In [1]:
```
# Using the yfinance API to collect stock market data
#pip install yfinance
```

In [2]:
```
import pandas as pd
import yfinance as yf
from datetime import date, timedelta
```

In [3]:
```
endDate = "2024-07-31"
startDate = (date.today() - timedelta(days=365)).strftime("%Y-%m-%d")
```

In [4]:
```
tickers = ['AAPL', 'META', 'NVDA', 'GOOGL', 'AMZN']
```

In [5]:
```
data = yf.download(tickers, start = startDate, end = endDate, progress
```

In [6]:
```
data = data.reset_index()
dataMelted = data.melt(id_vars=['Date'], var_name=['Attribute', 'Ticke
dataPivoted = dataMelted.pivot_table(index=['Date', 'Ticker'], columns
stockData = dataPivoted.reset_index()

print(stockData.tail())
```

```
Attribute       Date Ticker   Adj Close       Close        High
Low  \
1245      2024-07-30   AAPL  218.800003  218.800003  220.330002  21
6.119995
1246      2024-07-30   AMZN  181.710007  181.710007  185.860001  17
9.380005
1247      2024-07-30  GOOGL  170.289993  170.289993  171.229996  16
8.440002
1248      2024-07-30   META  463.190002  463.190002  472.730011  45
6.700012
1249      2024-07-30   NVDA  103.730003  103.730003  111.989998  10
2.540001

Attribute        Open       Volume
1245       219.190002   41643800.0
1246       184.720001   39508600.0
1247       170.240005   18959700.0
1248       467.000000   11390400.0
1249       111.519997  486833300.0
```

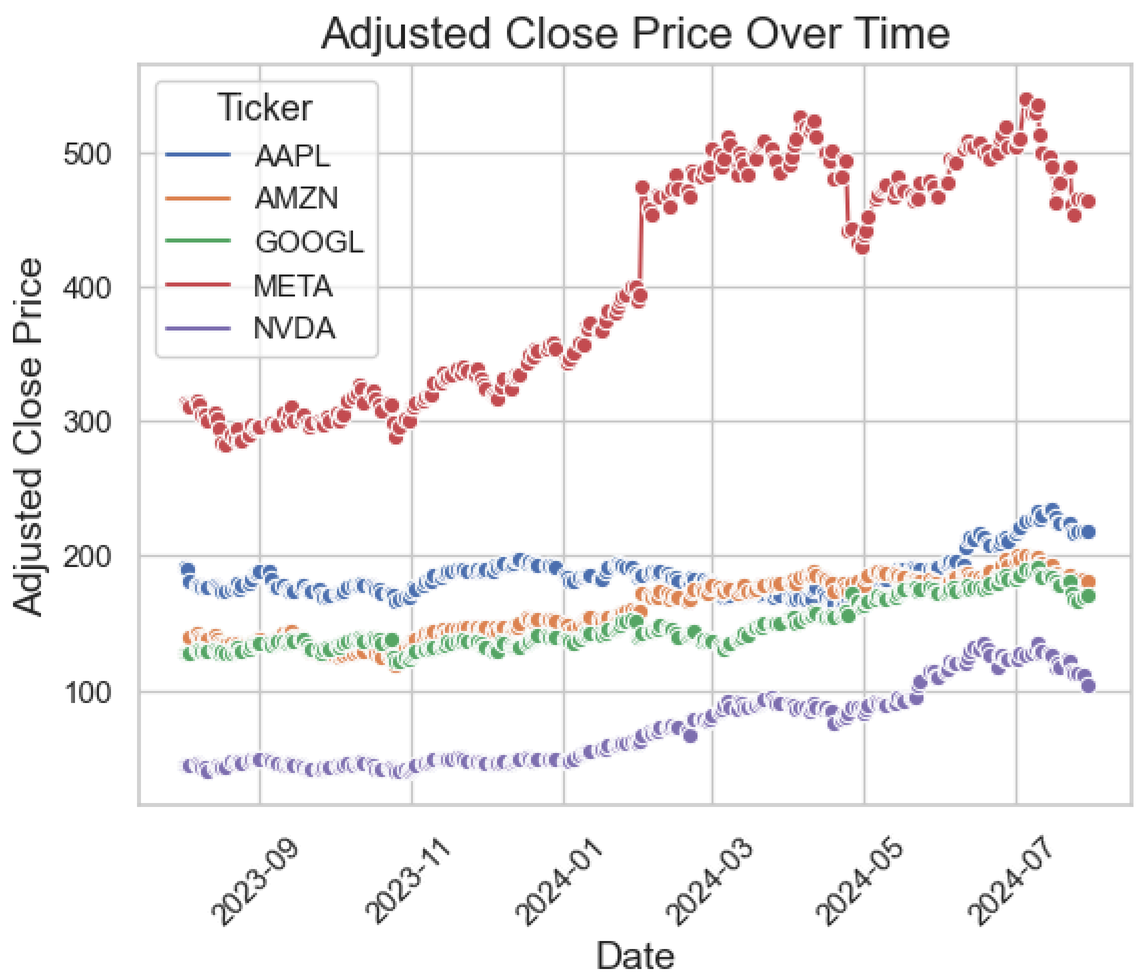
In [7]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

