Assignment 2: YINGWEI CHEN

Question 1

Get 1-day trading data (open, close, high, low and volume) with 1 minute intveral for Therma Bright Inc. (THRM.V). Plot candle stick plot and VWAP in one figure using the 1-day data above.

In [1]:

```
import finnhub
import yfinance as yf
import datetime
import pandas as pd
import numpy as np
from pandas_datareader import data as pdr
import plotly.graph_objects as go

stock_name='THRM.V'
#stock_name='AMZN'
stock_candle=yf.Ticker(stock_name).history(interval='1m', period='1d')
stock_candle=stock_candle.reset_index()
stock_candle['trading_date']=stock_candle['Datetime'].apply(lambda x: x.date())
stock_candle['trading_time']=stock_candle['Datetime'].apply(lambda x: x.time().strftime('%H:%M'))
stock_candle['volume_cumsum']=stock_candle['Volume'].cumsum()

range_high=stock_candle['High'].max()*1.01
range_low=stock_candle['Low'].min()*0.99

price_list=['Open','High','Low','Close']
price_vwap=['VWAP_'+x for x in price_list]
for price_type, price_vwap in zip(price_list,price_vwap):
    stock_candle[price_vwap]=(stock_candle[price_type]*stock_candle['Volume']).cumsum()/stock_candle['volume_cumsum']

raw_trading_data=go.Candlestick(x=stock_candle['trading_time'], open=stock_candle['Open'], high=stock_candle['High'], low
vwap_trading_data=go.Scatter(x=stock_candle['trading_time'], y=stock_candle['VWAP_Close'])

fig=go.Figure(data=[raw_trading_data,vwap_trading_data])
fig.update_layout(title=f'Stock Candle and VWAP ---- {stock_name} ---- Intraday ', title_font_size=30, showlegend=True,
                  width=1200, height=600, yaxis_title=f'Stock Price', xaxis_title=f'Trading Time')

fig.update_yaxes(range=(range_low,range_high))
fig.show()
```


Stock Candle and VWAP ---- THRM.V ---- Intraday
Stock Price
0.53
0.52
0.51
0.5
0.49
0.48
09:30
09:33
09:36
09:39
09:42
09:49
09:53
09:56
10:00
10:08
10:14
10:17
10:20
10:24
10:27
10:31
10:34
10:37
10:50
10:58
11:10
11:19
11:28
11:33
11:37
11:45
11:57
12:13
12:45
12:51
12:57
13:04
13:09
13:17
13:30
13:50
14:01
14:18
14:31
14:34
14:42
Trading Time

