삼성 주가 예측

4팀 이은지, 김지은, 신아진, 현희섭

데이터 수집: Yahoo Finance 데이터 불러오기

```
%matplotlib inline
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
!pip install yfinance

import yfinance as yf


samsung_df = yf.download('005930.KS',
                         start='2020-01-01',
                         end='2021-04-20',
                         progress=False)

samsung_df = samsung_df[["Close"]]

samsung_df = samsung_df.reset_index()

samsung_df.columns = ['day', 'price']

samsung_df['day'] = pd.to_datetime(samsung_df['day'])

samsung_df.index = samsung_df['day']
samsung_df.set_index('day', inplace=True)

samsung_df
```

야후 파이낸스(http://finance.yahoo.com)

	price
day	
2020-01-02	55200.0
2020-01-03	55500.0
2020-01-06	55500.0
2020-01-07	55800.0
2020-01-08	56800.0
...	...
2021-04-14	84000.0
2021-04-15	84100.0
2021-04-16	83900.0
2021-04-19	83300.0
2021-04-20	83900.0

322 rows × 1 columns

데이터 탐색

```
samsung_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 322 entries, 2020-01-02 to 2021-04-20
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   price   322 non-null    float64
dtypes: float64(1)
memory usage: 5.0 KB
```

```
samsung_train_df = samsung_df[:317]
samsung_train_df
```

day	price
2020-01-02	55200.0
2020-01-03	55500.0
2020-01-06	55500.0
2020-01-07	55800.0
2020-01-08	56800.0
...	...
2021-04-07	85600.0
2021-04-08	84700.0
2021-04-09	83600.0
2021-04-12	83200.0
2021-04-13	84000.0

317 rows × 1 columns

```
samsung_test_df = samsung_df[317:]
samsung_test_df
```

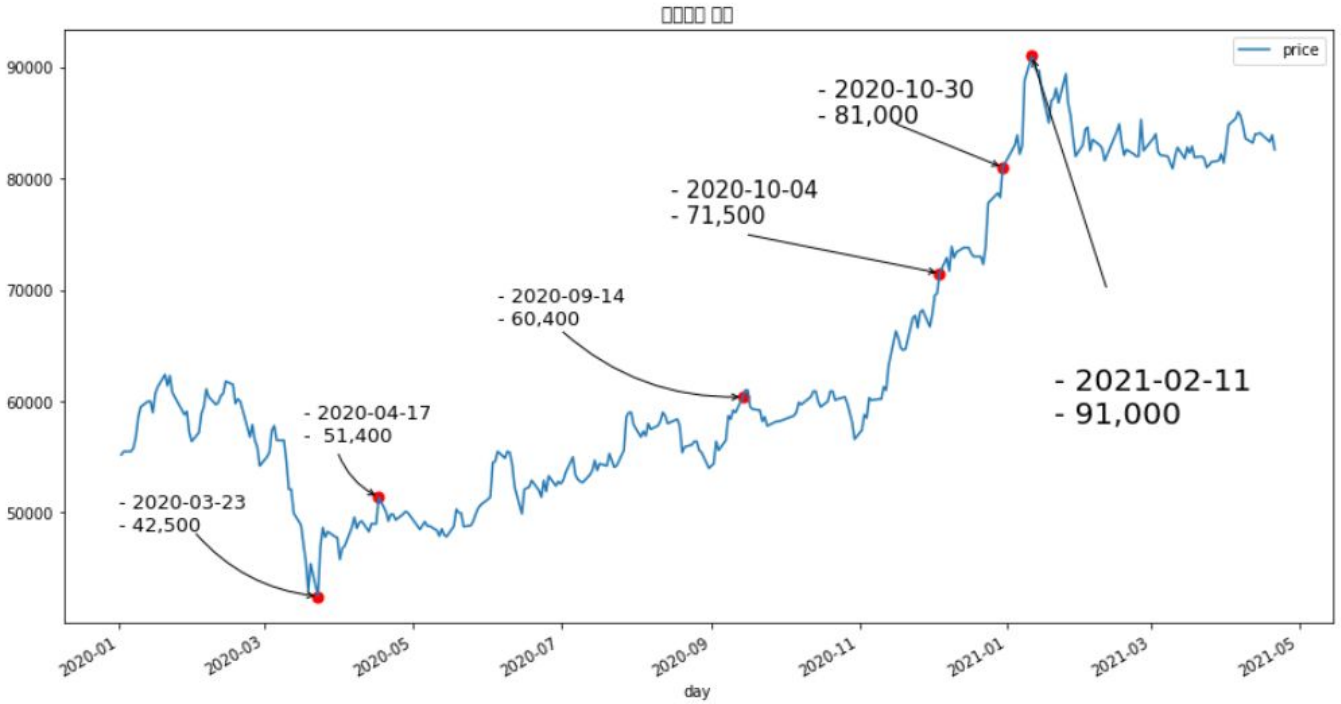
day	price
2021-04-14	84000.0
2021-04-15	84100.0
2021-04-16	83900.0
2021-04-19	83300.0
2021-04-20	83900.0