In [8]:
```
stockData['Date'] = pd.to_datetime(stockData['Date'])
stockData.set_index('Date', inplace = True)
stockData.reset_index(inplace = True)
plt.figure(figsize = (14,7))
sns.set(style='whitegrid')
```

```
<Figure size 1400x700 with 0 Axes>
```

In [9]:
```
sns.lineplot(data=stockData, x='Date', y='Adj Close', hue='Ticker', ma

plt.title('Adjusted Close Price Over Time', fontsize=16)
plt.xlabel('Date', fontsize=14)
plt.ylabel('Adjusted Close Price', fontsize=14)
plt.legend(title='Ticker', title_fontsize='13', fontsize='11')
plt.grid(True)

plt.xticks(rotation=45)

plt.show()
```

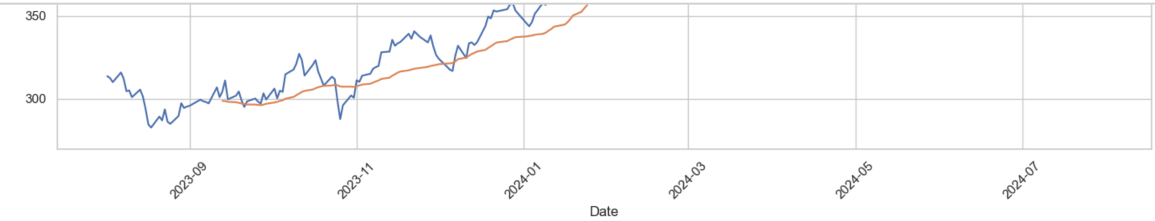
```
/opt/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:111
9: FutureWarning: use_inf_as_na option is deprecated and will be rem
oved in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:111
9: FutureWarning: use_inf_as_na option is deprecated and will be rem
oved in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

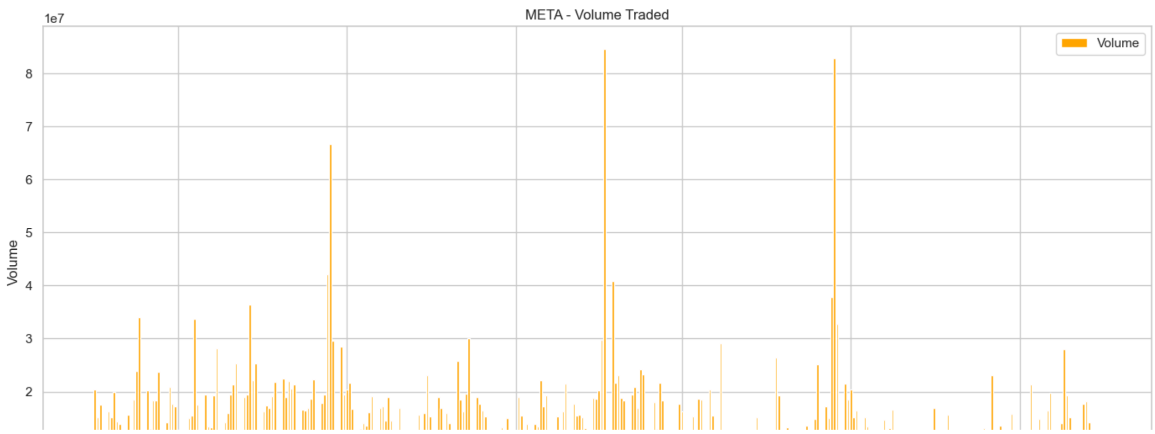


The above graph displays the adjusted closed price of the 5 stocks we are analysing (AAPL, AMZN, GOOGL, META, NVIDIA) over time from July 2023 to July 2024. META has the highest adjusted closed price followed by AAPL, AMZN, GOOGL and finally NVDA.

Meta has noticable upward trends but relatively volatile, whereas AAPL, AMZN, GOOGL, and NVDA exhibit more stability with less price fluctuation over the year.

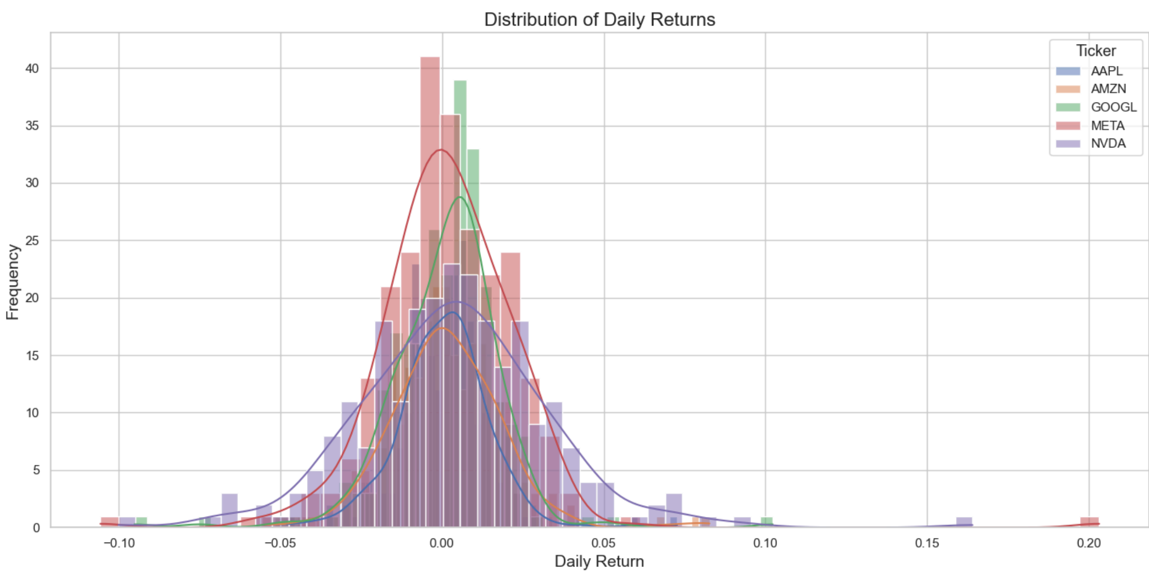
In [10]:
```
shortWindow = 30
longWindow = 300
stockData.set_index('Date', inplace = True)
uniqueTickers = stockData['Ticker'].unique()