Question 2

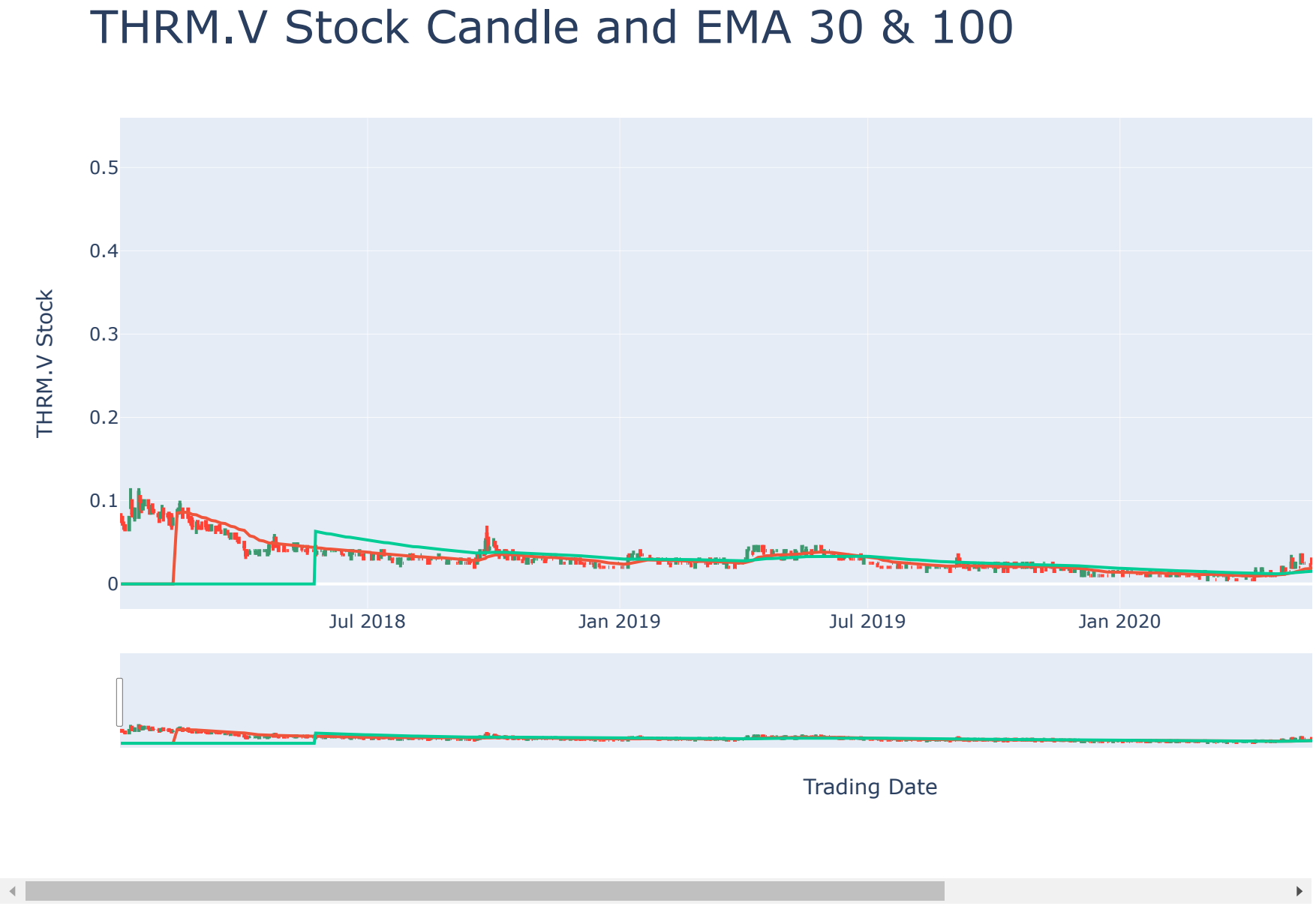
Get daily trading data (pen, close, high, low and volume) from 20180-01-01 to 2020-12-31 for Therma Bright Inc. (THRM.V) Plot candle stick plot and 30-day and 100 day expenential moving averages on one figure.

In [2]:

```
stock_name='THRM.V'
hist_2018_2020=yf.download(stock_name, start='2018-01-01', end='2021-01-01' ).reset_index()
alpha_30=2.0/(30+1)
alpha_100=2.0/(100+1)
hist_2018_2020.loc[:,'sma_30'] =hist_2018_2020['Close'].rolling(30).mean()
hist_2018_2020.loc[:,'sma_100']=hist_2018_2020['Close'].rolling(100).mean()

# Comment out 2 lines code are not correct and keep here for reference
#hist_2018_2020['EMA_30']=np.where(hist_2018_2020['EMA_30'].shift(periods=1)>0,
#    alpha_30*hist_2018_2020['Close']+hist_2018_2020['EMA_30'].shift(periods=1)*(1-alpha_30), hist_2018_2020['EMA_30'])

def ema_from_df(array_df, ema_cnt):
    alpha=2.0/(1+ema_cnt)
    if len(array_df)<=ema_cnt:
        print(f"The input columns is shorter than {ema_cnt}")
    else:
        array_size=array_df.size
        temp_arr=np.zeros(array_size)
        copy_arr=np.copy(array_df)
        temp_arr[ema_cnt-1]=copy_arr[:ema_cnt].mean()
        print( temp_arr[ema_cnt-1])
        for i in range(ema_cnt, array_size):
            temp_arr[i]=temp_arr[i-1]*(1-alpha)+copy_arr[i]*alpha
        return temp_arr

hist_2018_2020.loc[:,'ema_30']=ema_from_df(hist_2018_2020['Close'], 30)
hist_2018_2020.loc[:,'ema_100']=ema_from_df(hist_2018_2020['Close'], 100)

daily_trading_data=go.Candlestick(x=hist_2018_2020['Date'], open=hist_2018_2020['Open'], high=hist_2018_2020['High'],
                                  low=hist_2018_2020['Low'], close=hist_2018_2020['Close'])
ema30_trading_data=go.Scatter(x=hist_2018_2020['Date'], y=hist_2018_2020['ema_30'])
ema100_trading_data=go.Scatter(x=hist_2018_2020['Date'], y=hist_2018_2020['ema_100'])

fig=go.Figure(data=[daily_trading_data,ema30_trading_data, ema100_trading_data])
fig.update_layout(title=f'{stock_name} Stock Candle and EMA 30 & 100', title_font_size=30, showlegend=True,
                  width=1200, height=600, xaxis_title='Trading Date', yaxis_title=f'{stock_name} Stock')

fig.show()
```

```
[**********100%**********]  1 of 1 completed
0.0851666676501433
```

```
0.06315000016242266
```



