금융데이터 불러오기

In [ ]:

```
import matplotlib as mpl
import matplotlib.pyplot as plt
%config InlineBackend.figure_format = 'retina'
!apt -qq -y install fonts-nanum
import matplotlib.font_manager as fm
fontpath = '/usr/share/fonts/truetype/nanum/NanumBarunGothic.ttf'
font = fm.FontProperties(fname=fontpath, size=9)
plt.rc('font', family='NanumBarunGothic')
mpl.font_manager._rebuild()
```

In [2]:

```
!pip install -U finance-datareader
import FinanceDataReader as fdr
df_krx=fdr.StockListing('KRX')
df_krx.info()
```

```
Collecting finance-datareader
  Downloading https://files.pythonhosted.org/packages/83/5e/54306e72b5ff5d5ec6cc9f
32cdf19602237f9bb70d64efcd527338388be3/finance_datareader-0.9.31-py3-none-any.whl
 (https://files.pythonhosted.org/packages/83/5e/54306e72b5ff5d5ec6cc9f32cdf1960223
7f9bb70d64efcd527338388be3/finance_datareader-0.9.31-py3-none-any.whl)
Collecting requests-file
  Downloading https://files.pythonhosted.org/packages/77/86/cdb5e8eaed90796aa83a6d
9f75cfbd37af553c47a291cd47bc410ef9bdb2/requests_file-1.5.1-py2.py3-none-any.whl (h
ttps://files.pythonhosted.org/packages/77/86/cdb5e8eaed90796aa83a6d9f75cfbd37af553
c47a291cd47bc410ef9bdb2/requests_file-1.5.1-py2.py3-none-any.whl)
Requirement already satisfied, skipping upgrade: requests>=2.3.0 in /usr/local/li
b/python3.7/dist-packages (from finance-datareader) (2.23.0)
Requirement already satisfied, skipping upgrade: lxml in /usr/local/lib/python3.7/
dist-packages (from finance-datareader) (4.2.6)
Requirement already satisfied, skipping upgrade: tqdm in /usr/local/lib/python3.7/
dist-packages (from finance-datareader) (4.41.1)
Requirement already satisfied, skipping upgrade: pandas>=0.19.2 in /usr/local/lib/
python3.7/dist-packages (from finance-datareader) (1.1.5)
Requirement already satisfied, skipping upgrade: six in /usr/local/lib/python3.7/d
ist-packages (from requests-file->finance-datareader) (1.15.0)
```

데이터에서 코스피, 코스닥 상장 주식 가져오기

In [3]:

```
df_krx=df_krx[df_krx['ListingDate'].notna()] #ListingDate
df_krx=df_krx[df_krx['Market']!='KONEX'] #비상장주식(Konex)제외
df_krx.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2314 entries, 0 to 7125
Data columns (total 10 columns):
 #   Column          Non-Null Count  Dtype         
---  ------          --------------  -----         
 0   Symbol          2314 non-null   object        
 1   Market          2314 non-null   object        
 2   Name            2314 non-null   object        
 3   Sector          2314 non-null   object        
 4   Industry        2297 non-null   object        
 5   ListingDate     2314 non-null   datetime64[ns]
 6   SettleMonth     2314 non-null   object        
 7   Representative  2314 non-null   object        
 8   HomePage        2134 non-null   object        
 9   Region          2314 non-null   object        
dtypes: datetime64[ns](1), object(9)
memory usage: 198.9+ KB
```

In [4]:

```
#주가_종가, 거래량 가져오기 df_stock (2021년 상반기 데이터)
import pandas as pd
start = '2021-01-01'
end = '2021-06-30'
df_stock=pd.DataFrame(fdr.DataReader('KS11', start, end)[['Close','Volume']])
df_stock.columns=['KOSPI_Close','KOSPI_Volume']
tmp=pd.DataFrame(fdr.DataReader('KQ11', start, end)[['Close','Volume']])
tmp.columns=['KOSDAK_Close','KOSDAK_Volume']
df_stock=pd.concat([df_stock,tmp], axis=1)
for i in df_krx['Symbol']:
  tmp = pd.DataFrame(fdr.DataReader(i,start,end)[['Close','Volume']])
  tmp.columns=[str(df_krx[df_krx['Symbol']==i].iloc[0,2])+str('_Close'),str(df_krx[df_krx['Symb
  df_stock = pd.concat([df_stock,tmp],axis=1)
```

In [7]:

```
#종가 데이터만 가져오기
df_stock_close=df_stock.filter(regex='Close')
df_stock_close.head(6)
```

Out[7]:

Date	KOSPI_Close	KOSDAK_Close	3S_Close	AJ네트웍스_Close	AK홀딩스_Close	APS홀딩스_Close	AP시스템_Close	AP위성_Close	BGF_
2021-01-04	2944.45	977.62	2260	4580	25250	8000	25500	8330	
2021-01-05	2990.57	985.76	2250	4935	25050	7900	25150	8160	
2021-01-06	2968.21	981.39	2290	4710	24900	7670	25500	8190	
2021-01-07	3031.68	988.86	2290	4695	25200	7650	26100	8350	
2021-01-08	3152.18	987.79	2245	4540	25350	7500	26000	8100	
2021-01-11	3148.45	976.63	2175	4360	24800	7200	25000	7900	

6 rows × 2316 columns

월수익 가장 큰 5개기업 가져오기

In [9]:

```
df_m = df_stock_close.resample('M').last().pct_change()#Y, M, 2W, W, D
df_m
```

Out[9]:

Date	KOSPI_Close	KOSDAK_Close	3S_Close	AJ네트웍스_Close	AK홀딩스_Close	APS홀딩스_Close	AP시스템_Close	AI_C
2021-01-31	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
2021-02-28	0.012345	-0.015925	0.074661	-0.031172	0.079447	0.070735	0.158111	0.75
2021-03-31	0.016087	0.046207	0.008421	0.159588	-0.145600	0.256477	0.015957	-0.15
2021-04-30	0.028235	0.028530	0.183716	0.234184	0.164794	0.242268	-0.012216	-0.1
2021-05-31	0.017809	-0.001698	0.019400	0.064748	0.138264	0.211618	0.028269	0.09
2021-06-30	0.028952	0.049074	0.060554	0.027027	-0.079096	0.082192	0.091065	0.19

6 rows × 2316 columns

In [10]:

```
import numpy as np
n=5
best_stock=df_m.mean().sort_values(ascending=False)[0:n].index.to_list()
df=df_stock_close[best_stock]
df
```

Out[10]:

Date	쎄미시스코_Close	NE능률_Close	덕성_Close	이스타코_Close	휴마시스_Close
2021-01-04	7680	2780	8100	678	9050
2021-01-05	7580	2840	7940	685	9100
2021-01-06	7570	2830	8350	685	9030
2021-01-07	7550	2830	8630	701	9030
2021-01-08	7660	2790	8360	699	8820
...	...	...	...	...	...
2021-06-24	38550	26500	27200	4600	18700
2021-06-25	41200	26150	26800	5130	19100
2021-06-28	41500	26700	26900	5990	18400
2021-06-29	41650	23500	24300	7200	19300
2021-06-30	40700	22400	22950	6650	18900

123 rows × 5 columns

포트폴리오 구성

- 위의 최대수익 5개 기업으로 포트폴리오 구성

In [11]:

```
#title 주식별 (월)수익, 위험, 공분산 구하기
# Yearly returns for individual companies
ind_er=df.resample('m').last().pct_change().mean()
# Volatility
ann_sd=df.pct_change().apply(lambda x: np.log(1+x)).std().apply(lambda x: x*np.sqrt(df.shape[0]
# Log of percentage change
cov_matrix = df.pct_change().apply(lambda x: np.log(1+x)).cov()
```

In [12]:

```
assets = pd.concat([ind_er, ann_sd], axis=1) # Creating a table for visualising returns and vol
assets.columns = ['Returns', 'Volatility']
assets
```

Out[12]:

	Returns	Volatility
쎄미시스코_Close	0.748925	0.762761
NE능률_Close	0.722106	0.840783
덕성_Close	0.567284	0.766418
이스타코_Close	0.565495	0.865522
휴마시스_Close	0.515763	0.752524

포트폴리오 수익,위험,가중치 데이터

In [13]:

```
p_ret = [] # Define an empty array for portfolio returns
p_vol = [] # Define an empty array for portfolio volatility
p_weights = [] # Define an empty array for asset weights

num_assets = len(df.columns)
num_portfolios = 10000
```

In [14]:

```
for portfolio in range(num_portfolios):
    weights = np.random.random(num_assets)
    weights = weights/np.sum(weights)
    p_weights.append(weights)
    returns = np.dot(weights, ind_er) # Returns are the product of individual expected returns
                                      # weights
    p_ret.append(returns)
    var = cov_matrix.mul(weights, axis=0).mul(weights, axis=1).sum().sum()# Portfolio Variance
    sd = np.sqrt(var) # Daily standard deviation
    ann_sd = sd*np.sqrt(250) # Annual standard deviation = volatility
    p_vol.append(ann_sd)
```

In [15]:

```
data = {'Returns':p_ret, 'Volatility':p_vol}

for counter, symbol in enumerate(df.columns.tolist()):
    #print(counter, symbol)
    data[symbol+' weight'] = [w[counter] for w in p_weights]
```

In [16]:

```
portfolios  = pd.DataFrame(data)
portfolios.head() # Dataframe of the 10000 portfolios created
```

Out[16]:

	Returns	Volatility	쎄미시스코_Close weight	NE능률_Close weight	덕성_Close weight	이스타코_Close weight	휴마시스_Close weight
0	0.615378	0.638587	0.126107	0.247531	0.250069	0.125713	0.250580
1	0.601816	0.623814	0.091469	0.201098	0.181795	0.278776	0.246862
2	0.615788	0.627985	0.316066	0.043742	0.306105	0.030833	0.303253
3	0.630856	0.600670	0.220392	0.223837	0.199616	0.145468	0.210687
4	0.607552	0.617079	0.264344	0.082180	0.199892	0.058270	0.395315

In [17]:

```
#Minimum volatility (left most point)
min_vol_port = portfolios.iloc[portfolios['Volatility'].idxmin()]
# idxmin() gives us the minimum value in the column specified.
min_vol_port
```

Out[17]:

```
Returns                 0.618336
Volatility              0.571730
쎄미시스코_Close weight     0.275204
NE능률_Close weight      0.088683
덕성_Close weight        0.175911
이스타코_Close weight      0.222075
휴마시스_Close weight      0.238128
Name: 1379, dtype: float64
```

In [18]:

```
# Finding the optimal portfolio
#Optimal Risky Portfolio
#An optimal risky portfolio can be considered as one that has highest Sharpe ratio.
rf = 0.01 # risk factor
optimal_risky_port = portfolios.iloc[((portfolios['Returns']-rf)/portfolios['Volatility']).idxm
optimal_risky_port
```