데이터 시각화

```
fig, ax = plt.subplots(figsize=(15, 8))
samsung_df.plot(ax=ax)

# 4만 최저점
ax.annotate('', xy=('2020-03-23' ,42500.0), xytext=('2020-02-01', 48300.0),
            arrowprops=dict(arrowstyle="->",
                            connectionstyle="arc3,rad=0.2"),
            )

plt.text('2020-01-01',48300, "₩n- 2020-03-23 ₩n- 42,500",fontsize=13)

# 5만
ax.annotate('', xy=('2020-04-17' ,51400.0), xytext=('2020-03-31', 55500),
            arrowprops=dict(arrowstyle="->",
                            connectionstyle="arc3,rad=0.2"),
            )

plt.text('2020-03-17',56400, "₩n- 2020-04-17₩n-  51,400",fontsize=13)

# 6만
ax.annotate('', xy=('2020-09-14' ,60400.0), xytext=('2020-07-01', 66400),
            arrowprops=dict(arrowstyle="->",
                            connectionstyle="arc3,rad=0.2"),
            )

plt.text('2020-06-05',66900, "₩n- 2020-09-14₩n- 60,400",fontsize=13)

# 7만
ax.annotate('', xy=('2020-12-04' ,71500.0), xytext=('2020-09-15', 75000),
            arrowprops=dict(arrowstyle="->",
                            connectionstyle="arc3,rad=0"),
            )

plt.text('2020-08-15',76000, "₩n- 2020-10-04₩n- 71,500",fontsize=15)
```

```
# 8만
ax.annotate('', xy=('2020-12-30' ,81000.0), xytext=('2020-11-15', 85000),
            arrowprops=dict(arrowstyle="->",
                            connectionstyle="arc3,rad=0"),
            )

plt.text('2020-10-15',85000, " ₩n- 2020-10-30₩n- 81,000",fontsize=16)

# 9만
ax.annotate('', xy=('2021-01-11' ,91000.0), xytext=('2021-02-11', 70000),
            arrowprops=dict(arrowstyle="->",
                            connectionstyle="arc3,rad=0"),
            )

plt.text('2021-01-20',58000, " ₩n- 2021-02-11 ₩n- 91,000",fontsize=20)



# Scatter plot 추가
y1 = ['2020-03-23','2020-04-17' ,'2020-09-14','2020-12-04','2020-12-30','2021-01-11']
y2 = [42500,51400,60400,71500,81000,91000]


plt.scatter(y1,y2,s=50,color='r')

plt.title("삼성전자 주가")
plt.show()
```

데이터 시각화


price
- 2020-10-30
- 81,000
- 2020-10-04
- 71,500
- 2020-09-14
- 60,400
- 2020-04-17
- 51,400
- 2020-03-23
- 42,500
- 2021-02-11
- 91,000
90000
80000
70000
60000
50000
2020-01
2020-03
2020-05
2020-07
2020-09
2020-11
2021-01
2021-03
2021-05
day

Model 1: ARIMA

ARIMA란?

- 전통적인 시계열 예측 방법
- ARMA모델 + 추세 변동의 경향성 / ARMA 모델 : AR(Autoregression) + MA(Moving Average)
- statsmodel 모듈로 ARIMA 분석 수행 가능

In [10]:
```
from statsmodels.tsa.arima_model import ARIMA
import statsmodels.api as sm
```

ARIMA 분석 : order = (p,d,q)

- p : AR이 몇 번째 과거까지를 바라보는지
- d : 차분(difference), 즉 현재 상태의 변수에서 바로 전 상태의 변수를 빼준 것
- q : MA가 몇 번째 과거까지를 바라보는지

In [10]:

```
# (AR = 2, 차분 =1, MA=2) 파라미터로 ARIMA 모델을 학습한다.
model = ARIMA(samsung_train_df.price.values, order = (2,1,2))
model_fit = model.fit(trend = 'c', full_output = True, disp = True)
print(model_fit.summary())
```

Out [10]:

ARIMA Model Results

Dep. Variable:	D.y	No. Observations:	316
Model:	ARIMA(2, 1, 2)	Log Likelihood	-2693.879
Method:	css-mle	S.D. of innovations	1212.701
Date:	Tue, 21 Dec 2021	AIC	5399.758
Time:	11:36:09	BIC	5422.293
Sample:	1	HQIC	5408.761

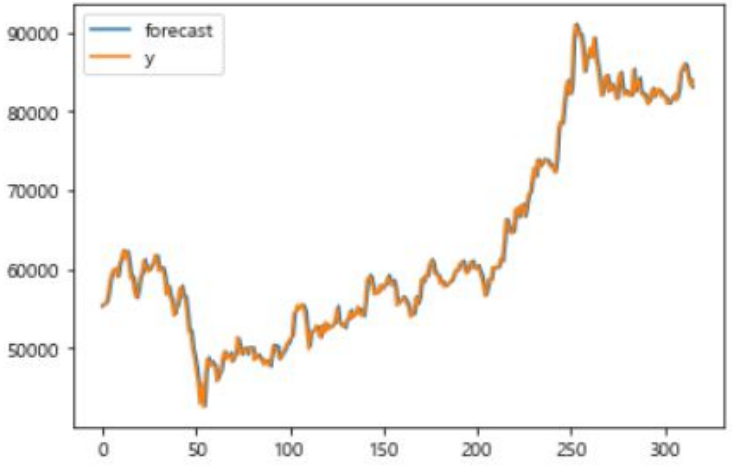
	coef	std err	z	P>|z|	[0.025	0.975]
const	91.4546	68.804	1.329	0.184	-43.398	226.307
ar.L1.D.y	-1.6290	0.017	-94.431	0.000	-1.663	-1.595
ar.L2.D.y	-0.9743	0.016	-60.958	0.000	-1.006	-0.943
ma.L1.D.y	1.6342	0.017	98.172	0.000	1.602	1.667
ma.L2.D.y	1.0000	0.019	51.939	0.000	0.962	1.038