Question 3: (1)

Use the same data in Question 2. (1) Calculate daily return (return=log(today close/previous close)

In [3]:
```
hist_2018_2020.loc[:,'daily_return']=np.log(hist_2018_2020['Close']/hist_2018_2020['Close'].shift(periods=1))
hist_2018_2020=hist_2018_2020[hist_2018_2020['daily_return'].notnull()]
hist_2018_2020[['Close','daily_return']]
```

Out[3]:

	Close	daily_return
1	0.075	0.000000
2	0.070	-0.068993
3	0.065	-0.074108
4	0.080	0.207639
5	0.100	0.223144
...	...	...
748	0.210	-0.023531
749	0.225	0.068993
750	0.225	0.000000
751	0.225	0.000000
752	0.215	-0.045462

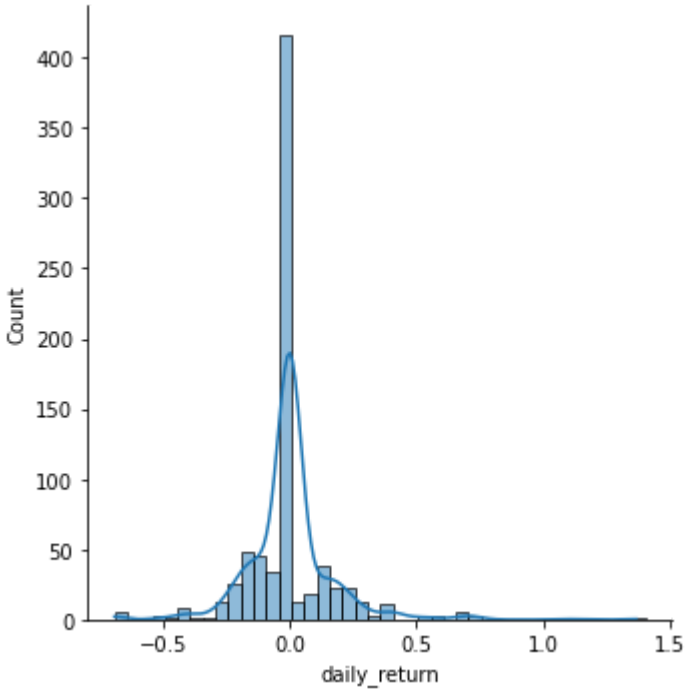
752 rows × 2 columns

Question 3: (2)

Use the same data in Question 2. (2) Conduct the hypothesis testing to check if the distribution of daily return is normal

In [4]:
```
# Visualization can provide the first impresssion of statistic distribution
import seaborn as sns
from scipy import stats
sns.displot(hist_2018_2020['daily_return'],kde=True, binwidth=0.05)
```

Out[4]:
```
<seaborn.axisgrid.FacetGrid at 0x1fadb40c048>
```



In [5]:

```
# Calculate the sample mean and standard error
# Make assumption of using sample mean and standard error as the mean and stardard deviation of normal distribution.
# Print out H0 and H1 content

from scipy import stats
hist_2018_2020.columns
#print(hist_2018_2020.head())
sample_mean=hist_2018_2020['daily_return'].mean()
sample_std=hist_2018_2020['daily_return'].std()
print("-----------------------------------------------------------------------------------------")
print(f"The sample Mean: {sample_mean} \nThe sample standard deviation: {sample_std} \n")
print(f"Assume the sample mean and standard error are the best estimate of population mean and standard deviation")
print("We make Hypothesis H0 and H1 as follow:")
print(f"H0: the stock daily return follow normal distribution of N({sample_mean},{sample_std})")
print(f"H1: the stock daily return does not follow normal distribution")
```

```
-----------------------------------------------------------------------------------------
The sample Mean: 0.001400465281235082
The sample standard deviation: 0.17459207289146936

Assume the sample mean and standard error are the best estimate of population mean and standard deviation
We make Hypothesis H0 and H1 as follow:
```

```
H0: the stock daily return follow normal distribution of N(0.001400465281235082,0.17459207289146936)
H1: the stock daily return does not follow normal distribution
```