Out[18]:

```
Returns                 0.643012
Volatility              0.583791
쎄미시스코_Close weight     0.326937
NE능률_Close weight      0.176731
덕성_Close weight        0.100403
이스타코_Close weight      0.188612
휴마시스_Close weight      0.207317
Name: 9935, dtype: float64
```

In [19]:

```
import warnings
warnings.filterwarnings("ignore")
```

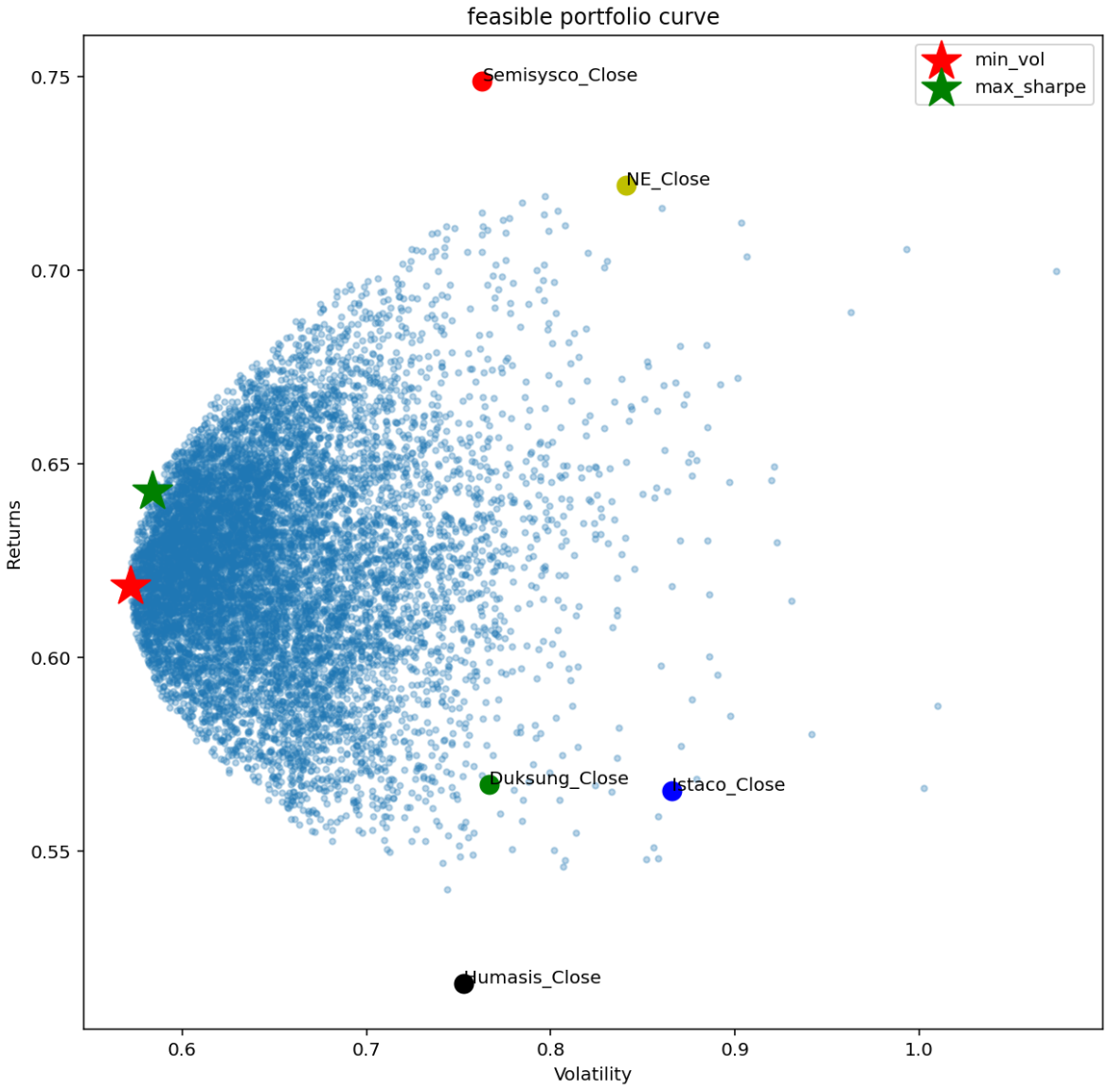
포트폴리오 투자선 및 최적포트폴리오 위치

In [23]:

```
# Plotting optimal portfolio
plt.subplots(figsize=(10, 10))
plt.scatter(portfolios['Volatility'], portfolios['Returns'],marker='o', s=10, alpha=0.3)
a = plt.scatter(min_vol_port[1], min_vol_port[0], color='r', marker='*', s=500 )
b = plt.scatter(optimal_risky_port[1], optimal_risky_port[0], color='g', marker='*', s=500)
plt.scatter(assets.iloc[0,1],assets.iloc[0,0],color='r',s=100) # 쎄미시스코_Close
plt.scatter(assets.iloc[1,1],assets.iloc[1,0],color='y',s=100) # NE능률_Close
plt.scatter(assets.iloc[2,1],assets.iloc[2,0],color='g',s=100) # 덕성_Close
plt.scatter(assets.iloc[3,1],assets.iloc[3,0],color='b',s=100) # 이스타코_Close
plt.scatter(assets.iloc[4,1],assets.iloc[4,0],color='k',s=100) # 휴마시스_Close
plt.title("feasible portfolio curve")
plt.xlabel("Volatility")
plt.ylabel("Returns")
plt.legend((a,b),('min_vol','max_sharpe'))
plt.text(assets.iloc[0,1],assets.iloc[0,0],'Semisysco_Close')
plt.text(assets.iloc[1,1],assets.iloc[1,0],'NE_Close')
plt.text(assets.iloc[2,1],assets.iloc[2,0],'Duksung_Close')
plt.text(assets.iloc[3,1],assets.iloc[3,0],'Istaco_Close')
plt.text(assets.iloc[4,1],assets.iloc[4,0],'Humasis_Close')
```