Roots

	Real	Imaginary	Modulus	Frequency
AR.1	-0.8360	-0.5723j	1.0131	-0.4045
AR.2	-0.8360	+0.5723j	1.0131	0.4045
MA.1	-0.8171	-0.5765j	1.0000	-0.4022
MA.2	-0.8171	+0.5765j	1.0000	0.4022

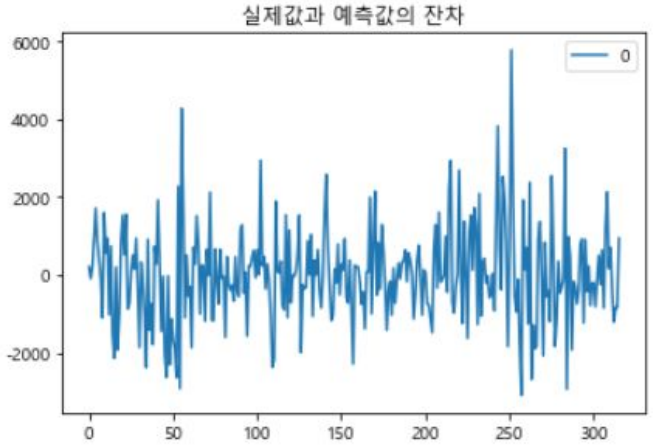
ARIMA 모델 학습결과

In [11]:
```
# 학습 데이터에 대한 예측 결과

fig = model_fit.plot_predict()
```



In [12]:
```
residuals = pd.DataFrame(model_fit.resid)
residuals.plot(title = "실제값과 예측값의 잔차")
```

Out[12]: <AxesSubplot:title={'center':'실제값과 예측값의 잔차'}>



ARIMA 모델 평가하기

- model_fit.forecast(steps = 5)로 향후 5일의 가격을 예측하여 pred_y로 정의

```
In [13]: forecast_data = model_fit.forecast(steps=5)
```

- pred_arima_y 변수에 마지막 5일의 예측 데이터를 리스트로 저장

```
In [14]: # 마지막 5일의 예측 데이터 (2021-04-15 ~ 2021-04-19)
pred_arima_y = forecast_data[0].tolist()
```

- test_y = samsung_test_df.price.values 을 통해 samsung_df의 마지막 5일을 test_y로 정의

input ->

```
# 실제 5일의 데이터 (2021-04-15 ~ 2021-04-19 -> step1의 samsung_test_df 변수 이용
test_y = samsung_test_df.price.values
```

ARIMA 모델 평가하기

- 모델의 예측한 상한값을 pred_y_upper, 하한값을 pred_y_lower로 정의

input ->

```
# 마지막 5일의 예측 데이터 최소값
pred_y_lower = []
# 마지막 5일의 예측 데이터 최대값
pred_y_upper = []
```

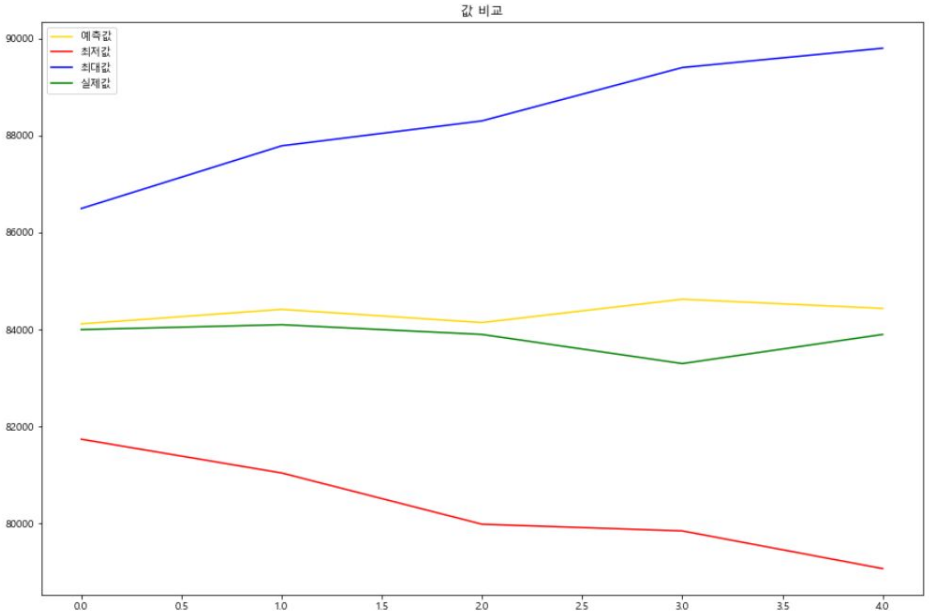
- 정의한 모든 값을 비교하여 5일동안의 상승 경향 예측이 얼마나 맞는지 평가하기

input ->

```
for lower_upper in forecast_data[2]:
    lower = lower_upper[0]
    upper = lower_upper[1]
    pred_y_lower.append(lower)
    pred_y_upper.append(upper)
```

ARIMA 모델 예측데이터 시각화

In [15]:
```
plt.figure(figsize=(15, 10))

# 모델이 예측한 가격 그래프
plt.plot(pred_arima_y, color = 'gold')

# 모델이 예측한 최저 가격 그래프
plt.plot(pred_y_lower, color = 'red')

# 모델이 예측한 최고 가격 그래프
plt.plot(pred_y_upper, color = 'blue')

# 실제 가격 그래프
plt.plot(test_y, color = 'green')

plt.legend(['예측값', '최저값','최대값','실제값'])
plt.title("값 비교")
plt.show()
```



Model 2: Facebook Prophet

Facebook Prophet?

