Historical Volatility & Risk Return Measures

Takao Oba

Assisted by quantpy

Assess the following Risk Measures: Sharpe ratio, Sortino ratio, Modigliani ratio (M2 ratio), Calmar ratio, Max Drawdown

In [1]:
```
# Importing dependencies

import datetime as dt
import pandas as pd
import numpy as np

from pandas_datareader import data as pdr
import plotly.offline as pyo
import plotly.graph_objects as go
from plotly.subplots import make_subplots

pyo.init_notebook_mode(connected = True)
pd.options.plotting.backend = 'plotly'
```

In [2]:
```
# Getting stock market data

end = dt.datetime.now()
start = dt.datetime(2015,1,1)

import yfinance as yfin
yfin.pdr_override()

# Let QQQ be the market index
df = pdr.get_data_yahoo(['QQQ','AAPL', 'MSFT', 'NFLX', 'GOOGL'], start, end)
Close = df.Close # indexing on Close
Close.head()
```

```
[*********************100%***********************]  5 of 5 completed
```

Out[2]:

	AAPL	GOOGL	MSFT	NFLX	QQQ
Date					
2015-01-02	27.332500	26.477501	46.759998	49.848572	102.940002

2015-01-05	26.562500	25.973000	46.330002	47.311428	101.430000
2015-01-06	26.565001	25.332001	45.650002	46.501431	100.070000
2015-01-07	26.937500	25.257500	46.230000	46.742859	101.360001
2015-01-08	27.972500	25.345501	47.590000	47.779999	103.300003

Determine log returns

In [3]:

```
log_returns = np.log(df.Close / df.Close.shift(1)).dropna()
log_returns
```

Out[3]:

	AAPL	GOOGL	MSFT	NFLX	QQQ
Date					
2015-01-05	-0.028576	-0.019238	-0.009238	-0.052238	-0.014777
2015-01-06	0.000094	-0.024989	-0.014786	-0.017269	-0.013499
2015-01-07	0.013925	-0.002945	0.012625	0.005178	0.012809
2015-01-08	0.037703	0.003478	0.028994	0.021946	0.018959
2015-01-09	0.001072	-0.012286	-0.008441	-0.015578	-0.006605
...	...	...	...	...	...
2022-12-23	-0.002802	0.016612	0.002265	-0.009414	0.002247
2022-12-27	-0.013976	-0.020836	-0.007442	-0.037267	-0.014239
2022-12-28	-0.031166	-0.015801	-0.010308	-0.025988	-0.013291
2022-12-29	0.027931	0.027858	0.027255	0.050151	0.024083
2022-12-30	0.002466	-0.002490	-0.004950	0.012833	-0.000601

2013 rows × 5 columns

Calculate the daily standard deviation of returns

We can see the volatility from this

In [4]:

```
daily_std = log_returns.std()
daily_std
```

Out[4]:

```
AAPL     0.018883
GOOGL    0.017780
```

```
MSFT     0.017743
NFLX     0.029264
QQQ      0.014343
dtype: float64
```

In [5]:
```
# above if for daily, compute annualized
annualized_vol = daily_std*np.sqrt(252)
annualized_vol*100 # times by 100 to get percentage
```

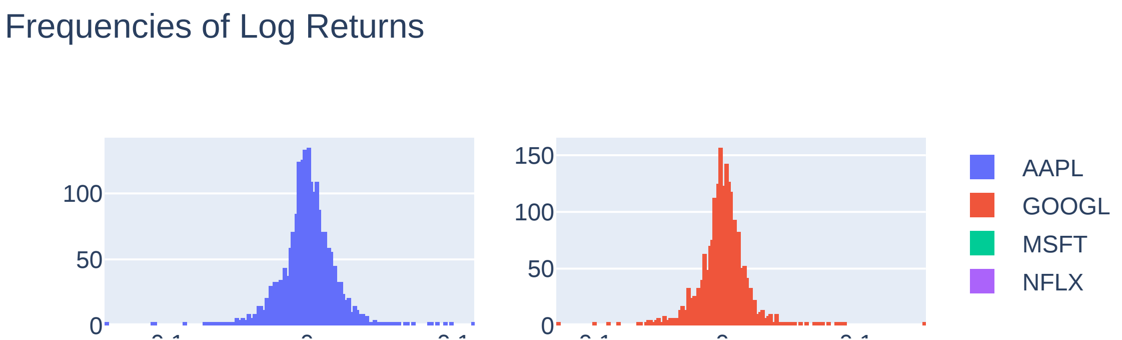
Out[5]:
```
AAPL     29.976389
GOOGL    28.225055
MSFT     28.166151
NFLX     46.454466
QQQ      22.768026
dtype: float64
```

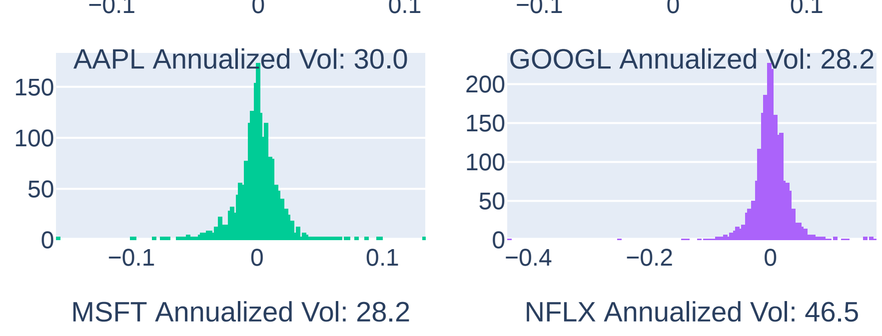
In [6]:
```
# Plotting the histogram of log returns with annualized volatility
fig = make_subplots(rows = 2, cols = 2)
trace0 = go.Histogram(x = log_returns['AAPL'], name = 'AAPL')
trace1 = go.Histogram(x = log_returns['GOOGL'], name = 'GOOGL')
trace2 = go.Histogram(x = log_returns['MSFT'], name = 'MSFT')
trace3 = go.Histogram(x = log_returns['NFLX'], name = 'NFLX')

fig.append_trace(trace0, 1,1)
fig.append_trace(trace1, 1,2)
fig.append_trace(trace2, 2,1)
fig.append_trace(trace3, 2,2)

fig.update_layout(autosize = False, width = 600, height = 400, title = 'Frequencies of Log Returns',
                  xaxis = dict(title = 'AAPL Annualized Vol: ' + str(np.round(annualized_vol['AAPL']*100,1))),
                  xaxis2 = dict(title = 'GOOGL Annualized Vol: ' + str(np.round(annualized_vol['GOOGL']*100,1))),
                  xaxis3 = dict(title = 'MSFT Annualized Vol: ' + str(np.round(annualized_vol['MSFT']*100,1))),
                  xaxis4 = dict(title = 'NFLX Annualized Vol: ' + str(np.round(annualized_vol['NFLX']*100,1))),
                  )

fig.show()
```





Trailing volatility over time

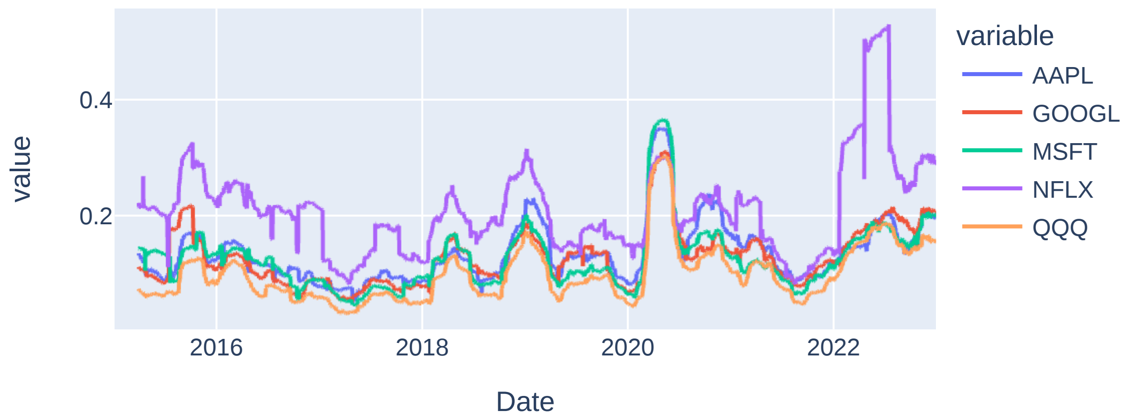
In [7]:
```
TRADING_DAYS = 60
volatility = log_returns.rolling(window = TRADING_DAYS).std()*np.sqrt(TRADING_DAYS)
volatility
```

Out[7]:

Date	AAPL	GOOGL	MSFT	NFLX	QQQ
2015-01-05	NaN	NaN	NaN	NaN	NaN
2015-01-06	NaN	NaN	NaN	NaN	NaN
2015-01-07	NaN	NaN	NaN	NaN	NaN
2015-01-08	NaN	NaN	NaN	NaN	NaN
2015-01-09	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...
2022-12-23	0.197299	0.208290	0.202486	0.289170	0.156609
2022-12-27	0.195495	0.208511	0.201652	0.291260	0.156322
2022-12-28	0.195146	0.206368	0.199124	0.292436	0.155015
2022-12-29	0.195538	0.206044	0.198180	0.296276	0.153728
2022-12-30	0.195548	0.206044	0.198219	0.295727	0.153728

2013 rows × 5 columns

In [8]:
```
volatility.plot().update_layout(autosize = False, width = 600, height = 300)
```

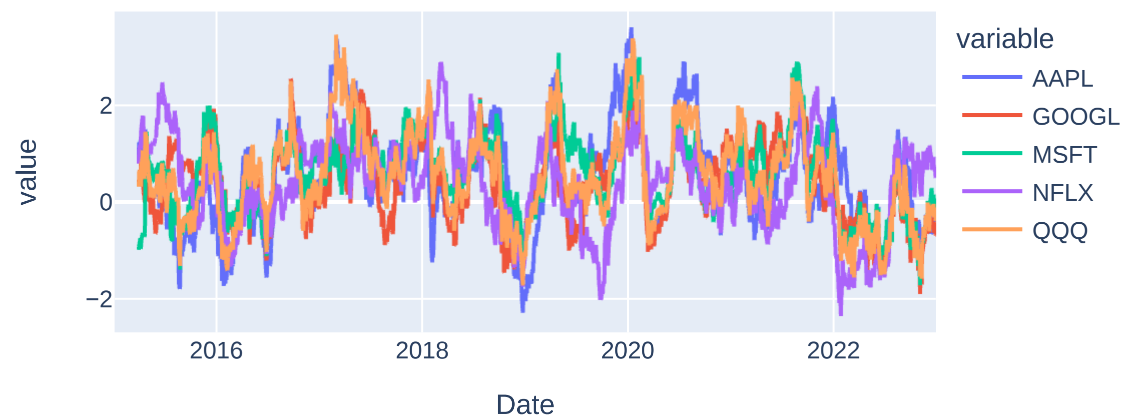


Sharpe Ratio

The Sharpe ratio which was introducted in 1966 by Nobel laureate William F. Sharpe is a measure for calculating risk-adjusted return. The Sharpe ratio is the average return earned in excess of the risk-free rate per unit of volatility. It is a risk-adjusted return measure that divides the excess return of an investment over the risk-free rate by the standard deviation of returns. It is used to evaluate the performance of an investment compared to a risk-free asset, such as a US Treasury bond.

In [9]:
```
Rf= 0.01/252 # a year is 252 trading days
sharpe_ratio = (log_returns.rolling(window = TRADING_DAYS).mean() - Rf)*TRADING_DAYS/volatility # mutliple by trading days becau
```

In [10]:
```
# sharpe ratio over time
sharpe_ratio.plot().update_layout(autosize = False, width = 600, height = 300)
```

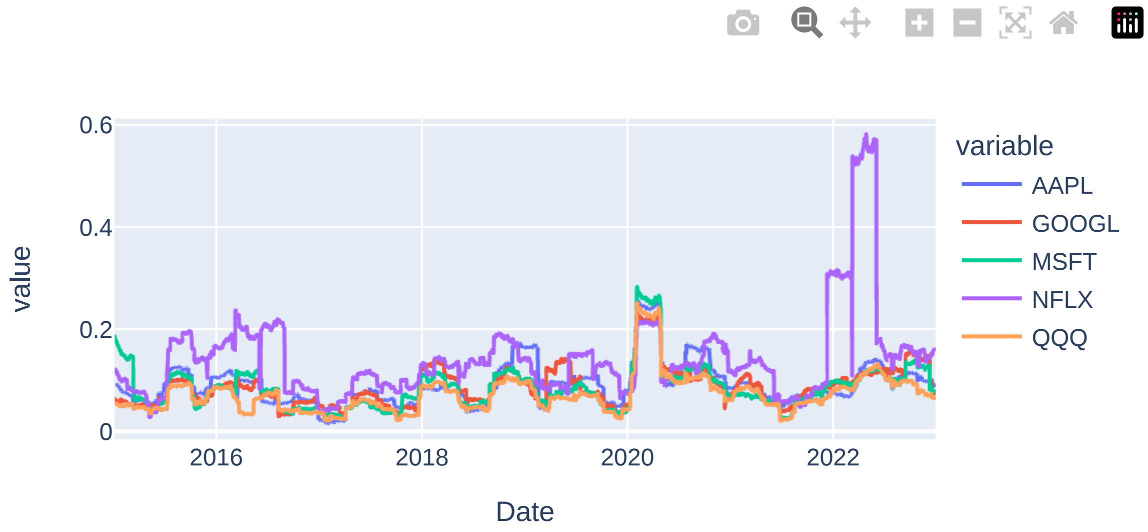


Sortino Ratio

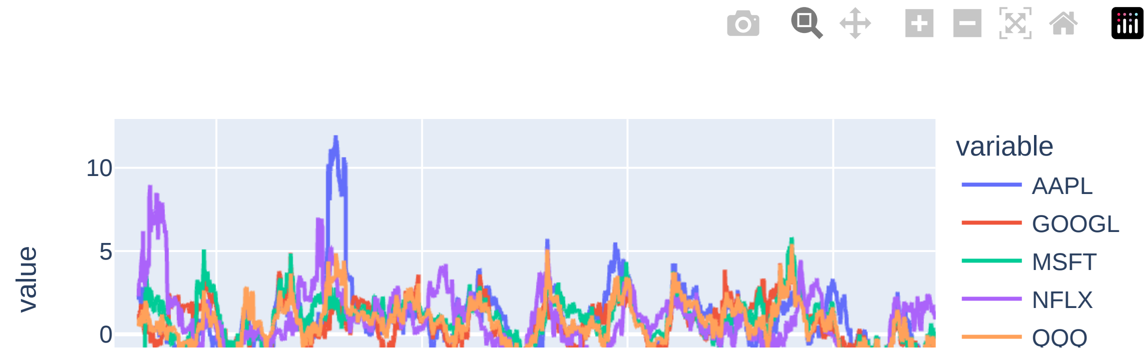
The Sortino ratio is very similar to the Sharpe ratio, the only difference being that where the Sharpe ratio uses all the observations for calculating the standard deviation for the Sortino ratio only considers the harmful variance. This makes it a useful measure for investors who are more concerned with minimizing potential losses.

In [11]:
```
sortino_vol = log_returns[log_returns < 0].rolling(window = TRADING_DAYS, center = True, min_periods = 10).std()*np.sqrt(TRADIN(
sortino_ratio = (log_returns.rolling(window = TRADING_DAYS).mean() - Rf)*TRADING_DAYS/sortino_vol
```

In [12]:
```
# slightly different than the volatility plot
sortino_vol.plot().update_layout(autosize = False, width = 600, height = 300)
```



In [13]:
```
sortino_ratio.plot().update_layout(autosize = False, width = 600, height = 300)
```



Modigliani Ratio (M2 Ratio)

The Modigliani ratio measures the returns of the portfolio, adjusted for the risk of the portfolio relative to that of some benchmark. It divides the portfolio's alpha by its standard deviation of returns. A high M2 ratio indicates that the portfolio has a higher risk-adjusted return.

In [15]:
```
m2_ratio = pd.DataFrame()

benchmark_vol = volatility['QQQ']

for c in log_returns.columns:
    if c != 'QQQ': # we don't want to compare with benchmark
        m2_ratio[c] = (sharpe_ratio[c] * benchmark_vol/TRADING_DAYS + Rf)*TRADING_DAYS
```

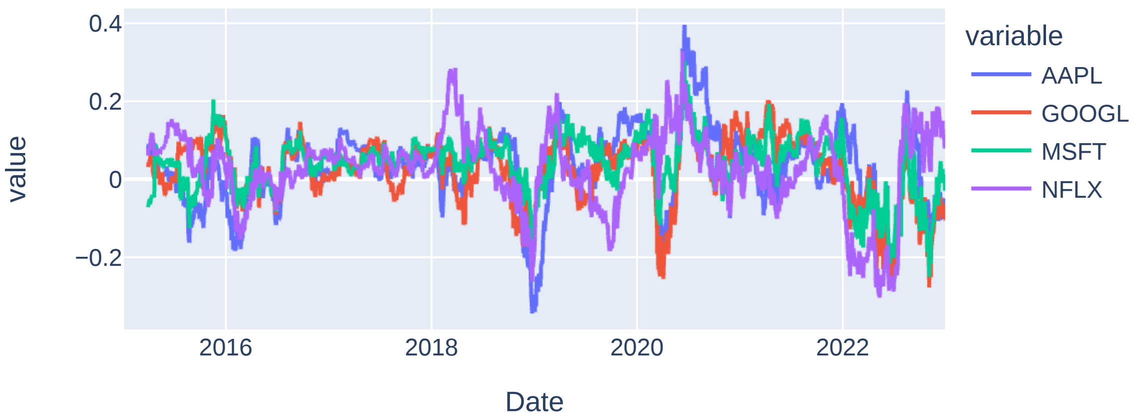
In [16]:
```
m2_ratio
```

Out[16]:

Date	AAPL	GOOGL	MSFT	NFLX
2015-01-05	NaN	NaN	NaN	NaN
2015-01-06	NaN	NaN	NaN	NaN
2015-01-07	NaN	NaN	NaN	NaN
2015-01-08	NaN	NaN	NaN	NaN
2015-01-09	NaN	NaN	NaN	NaN
...	...	...	...	...
2022-12-23	-0.060995	-0.065435	0.004535	0.113421
2022-12-27	-0.048249	-0.067114	0.013933	0.102067
2022-12-28	-0.096733	-0.102239	-0.019818	0.079016
2022-12-29	-0.093645	-0.103102	-0.024391	0.099740
2022-12-30	-0.093314	-0.103417	-0.029221	0.115323

2013 rows × 4 columns

In [17]:
```python
# This is with respect to market volatility
m2_ratio.plot().update_layout(autosize = False, width = 600, height = 300)
```



Max Drawdown

Max drawndown quantifies the steepest decline from peak to trough observed for an investment. This is useful for a number of reasons, mainly the fact that it doesn't rely on the underlying returns being normally distributed. It is often used to evaluate the risk of an investment and can help investors understand the potential downside of an investment.

In [18]:
```python
# function for max drawdown

def max_drawdown(returns):
    cumulative_returns = (1+returns).cumprod() # use simple return
    peak = cumulative_returns.expanding(min_periods = 1).max() # find absolute max of cumulative returns
    drawdown = (cumulative_returns/peak) - 1
    return drawdown.min() # when this is min, max of drawdown

returns = df.Close.pct_change().dropna()
max_drawdowns = returns.apply(max_drawdown, axis = 0)
print(max_drawdowns*100)
```

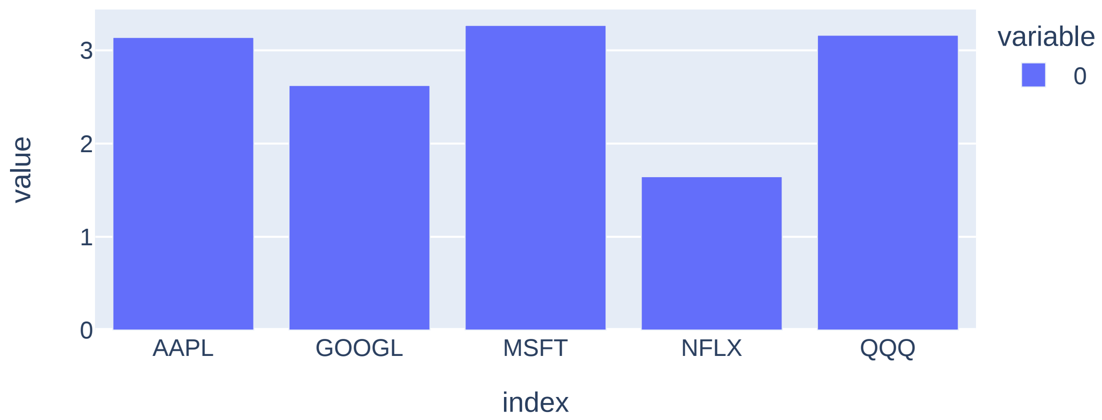
```
AAPL    -38.729695
GOOGL   -44.320051
MSFT    -37.556466
NFLX    -75.947318
QQQ     -35.617215
dtype: float64
```

Calmar Ratio

Calmar Ratio uses max drawdown (computed above) in the denominator as opposed to standard deviation

In [20]:
```
calmars = np.exp(log_returns.mean()*252)/abs(max_drawdowns)
calmars.plot.bar().update_layout(autosize = False, width = 600, height = 300)

# returns vs risk metric utilizing max drawdowns
```



In conclusion, I have assessed and exmamined various measures of risk-adjusted returns. I initially calculated the annualized volatility and computed and plotted the Sharpe ratio, Sortino ratio, Modigliani ratio, and the Calmar ratio utilizing max drawdown.