for ticker in uniqueTickers:
    tickerData = stockData[stockData['Ticker'] == ticker].copy()
    tickerData['30_MA'] = tickerData['Adj Close'].rolling(window=short
    tickerData['300_MA'] = tickerData['Adj Close'].rolling(window=long

    plt.figure(figsize=(14, 7))
    plt.plot(tickerData.index, tickerData['Adj Close'], label='Adj Clo
    plt.plot(tickerData.index, tickerData['30_MA'], label='30-Day MA')
    plt.plot(tickerData.index, tickerData['300_MA'], label='300-Day MA
    plt.title(f'{ticker} - Adjusted Close and Moving Averages')
    plt.xlabel('Date')
    plt.ylabel('Price')
    plt.legend()
    plt.grid(True)
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()

    plt.figure(figsize=(14, 7))
    plt.bar(tickerData.index, tickerData['Volume'], label='Volume', co
    plt.title(f'{ticker} - Volume Traded')
    plt.xlabel('Date')
    plt.ylabel('Volume')
    plt.legend()
    plt.grid(True)
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()
```





In [11]:
```
stockData['Daily Return'] = stockData.groupby('Ticker')['Adj Close'].p
plt.figure(figsize=(14, 7))
sns.set(style='whitegrid')

for ticker in uniqueTickers:
    tickerData = stockData[stockData['Ticker'] == ticker]
    sns.histplot(tickerData['Daily Return'].dropna(), bins=50, kde=Tru

