_boxcox=True,
    initialization_method="estimated",
).fit(smoothing_level=0.2, smoothing_trend=0.2, smoothing_seasonal=0.2, optimized=False)

ax = data.plot(
    figsize=(10, 6),
    marker="o",
    color="black",
    title="Forecasts from Holt-Winters' additive method",
)
ax.set_ylabel("Search for nfl")
ax.set_xlabel("Time")
fit3.fittedvalues.plot(ax=ax, style="--", color="red")

fit3.forecast(26).rename("Holt-Winters (add-add-seasonal)").plot(
    ax=ax, style="--", marker="o", color="red", legend=True
)

plt.show()
```



계절성을 잘 반영하는 편이다.

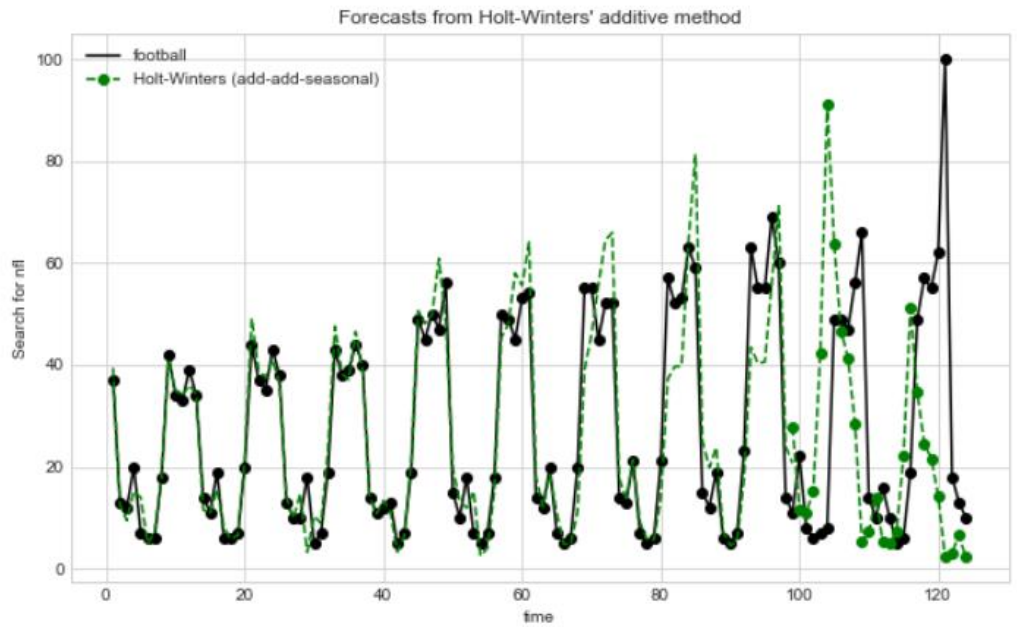
Case 2) α =0.5 γ=0.5 δ=0.5 Additive

```
fit4 = ExponentialSmoothing(
    train_data,
    seasonal_periods=12,
    trend="add",
    seasonal="add",
    use_boxcox=True,
    initialization_method="estimated",
).fit(smoothing_level=0.5, smoothing_trend=0.5, smoothing_seasonal=0.5

ax = data.plot(
    figsize=(10, 6),
    marker="o",
    color="black",
    title="Forecasts from Holt-Winters' additive method",
)
ax.set_ylabel("Search for nfl")
ax.set_xlabel("Time")
fit4.fittedvalues.plot(ax=ax, style="--", color="green")


fit4.forecast(26).rename("Holt-Winters (add-add-seasonal)").plot(
    ax=ax, style="--", marker="o", color="green", legend=True
)

plt.show()
```

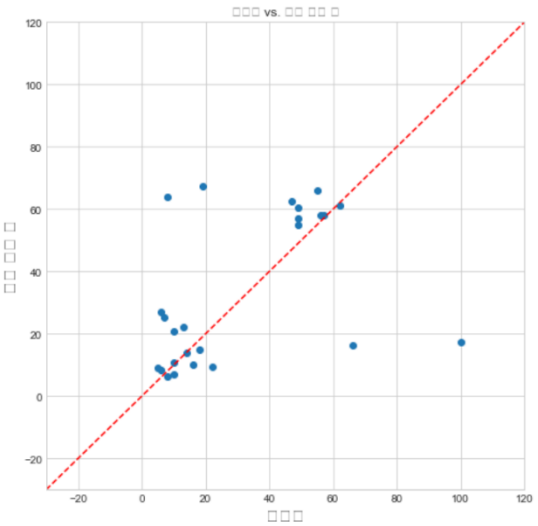


계절성을 반영하기는 하지만, 조금 더 부정확하다.