→ ARIMA보다 조금 더 정확한 트렌드 예측 분석을 제공하는 라이브러리이다.

Prophet?

→ Prophet은 Additive 모델이라는 모델링 방법에 기반한 시계열 예측모델로, 시계열 데이터의 트렌드성(연간/월간/일간)을 예측하는 것에 초점이 맞추어져 있다.

Additive?

→ Additive 모델은 선형회귀 분석의 단점을 극복하기 위해 개량된 분석 방법의 하나이다.

In [10]:
```
from fbprophet import Prophet


samsung_df = samsung_df.reset_index()

samsung_df.columns = ['ds', 'y']

samsung_train_df = samsung_df[:482]
samsung_test_df = samsung_df[482:]
```

```
Importing plotly failed. Interactive plots will not work.
```

In [11]:
```
prophet = Prophet(seasonality_mode = 'multiplicative',
                  yearly_seasonality=True,
                  weekly_seasonality=True,
                  daily_seasonality=True,
                  changepoint_prior_scale=0.5)

prophet.fit(samsung_train_df)
```

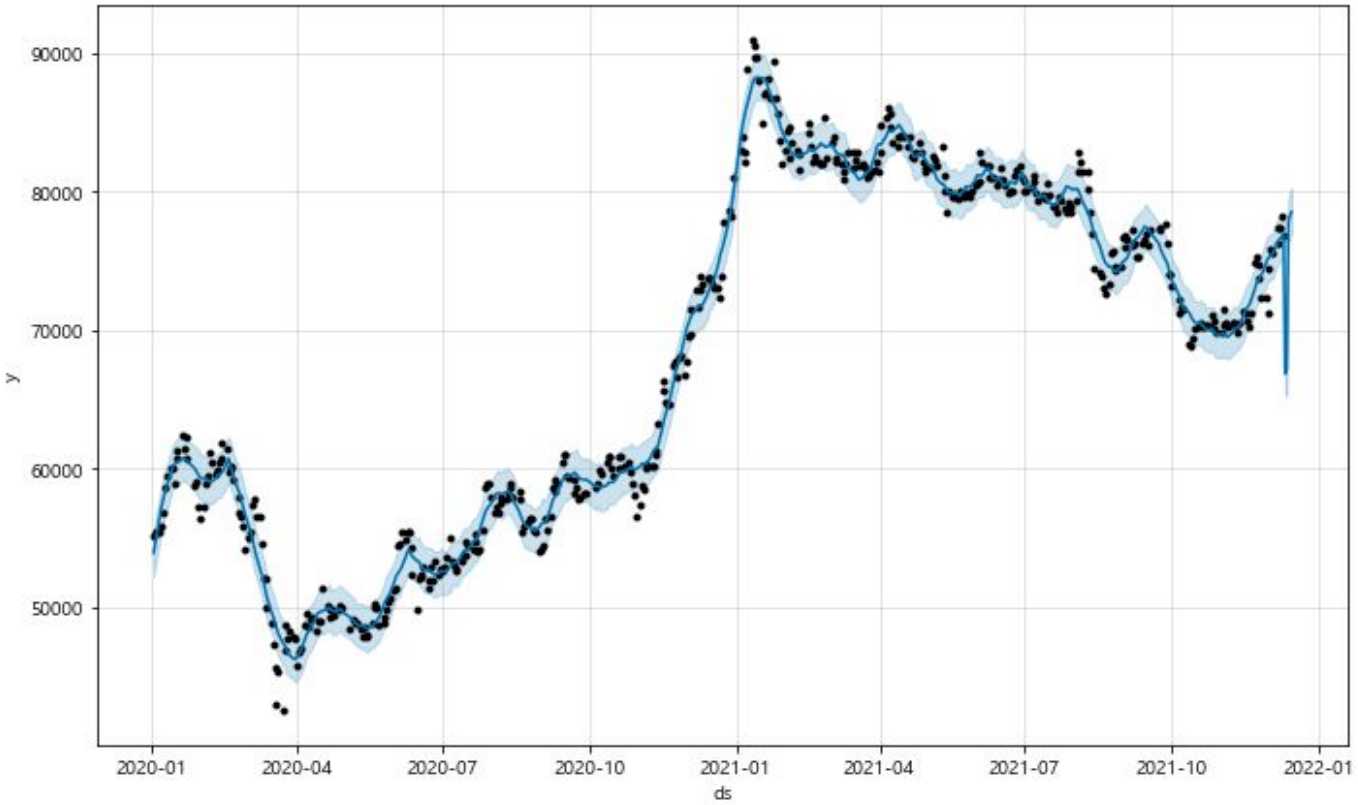
Out[11]:
```
<fbprophet.forecaster.Prophet at 0x205058f8b48>
```

In [12]:
```
future_data = prophet.make_future_dataframe(periods = 5, freq = 'd')
forecast_data = prophet.predict(future_data)
forecast_data[['ds','yhat', 'yhat_lower', 'yhat_upper']].tail(5)
```

Out[12]:

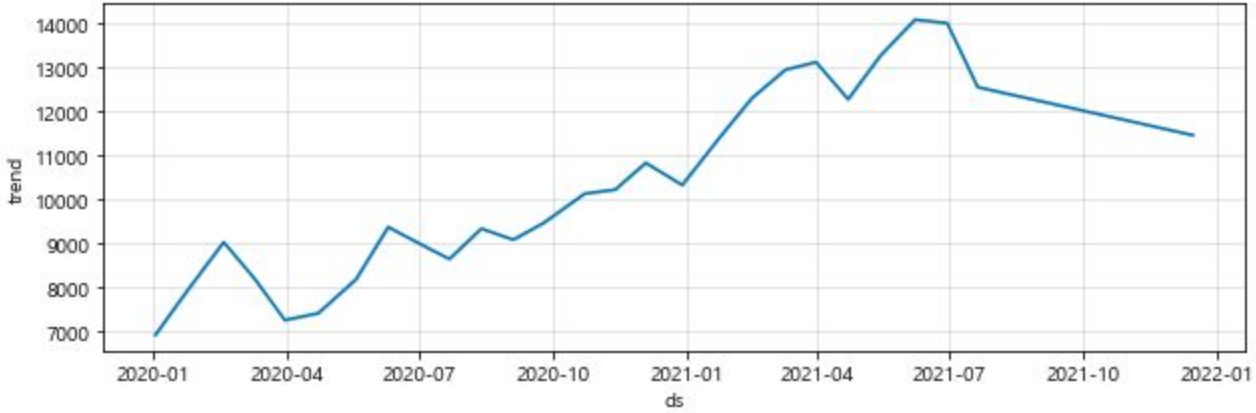
	ds	yhat	yhat_lower	yhat_upper
482	2021-12-11	66872.637714	65260.217789	68460.583096
483	2021-12-12	67145.736514	65343.682565	68786.482062
484	2021-12-13	77820.727788	76183.453668	79536.796971
485	2021-12-14	78290.146062	76699.750081	79994.363475
486	2021-12-15	78528.098654	76977.042727	80270.776830

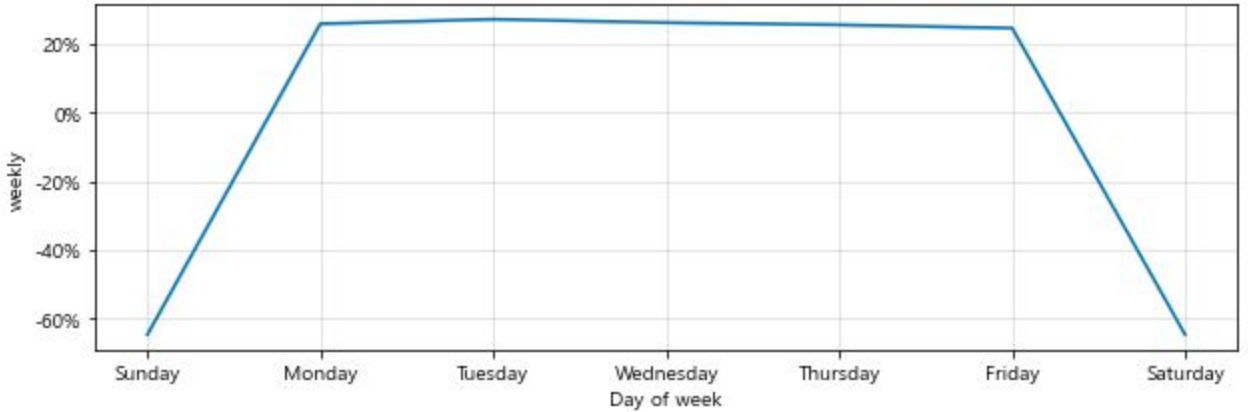
```
In [13]: fig1 = prophet.plot(forecast_data)
```