plt.title('Distribution of Daily Returns', fontsize=16)
plt.xlabel('Daily Return', fontsize=14)
plt.ylabel('Frequency', fontsize=14)
plt.legend(title='Ticker', title_fontsize='13', fontsize='11')
plt.grid(True)
plt.tight_layout()
plt.show()
```

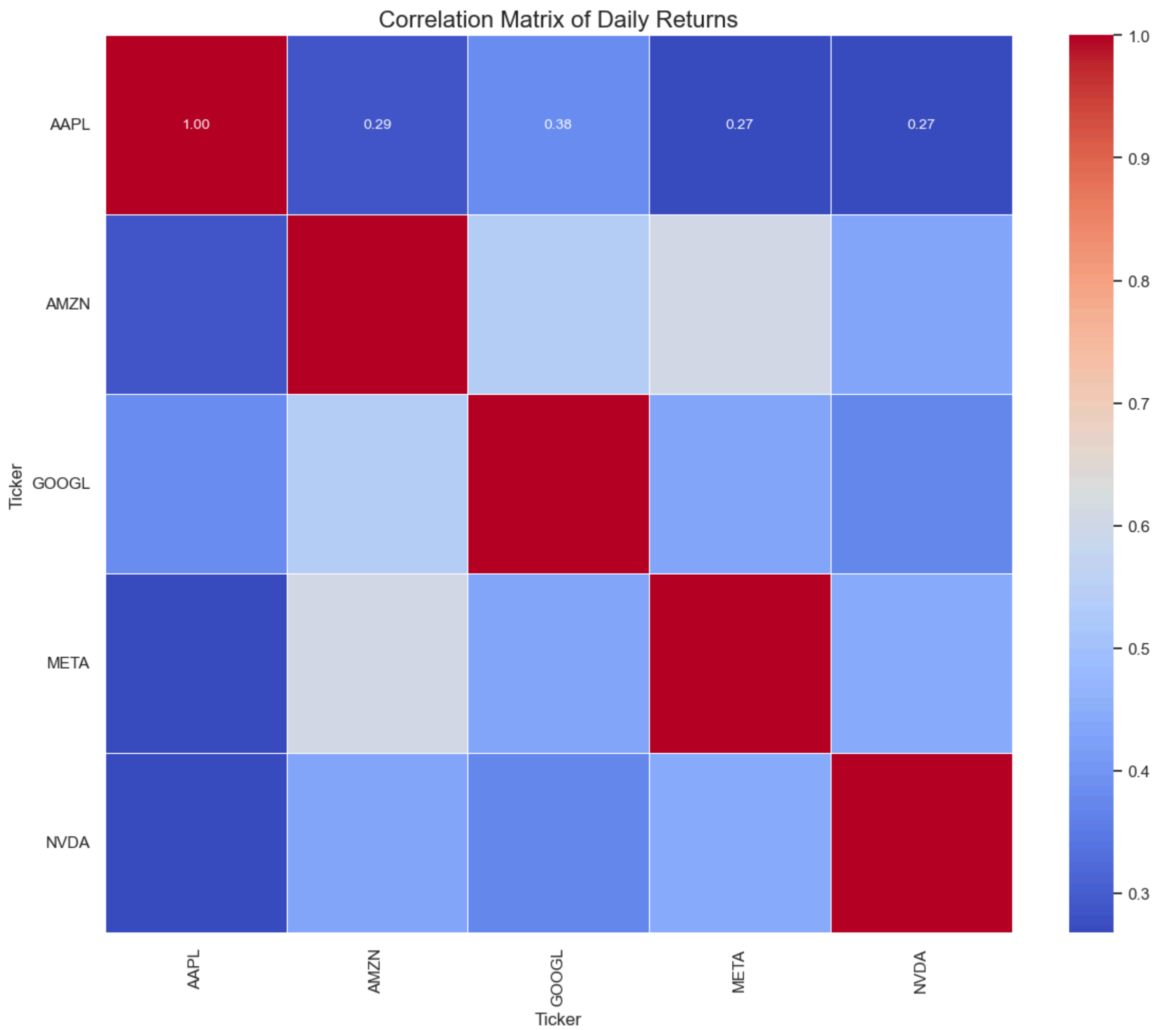
```
/opt/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:111
9: FutureWarning: use_inf_as_na option is deprecated and will be rem
oved in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:111
9: FutureWarning: use_inf_as_na option is deprecated and will be rem
oved in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:111
9: FutureWarning: use_inf_as_na option is deprecated and will be rem
oved in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:111
9: FutureWarning: use_inf_as_na option is deprecated and will be rem
oved in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:111
9: FutureWarning: use_inf_as_na option is deprecated and will be rem
oved in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



The distributions are approximately normal, centred around zero, which indicates that most daily returns are close to the average return. There are tails on both sides which reflects significant gains or losses.

NVDIA and META shows a wider distribution, which suggests a higher volatility compared to the other stocks.

