Homework 1

This notebook is created by Kaiyi Li at 2020-02-22 09:04:42 for the homework1 of volatility class.
Email: kaiyi.li@nyu.edu net id: kl2538 github link: https://github.com/LiKaiy1/volatility/blob/master/HW_1_remastered.ipynb

1.

Use the attached data of daily equity prices given by the Shanghai Composite Index

```
import pandas as pd
%matplotlib inline
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import plotly.express as px
from scipy.stats import kurtosis
from scipy.stats import skew
import numpy as np
```

I downloaded the data from yahoo finance for the data of SCI over the same period for some datetime converting issue. The Adj Close and Date are exactly the same.

```
sci = pd.read_csv('SCI_OHLC.csv')[['Date','Open','High','Low','Close','Adj Close']]
```

```
sci['DATE'] = pd.to_datetime(sci['Date'],format='%Y-%m-%d')
sci = sci[['DATE','Open','High','Low','Close','Adj Close']]
```

```
sci.set_index('DATE', inplace=True)
```

Plot the index candlestick.

```
fig = go.Figure(data=[go.Candlestick(
    x=sci.index,
    open=sci['Open'], high=sci['High'],
    low=sci['Low'], close=sci['Adj Close'],
    increasing_line_color= 'red', decreasing_line_color= 'green'
)])
fig.show()
```

a) Calculate the mean annualized return since 2000

```
# To calculate the mean annualized return, first we resample the dataset to 20 years.
sci_annually_price = sci['Adj Close'].resample('Y').last()
sci_annually_price = pd.DataFrame(sci_annually_price)
```

```
sci_annually_price = sci_annually_price.append(pd.DataFrame(sci['Adj Close'].iloc[sci.index == pd.to_datetime('2000-01-03',format='%Y-%m-%d')]))
sci_annually_price.sort_index(inplace=True)
```

```
sci_annual_returns = pd.DataFrame(sci_annually_price.pct_change())
sci_annual_returns['ann.return'] = sci_annual_returns['Adj Close']
sci_annual_returns = sci_annual_returns[['ann.return']]
```

```
sci_annual_returns=sci_annual_returns[1:len(sci_annual_returns)-1]
```

```
# Calculate the mean annualized return.
sci_annual_returns['ann.return'].mean()
0.12824764612628745
```

Mean annualized return is 12.82%.

b) Calculate the annualized volatility since 2000

Annualized Volatility is calculated as

$$\text{Annualized Volatility is calculated as}$$

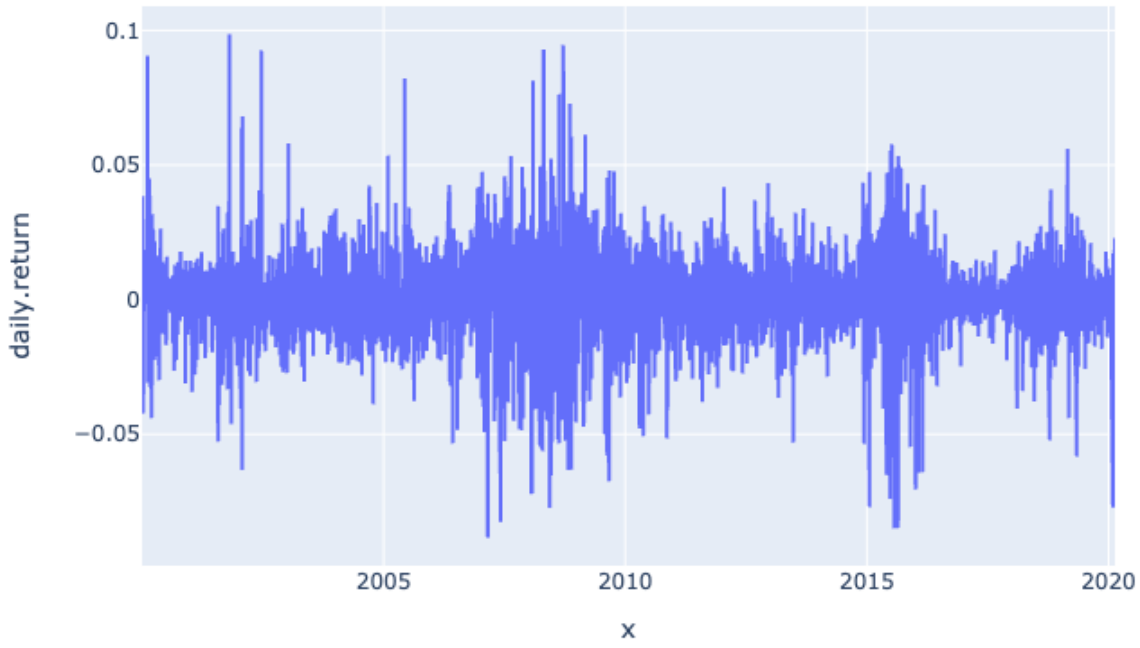
$$ann.vol = \sqrt{252} \times \sigma_{daily}$$

First, we calculate the daily return (price change in percentages).

```
sci_daily_returns = pd.DataFrame(sci['Adj Close'].pct_change())
sci_daily_returns['daily.return'] = sci_daily_returns['Adj Close']
sci_daily_returns = sci_daily_returns[['daily.return']]
```

Plot the daily return.

```
fig = px.line(sci_daily_returns, x=sci_daily_returns.index, y='daily.return')
fig.show()
```



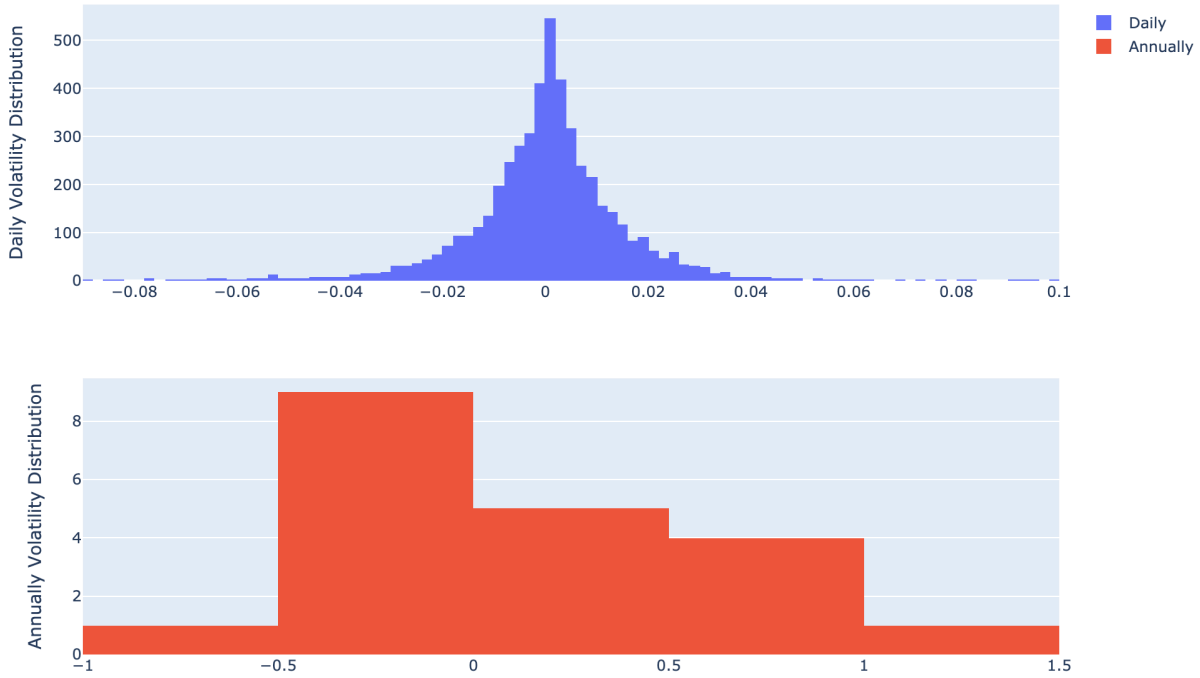

```
# Annualized volatility is daily volatility * sqrt(252)
float(sci_daily_returns.std()*(252**0.5))
0.24424626314336378
```

The annualized volatility is 24.42% since 2000.

c) Calculate the skewness and kurtosis over the same period. Is there evidence of fat tail?

Plot the histogram of daily returns.

```
# Initialize figure with subplots
fig = make_subplots(rows=2, cols=1)
fig.append_trace(go.Histogram(x=sci_daily_returns['daily.return'],name =
"Daily"),row = 1,col = 1)
fig.append_trace(go.Histogram(x=sci_annual_returns['ann.return'],name =
"Annually"),row = 2,col = 1)
fig.update_yaxes(title_text="Daily Volatility Distribution", row=1, col=1)
fig.update_yaxes(title_text="Annually Volatility Distribution", row=2, col=1)
fig.update_layout(title_text="Return Distribution", height=700)
fig.show()
```




```
daily_return_series = np.array(sci_daily_returns['daily.return'])
#Drop nan numbers
daily_return_series = daily_return_series[~np.isnan(daily_return_series)]
# kurtosis(np.array(sci_daily_returns['daily.return']))
```

```
print("kurtosis of daily return price changes is
{}".format(kurtosis(daily_return_series)))
print("skewness of daily return price changes is
{}".format(skew(daily_return_series)))
print("kurtosis of annually return price changes is
{}".format(kurtosis(sci_annual_returns['ann.return'])))
print("skewness of annually return price changes is
{}".format(skew(sci_annual_returns['ann.return'])))
# stats.kurtosis(sci_daily_returns)
```

```
kurtosis of daily return price changes is 5.155707477450127
skewness of daily return price changes is -0.21556464062649736
kurtosis of annually return price changes is 0.4458631536004818
skewness of annually return price changes is 0.9856216289904693
```

Therfore, since the kurtosis of a normal distribution is 3, the distribution of daily returns of the SCI is leptokurtic. While the skew is lightly smaller compared to that of a normal distribution which is 0. **For daily returns, There are evidences of a fat tail. (large kurtosis and negative skewness)**. For annual return distribution, it is platyokurtic and has a positive skewness, and **therefore has evidence of a fat tail. (small kurtosis and positive skewness)**

d) Test the efficient market hypothesis by looking at the first 10 autocorrelations. What do you conclude?

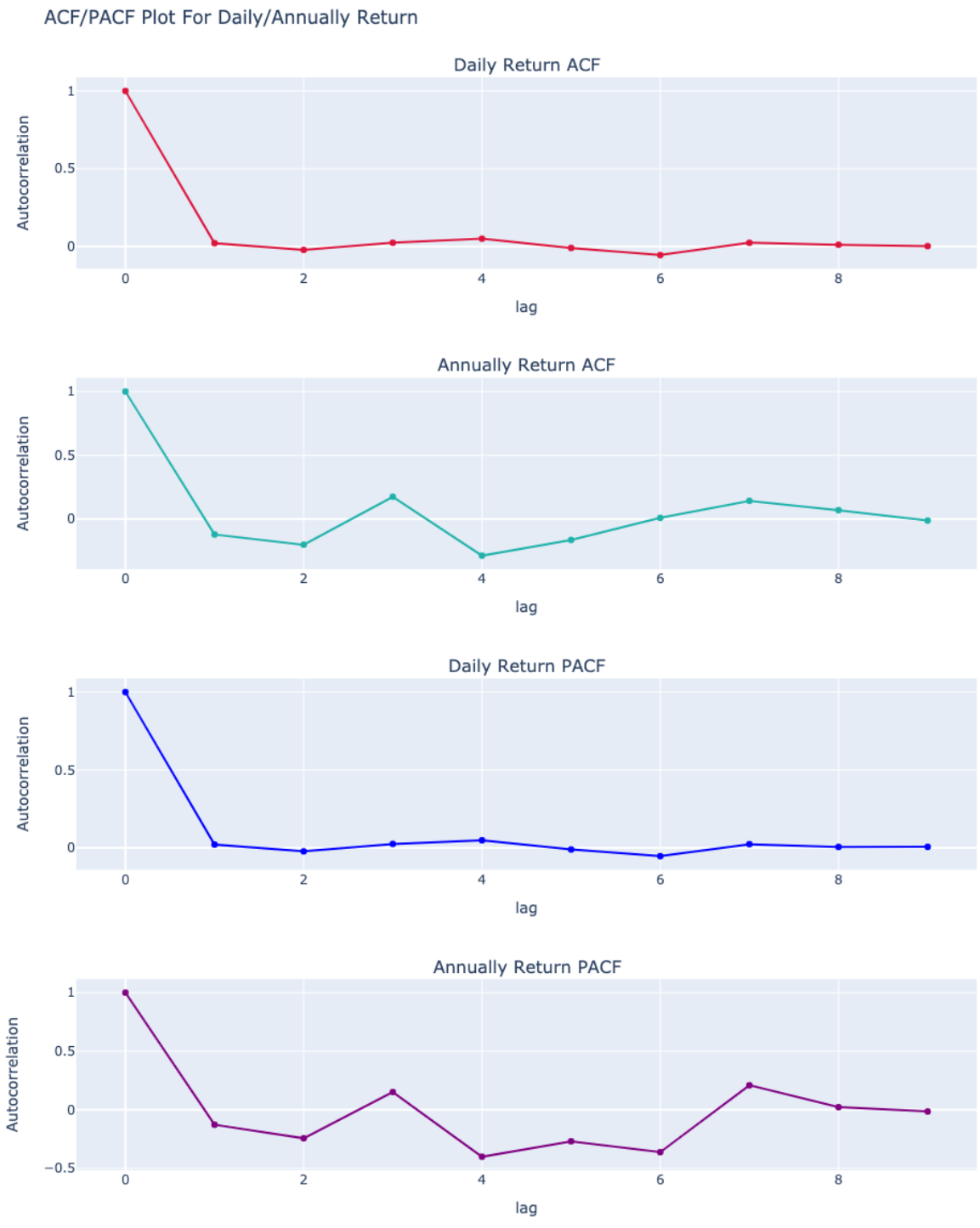
```
from statsmodels.tsa.stattools import acf, pacf
import warnings
warnings.filterwarnings('ignore')
```

```
daily_lag_acf = acf(daily_return_series, nlags=10)
annually_lag_acf = acf(sci_annual_returns['ann.return'],nlags = 10)
daily_lag_pacf = pacf(daily_return_series,nlags = 10)
annually_lag_pacf = pacf(sci_annual_returns['ann.return'],nlags = 10)
```

```
fig = make_subplots(rows=4, cols=1,
                    subplot_titles=("Daily Return ACF", "Annually Return ACF",
                                    "Daily Return PACF", "Annually Return
PACF"),
                     vertical_spacing=0.1)
fig.append_trace(go.Scatter(x = np.arange(10), y=daily_lag_acf,
line_color="crimson",
                            ),row=1,col=1,)
fig.append_trace(go.Scatter(x = np.arange(10), y=annually_lag_acf,
line_color="lightseagreen",
                            ),row=2,col=1)
fig.append_trace(go.Scatter(x = np.arange(10), y=daily_lag_pacf,
line_color="blue",
                            ),row=3,col=1)
fig.append_trace(go.Scatter(x = np.arange(10), y=annually_lag_pacf,
line_color="purple",
                            ),row=4,col=1)

fig.update_xaxes(title_text="lag", row=1, col=1)
fig.update_xaxes(title_text="lag", row=2, col=1)
fig.update_xaxes(title_text="lag", row=3, col=1)
fig.update_xaxes(title_text="lag", row=4, col=1)
fig.update_yaxes(title_text="Autocorrelation", row=1, col=1)
```

```
fig.update_yaxes(title_text="Autocorrelation", row=2, col=1)
fig.update_yaxes(title_text="Autocorrelation", row=3, col=1)
fig.update_yaxes(title_text="Autocorrelation", row=4, col=1)
fig.update_layout(
    title="ACF/PACF Plot For Daily/Annually Return",
    )
fig.update_layout(height=1200, width=1000,
                  showlegend = False)
fig.show()
```



From the Daily ACF and PACF plots, we can see that there are some (but small) autocorrelations for the firts few lags. However,the annually ACF and DACF plots exhibits larger autocorrelations. We conclude that: **Although autocorrelations exist in the data and the efficient market hypothesis is not strictly valid, but the market is very close to satisfy the efficient market hypothesis.**

e) Test for volatility clustering by looking at the first 10 autocorrelations of squared returns.

First, we square the annual and daily returns.

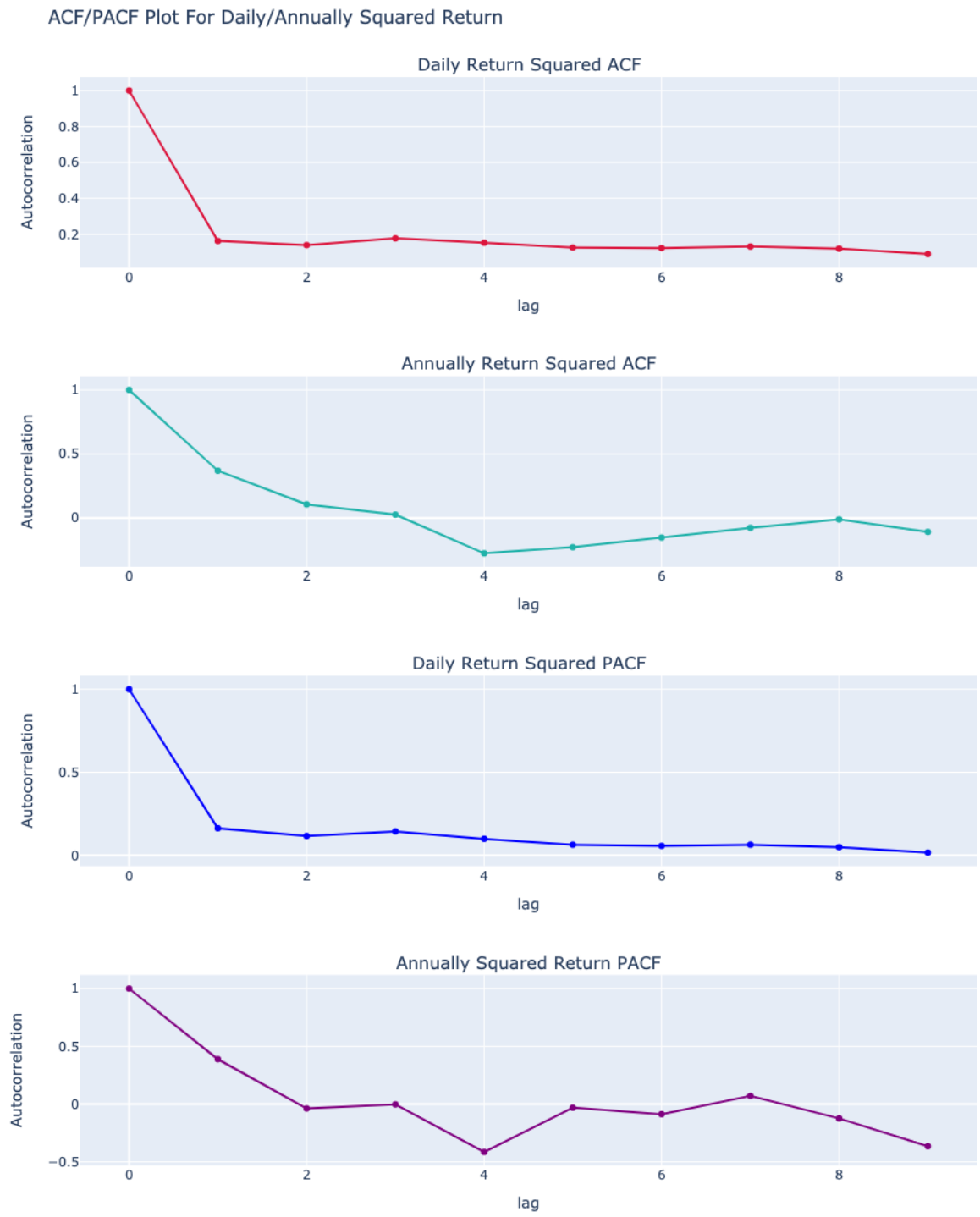
```
#squared all annual returns
squared_annual_returns_series = sci_annual_returns['ann.return']**2
squared_daily_returns_series = daily_return_series**2
# squared_annual_returns.head()
```

```
daily_squared_lag_acf = acf(squared_daily_returns_series, nlags=10)
annually_squared_lag_acf = acf(squared_annual_returns_series,nlags = 10)
daily_squared_lag_pacf = pacf(squared_daily_returns_series,nlags = 10)
annually_squared_lag_pacf = pacf(squared_annual_returns_series,nlags = 10)
```

```
fig = make_subplots(rows=4, cols=1,
                    subplot_titles=("Daily Return Squared ACF", "Annually
Return Squared ACF",
                                    "Daily Return Squared PACF", "Annually
Squared Return PACF"),
                     vertical_spacing=0.1)
fig.append_trace(go.Scatter(x = np.arange(10), y=daily_squared_lag_acf,
line_color="crimson",
                            ),row=1,col=1,)
fig.append_trace(go.Scatter(x = np.arange(10), y=annually_squared_lag_acf,
line_color="lightseagreen",
                            ),row=2,col=1)
fig.append_trace(go.Scatter(x = np.arange(10), y=daily_squared_lag_pacf,
line_color="blue",
                            ),row=3,col=1)
fig.append_trace(go.Scatter(x = np.arange(10), y=annually_squared_lag_pacf,
line_color="purple",
                            ),row=4,col=1)

fig.update_xaxes(title_text="lag", row=1, col=1)
fig.update_xaxes(title_text="lag", row=2, col=1)
fig.update_xaxes(title_text="lag", row=3, col=1)
fig.update_xaxes(title_text="lag", row=4, col=1)
fig.update_yaxes(title_text="Autocorrelation", row=1, col=1)
fig.update_yaxes(title_text="Autocorrelation", row=2, col=1)
fig.update_yaxes(title_text="Autocorrelation", row=3, col=1)
```

```
fig.update_yaxes(title_text="Autocorrelation", row=4, col=1)
fig.update_layout(
    title="ACF/PACF Plot For Daily/Annually Squared Return",
    )
fig.update_layout(height=1200, width=1000,
                  showlegend = False)
fig.show()
```



From the ACF plots and PACF plots of squared annual or daily returns for the first 10 lags, we could observe that **consistent autocorrelations exist, and thus there exists volatility clustering.**

2. Calculate annualized historical volatilities for each day over the whole sample period. Do it once with monthly window and again with annual window.

> *Historical volatility is calculated as a rolling 100-day annualized standard deviation of equity price changes. Volatilities are expressed in percent rate of change.* Source from Bloomberg.L.P

For daily window: Annualized Historical Volatility is

$$\sigma = \sqrt{252} \times \sigma_{daily}$$
$$= \sqrt{252} \times r_{daily}$$

```
daily_window_annuallized_historical_volatility = sci_daily_returns*(252**0.5)
```

For montly window, we calculate a rolling 21-day annualized standard deviation of equity price changes.

$$\sigma = \sqrt{12} \times \sigma_{month}$$

```
monthly_window_annuallized_historical_volatility =
sci_daily_returns.rolling(21).std()[21::]*(12**0.5)
```

For annually window: we calculate a rolling 252-day annualized standard deviation of equity price changes. Annualized Historical Volatility is

$$\sigma = \sqrt{1} \times \sigma_{year}$$

```
annually_window_annuallized_historical_volatility =
sci_daily_returns.rolling(252).std()[252::]
```

a) Which has the greatest differences between high and low volatility?

```
print("The difference between maximum and minimum of daily window volatility
is {}."
      .format(float(daily_window_annuallized_historical_volatility.max())
              -float(daily_window_annuallized_historical_volatility.min())))

print("The difference between maximum and minimum of monthly window volatility
is {}."
      .format(float(monthly_window_annuallized_historical_volatility.max())
             -float(monthly_window_annuallized_historical_volatility.min())))

print("The difference between maximum and minimum of annually volatility is
{}"
      .format(float(annually_window_annuallized_historical_volatility.max())
             -float(annually_window_annuallized_historical_volatility.min())))
```

```
The difference between maximum and minimum of daily window volatility is
2.9681642880389942.
The difference between maximum and minimum of monthly window volatility is
0.13571049614208.
The difference between maximum and minimum of annually volatility is
0.023200555508148885
```

Therefore, **daily window has the greatest difference between highest and lowest.**

```
daily_window_annuallized_historical_volatility_series =
np.array(daily_window_annuallized_historical_volatility['daily.return'])
#Drop nan numbers
daily_window_annuallized_historical_volatility_series =
daily_window_annuallized_historical_volatility_series[~np.isnan(daily_window_a
nnuallized_historical_volatility_series)]
```

```
print("The mean of daily volatility is
{}.".format(float(daily_window_annuallized_historical_volatility.mean())))
print("The mean of monthly volatility is
{}.".format(float(monthly_window_annuallized_historical_volatility.mean())))
print("The mean of annually volatility is
{}.".format(float(annually_window_annuallized_historical_volatility.mean())))
print("-----------------------------------------------------")
print("The standard deviation of daily volatility is
{}.".format(float(daily_window_annuallized_historical_volatility.std())))
print("The standard deviation of monthly volatility is
{}.".format(float(monthly_window_annuallized_historical_volatility.std())))
print("The standard deviation of annually volatility is
{}.".format(float(annually_window_annuallized_historical_volatility.std())))
print("-----------------------------------------------------")
print("The kurtosis of daily volatility is
{}.".format(float(kurtosis(daily_window_annuallized_historical_volatility_seri
es))))
print("The kurtosis of monthly volatility is
{}.".format(float(kurtosis(monthly_window_annuallized_historical_volatility['d
aily.return']))))
print("The kurtosis of annually volatility is
{}.".format(float(kurtosis(annually_window_annuallized_historical_volatility['
daily.return']))))
```

```
The mean of daily volatility is 0.0043661289510826835.
The mean of monthly volatility is 0.04711454272326737.
The mean of annually volatility is 0.01447379120709604.
-----------------------------------------------------
The standard deviation of daily volatility is 0.24424626314336306.
The standard deviation of monthly volatility is 0.024197932394545107.
The standard deviation of annually volatility is 0.00530994786342295.
-----------------------------------------------------
The kurtosis of daily volatility is 5.1557074774501235.
The kurtosis of monthly volatility is 1.6079959125323846.
The kurtosis of annually volatility is 0.20665015045710033.
```

The monthly window generates highest mean, while daily window has the highest standard deviation and kurtosis.

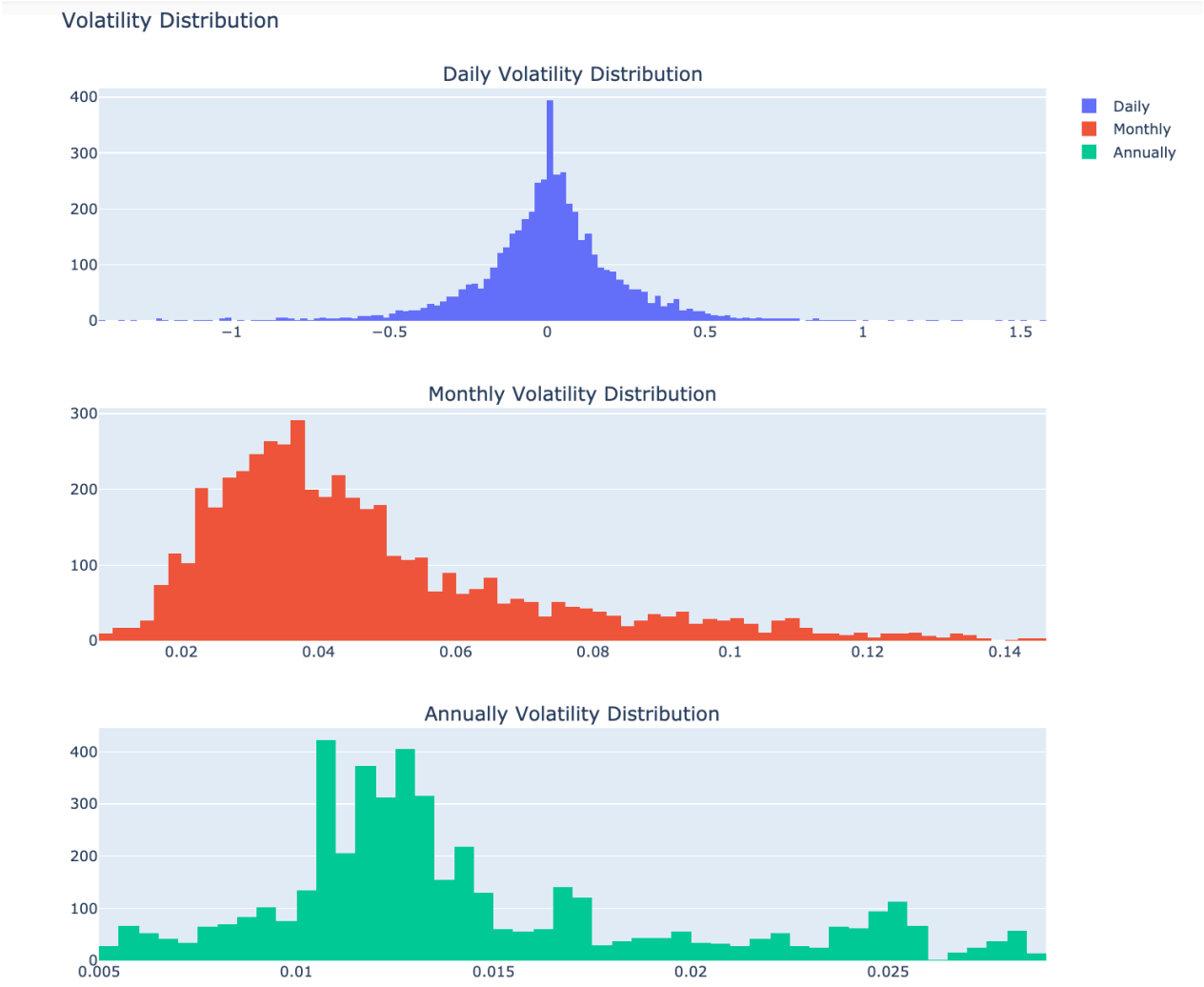
c) Draw the histograms of volatilities.

```
# Initialize figure with subplots
fig = make_subplots(rows=3, cols=1,
                    subplot_titles=("Daily Volatility Distribution", "Monthly
Volatility Distribution",
                                    "Annually Volatility Distribution"),
                    vertical_spacing=0.1)
```

```
fig.append_trace(go.Histogram(x=daily_window_annuallized_historical_volatility
['daily.return'],name = "Daily"),row = 1,col = 1)

fig.append_trace(go.Histogram(x=monthly_window_annuallized_historical_volatili
ty['daily.return'],name = "Monthly"),row = 2,col = 1)

fig.append_trace(go.Histogram(x=annually_window_annuallized_historical_volatil
ity['daily.return'],name = "Annually"),row = 3,col = 1)
# fig.update_yaxes(title_text="yaxis 4 title", row=2, col=2)
# Update title and height
fig.update_layout(title_text="Volatility Distribution", height=900)
fig.show()
```



d) What events correspond to these volatilities?

Correspondig events for daily volatilities are daily event, such as breaking news, trade war announcement, etc. Correspondig events for monthly volatilities are monthly or quaterly event, such as revenue, quanterly report announcement, etc. Correspondig events for annualy volatilities are annually event, such as GDP downturn, monetary policy , etc.

3. Calculate the exponentially weighted annualized volatilities using .06 and .02 as weights.

a) Plot these series on the same graph and attach it.

To calculate the EWAV, we need to prepare a few data series that will be needed.

```
weight = [0.06,0.02]
window = [17,50]
```

```
initial_var =
[squared_daily_returns_series[0:window[0]-1].std()**2,squared_daily_returns_se
ries[0:window[1]-1].std()**2]
initial_var
```

```
[3.232725779380975e-07, 1.473244755581626e-06]
```

```
var_result = []
var_result.append(initial_var[0])#place holder for result.
```

```
lamda_times_squared_return =
squared_daily_returns_series[window[0]::]*weight[0]
# len(lamda_times_squared_return)
```

```
def
calculate_next_var(weight,var_result=var_result,lamda_times_squared_return=lam
da_times_squared_return):
    #Select the previous sigma_square from the list
    var_last = var_result[len(var_result)-1]
    return_last = lamda_times_squared_return[len(var_result)-1]
    next_var = var_last*(1-weight)+return_last
    var_result.append(next_var)
    return var_result
```

```
for i in range(len(lamda_times_squared_return)):
    calculate_next_var(weight[0])
```

```
var_results = []
var_results.append((np.array(var_result)*252)**0.5)
```

```
#do it for the weight = 0.02, window = 50
var_result = []
var_result.append(initial_var[1])
lamda_times_squared_return =
squared_daily_returns_series[window[1]::]*weight[1]
```

```
def
calculate_next_var(weight,var_result=var_result,lamda_times_squared_return=lam
da_times_squared_return):
    #Select the previous sigma_square from the list
    var_last = var_result[len(var_result)-1]
    return_last = lamda_times_squared_return[len(var_result)-1]
    next_var = var_last*(1-weight)+return_last
    var_result.append(next_var)
    return var_result
```

```
for i in range(len(lamda_times_squared_return)):
    calculate_next_var(weight[1])
var_results.append((np.array(var_result)*252)**0.5)
```

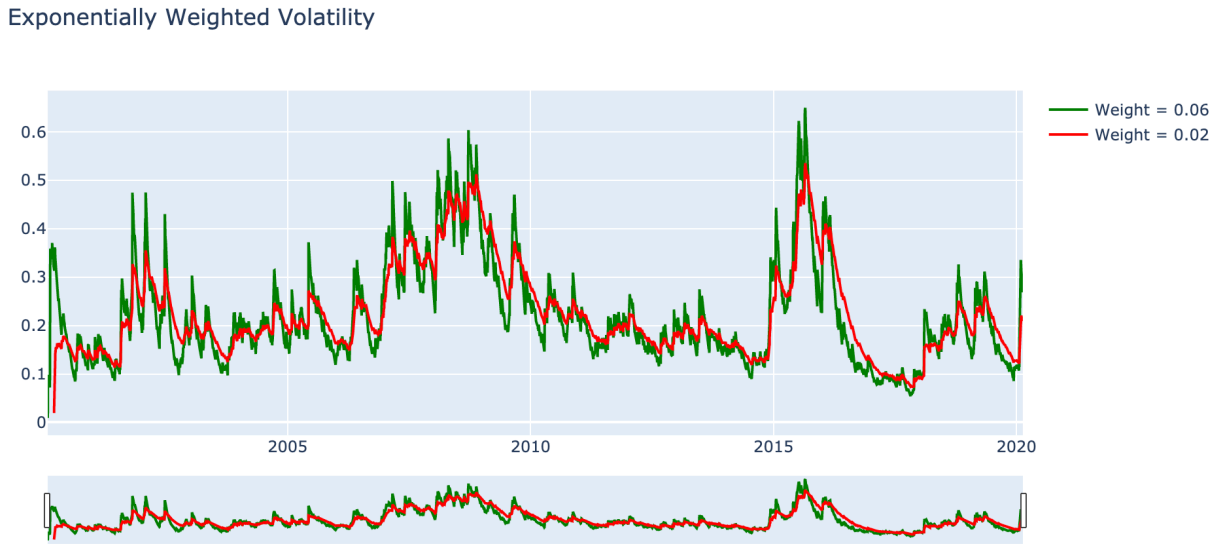
```
[array([0.00902578, 0.06790639, 0.09819363, ..., 0.27069478, 0.27706359,
        0.26862888]),
 array([0.01926805, 0.05966839, 0.05929392, ..., 0.20997316, 0.21409186,
        0.21194256])]
```

```
#Convert vol series to pandas data frame
# ewav_first = []
ewav_first = pd.DataFrame({
    'Date':sci_daily_returns.index[window[0]:],
    'Estimated Volatility (0.06)':var_results[0]
})
ewav_first.set_index('Date',inplace = True)
# ewav_first.head()
ewav_second = pd.DataFrame({
    'Date':sci_daily_returns.index[window[1]:],
    'Estimated Volatility (0.02)':var_results[1]
})
ewav_second.set_index('Date',inplace = True)
ewav = ewav_first.join(ewav_second)
```

```
fig = go.Figure()
fig.add_trace(go.Scatter(x=ewav.index, y=ewav['Estimated Volatility (0.06)'],
name="Weight = 0.06",
                         line_color='green'))

fig.add_trace(go.Scatter(x=ewav.index, y=ewav['Estimated Volatility (0.02)'],
name="Weight = 0.02",
                         line_color='red'))

fig.update_layout(title_text='Exponentially Weighted Volatility',
                  xaxis_rangeslider_visible=True)
fig.show()
```



b) What is the maximum volatility over this sample period and when does it occur?

```
print(ewav['Estimated Volatility (0.06)'].max())
print(ewav['Estimated Volatility (0.02)'].max())
```

```
0.6498773787462452
0.5356121683201505
```

```
ewav.idxmax()
```

```
Estimated Volatility (0.06)   2015-08-28
Estimated Volatility (0.02)   2015-08-28
dtype: datetime64[ns]
```

The maximum volatility since 2000 is 64% (estimated using weight=0.06) or 53% (estimated using weight=0.06). It occured on August 28th, 2015.

4. If a tax were put on stock market transactions, it would probably substantially reduce trading. What do you expect it to do to volatility of individual stocks and stock indices? Explain.

The volatility of each individual stock and the stock indices will increased substantially because the trading volume decreased substantially due to the increasing tax.