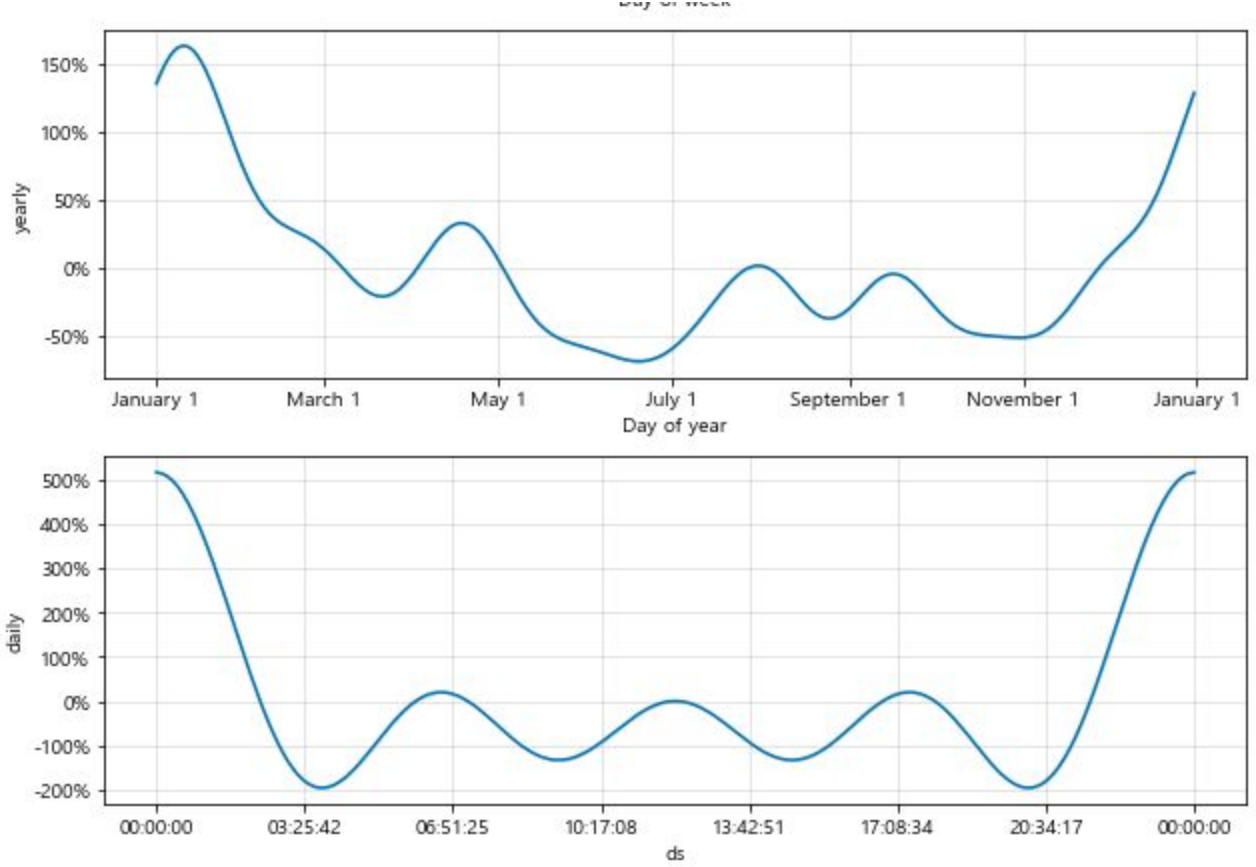


In [14]:
```
fig2 = prophet.plot_components(forecast_data)
```

```
D:₩anaconda3₩envs₩pp37₩lib₩site-packages₩fbprophet₩plot.py:422: UserWarning: FixedFormatter should only be used together with FixedLocator
  ax.set_yticklabels(yticklabels)
D:₩anaconda3₩envs₩pp37₩lib₩site-packages₩fbprophet₩plot.py:422: UserWarning: FixedFormatter should only be used together with FixedLocator
  ax.set_yticklabels(yticklabels)
D:₩anaconda3₩envs₩pp37₩lib₩site-packages₩fbprophet₩plot.py:422: UserWarning: FixedFormatter should only be used together with FixedLocator
  ax.set_yticklabels(yticklabels)
```


14000
13000
12000
11000
10000
9000
8000
7000
trend
2020-01
2020-04
2020-07
2020-10
2021-01
2021-04
2021-07
2021-10
2022-01
ds