In [6]:

```
# There are multiple testing methodologies avaialbe in Python Scipy package
# Kolmogorov-Smirnov test is presented as follow

stats_ks, pvalue_ks=stats.kstest(hist_2018_2020['daily_return'],'norm',(sample_mean,sample_std))
print(f"KS test result: \n *** Test Statistic: {stats_ks} *** P-Value: {pvalue_ks} \n")
print ("Conclusion: (when set critical value as 0.05)")
if pvalue_ks<0.05:
    print(f"The pvalue of KS test is {pvalue_ks} and less than 0.05, We REJECT H0 and conclude that daily return does NOT
else:
   print(f"The pvalue of KS test is {pvalue_ks} and large than 0.05, We ACCEPT H0 and conclude that daily return folow nc
```

```
KS test result:
 *** Test Statistic: 0.29841278994676507 *** P-Value: 7.165538917336539e-60

Conclusion: (when set critical value as 0.05)
The pvalue of KS test is 7.165538917336539e-60 and less than 0.05, We REJECT H0 and conclude that daily return does NOT f
olow normal distribution!
```

Question 4

Take 10 stocks in S&P 500 and collect daily close price from 2020-06-01 to 2020-12-31 for slected stocks (1) Calculate daily return for each of 10 stocks.

In [7]:

```
df_sp500=pd.read_csv(r"C:\Finnhub_Data\finnhub_data\S_P550_Symbol_list.csv")
#yf_stock=yf.download(sp500_10, start='2020-06-01', end='2020-12-31')
stock_list=list(df_sp500.loc[:9,'Symbol'])
stock_daily=[]
for stock_name in stock_list:
    yf_stock=yf.download(stock_name,  start='2020-06-01', end='2020-12-31')
    yf_stock['symbol']=stock_name
    yf_stock[f'return_{stock_name}']=np.log(yf_stock['Close']/yf_stock['Close'].shift(periods=1))
    stock_daily.append(yf_stock.loc[:,[f'return_{stock_name}']])

stock_10sp=pd.concat(stock_daily,axis=1)
print(stock_10sp)
```

```
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
```

```
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
            return_MMM  return_ABT  return_ABBV  return_ABMD  return_ACN  \
Date
2020-06-01         NaN         NaN          NaN          NaN         NaN
2020-06-02    0.013154   -0.004202     0.003742     0.019791    0.011731
2020-06-03    0.022394   -0.013260    -0.001649     0.031494    0.007263
2020-06-04    0.008278   -0.035072     0.023380    -0.008522   -0.020372
2020-06-05    0.029460    0.018079     0.008668     0.056784    0.026193
...                ...         ...          ...          ...         ...
2020-12-23   -0.004530   -0.007695     0.004658     0.005673   -0.007854
2020-12-24    0.003042    0.008341    -0.000194    -0.002107   -0.000544
2020-12-28    0.001088   -0.005182     0.001838     0.030831    0.009207
2020-12-29   -0.004935    0.004997     0.012011     0.025307   -0.007421
2020-12-30    0.001494    0.001015     0.005429     0.009274   -0.000854

            return_ATVI  return_ADBE  return_AMD  return_AAP  return_AES
Date
2020-06-01          NaN          NaN         NaN         NaN         NaN
2020-06-02    -0.000137     0.001948   -0.001680   -0.002148    0.043655
2020-06-03    -0.042834    -0.001333   -0.015245    0.020085    0.038898
2020-06-04    -0.001574    -0.010623   -0.001898   -0.000844    0.015290
2020-06-05    -0.005458     0.018236    0.008891    0.027057    0.032697
...                 ...          ...         ...         ...         ...
2020-12-23     0.008796    -0.014505   -0.017433   -0.000690    0.028937
2020-12-24     0.008280     0.005919    0.002836    0.008688   -0.009410
2020-12-28     0.005154    -0.001822   -0.002290   -0.011769    0.001288
2020-12-29    -0.000656     0.006313   -0.010756   -0.013376    0.000429
2020-12-30     0.002296    -0.009324    0.018261    0.006171   -0.022999

[149 rows x 10 columns]
```

Question 4 - (2)

Run PCA on calculated daily return and find the first principal component.