```
results = pd.DataFrame(
    index=[r"$\alpha$", r"$\beta$", r"$\phi$", r"$\gamma$", r"$l_0$",
)
params = [
    "smoothing_level",
    "smoothing_trend",
    "damping_trend",
    "smoothing_seasonal",
    "initial_level",
    "initial_trend",
]
results["case1"] = [fit3.params[p] for p in params] + [fit3.sse]
results["case2"] = [fit4.params[p] for p in params] + [fit4.sse]
results
```

	case1	case2
α	0.200000	0.500000
β	0.200000	0.500000
ϕ	NaN	NaN
γ	0.200000	0.500000
l_0	3.566255	3.566255
b_0	0.000423	0.000423
SSE	2089.455479	4941.675657

파라미터에 따른 SSE 값을 비교하면, case 2 가 더 크기 때문에 case 1 이 더 정교하게 fitted 되었음을 알 수 있다.

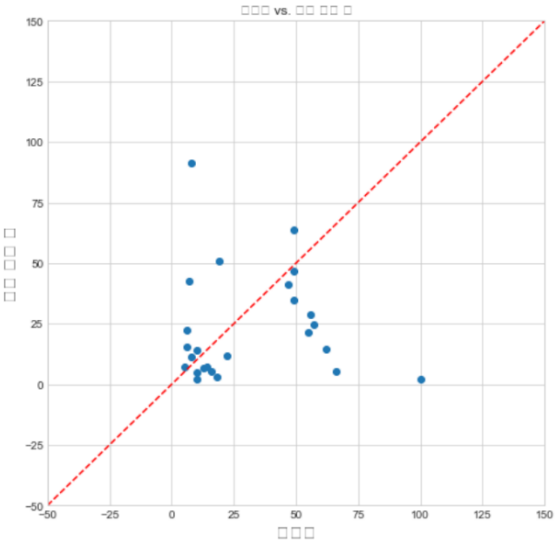


Case 1 의 예측정확도를 정성적으로 분석해보면 y=x 추세를 따르지 않는 데이터값이 일부 있지만 대부분 따르고 있다. 따라서 예측정확도가 무난할 것이라고 분석했다.

```
print('Testing MSE: {:.3f}'.format(mean_squared_error(test_data, fit3.forecast(26)[1:])))
print('Testing RMSE: {:.3f}'.format(np.sqrt(mean_squared_error(test_data, fit3.forecast(26)[1:]))))
print('Testing MAE: {:.3f}'.format(mean_absolute_error(test_data, fit3.forecast(26)[1:])))
print('Testing MAPE: {:.3f}'.format(mean_absolute_percentage_error(test_data, fit3.forecast(26)[1:])))
print('Testing R2: {:.3f}'.format(r2_score(test_data, fit3.forecast(26)[1:])))
```

```
Testing MSE: 661.209
Testing RMSE: 25.714
Testing MAE: 15.389
Testing MAPE: 179.851
Testing R2: -0.023
```

예측정확도를 살펴보면 R2 값이 음수이기는 하지만 절댓값이 크지 않고, 오차 지표들도 최적 파라미터 Additive Holt-winter 방법과 비슷하다.



Case 2 의 예측정확도를 정성적으로 분석해보면 y=x 추세를 따르지 않는 데이터값이 일부 있다. 따라서 예측정확도가 낮을 것이라고 분석했다.

```
print('Testing MSE: {:.3f}'.format(mean_squared_error(test_data, fit4.forecast(26)[1:])))
print('Testing RMSE: {:.3f}'.format(np.sqrt(mean_squared_error(test_data, fit4.forecast(26)[1:]))))
print('Testing MAE: {:.3f}'.format(mean_absolute_error(test_data, fit4.forecast(26)[1:])))
print('Testing MAPE: {:.3f}'.format(mean_absolute_percentage_error(test_data, fit4.forecast(26)[1:])))
print('Testing R2: {:.3f}'.format(r2_score(test_data, fit4.forecast(26)[1:])))
```

```
Testing MSE: 1162.712
Testing RMSE: 34.099
Testing MAE: 23.355
Testing MAPE: 136.821
Testing R2: -0.799
```

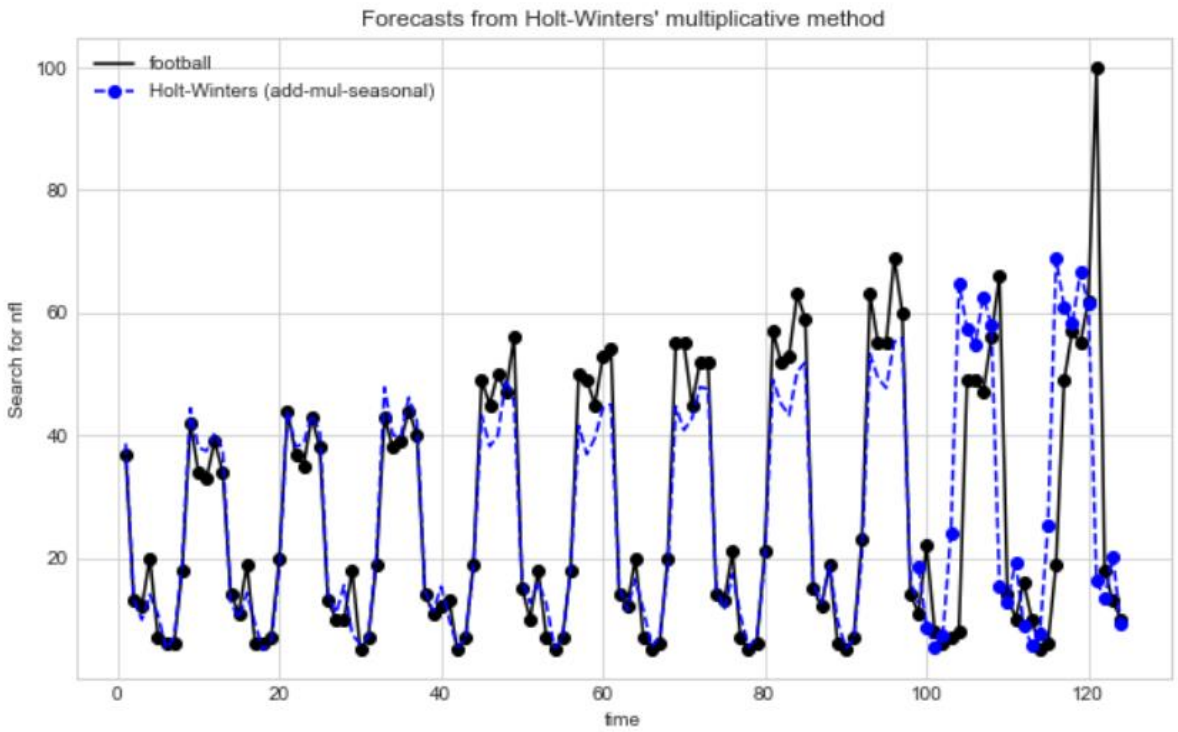
예측 정확도도 case 1 에 비해 오차 지표들도 크고, R2 값도 매우 작다.

Case 3) α =0.2 γ=0.2 δ=0.2 Multiplicative

```
fit5 = ExponentialSmoothing(
    train_data,
    seasonal_periods=12,
    trend="add",
    seasonal="mul",
    use_boxcox=True,
    initialization_method="estimated",
).fit(smoothing_level=0.2, smoothing_trend=0.2, smoothing_seasonal=0.2, optimized=False)

ax = data.plot(
    figsize=(10, 6),
    marker="o",
    color="black",
    title="Forecasts from Holt-Winters' multiplicative method",
)
ax.set_ylabel("Search for nfl")
ax.set_xlabel("Time")
fit5.fittedvalues.plot(ax=ax, style="--", color="green")