In [12]:
```
dailyReturns = stockData.pivot_table(index='Date', columns='Ticker', v

correlationMatrix = dailyReturns.corr()

plt.figure(figsize=(12, 10))
sns.set(style='whitegrid')

sns.heatmap(correlationMatrix, annot=True, cmap='coolwarm', linewidths
plt.title('Correlation Matrix of Daily Returns', fontsize=16)
plt.xticks(rotation=90)
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()
```



Correlation represents the defree of which a pair of variables are related. The above correlation matrix displays how much each stock correlate to each other in pairs.

As discovered in the above correlation matrix, META and AMZN seem to have the highest correlation compared to any other stock in our study, indicating that they tend to move in the same direction.

There are also multiple low correlation pairs such as, META and AAPL, NVDA and AAPL.

These varying correlations suggest potential diversification benefits; combining stocks with lower correlation can reduce the overall portfolio risk.

In [13]:
```
import numpy as np

# 252 because there are 252 trading days a year
expectedReturns = dailyReturns.mean() * 252
volatility = dailyReturns.std() * np.sqrt(252)

stockStats = pd.DataFrame({
    'Expected Return': expectedReturns,
    'Volatility': volatility
})

stockStats
```

Out[13]:

Ticker	Expected Return	Volatility
AAPL	0.160192	0.227276
AMZN	0.390893	0.276377
GOOGL	0.325096	0.275125
META	0.459535	0.365793
NVDA	0.973097	0.471862

NVDA has the highest expected return at an astonishing 92.85%, but a volatility of 47.26%, the highest amongst all the studied stocks, indicating a potentially high-risk investment with a relatively higher risk.

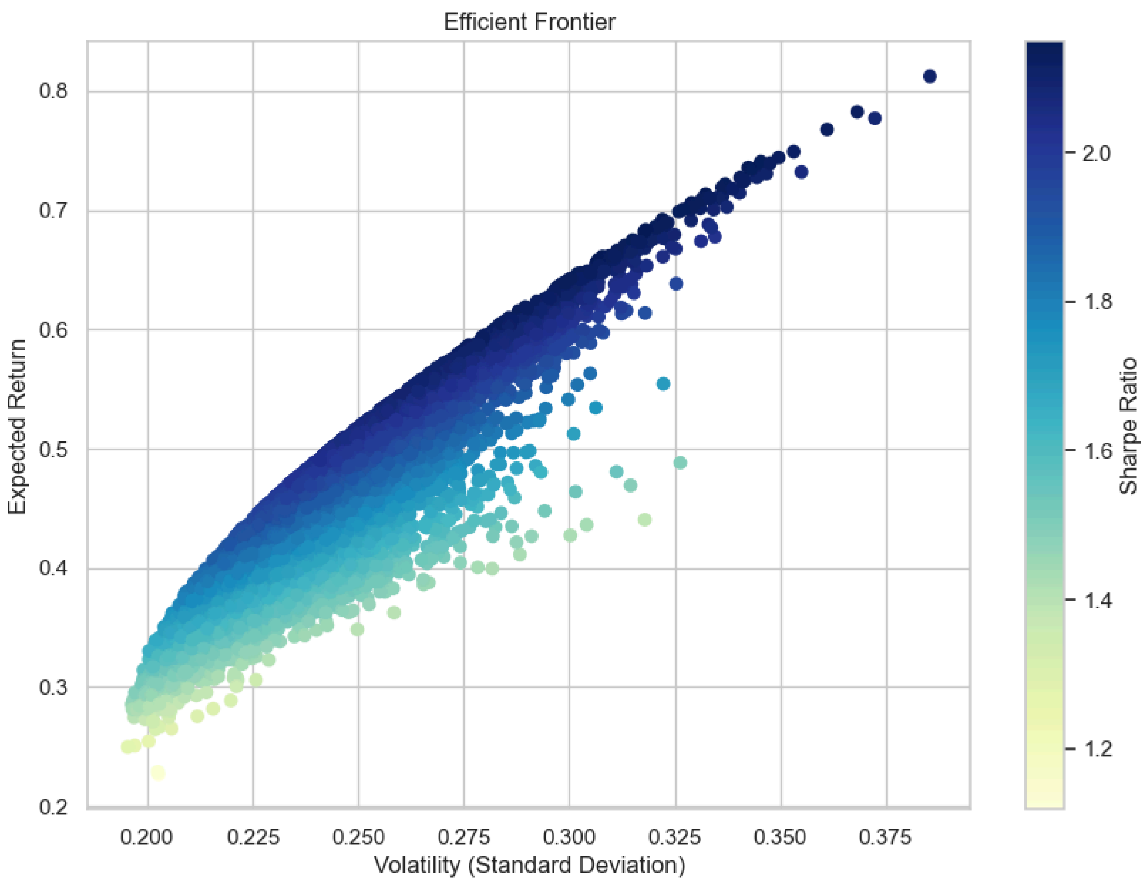
META and AMZN also have relatively high expected returns at 43.36% and 36.32% respectively but also a moderate volatility of 36.61% and 27.72% respectively.

GOOGL has a similar volatility compared to AMZN (27.57% GOOGL vs 27.72% AMZN) but has a lower expected return compared to AMZN (29.42% GOOGL vs 36.32% AMZN).

Whereas, AAPL has the lowest expected return of 13.86% but also has the lowest volatility at 22.74%.

After analysing the above, NVDA could potentially be the least attractive investment in terms of risk-adverse behaviour. Whereas AMZN in my opinion is the most attrative investment opportunity.

In [14]:
```
def portfolioPerformance(weights, returns, covarianceMatrix):
    portfolioReturn = np.dot(weights, returns)
    portfolioVolatility = np.sqrt(np.dot(weights.T, np.dot(covarianceM
    return portfolioReturn, portfolioVolatility

numOfPortfolios = 10000

results = np.zeros((3, numOfPortfolios))

covarianceMatrix = dailyReturns.cov() * 252

np.random.seed(43)

for i in range(numOfPortfolios):
    weights = np.random.random(len(uniqueTickers))
    weights /= np.sum(weights)

    portfolioReturn, portfolioVolatility = portfolioPerformance(weight

    results[0,i] = portfolioReturn
    results[1,i] = portfolioVolatility
    results[2,i] = portfolioReturn / portfolioVolatility  # Sharpe Rat

plt.figure(figsize=(10, 7))
plt.scatter(results[1,:], results[0,:], c=results[2,:], cmap='YlGnBu',
plt.title('Efficient Frontier')
plt.xlabel('Volatility (Standard Deviation)')
plt.ylabel('Expected Return')
plt.colorbar(label='Sharpe Ratio')
plt.grid(True)
plt.show()
```



Sharpe ratio is a measure of risk-adjusted returns.

Each dot represent a simulated portfolio and the colour represents the Sharpe Ratio, the darker the colour the better the risk-adjusted returns.

Portfolios on the leftmost edge (closer to the y-axis) offer the highest expected returns for a given level of volatility, which is the idea investment strategy when considering the Moden Portfolio Theory

In [15]:
```
maxSharpeIdx = np.argmax(results[2])
maxSharpeReturn = results[0, maxSharpeIdx]
maxSharpeVolatility = results[1, maxSharpeIdx]
maxSharpeRatio = results[2, maxSharpeIdx]

maxSharpeReturn, maxSharpeVolatility, maxSharpeRatio
```

Out[15]:
```
(0.692086398270254, 0.32210031771006825, 2.1486672325893847)
```

```
The above shows that the a portfolio with maximum Sharpe ratio has
the following:

Expected Return = 69.21%
Volatility = 32.21%
Sharpe Ratio ~ 2.149

Identifying the weight of the stocks in a portfolio to achieve the
following Sharpe Ratio are calculated below.
```

In [16]:
```
maxSharpeWeights = np.zeros(len(uniqueTickers))

for i in range(numOfPortfolios):
    weights = np.random.random(len(uniqueTickers))
    weights /= np.sum(weights)

    portfolioReturn, portfolioVolatility = portfolioPerformance(weight

    if results[2, i] == maxSharpeRatio:
        maxSharpeWeights = weights
        break

portfolioWeightsDF = pd.DataFrame({
    'Ticker': uniqueTickers,
    'Weight': maxSharpeWeights
})

portfolioWeightsDF
```

Out[16]:

	Ticker	Weight
0	AAPL	0.084441
1	AMZN	0.312022
2	GOOGL	0.273321
3	META	0.048030
4	NVDA	0.282187

```
Hence, it can be seen that to achieve the maximum Sharpe Ratio, the
portfolio diversification is as follows:
```

```
AAPL : 8.44%
AMZN : 31.20%
GOOGL : 27.33%
META : 4.80%
NVDA : 28.22%
```

AMZN has the highest allocation, indicating that a significant contribution of the portfolio's performance, where META has the smallest allocation. This balanced allocation aims to maximise returns while minimizing risk by leveraging individual stock performance and their correlations.