In [8]:

```python
# Method 1: Self-Implementation Notice: set STD(ddof=0) is to be consistent with the PYTHON preprocessing package. ddof=1

stock_10sp=stock_10sp[stock_10sp.index>'2020-06-01']
print(len(stock_10sp))
#adjust_std=np.sqrt(len(stock_10sp)/(len(stock_10sp)-1))
```

```python
#print(adjust_std)
for stock_name in stock_list:
    stock_10sp[f'return_{stock_name}_u']=stock_10sp[f'return_{stock_name}'].mean()
    stock_10sp[f'return_{stock_name}_s']=stock_10sp[f'return_{stock_name}'].std()
    stock_10sp[f'return_{stock_name}_standard']=(stock_10sp[f'return_{stock_name}']-stock_10sp[f'return_{stock_name}_u'])
    #stock_10sp[f'return_{stock_name}_standard']=stock_10sp[f'return_{stock_name}_standard']*adjust_std
col_return_standard=[f'return_{x}_standard' for x in stock_list]
self_preprocessing_df=stock_10sp[col_return_standard]
print(self_preprocessing_df)
```

```
148
            return_MMM_standard  return_ABT_standard  return_ABBV_standard  \
Date
2020-06-02             0.796493            -0.314648              0.179834
2020-06-03             1.390323            -0.858644             -0.174605
2020-06-04             0.483121            -2.168655              1.471036
2020-06-05             1.844451             1.023528              0.503759
2020-06-08            -0.256501             1.715702              0.450222
...                         ...                  ...                   ...
2020-12-23            -0.340003            -0.524430              0.240107
2020-12-24             0.146605             0.438671             -0.078914
2020-12-28             0.021067            -0.373500              0.054686
2020-12-29            -0.365994             0.237842              0.723529
2020-12-30             0.047177            -0.001333              0.290801

            return_ABMD_standard  return_ACN_standard  return_ATVI_standard  \
Date
2020-06-02              0.731912             0.580179             -0.090747
2020-06-03              1.225266             0.324406             -2.410885
2020-06-04             -0.461599            -1.257373             -0.168862
2020-06-05              2.291365             1.407912             -0.379902
2020-06-08              0.482501             0.669569              0.924742
...                          ...                  ...                   ...
2020-12-23              0.136794            -0.540884              0.394664
2020-12-24             -0.191175            -0.122449              0.366605
2020-12-28              1.197303             0.435707              0.196752
2020-12-29              0.964475            -0.516083             -0.118976
2020-12-30              0.288575            -0.140185              0.041443

            return_ADBE_standard  return_AMD_standard  return_AAP_standard  \
Date
2020-06-02              0.013218            -0.162424            -0.182895
2020-06-03             -0.131998            -0.574458             1.190362
2020-06-04             -0.543147            -0.169066            -0.102310
2020-06-05              0.734071             0.158643             1.620974
2020-06-08              0.473301            -0.185861             0.055395
...                          ...                  ...                  ...
2020-12-23             -0.714958            -0.640938            -0.092844
```

```
2020-12-24              0.188950           -0.025268            0.486436
2020-12-28             -0.153659           -0.180964           -0.777135
2020-12-29              0.206396           -0.438128           -0.876393
2020-12-30             -0.485681            0.443263            0.330980

            return_AES_standard
Date
2020-06-02             1.757103
2020-06-03             1.546187
2020-06-04             0.499357
2020-06-05             1.271207
2020-06-08             1.851732
...                         ...
2020-12-23             1.104521
2020-12-24            -0.595859
2020-12-28            -0.121487
2020-12-29            -0.159587
2020-12-30            -1.198447

[148 rows x 10 columns]
```

In [9]:

```
#Method 2 Standardize return using Python preprocessing package
# For demonstration purpose only
from sklearn.preprocessing import StandardScaler
raw_feature=['return_'+x for x in stock_list]
sklearn_preprocessing=stock_10sp.loc[:,raw_feature]
#print(sklearn_preprocessing)
sklearn_preprocessing=StandardScaler().fit_transform(sklearn_preprocessing)
sklearn_preprocessing_df=pd.DataFrame(sklearn_preprocessing)
sklearn_preprocessing_df.columns=col_return_standard
#print(sklearn_preprocessing_df)
```