fit5.forecast(26).rename("Holt-Winters (add-mul-seasonal)").plot(
    ax=ax, style="--", marker="o", color="green", legend=True
)
plt.show()
```



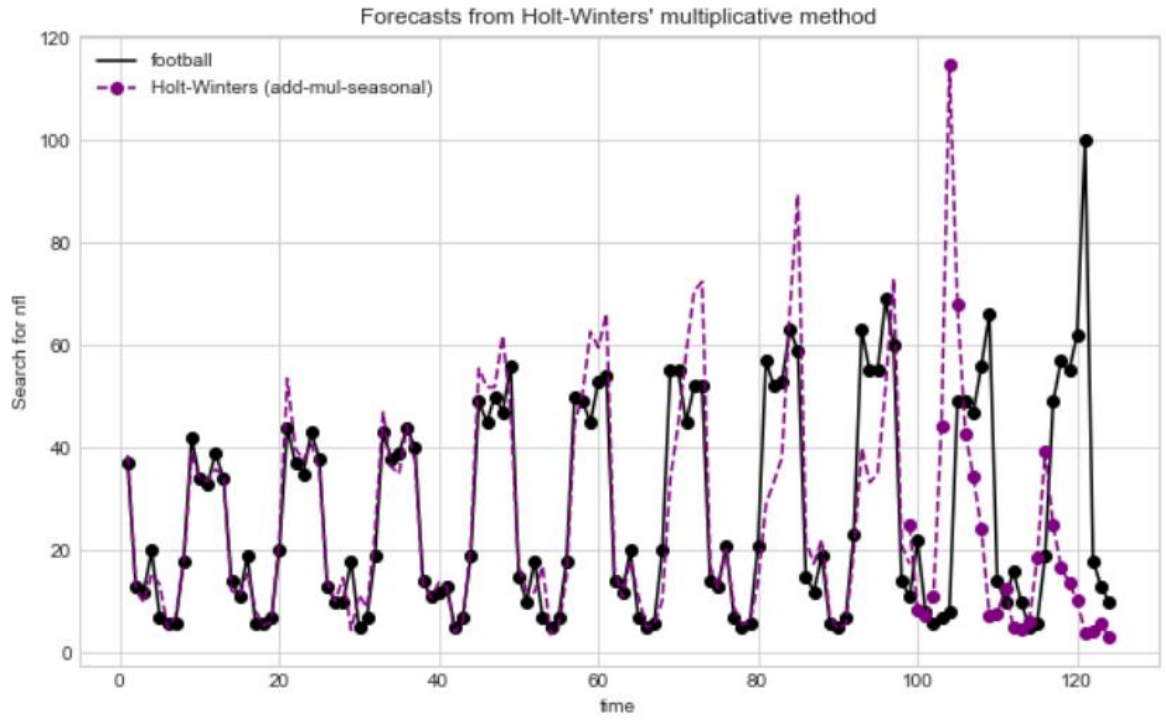
계절성을 반영하지만, 오차가 약간 있다.

Case 4) α =0.5 γ=0.5 δ=0.5 Multiplicative

```
fit6 = ExponentialSmoothing(
    train_data,
    seasonal_periods=12,
    trend="add",
    seasonal="mul",
    use_boxcox=True,
    initialization_method="estimated",
).fit(smoothing_level=0.5, smoothing_trend=0.5, smoothing_seasonal=0.5, optimized=False)

ax = data.plot(
    figsize=(10, 6),
    marker="o",
    color="black",
    title="Forecasts from Holt-Winters' multiplicative method",
)
ax.set_ylabel("Search for nfl")
ax.set_xlabel("Time")
fit6.fittedvalues.plot(ax=ax, style="--", color="purple")

fit6.forecast(26).rename("Holt-Winters (add-mul-seasonal)").plot(
    ax=ax, style="--", marker="o", color="purple", legend=True
)
plt.show()
```

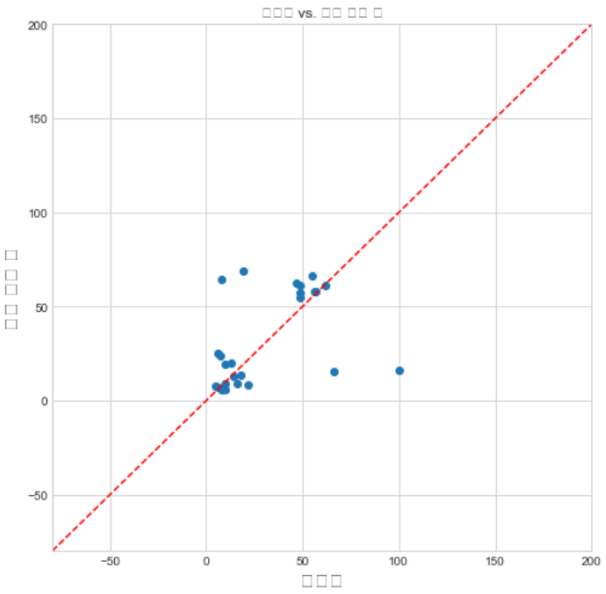


계절성을 반영하기는 하지만, 오차가 큰 편이다.

```
results = pd.DataFrame(
    index=[r"$\alpha$", r"$\beta$", r"$\phi$", r"$\gamma$", r"$l_0$", "$b_0$", "SSE"]
)
params = [
    "smoothing_level",
    "smoothing_trend",
    "damping_trend",
    "smoothing_seasonal",
    "initial_level",
    "initial_trend",
]
results["case3"] = [fit5.params[p] for p in params] + [fit5.sse]
results["case4"] = [fit6.params[p] for p in params] + [fit6.sse]
results
```

	case3	case4
α	0.200000	0.500000
β	0.200000	0.500000
ϕ	NaN	NaN
γ	0.200000	0.500000
l_0	3.566255	3.566255
b_0	0.000423	0.000423
SSE	2370.977546	7668.073064

SSE 를 비교하면 case 4 가 매우 크기 때문에 case 3 이 더 정교하게 fitted 된 모델이다.

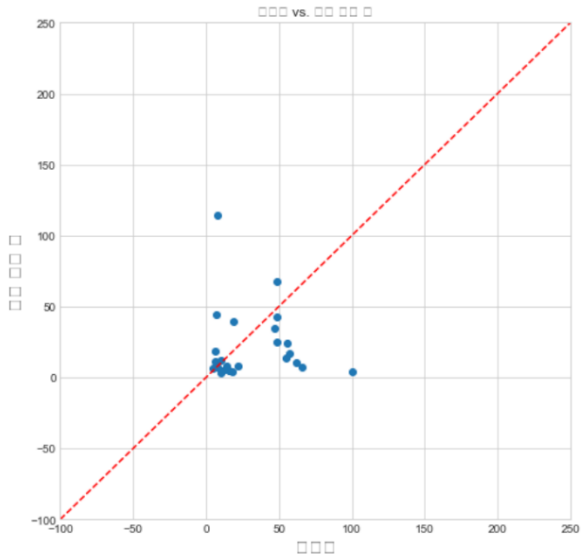


정성적으로 case 3 의 예측정확도를 분석하면, 대부분 y=x 추세를 따르지만, 일부 벗어난 데이터가 있기 때문에 예측정확도는 무난할 것이라고 분석했다.

```
print('Testing MSE: {:.3f}'.format(mean_squared_error(test_data, fit5.forecast(26)[1:])))
print('Testing RMSE: {:.3f}'.format(np.sqrt(mean_squared_error(test_data, fit5.forecast(26)[1:]))))
print('Testing MAE: {:.3f}'.format(mean_absolute_error(test_data, fit5.forecast(26)[1:])))
print('Testing MAPE: {:.3f}'.format(mean_absolute_percentage_error(test_data, fit5.forecast(26)[1:])))
print('Testing R2: {:.3f}'.format(r2_score(test_data, fit5.forecast(26)[1:])))
```

```
Testing MSE: 679.679
Testing RMSE: 26.071
Testing MAE: 15.547
Testing MAPE: 179.133
Testing R2: -0.051
```

예측정확도 자체는 case 1 과 유사하다. 하지만 오차 지표가 약간 더 크고, R2 값이 약간 더 작기 때문에 case 1 이 조금 더 좋은 모델이다.



Case 4 의 예측정확도를 정성적으로 분석하면 y=x 추세를 벗어나는 데이터가 꽤 있다. 따라서 예측정확도가 높지 않을 것이라고 분석했다.

```
print('Testing MSE: {:.3f}'.format(mean_squared_error(test_data, fit6.forecast(26)[1:])))
print('Testing RMSE: {:.3f}'.format(np.sqrt(mean_squared_error(test_data, fit6.forecast(26)[1:]))))
print('Testing MAE: {:.3f}'.format(mean_absolute_error(test_data, fit6.forecast(26)[1:])))
print('Testing MAPE: {:.3f}'.format(mean_absolute_percentage_error(test_data, fit6.forecast(26)[1:])))
print('Testing R2: {:.3f}'.format(r2_score(test_data, fit6.forecast(26)[1:])))
```

```
Testing MSE: 1393.544
Testing RMSE: 37.330
Testing MAE: 25.272
Testing MAPE: 130.371
Testing R2: -1.156
```

예측정확도를 보면 오차지표들도 상당히 크고, R2 값도 상당히 작기 때문에 제일 좋지 않는 모델이다.

<case 1>

```
print('Testing MSE: {:.3f}'.format(mean_squared_error(test_data, fit3.forecast(26)[1:])))
print('Testing RMSE: {:.3f}'.format(np.sqrt(mean_squared_error(test_data, fit3.forecast(26)[1:]))))
print('Testing MAE: {:.3f}'.format(mean_absolute_error(test_data, fit3.forecast(26)[1:])))
print('Testing MAPE: {:.3f}'.format(mean_absolute_percentage_error(test_data, fit3.forecast(26)[1:])))
print('Testing R2: {:.3f}'.format(r2_score(test_data, fit3.forecast(26)[1:])))
```

```
Testing MSE: 661.209
Testing RMSE: 25.714
Testing MAE: 15.389
Testing MAPE: 179.851
Testing R2: -0.023
```

<case 2>

```
print('Testing MSE: {:.3f}'.format(mean_squared_error(test_data, fit4.forecast(26)[1:])))
print('Testing RMSE: {:.3f}'.format(np.sqrt(mean_squared_error(test_data, fit4.forecast(26)[1:]))))
print('Testing MAE: {:.3f}'.format(mean_absolute_error(test_data, fit4.forecast(26)[1:])))
print('Testing MAPE: {:.3f}'.format(mean_absolute_percentage_error(test_data, fit4.forecast(26)[1:])))
print('Testing R2: {:.3f}'.format(r2_score(test_data, fit4.forecast(26)[1:])))
```

```
Testing MSE: 1162.712
Testing RMSE: 34.099
Testing MAE: 23.355
Testing MAPE: 136.821
Testing R2: -0.799
```

<case 3>

```
print('Testing MSE: {:.3f}'.format(mean_squared_error(test_data, fit5.forecast(26)[1:])))
print('Testing RMSE: {:.3f}'.format(np.sqrt(mean_squared_error(test_data, fit5.forecast(26)[1:]))))
print('Testing MAE: {:.3f}'.format(mean_absolute_error(test_data, fit5.forecast(26)[1:])))
print('Testing MAPE: {:.3f}'.format(mean_absolute_percentage_error(test_data, fit5.forecast(26)[1:])))
print('Testing R2: {:.3f}'.format(r2_score(test_data, fit5.forecast(26)[1:])))
```

```
Testing MSE: 679.679
Testing RMSE: 26.071
Testing MAE: 15.547
Testing MAPE: 179.133
Testing R2: -0.051
```

<case 4>

```
print('Testing MSE: {:.3f}'.format(mean_squared_error(test_data, fit6.forecast(26)[1:])))
print('Testing RMSE: {:.3f}'.format(np.sqrt(mean_squared_error(test_data, fit6.forecast(26)[1:]))))
print('Testing MAE: {:.3f}'.format(mean_absolute_error(test_data, fit6.forecast(26)[1:])))
print('Testing MAPE: {:.3f}'.format(mean_absolute_percentage_error(test_data, fit6.forecast(26)[1:])))
print('Testing R2: {:.3f}'.format(r2_score(test_data, fit6.forecast(26)[1:])))
```

```
Testing MSE: 1393.544
Testing RMSE: 37.330
Testing MAE: 25.272
Testing MAPE: 130.371
Testing R2: -1.156
```

한번에 비교해보면 파라미터가 작은 case 들이 MAPE 값이 크기 때문에 예측을 크게 한 것이라고 분석할 수 있고, 나머지 오차 지표에서 파라미터가 작은 case 들이 작고, R2

값도 크다. 그리고 Case 1 과 3 을 비교했을 때 1 이 미묘하게 예측정확도가 더 좋다는 것을 알 수 있다.

고려대학교

교훈 : 자유, 정의, 진리 LIBERTAS, JUSTITIA, VERITAS

상징동물 : 호랑이

교목 : 잣나무

교색 : 크림슨

교가 :

북악산 기슭에 우뚝 솟은 집을 보라

안암의 언덕에 퍼져나는 빛을 보라
겨레의 보람이요 정성이 뭉쳐
드높이 쌓아 올린 공든 탑
자유 정의 진리의 전당이 있다