Out[23]:

```
Text(0.7525243396784821, 0.5157633341413592, 'Humasis_Close')
```



수익률 높은 주식과 상관관계 높은 기업

In [24]:

```
df_corr=df_stock_close.corr()
target_feature=df.columns[0] #수익 최대 기업 - 쎄미시스코
cor_target=abs(df_corr[target_feature]) #Selecting highly correlated features
df_corr[target_feature][cor_target[cor_target>0.7].index.values.tolist()]
```

Out[24]:

```
CJ ENM_Close       0.759498
CJ프레시웨이_Close     0.797159
DB_Close           0.766417
DRB동일_Close        0.851986
E1_Close           0.755617
                     ...
화승코퍼레이션_Close     0.784216
화신_Close           0.849889
화신정공_Close         0.723099
휴먼엔_Close         -0.707547
휴젤_Close           0.830951
Name: 쎄미시스코_Close, Length: 304, dtype: float64
```

시간도표,거래량,이동평균선

- 거래량과 주가 상승/하락 관계
- 주가가 바닥에 있을 때 거래량 상승 / "주가가 천장일때 거래량이 급증한다-주가하락가능성 농후"

    1) 주가 하락 + 거래량 증가 => 위험
    2) 주가 하락 + 거래량 감소 => 판단, 악재가 없다면 사자
    3) 주가 상승 + 거래량 증가 => 사는 시점
    4) 주가 상승 + 거래량 감소 => 위험

In [25]:

```
!pip install --upgrade mplfinance
```

```
Collecting mplfinance
  Downloading https://files.pythonhosted.org/packages/3f/4e/a3de915df8d112579beb3df2
2e47b3929670b10e9c5dfd4a2b31bf1a34a7/mplfinance-0.12.7a17-py3-none-any.whl (https://
files.pythonhosted.org/packages/3f/4e/a3de915df8d112579beb3df22e47b3929670b10e9c5dfd
4a2b31bf1a34a7/mplfinance-0.12.7a17-py3-none-any.whl) (62kB)
    |████████████████████████████████| 71kB 3.8MB/s eta 0:00:011
Requirement already satisfied, skipping upgrade: matplotlib in /usr/local/lib/python
3.7/dist-packages (from mplfinance) (3.2.2)
Requirement already satisfied, skipping upgrade: pandas in /usr/local/lib/python3.7/
dist-packages (from mplfinance) (1.1.5)
Requirement already satisfied, skipping upgrade: kiwisolver>=1.0.1 in /usr/local/li
b/python3.7/dist-packages (from matplotlib->mplfinance) (1.3.1)
Requirement already satisfied, skipping upgrade: python-dateutil>=2.1 in /usr/local/
lib/python3.7/dist-packages (from matplotlib->mplfinance) (2.8.1)
Requirement already satisfied, skipping upgrade: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=
2.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib->mplfinance) (2.4.
7)
Requirement already satisfied, skipping upgrade: numpy>=1.11 in /usr/local/lib/pytho
n3.7/dist-packages (from matplotlib->mplfinance) (1.19.5)
Requirement already satisfied, skipping upgrade: cycler>=0.10 in /usr/local/lib/pyth
on3.7/dist-packages (from matplotlib->mplfinance) (0.10.0)
Requirement already satisfied, skipping upgrade: pytz>=2017.2 in /usr/local/lib/pyth
on3.7/dist-packages (from pandas->mplfinance) (2018.9)
Requirement already satisfied, skipping upgrade: six>=1.5 in /usr/local/lib/python3.
7/dist-packages (from python-dateutil>=2.1->matplotlib->mplfinance) (1.15.0)
Installing collected packages: mplfinance
Successfully installed mplfinance-0.12.7a17
```

In [26]:

```
df.columns[0] #최대 수익 사업체명
```

Out[26]:

```
'쎄미시스코_Close'
```

In [45]:

```
#주식종목 코드 찾기
name=input(" 주식 코드 기업명" ) #find 종목코드 및 정보
df_krx[df_krx['Name'].str.contains(name)]
```

주식 코드 기업명쎄미시스코

Out[45]:

	Symbol	Market	Name	Sector	Industry	ListingDate	SettleMonth	Representative	
4038	136510	KOSDAQ	쎄미시스코	특수목적용 기계 제조업	EGIS(유리기판 검사장비), Smart-EPD 및 Smart-HMS(플라즈마 검사장비)	2011-11-18	12월	이순종	http

In [46]:

```
#주가 데이터 가져오기 함수
df_top=fdr.DataReader('136510','2021-01-01','2021-06-30')  #최대 수익 주식 데이터 가져오기
```

골든크로스/데드크로스

In [48]:

```
df=df_top['Close']
import pandas as pd
df_ma_s=df.rolling(window=50).mean()
df_ma_L=df.rolling(window=200).mean()
df_all=pd.concat([df_ma_s, df_ma_L], axis=1)
df_all.columns=['ma50','ma200']
df_all['ma50_diff']=df_all['ma50'].diff()
df_all['ma200_diff']=df_all['ma200'].diff()
df_all.dropna(inplace=True)
#Golden Cross
for k in range(0,df_all.shape[0]):
  if (df_all.ma50_diff[k]>0) & (df_all.ma50_diff[k-1]>0) & (df_all.ma50_diff[k]>df_all.ma50_dif
    if (df_all.ma50[k-1]<df_all.ma200[k-1]) & (df_all.ma50[k]>df_all.ma200[k]):
      print('Golden Cross happens on',df_all.index[k])
#Dead Cross
for k in range(0,df_all.shape[0]):
  if (df_all.ma50_diff[k]<0) & (df_all.ma50_diff[k-1]<0) & (df_all.ma50_diff[k]<df_all.ma50_dif
    if (df_all.ma50[k-1]>df_all.ma200[k-1]) & (df_all.ma50[k]<df_all.ma200[k]):
      print('Dead Cross happens on',df_all.index[k])
```

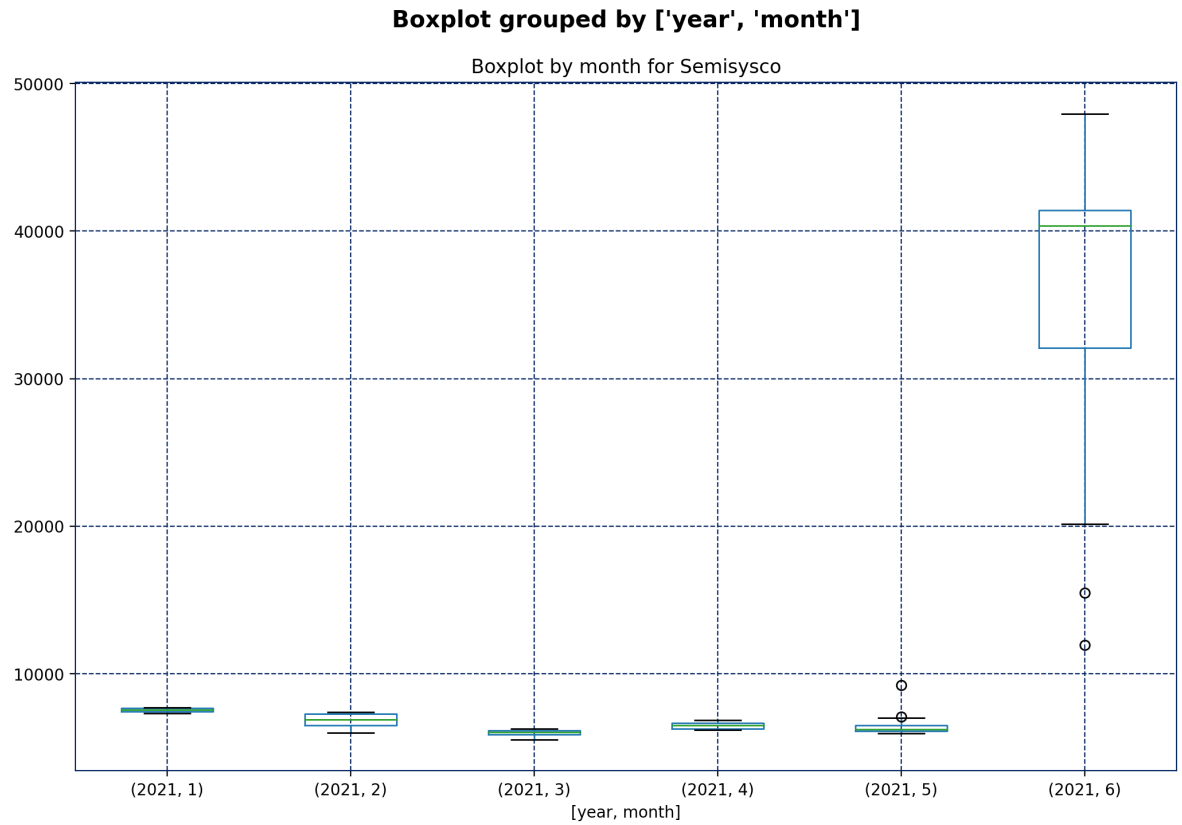
월별 주가 나무상자그림

In [49]:

```
df=df_top
df['year']=df.index.year.astype(str)
df['month']=df.index.month
df['weekday']=df.index.dayofweek
```

In [50]:

```
df.boxplot(column=['Close'], by=['year','month'],figsize=(12,8))
plt.title('Boxplot by month for Semisysco')
plt.show()
```



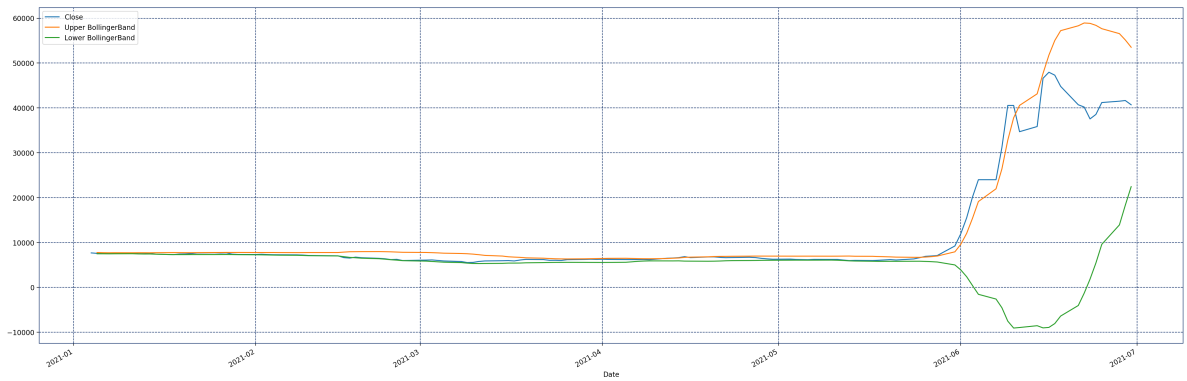
볼린저밴드

In [51]:

```
w_size = 20
k = 2
df['Moving Average']      = df['Close'].rolling(window=w_size, min_periods=1).mean()
df['Standard Deviation']  = df['Close'].rolling(window=w_size, min_periods=1).std()
df['Upper BollingerBand'] = df['Moving Average'] + (df['Standard Deviation'] * k)
df['Lower BollingerBand'] = df['Moving Average'] - (df['Standard Deviation'] * k)
df[['Close', 'Upper BollingerBand', 'Lower BollingerBand']].plot(figsize=(30,10))
```

Out[51]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbf540be150>
```



AUTO ARMA 추정

In [52]:

```
1 !pip install pmdarima
```

```
Collecting pmdarima
  Downloading https://files.pythonhosted.org/packages/f4/c2/2a94bce6bba1deac3c0b16db
b14f28de0b9035e3211919cae8448455aa65/pmdarima-1.8.2-cp37-cp37m-manylinux1_x86_64.whl
(https://files.pythonhosted.org/packages/f4/c2/2a94bce6bba1deac3c0b16dbb14f28de0b903
5e3211919cae8448455aa65/pmdarima-1.8.2-cp37-cp37m-manylinux1_x86_64.whl) (1.5MB)
     |████████████████████████████████| 1.5MB 5.2MB/s
Requirement already satisfied: pandas>=0.19 in /usr/local/lib/python3.7/dist-package
s (from pmdarima) (1.1.5)
Requirement already satisfied: scikit-learn>=0.22 in /usr/local/lib/python3.7/dist-p
ackages (from pmdarima) (0.22.2.post1)
Requirement already satisfied: numpy~=1.19.0 in /usr/local/lib/python3.7/dist-packag
es (from pmdarima) (1.19.5)
Requirement already satisfied: Cython!=0.29.18,>=0.29 in /usr/local/lib/python3.7/di
st-packages (from pmdarima) (0.29.23)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.7/dist-package
s (from pmdarima) (1.4.1)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-package
s (from pmdarima) (1.0.1)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.7/dist-packages (fr
om pmdarima) (1.24.3)
Requirement already satisfied: setuptools!=50.0.0,>=38.6.0 in /usr/local/lib/python
3.7/dist-packages (from pmdarima) (57.0.0)
Collecting statsmodels!=0.12.0,>=0.11
  Downloading https://files.pythonhosted.org/packages/da/69/8eef30a6237c54f3c0b52414
0e2975f4b1eea3489b45eb3339574fc8acee/statsmodels-0.12.2-cp37-cp37m-manylinux1_x86_6
4.whl (https://files.pythonhosted.org/packages/da/69/8eef30a6237c54f3c0b524140e2975f
4b1eea3489b45eb3339574fc8acee/statsmodels-0.12.2-cp37-cp37m-manylinux1_x86_64.whl)
 (9.5MB)
     |████████████████████████████████| 9.5MB 21.9MB/s
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/di
st-packages (from pandas>=0.19->pmdarima) (2.8.1)
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-package
s (from pandas>=0.19->pmdarima) (2018.9)
Requirement already satisfied: patsy>=0.5 in /usr/local/lib/python3.7/dist-packages
 (from statsmodels!=0.12.0,>=0.11->pmdarima) (0.5.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (f
rom python-dateutil>=2.7.3->pandas>=0.19->pmdarima) (1.15.0)
Installing collected packages: statsmodels, pmdarima
  Found existing installation: statsmodels 0.10.2
    Uninstalling statsmodels-0.10.2:
      Successfully uninstalled statsmodels-0.10.2
Successfully installed pmdarima-1.8.2 statsmodels-0.12.2
```

In [53]:

```
df=df_top["Close"]
import pmdarima as pm
fit_auto=pm.auto_arima(df)
print(fit_auto.summary())
```

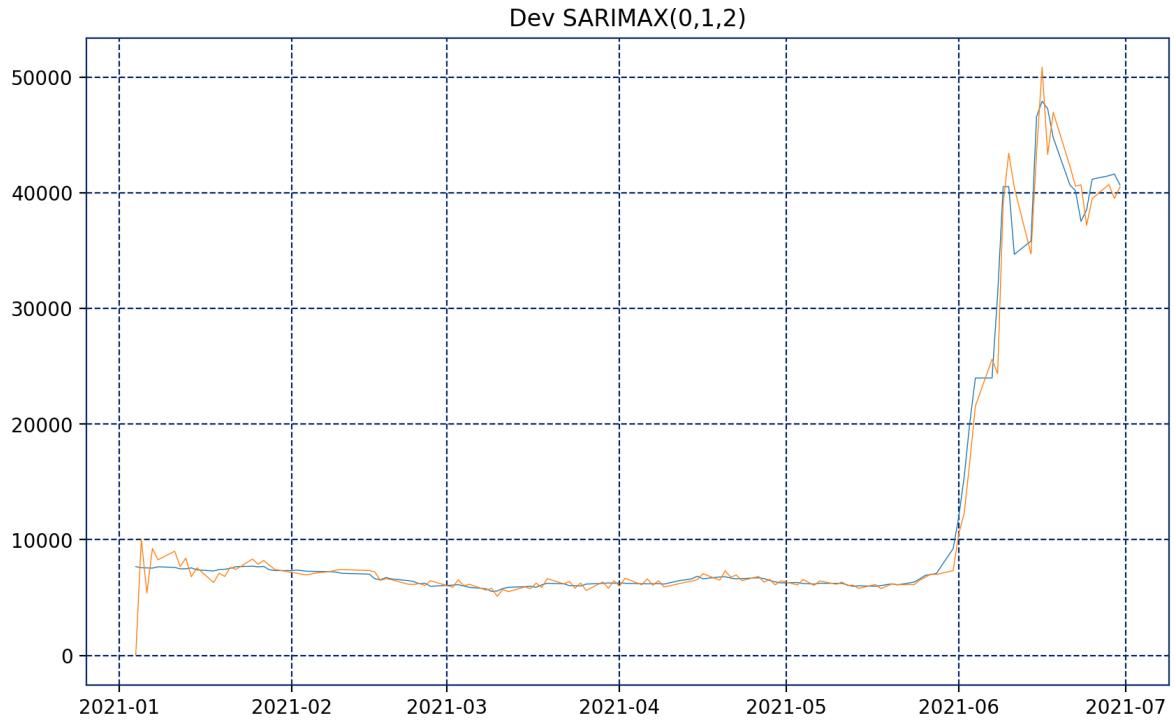
```
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  123
Model:               SARIMAX(1, 1, 5)   Log Likelihood               -1050.453
Date:                Tue, 06 Jul 2021   AIC                           2114.906
Time:                        10:20:32   BIC                           2134.534
Sample:                             0   HQIC                          2122.878
                                - 123
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.5046      0.062      8.200      0.000       0.384       0.625
ma.L1          0.0702      0.051      1.382      0.167      -0.029       0.170
ma.L2         -0.6096      0.058    -10.529      0.000      -0.723      -0.496
ma.L3          0.0700      0.041      1.707      0.088      -0.010       0.150
ma.L4          0.3314      0.046      7.214      0.000       0.241       0.421
ma.L5          0.4581      0.060      7.631      0.000       0.340       0.576
sigma2      1.639e+06   9.37e+04     17.492      0.000    1.46e+06    1.82e+06
===================================================================================
Ljung-Box (L1) (Q):                   0.01   Jarque-Bera (JB):               405.20
Prob(Q):                              0.94   Prob(JB):                         0.00
Heteroskedasticity (H):              12.00   Skew:                             0.62
Prob(H) (two-sided):                  0.00   Kurtosis:                        11.84
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-ste
p).
```

In [54]:

```
#원 데이터 예측값 구하기
fit_auto.fitted=pd.DataFrame(fit_auto.predict_in_sample())
import pandas as pd
fit_auto.fitted.index=df.index
```

In [55]:

```
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (10,6)
plt.plot(df,linewidth=0.5)
plt.plot(fit_auto.fitted,linewidth=0.5)
plt.title('Dev SARIMAX(0,1,2)')
plt.show()
```



In [56]:

```
#향후 10일 주가 예측
fit_auto.predict(n_periods=10)
```

Out[56]:

```
array([40174.04941617, 40971.46965397, 42433.77153108, 44197.60921121,
       45169.37785174, 45659.68837585, 45907.07690028, 46031.89796572,
       46094.87703241, 46126.65342229])
```

LSTM 주가예측

In [57]:

```
import torch
import os
import numpy as np
import pandas as pd
from tqdm import tqdm
import seaborn as sns
from pylab import rcParams
import matplotlib.pyplot as plt
from matplotlib import rc
from sklearn.preprocessing import MinMaxScaler
from pandas.plotting import register_matplotlib_converters
from torch import nn, optim
```

In [58]:

```
def create_sequences(data, seq_length):
    xs = []
    ys = []
    for i in range(len(data)-seq_length-1):
        x = data[i:(i+seq_length)]
        y = data[i+seq_length]
        xs.append(x)
        ys.append(y)
    return np.array(xs), np.array(ys)
```

In [59]:

```
class Lstm_Predictor(nn.Module):
  def __init__(self, n_features, n_hidden, seq_len, n_layers=2):
    super(Lstm_Predictor, self).__init__()
    self.n_hidden = n_hidden
    self.seq_len = seq_len
    self.n_layers = n_layers
    self.lstm = nn.LSTM(
      input_size=n_features,
      hidden_size=n_hidden,
      num_layers=n_layers,
      dropout=0.5
    )
    self.linear = nn.Linear(in_features=n_hidden, out_features=1)
  def reset_hidden_state(self):
    self.hidden = (
        torch.zeros(self.n_layers, self.seq_len, self.n_hidden),
        torch.zeros(self.n_layers, self.seq_len, self.n_hidden)
    )
  def forward(self, sequences):
    lstm_out, self.hidden = self.lstm(
      sequences.view(len(sequences), self.seq_len, -1),
      self.hidden
    )
    last_time_step = W
      lstm_out.view(self.seq_len, len(sequences), self.n_hidden)[-1]
    y_pred = self.linear(last_time_step)
    return y_pred
```

In [60]:

```
def train_model(model,train_data,train_labels,test_data=None,test_labels=None):
  loss_fn = torch.nn.MSELoss(reduction='sum')
  optimiser = torch.optim.Adam(model.parameters(), lr=1e-3)
  num_epochs = 60
  train_hist = np.zeros(num_epochs)
  test_hist = np.zeros(num_epochs)
  for t in range(num_epochs):
    model.reset_hidden_state()
    y_pred = model(X_all)
    loss = loss_fn(y_pred.float(), y_all)
    if test_data is not None:
      with torch.no_grad():
        y_test_pred = model(X_all)
        test_loss = loss_fn(y_test_pred.float(), y_all)
      test_hist[t] = test_loss.item()
      if t % 10 == 0:
        print(f'Epoch {t} train loss: {loss.item()} test loss: {test_loss.item()}')
    elif t % 10 == 0:
      print(f'Epoch {t} train loss: {loss.item()}')
    train_hist[t] = loss.item()
    optimiser.zero_grad()
    loss.backward()
    optimiser.step()
  return model.eval(), train_hist, test_hist
```

In [61]:

```
# 데이터 변환
df=df_top['Close'] #분석 데이터
scaler = MinMaxScaler()
scaler = scaler.fit(np.expand_dims(df, axis=1))
all_data = scaler.transform(np.expand_dims(df, axis=1))
all_data.shape
```

Out[61]:

```
(123, 1)
```

In [62]:

```
seq_length = 5
X_all, y_all = create_sequences(all_data, seq_length)
X_all = torch.from_numpy(X_all).float()
y_all = torch.from_numpy(y_all).float()
model = Lstm_Predictor(n_features=1,n_hidden=512,seq_len=seq_length,n_layers=2)
model, train_hist, _ = train_model(model, X_all, y_all)
```

```
Epoch 0 train loss: 11.121109008789062
Epoch 10 train loss: 4.257625102996826
Epoch 20 train loss: 3.232571840286255
Epoch 30 train loss: 3.3310368061065674
Epoch 40 train loss: 2.192603349685669
Epoch 50 train loss: 2.0739121437072754
```

In [63]:

```
DAYS_TO_PREDICT = 10
with torch.no_grad():
  test_seq = X_all[:1]
  preds = []
  for _ in range(DAYS_TO_PREDICT):
    y_test_pred = model(test_seq)
    pred = torch.flatten(y_test_pred).item()
    preds.append(pred)
    new_seq = test_seq.numpy().flatten()
    new_seq = np.append(new_seq, [pred])
    new_seq = new_seq[1:]
    test_seq = torch.as_tensor(new_seq).view(1, seq_length, 1).float()
```

In [64]:

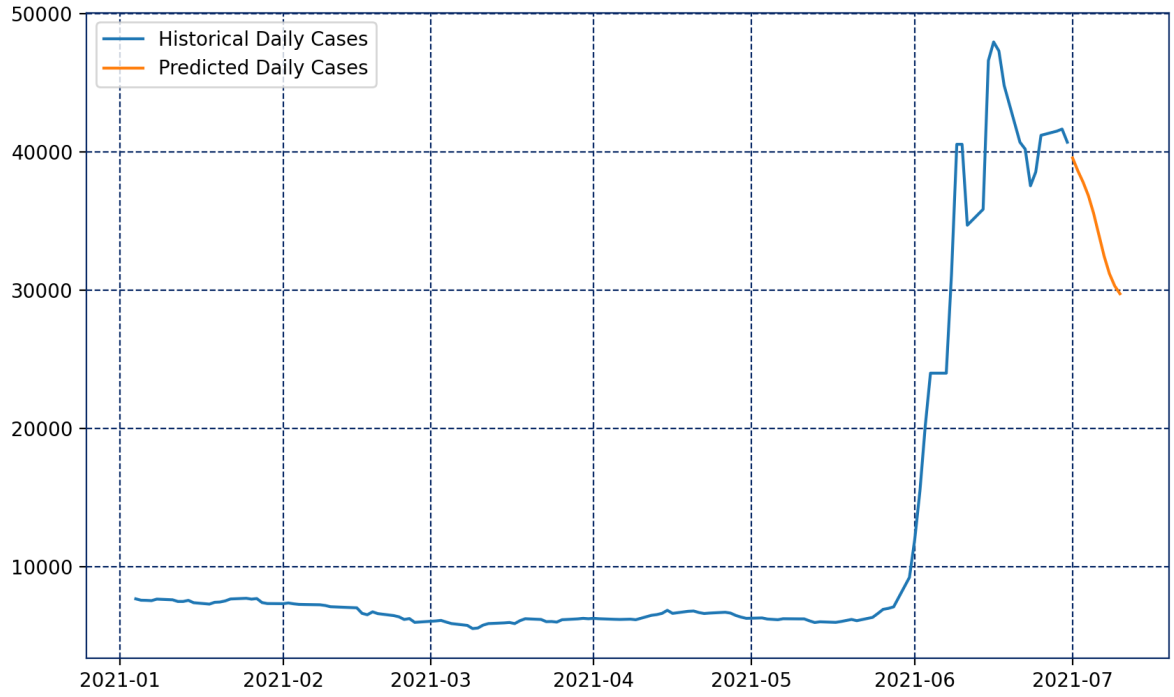
```
predicted_cases = scaler.inverse_transform(
    np.expand_dims(preds, axis=0)
).flatten()
```

In [65]:

```
predicted_index = pd.date_range(start=df.index[-1],periods=DAYS_TO_PREDICT + 1,closed='right')
predicted_cases = pd.Series(data=predicted_cases,index=predicted_index)
```

In [66]:

```
#분석대상 주가 및 예측주가
plt.plot(df, label='Historical Daily Cases')
plt.plot(predicted_cases, label='Predicted Daily Cases')
plt.legend();
```



In [67]:

```
#향후 10일 예측결과
predicted_cases
```

Out[67]:

```
2021-07-01    39566.204677
2021-07-02    38630.802243
2021-07-03    37809.007182
2021-07-04    36835.134701
2021-07-05    35549.989849
2021-07-06    33998.204458
2021-07-07    32453.840005
2021-07-08    31194.267685
2021-07-09    30310.103866
2021-07-10    29744.648526
Freq: D, dtype: float64
```

In [ ]:

```

```