In [10]:

```
# Princial Component Analysis on Standardized Data
# Merge back with standardized return columns
from sklearn.decomposition import PCA
pca=PCA(n_components=2)
principal_components=pca.fit_transform(self_preprocessing_df)
PCA_Final_df=pd.concat([pd.DataFrame(principal_components,columns=['Return_Close_PCA1','Return_Close_PCA2']),
                        self_preprocessing_df.reset_index()],axis=1).set_index('Date')
print(PCA_Final_df)
```

```
            Return_Close_PCA1  Return_Close_PCA2  return_MMM_standard  \
Date
2020-06-02          -0.939493          -1.233198             0.796493
2020-06-03          -0.385581          -3.067722             1.390323
```

```
2020-06-04          1.089927         -0.831744            0.483121
2020-06-05         -3.356692         -1.840471            1.844451
2020-06-08         -1.973215         -0.220244           -0.256501
...                      ...               ...                 ...
2020-12-23          0.447512         -0.589730           -0.340003
2020-12-24         -0.239738          0.307374            0.146605
2020-12-28         -0.153035          0.190966            0.021067
2020-12-29          0.072545          0.307427           -0.365994
2020-12-30          0.090792          0.394585            0.047177

            return_ABT_standard  return_ABBV_standard  return_ABMD_standard  \
Date
2020-06-02            -0.314648              0.179834              0.731912
2020-06-03            -0.858644             -0.174605              1.225266
2020-06-04            -2.168655              1.471036             -0.461599
2020-06-05             1.023528              0.503759              2.291365
2020-06-08             1.715702              0.450222              0.482501
...                         ...                   ...                   ...
2020-12-23            -0.524430              0.240107              0.136794
2020-12-24             0.438671             -0.078914             -0.191175
2020-12-28            -0.373500              0.054686              1.197303
2020-12-29             0.237842              0.723529              0.964475
2020-12-30            -0.001333              0.290801              0.288575

            return_ACN_standard  return_ATVI_standard  return_ADBE_standard  \
Date
2020-06-02             0.580179             -0.090747              0.013218
2020-06-03             0.324406             -2.410885             -0.131998
2020-06-04            -1.257373             -0.168862             -0.543147
2020-06-05             1.407912             -0.379902              0.734071
2020-06-08             0.669569              0.924742              0.473301
...                         ...                   ...                   ...
2020-12-23            -0.540884              0.394664             -0.714958
2020-12-24            -0.122449              0.366605              0.188950
2020-12-28             0.435707              0.196752             -0.153659
2020-12-29            -0.516083             -0.118976              0.206396
2020-12-30            -0.140185              0.041443             -0.485681

            return_AMD_standard  return_AAP_standard  return_AES_standard
Date
2020-06-02            -0.162424            -0.182895             1.757103
2020-06-03            -0.574458             1.190362             1.546187
2020-06-04            -0.169066            -0.102310             0.499357
2020-06-05             0.158643             1.620974             1.271207
2020-06-08            -0.185861             0.055395             1.851732
...                         ...                  ...                  ...
2020-12-23            -0.640938            -0.092844             1.104521
2020-12-24            -0.025268             0.486436            -0.595859
2020-12-28            -0.180964            -0.777135            -0.121487
```

```
2020-12-29           -0.438128           -0.876393           -0.159587
2020-12-30            0.443263            0.330980           -1.198447

[148 rows x 12 columns]
```

