Historical Volatility & Risk-Return Measures

In this notebook we compute and track historical volatility over time.

In [1]:
```
import datetime as dt
import pandas as pd
import numpy as np

import yfinance as yf
import plotly.offline as pyo
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import plotly.io as pio

pio.templates.default = "ggplot2"

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

import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

Get stock data with yfinance

In [2]:
```
stocks = ['TSLA', 'SMCI', 'GOOGL', 'NVDA', 'AAPL', 'MSFT', 'AMZN', 'META'

df = yf.download(stocks, period="5y")
Close = df.Close
Close.tail()
```

```
[*********************100%***********************]  10 of 10 completed
```

Out[2]:

Ticker	AAPL	AMZN	GOOGL	META	MSFT	NFLX	
Date							
2024-11-18	228.020004	201.699997	175.300003	554.400024	415.760010	847.049988	140.
2024-11-19	228.279999	204.610001	178.119995	561.090027	417.790009	871.320007	147.
2024-11-20	229.000000	202.880005	175.979996	565.520020	415.489990	883.849976	145.
2024-11-21	228.520004	198.380005	167.630005	563.090027	412.869995	897.479980	146.
2024-11-22	229.869995	197.119995	164.759995	559.140015	417.000000	897.789978	141.

Compute log returns

In [3]:
```
log_returns = np.log(df.Close/df.Close.shift(1)).dropna()
log_returns.tail()
```

Out[3]:

Ticker	AAPL	AMZN	GOOGL	META	MSFT	NFLX	NVDA	
Date								
2024-11-18	0.013333	-0.004502	0.016160	0.000577	0.001830	0.027638	-0.012973	-0.0
2024-11-19	0.001140	0.014324	0.015959	0.011995	0.004871	0.028250	0.047787	0.0
2024-11-20	0.003149	-0.008491	-0.012087	0.007864	-0.005520	0.014278	-0.007648	-0.0
2024-11-21	-0.002098	-0.022430	-0.048611	-0.004306	-0.006326	0.015303	0.005332	-0.0
2024-11-22	0.005890	-0.006372	-0.017269	-0.007040	0.009953	0.000345	-0.032710	0.0

Calculate daily standard deviation of returns

In [4]:
```
daily_std = log_returns.std()
daily_std
```

Out[4]:
```
Ticker
AAPL     0.017198
AMZN     0.022261
GOOGL    0.019275
META     0.028455
MSFT     0.016527
NFLX     0.029660
NVDA     0.032698
PLTR     0.044420
SMCI     0.047998
TSLA     0.037553
dtype: float64
```

In [5]:
```
annualized_std = daily_std * np.sqrt(252)
annualized_std
```

Out[5]:
```
Ticker
AAPL     0.273007
AMZN     0.353390
GOOGL    0.305987
META     0.451712
MSFT     0.262352
NFLX     0.470836
NVDA     0.519059
PLTR     0.705153
SMCI     0.761946
TSLA     0.596137
dtype: float64
```

Plot histogram of log returns with annualized volatility

In [6]:

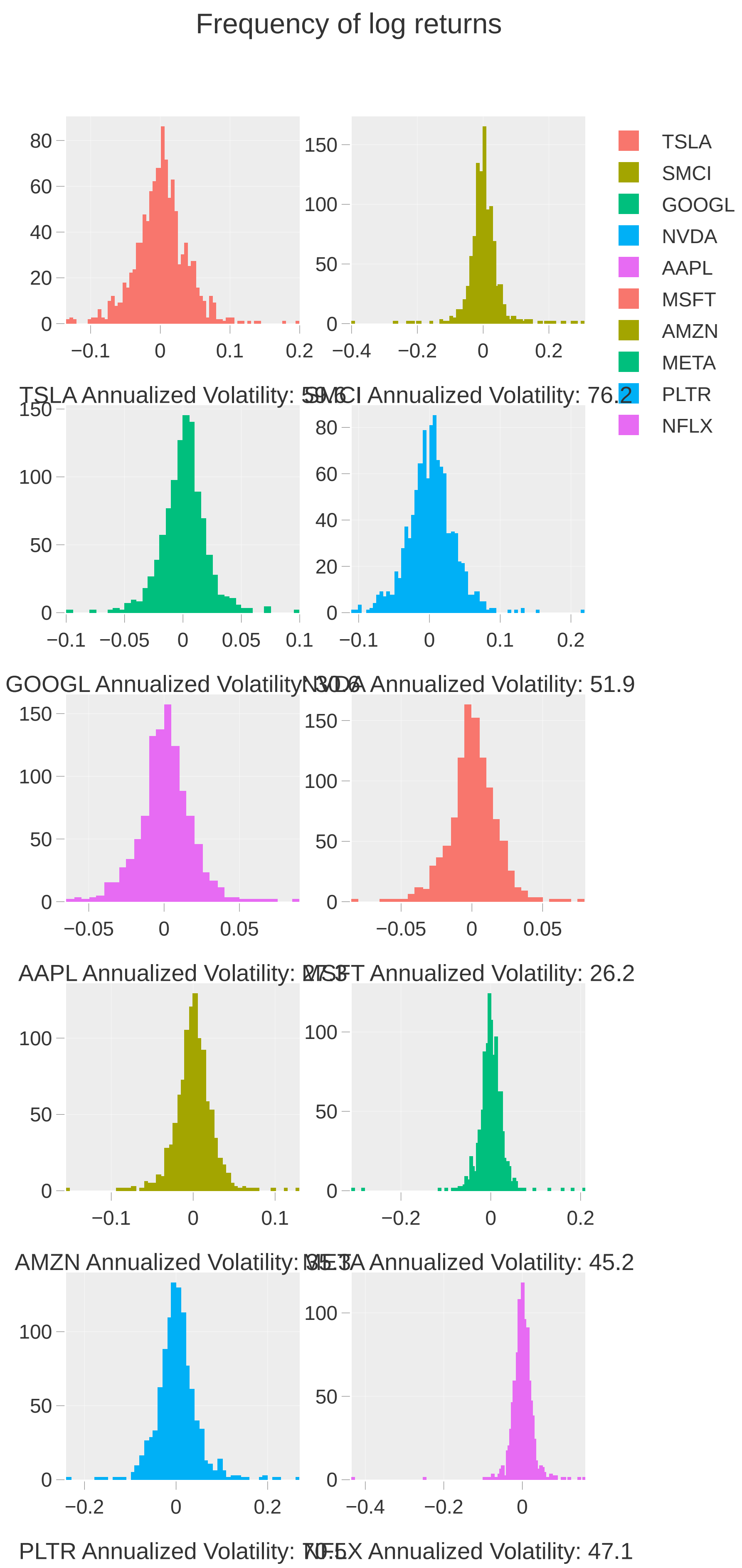
```python
fig = make_subplots(rows=5, cols=2)  # Create a 5x2 grid for 10 charts

# Create the histograms for each ticker
trace0 = go.Histogram(x=log_returns['TSLA'], name='TSLA')
trace1 = go.Histogram(x=log_returns['SMCI'], name='SMCI')
trace2 = go.Histogram(x=log_returns['GOOGL'], name='GOOGL')
trace3 = go.Histogram(x=log_returns['NVDA'], name='NVDA')
trace4 = go.Histogram(x=log_returns['AAPL'], name='AAPL')
trace5 = go.Histogram(x=log_returns['MSFT'], name='MSFT')
trace6 = go.Histogram(x=log_returns['AMZN'], name='AMZN')
trace7 = go.Histogram(x=log_returns['META'], name='META')
trace8 = go.Histogram(x=log_returns['PLTR'], name='PLTR')
trace9 = go.Histogram(x=log_returns['NFLX'], name='NFLX')

# Add the traces to the respective subplots
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.append_trace(trace4, 3, 1)
fig.append_trace(trace5, 3, 2)
fig.append_trace(trace6, 4, 1)
fig.append_trace(trace7, 4, 2)
fig.append_trace(trace8, 5, 1)
fig.append_trace(trace9, 5, 2)

# Update layout with axis titles for each chart
fig.update_layout(
    autosize=False,
    width=500,
    height=1000,  # Adjusted height to fit 10 charts
    title='Frequency of log returns',
    xaxis=dict(title='TSLA Annualized Volatility: ' + str(np.round(annual
    xaxis2=dict(title='SMCI Annualized Volatility: ' + str(np.round(annua
    xaxis3=dict(title='GOOGL Annualized Volatility: ' + str(np.round(annu
    xaxis4=dict(title='NVDA Annualized Volatility: ' + str(np.round(annua
    xaxis5=dict(title='AAPL Annualized Volatility: ' + str(np.round(annua
    xaxis6=dict(title='MSFT Annualized Volatility: ' + str(np.round(annua
    xaxis7=dict(title='AMZN Annualized Volatility: ' + str(np.round(annua
    xaxis8=dict(title='META Annualized Volatility: ' + str(np.round(annua
    xaxis9=dict(title='PLTR Annualized Volatility: ' + str(np.round(annua
    xaxis10=dict(title='NFLX Annualized Volatility: ' + str(np.round(annu
)

fig.show()
```

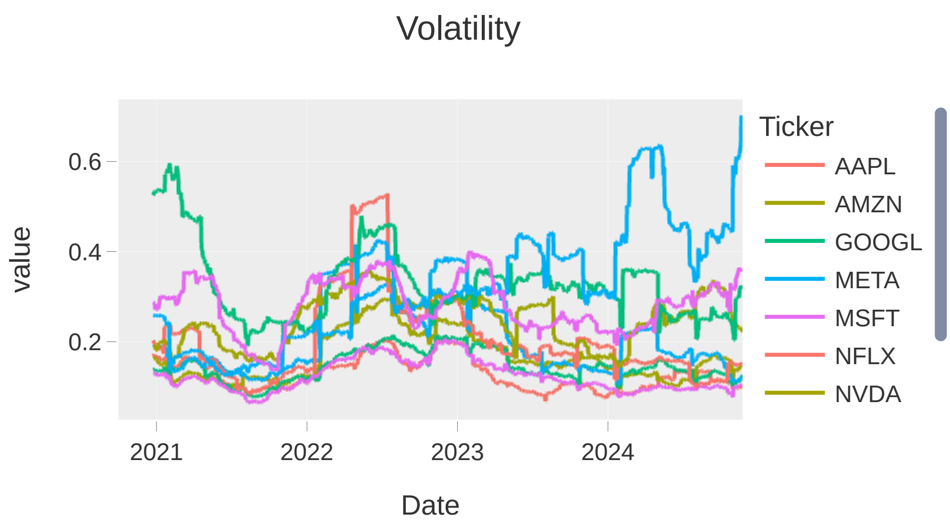

Frequency of log returns
TSLA
SMCI
GOOGL
NVDA
AAPL
MSFT
AMZN
META
PLTR
NFLX
TSLA Annualized Volatility:
SMCI Annualized Volatility: 76.2
GOOGL Annualized Volatility
NVDA Annualized Volatility: 51.9
AAPL Annualized Volatility:
MSFT Annualized Volatility: 26.2
AMZN Annualized Volatility:
META Annualized Volatility: 45.2
PLTR Annualized Volatility:
NFLX Annualized Volatility: 47.1

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

Out[7]:

Ticker	AAPL	AMZN	GOOGL	META	MSFT	NFLX	NVDA	PLT
Date								
2024-11-18	0.099739	0.144411	0.114084	0.119641	0.103386	0.150852	0.227258	0.32410
2024-11-19	0.099734	0.144481	0.115049	0.119303	0.103184	0.152863	0.230578	0.32186
2024-11-20	0.099714	0.143983	0.115371	0.119369	0.103338	0.153074	0.230474	0.32265
2024-11-21	0.099497	0.145273	0.125089	0.119360	0.103229	0.151942	0.229266	0.32238
2024-11-22	0.098608	0.145442	0.126154	0.119651	0.103528	0.151777	0.221429	0.32425

In [8]:
```
volatility.plot().update_layout(autosize = False, width=500, height=300,
```



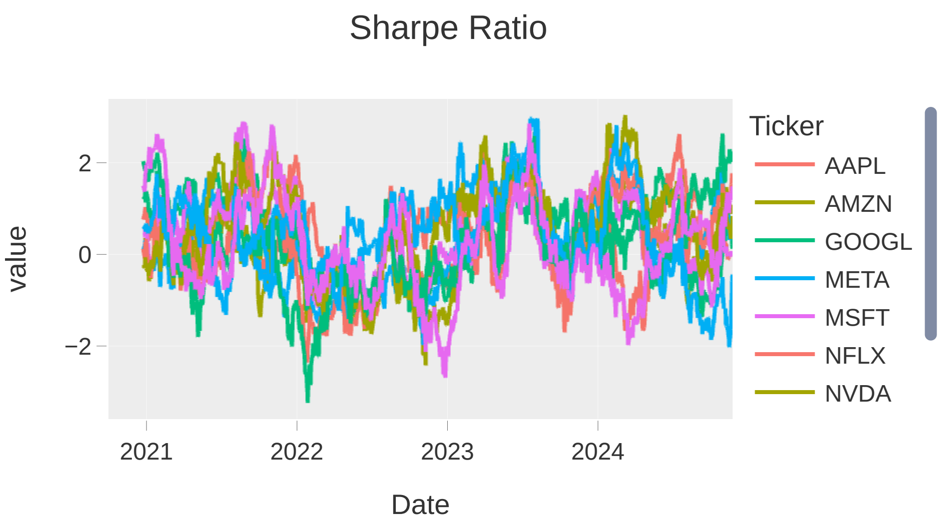
Sharpe ratio

The Sharpe ratio which was introduced 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.

$$\text{Sharpe Ratio} = \frac{R_p - R_f}{\sigma_p}$$

In [9]:
```
Rf = 0.01/255
sharpe_ratio = (log_returns.rolling(window=TRADING_DAYS).mean() - Rf)*TRA
```

In [10]:
```
sharpe_ratio.plot().update_layout(autosize = False, width=500, height=300
```



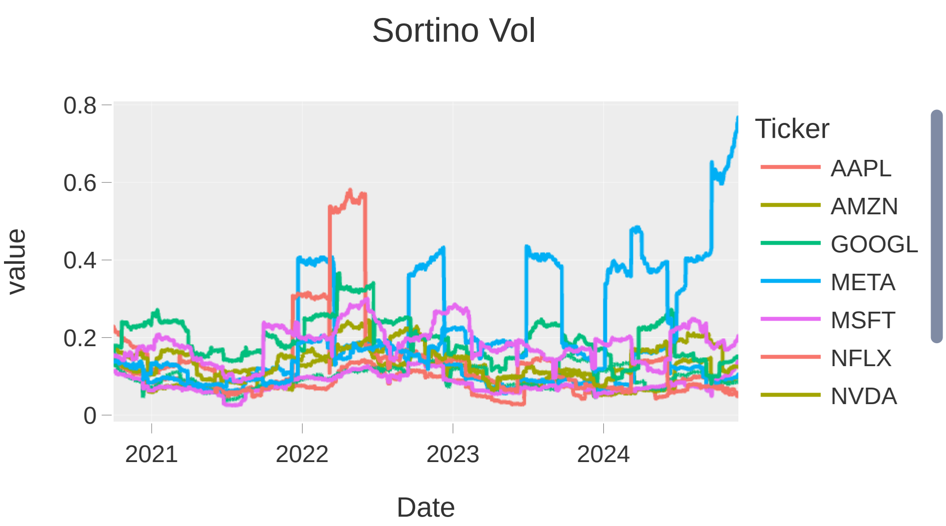
Sortino Ratio

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 the Sortino ratio only considers the harmful variance.

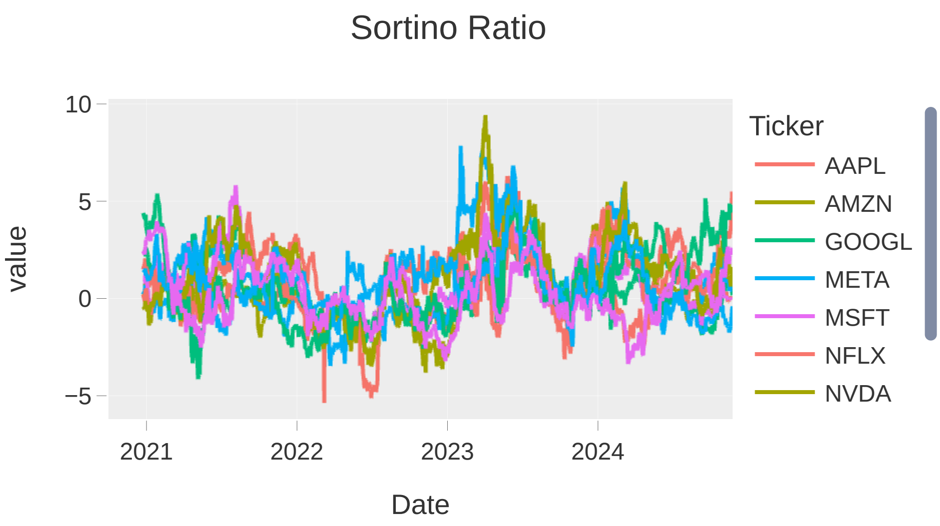
$$\text{Sortino Ratio} = \frac{R_p - R_f}{\sigma_d}$$

In [11]:
```
sortino_vol = log_returns[log_returns<0].rolling(window=TRADING_DAYS, cen
sortino_ratio = (log_returns.rolling(window=TRADING_DAYS).mean() - Rf)*TR
```

In [12]:
```
sortino_vol.plot().update_layout(autosize = False, width=500, height=300,
```



In [13]:
```
sortino_ratio.plot().update_layout(autosize = False, width=500, height=30
```



Modigliani ratio (M2 ratio)

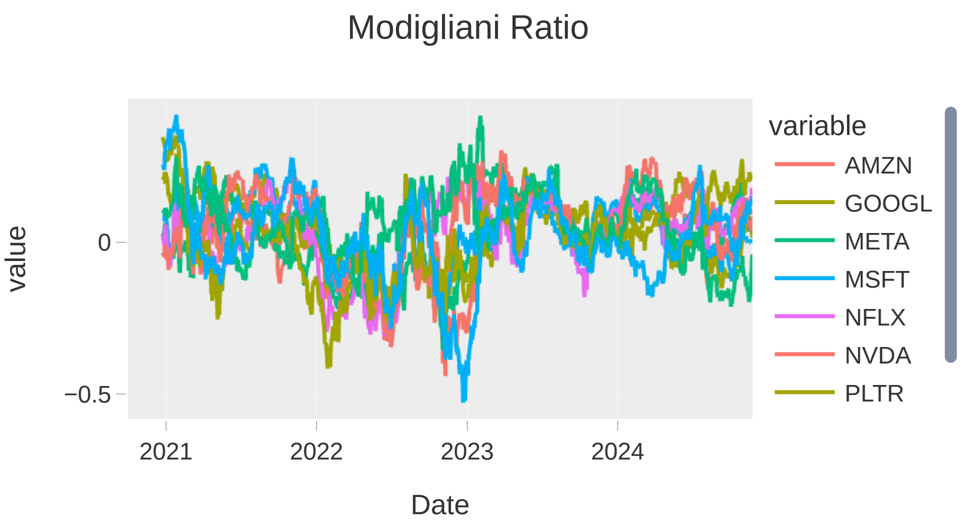
The Modigliani ratio measures the returns of the portfolio, adjusted for the risk of the portfolio relative to that of some benchmark.

$$M_A^2 = r_f + \left(\frac{R_A - r_f}{\sigma_A}\right)\sigma_M$$

In [14]:
```
m2_ratio = pd.DataFrame()

benchmark_vol = volatility['AAPL']
for c in log_returns.columns:
    if c != 'AAPL':
        m2_ratio[c] = (sharpe_ratio[c]*benchmark_vol/TRADING_DAYS + Rf)*T
```

In [15]:
```
m2_ratio.plot().update_layout(autosize = False, width=500, height=300, ti
```



Max Drawdown

Max drawdown 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.

In [16]:
```python
def max_drawdown(returns):
    cumulative_returns = (returns+1).cumprod()
    peak = cumulative_returns.expanding(min_periods=1).max()
    drawdown = (cumulative_returns/peak)-1
    return drawdown.min()


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

Out[16]:
```
Ticker
AAPL    -31.427266
AMZN    -56.145263
GOOGL   -44.320051
META    -76.736092
MSFT    -37.556466
NFLX    -75.947318
NVDA    -66.362055
PLTR    -84.615385
SMCI    -84.840960
TSLA    -73.632217
dtype: float64
```

Calmar Ratio

Calmar ratio uses max drawdown in the denominator as opposed to standard deviation.

$$Calmar\ Ratio = \frac{R - R_f}{Max\ Drawdown}$$

In [17]:
```python
calmars = np.exp(log_returns.mean()*255)/abs(max_drawdowns)
calmars.plot.bar().update_layout(autosize = False, width=500, height=300,
```