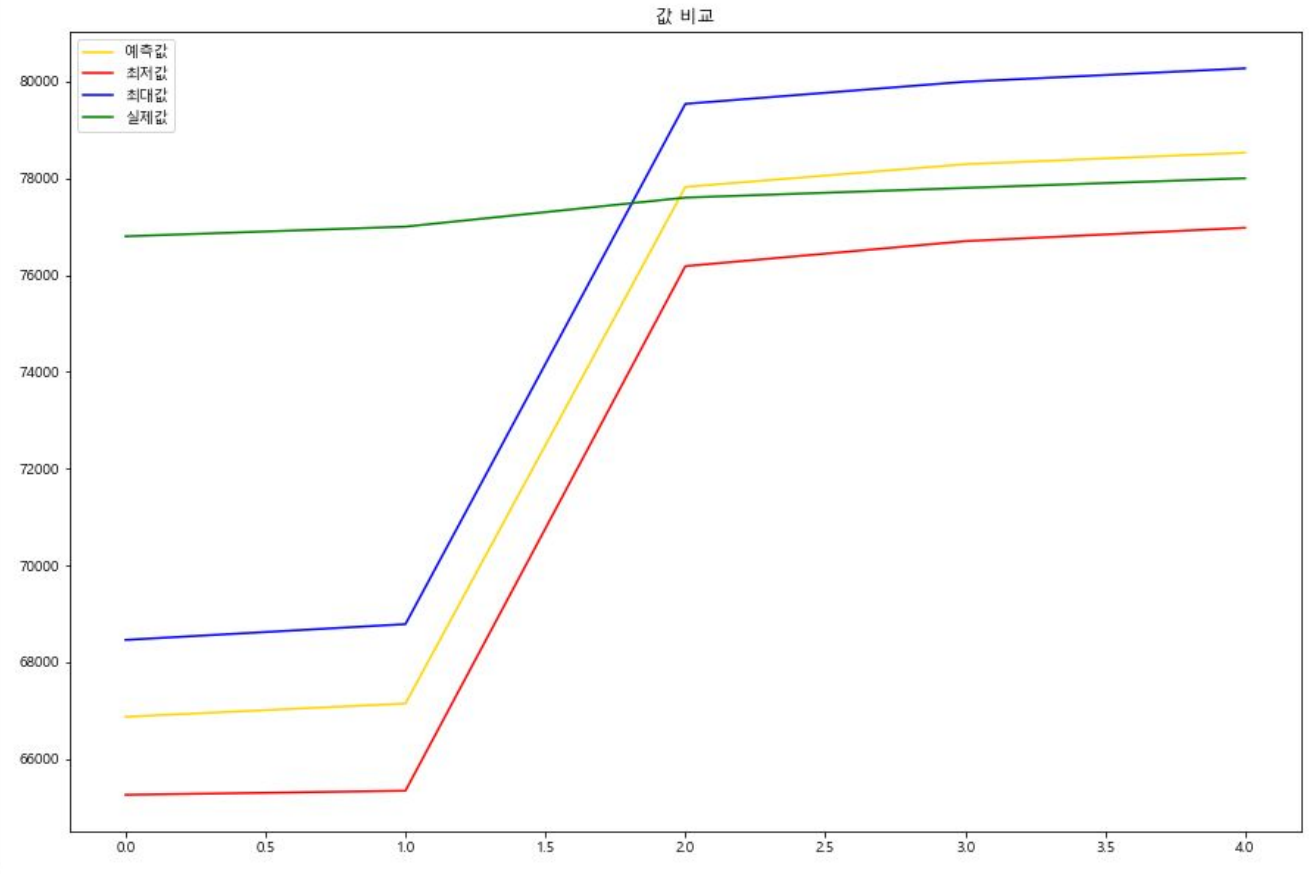

20%
0%
-20%
-40%
-60%
weekly
Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Day of week


150%
100%
50%
0%
-50%
yearly
January 1
March 1
May 1
July 1
September 1
November 1
January 1
Day of year
500%
400%
300%
200%
100%
0%
-100%
-200%
daily
00:00:00
03:25:42
06:51:25
10:17:08
13:42:51
17:08:34
20:34:17
00:00:00
ds

Testset 평가

In [15]:
```
plt.figure(figsize=(15, 10))

# 마지막 5일의 예측 데이터 (2021-04-15 ~ 2021-04-19)
pred_fbprophet_y = forecast_data.yhat.values[-5:]

# 실제 5일의 데이터 (2021-04-15 ~ 2021-04-19)
test_y = samsung_test_df.y.values

# 마지막 5일의 예측 데이터 최소값
pred_y_lower = forecast_data.yhat_lower.values[-5:]
# 마지막 5일의 예측 데이터 최대값
pred_y_upper = forecast_data.yhat_upper.values[-5:]

# 모델이 예측한 가격 그래프
plt.plot(pred_fbprophet_y, color = 'gold')

# 모델이 예측한 최저 가격 그래프
plt.plot(pred_y_lower, color = 'red')

# 모델이 예측한 최고 가격 그래프
plt.plot(pred_y_upper, color = 'blue')

# 실제 가격 그래프
plt.plot(test_y, color = 'green')

plt.legend(['예측값', '최저값','최대값','실제값'])
plt.title("값 비교")
```

Out [15]: Text(0.5, 1.0, '값 비교')

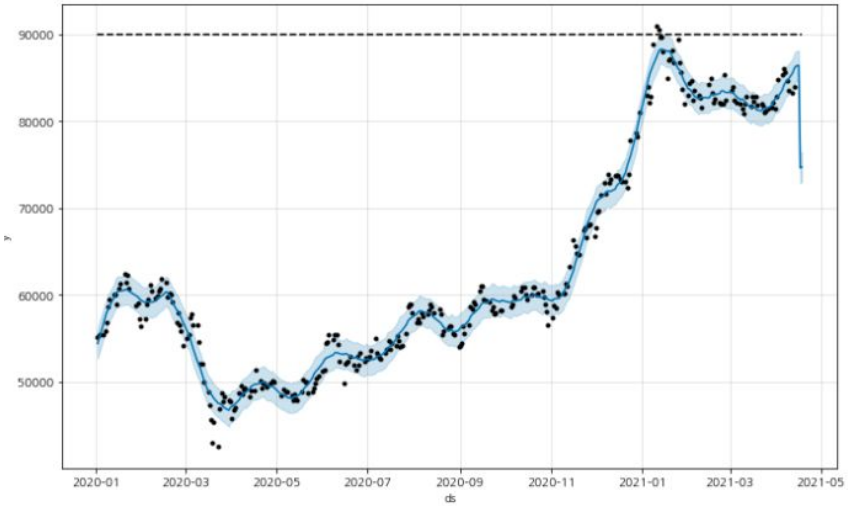


활용: 더 나은 결과를 위한 방법

1. 상한값 or 하한값 지정

→ Prophet 모델에서 future_data['cap'] = 96000 통해 데이터넷에 상한선을 설정할 수 있음.

```
# 상한가 설정
samsung_train_df['cap'] = 90000

# 상한가 적용을 위한 파라미터를 다음과 같이 설정
prophet = Prophet(seasonality_mode = 'multiplicative',
                  growth = 'logistic',
                  yearly_seasonality = True,
                  weekly_seasonality = True,
                  daily_seasonality = True,
                  changepoint_prior_scale = 0.5)

prophet.fit(samsung_train_df)

# 5일 예측
future_data = prophet.make_future_dataframe(periods = 5, freq = 'd')

# 상한가 설정
future_data['cap'] = 90000
forecast_data = prophet.predict(future_data)
fig = prophet.plot(forecast_data)
```