Question 4 (3) Note: Using Two Y-axis to handle different scale

Plot first principal compoent and daily return of S&P500 in one figure

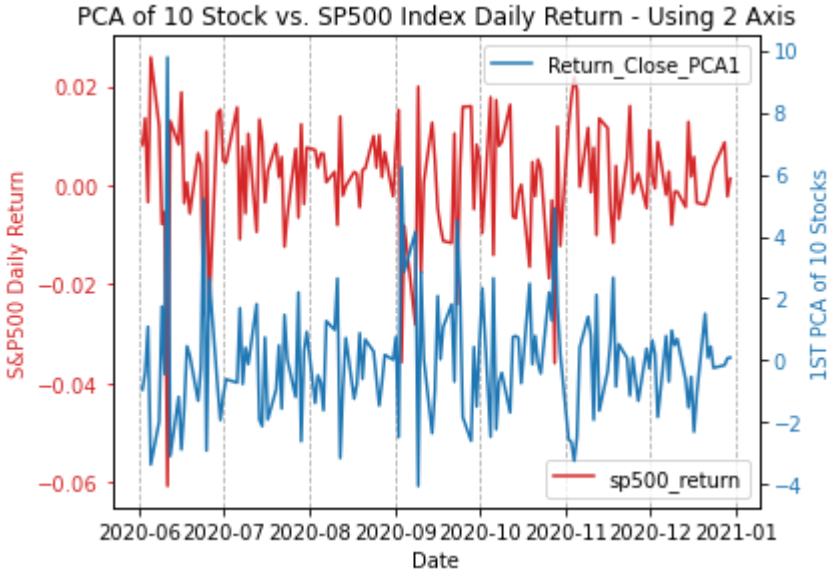
In [11]:

```
sp500=yf.download('^GSPC',  start='2020-06-01', end='2020-12-31')
sp500['sp500_return']=np.log(sp500['Close']/sp500['Close'].shift(periods=1))
sp500['sp500_return_pct']=sp500['sp500_return']*100
sp500_pca=pd.concat([sp500.loc[sp500.index>'2020-06-01',['sp500_return_pct','sp500_return']],PCA_Final_df.loc[:,'Return_C
print(sp500_pca)

import matplotlib.pyplot as plt
fig,ax1=plt.subplots()
color_ax1='tab:red'
color_ax2='tab:blue'
ax1.set_xlabel('Date')
ax1.set_ylabel('S&P500 Daily Return', color=color_ax1)
ax1.plot('Date', 'sp500_return', data=sp500_pca,color=color_ax1)
ax1.tick_params(axis='y',color='red',labelcolor=color_ax1)
plt.legend()
ax2=ax1.twinx()
ax2.set_ylabel('1ST PCA of 10 Stocks',color=color_ax2)
ax2.plot('Date', 'Return_Close_PCA1',data=sp500_pca, color=color_ax2)
ax2.tick_params(axis='y',labelcolor=color_ax2)
fig.tight_layout()
#plt.plot('Date', 'sp500_return_pct', data=sp500_pca)
#plt.plot('Date', 'Return_Close_PCA1',data=sp500_pca)

ax1.xaxis.grid(which='major',linestyle='--')
plt.legend()
plt.title("PCA of 10 Stock vs. SP500 Index Daily Return - Using 2 Axis")
```

```
[**********************100%**********************]  1 of 1 completed
          Date  sp500_return_pct  sp500_return  Return_Close_PCA1
0   2020-06-02          0.817731      0.008177          -0.939493
1   2020-06-03          1.355667      0.013557          -0.385581
2   2020-06-04         -0.337439     -0.003374           1.089927
3   2020-06-05          2.587401      0.025874          -3.356692
4   2020-06-08          1.196966      0.011970          -1.973215
..         ...               ...           ...                ...
```

```
143 2020-12-23          0.074553      0.000746          0.447512
144 2020-12-24          0.353035      0.003530         -0.239738
145 2020-12-28          0.868471      0.008685         -0.153035
146 2020-12-29         -0.222986     -0.002230          0.072545
147 2020-12-30          0.134065      0.001341          0.090792

[148 rows x 4 columns]
```

Out[11]: Text(0.5, 1.0, 'PCA of 10 Stock vs. SP500 Index Daily Return - Using 2 Axis')



Question 4 (4)

Calculate the correlation coefficient between first principal component and daily return of S&P500 index