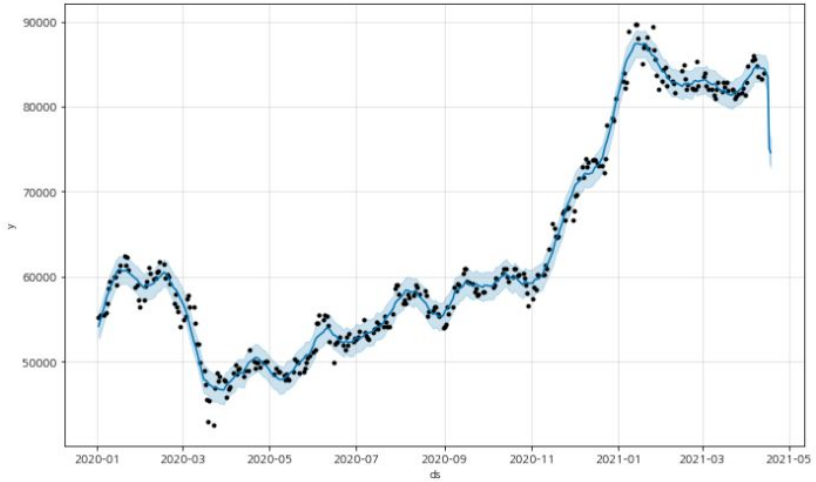


상한선을 적용할 학습 결과를 시각화 한 것

2. 이상치 제거

→ 이상치란 평균적인 수치에 비해 지나치게 높거나 낮은 수치의 데이터를 의미.

```
samsung_train_df.loc[samsung_train_df['y'] > 90000, 'y'] = None

# prophet 모델 학습
prophet = Prophet(seasonality_mode = 'multiplicative',
                  yearly_seasonality = True,
                  weekly_seasonality = True,
                  daily_seasonality = True,
                  changepoint_prior_scale = 0.5)

prophet.fit(samsung_train_df)

# 5일 예측
future_data = prophet.make_future_dataframe(periods = 5, freq = 'd')

forecast_data = prophet.predict(future_data)
fig = prophet.plot(forecast_data)
```




```
pred_fbprophet_y_1 = forecast_data.yhat.values[-5:]
```

→ fbprophet 모델이 이상치를 제거한 데이터로 학습하려면 이상치에 해당하는 데이터를 None로 설정

MODEL 평가

RMSE

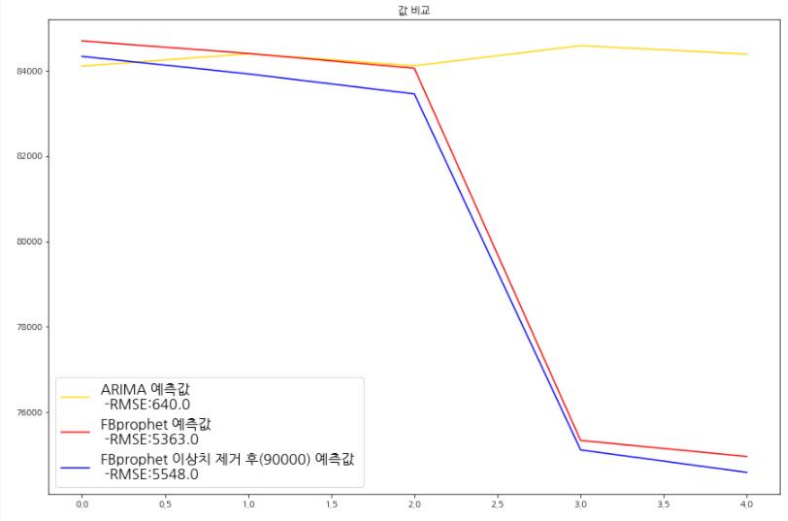
→ 평균 제곱근 오차(Root Mean Square Error; RMSE)를 의미

```
df = pd.DataFrame({'ARIMA 예측값':pred_arima_y,
                   'FBprophet 예측값':  pred_fbprophet_y,
                   'FBprophet 이상치 제거 후(90000) 예측값':pred_fbprophet_y_1,
                   '실제값':test_y})
```

	ARIMA 예측값	FBprophet 예측값	FBprophet 이상치 제거 후(90000) 예측값	실제값
0	84108.703039	84697.939052	84337.001557	84000.0
1	84397.233627	84406.375516	83926.677078	84100.0
2	84117.027275	84057.882277	83457.567176	83900.0
3	84588.083796	75340.001930	75118.134155	83300.0
4	84389.466677	74964.027647	74591.859907	83900.0

모델별 예측값 시각화

```
from sklearn.metrics import mean_squared_error, r2_score
from math import sqrt

plt.figure(figsize=(15, 10))

# arima 모델의  rmse
rmse_arima = sqrt(mean_squared_error(pred_arima_y, test_y))

# fbprophet 모델의  rmse
rmse_fbprophet = sqrt(mean_squared_error(pred_fbprophet_y, test_y))

# 전처리 진행한 fbprophet 모델의  rmse
rmse_fbprophet_1 = sqrt(mean_squared_error(pred_fbprophet_y_1, test_y))

# 모델이 예측한 가격 그래프
plt.plot(df[['ARIMA 예측값']], color = 'gold')

# 모델이 예측한 최저 가격 그래프
plt.plot(df[['FBprophet 예측값']], color = 'red')

# 모델이 예측한 최고 가격 그래프
plt.plot(df[['FBprophet 이상치 제거 후(90000) 예측값']], color = 'blue')

# 실제 가격 그래프
# plt.plot(test_y, color = 'green')

plt.rc('legend', fontsize=16)
plt.legend(['ARIMA 예측값 \n -RMSE:' + str(round(rmse_arima,0)),
            'FBprophet 예측값  \n -RMSE:' + str(round(rmse_fbprophet,0)),
            'FBprophet 이상치 제거 후(90000) 예측값 \n -RMSE:' + str(round(rmse_fbprophet_1,0))])
plt.title("값 비교")
```



ARIMA모델이 가장 좋은 효과를 보이는걸로 나타남

참고

[Python] 삼성전자 주가 예측 입니다 - DACON

https://minjejeon.github.io/learningstock/2016/07/12/getting-data-from-yahoo-finance.html