In [12]:

```
from scipy.stats import pearsonr
corr,pvalue=pearsonr(sp500_pca['sp500_return'],sp500_pca['Return_Close_PCA1'])
print('-------------------------------------------------------------------------------------')
print(f"The correlation Coefficient is: {corr} and pvalue is: {pvalue}")
print('-------------------------------------------------------------------------------------')
significant_level=0.05
corr_threshold=0.5
if pvalue<significant_level:
    if corr>corr_threshold:
        print(f"Conclusion:The 1st PCA of selected 10 Stocks and SP500 Index Daily Return are positive correlated when th
    if corr<corr_threshold*(-1):
```

```
        print(f"Conclusion:The 1st PCA of selected 10 stocks and SP500 Index Daily Return are negative correlated when th
print('-------------------------------------------------------------------------------------')
```

```
-------------------------------------------------------------------------------------
The correlation Coefficient is: -0.8835144305792828 and pvalue is: 6.061556245048064e-50
-------------------------------------------------------------------------------------
Conclusion:The 1st PCA of selected 10 stocks and SP500 Index Daily Return are negative correlated when threshold set as
0.5 and significant level 0.05!
-------------------------------------------------------------------------------------
```

Extra Credit

Pick 2 companies in the same indsutry, collect close, open, high and low for each company from 2020-06 to 202012. Variable set 1 - company A's close, open, high and low Variable set 2 - company B's close, open, high and low (1) Run canonical analysis using data stated above (2) Give some conclusion or finding from your analysis

In [13]:

```
# (1) Frun canonical analysis using data stated above
from sklearn.cross_decomposition import CCA
stock_symbol1='DAL'
stock_symbol2='UAL'
stock_same_industry=[stock_symbol1,stock_symbol2]
stock_df=['stock_'+x for x in stock_same_industry]
start_date='2020-06-01'
end_date='2020-12-31'
stock_1=yf.download(stock_symbol1,'2020-06-01','2020-12-31')
stock_2=yf.download(stock_symbol2,'2020-06-01','2020-12-31')
column_keep=['Open','High','Low','Close']
stock_1=stock_1[column_keep]
stock_2=stock_2[column_keep]
stock_1=(stock_1-stock_1.mean())/stock_1.std()
stock_2=(stock_2-stock_2.mean())/stock_2.std()
print(stock_1)
ca=CCA()
ca.fit(stock_1, stock_2)
stock1_cca, stock2_cca=ca.transform(stock_1, stock_2)
corr_first=np.corrcoef(stock1_cca[:,0],stock2_cca[:,0])
corr_second=np.corrcoef(stock1_cca[:,1],stock2_cca[:,1])
print(f"The correlation between first components of two CCA is {corr_first[0,1]}")
print(f"The correlation between second components of two CCA is {corr_second[0,1]}")
```

```
[**********************100%**********************]  1 of 1 completed
[**********************100%**********************]  1 of 1 completed
                Open      High       Low     Close
```

```
Date
2020-06-01 -1.348561 -1.212318 -1.281585 -1.200969
2020-06-02 -1.081040 -1.168388 -1.107746 -1.152838
2020-06-03 -1.015646 -0.844901 -0.954121 -0.739716
2020-06-04 -0.369634 -0.004235 -0.444732  0.044416
2020-06-05  1.003637  0.880362  0.410314  0.401386
...              ...       ...       ...       ...
2020-12-23  1.419779  1.525338  1.568567  1.620701
2020-12-24  1.641722  1.519347  1.653465  1.518423
2020-12-28  1.611998  1.587240  1.736342  1.602652
2020-12-29  1.724951  1.599221  1.714107  1.578586
2020-12-30  1.558494  1.613199  1.712086  1.684876

[149 rows x 4 columns]
The correlation between first components of two CCA is 0.9623537744088362
The correlation between second components of two CCA is 0.9063151681785587
```

(2) Give some conclusion or findings from your analysis

Conclusion: UAL and DAL are two companies in the Airlines industry which were highly negative affected by the covid-19 and positively affected by government support and vaccine information. The volatility from above reason would dominate over other factors such as financial performance during this special periods. Thus it is not surprise to see the first component show high correlation of 0.96. Since our attributes are limited to stock prices (Open, High,Low,Close) thus the variability among attributes are very limit. This leads to the unusually high correlation coefficience of the second components. Following heatmap shows the correlation matrix of stock prices (Open, High, Low, Close) using stock 1 (DAL) as example.

In [14]:
```
correlation_matrix=stock_1.corr()
sns.heatmap(correlation_matrix, annot=True)
```

Out[14]:
```
<AxesSubplot:>
```


Open
High
Low
Close
1
0.99
0.99
0.98
0.99
1
0.99
0.99
0.99
0.99
1
0.99
0.98
0.99
0.99
1
Open
High
Low
Close
1.000
0.998
0.996
0.994
0.992
0.990
0.988
0.986

In [ ]: