Data Science for Financial Markets

Table of Contents

Introduction

Data Science is a rapidly growing field that combines the power of statistical and computational techniques to extract valuable insights and knowledge from data. It brings together multiple disciplines such as

mathematics, statistics, computer science, and domain-specific knowledge to create a multi-faceted approach to understanding complex data patterns.

The goal of Data Science is to provide a complete picture of data and transform it into actionable information that can inform business decisions, scientific breakthroughs, and even public policy. With the increasing amount of data being generated every day, Data Science is becoming an increasingly vital part of our data-driven world.

When it comes to financial markets, Data Science can be applied in various ways, such as:

1. **Predictive Models:** Data Science professionals can use historical data to create predictive models that can identify trends and make predictions about future market conditions.

2. **Algorithmic Trading:** The use of algorithms that execute buy and sell orders autonomously, based on mathematical models through the analysis of price, volume, and volatility, among many others.

3. **Portfolio Optimization:** Algorithms and other mathematical models can be used to optimize portfolios, aiming for maximization of returns and risk minimization.

4. **Fraud Detection:** Data scientists can use machine learning algorithms to identify fraudulent activities in financial transactions.

5. **Risk Management:** Data science can be used to quantify and manage various types of financial risks, including market risk, credit risk, and operational risk.

6. **Customer Analysis:** Financial institutions can use data science to analyze customer data and gain insights into customer behavior and preferences, which can be used to improve customer engagement and retention.

In this notebook, I aim to demonstrate how Data Science, aswell as Python, can be powerful tools in extracting crucial insights from financial markets. I will demonstrate how these tools can be leveraged to build and optimize portfolios, develop effective trading strategies, and perform detailed stock analysis. This will showcase the versatility and usefulness of Data Science and Python in the finance industry and provide a valuable resource for those interested in utilizing these techniques to make informed investment decisions.

Essential Libraries

While developing this notebook, we will use four essential libraries specifically designed for handling financial data.
I will provide a brief introduction to each library and guide you through the steps required to install them in any Python environment.

yfinance

yfinance is probably the most popular Python library to extract data from financial markets! It allows you to obtain and analyze historical market data from Yahoo!Finance. It offers an easy-to-use API that allows users to fetch data for any publicly traded company, index, ETF, crypto and forex.

yfinance also provides tools for adjusting the data for dividends and splits, as well as for visualizing the data in different ways. Its simple interface and reliable data, makes it an excellent tool for analysis of financial data and explains why it is one of the most used library for traders and investors alike.

You can copy the code cell below in any Python environment to install it.

```
# Installing yfinance
!pip install yfinance
```

```
Collecting yfinance
  Downloading yfinance-0.2.13-py2.py3-none-any.whl (59 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 59.3/59.3 kB 2.3 MB/s eta
0:00:00
ultitasking>=0.0.7
  Downloading multitasking-0.0.11-py3-none-any.whl (8.5 kB)
Requirement already satisfied: appdirs>=1.4.4 in
/opt/conda/lib/python3.7/site-packages (from yfinance) (1.4.4)
Collecting pytz>=2022.5
  Downloading pytz-2022.7.1-py2.py3-none-any.whl (499 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 499.4/499.4 kB 12.9 MB/s eta
0:00:0000:01
ent already satisfied: frozendict>=2.3.4 in
/opt/conda/lib/python3.7/site-packages (from yfinance) (2.3.4)
Requirement already satisfied: cryptography>=3.3.2 in
/opt/conda/lib/python3.7/site-packages (from yfinance) (37.0.2)
Requirement already satisfied: html5lib>=1.1 in
```

```
/opt/conda/lib/python3.7/site-packages (from yfinance) (1.1)
Requirement already satisfied: beautifulsoup4>=4.11.1 in
/opt/conda/lib/python3.7/site-packages (from yfinance) (4.11.1)
Requirement already satisfied: numpy>=1.16.5 in
/opt/conda/lib/python3.7/site-packages (from yfinance) (1.21.6)
Requirement already satisfied: pandas>=1.3.0 in
/opt/conda/lib/python3.7/site-packages (from yfinance) (1.3.5)
Requirement already satisfied: requests>=2.26 in
/opt/conda/lib/python3.7/site-packages (from yfinance) (2.28.1)
Requirement already satisfied: lxml>=4.9.1 in
/opt/conda/lib/python3.7/site-packages (from yfinance) (4.9.2)
Requirement already satisfied: soupsieve>1.2 in
/opt/conda/lib/python3.7/site-packages (from beautifulsoup4>=4.11.1-
>yfinance) (2.3.1)
Requirement already satisfied: cffi>=1.12 in
/opt/conda/lib/python3.7/site-packages (from cryptography>=3.3.2-
>yfinance) (1.15.0)
Requirement already satisfied: webencodings in
/opt/conda/lib/python3.7/site-packages (from html5lib>=1.1->yfinance)
(0.5.1)
Requirement already satisfied: six>=1.9 in
/opt/conda/lib/python3.7/site-packages (from html5lib>=1.1->yfinance)
(1.15.0)
Requirement already satisfied: python-dateutil>=2.7.3 in
/opt/conda/lib/python3.7/site-packages (from pandas>=1.3.0->yfinance)
(2.8.2)
Requirement already satisfied: idna<4,>=2.5 in
/opt/conda/lib/python3.7/site-packages (from requests>=2.26->yfinance)
(3.3)
Requirement already satisfied: charset-normalizer<3,>=2 in
/opt/conda/lib/python3.7/site-packages (from requests>=2.26->yfinance)
(2.1.0)
Requirement already satisfied: certifi>=2017.4.17 in
/opt/conda/lib/python3.7/site-packages (from requests>=2.26->yfinance)
(2022.12.7)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in
/opt/conda/lib/python3.7/site-packages (from requests>=2.26->yfinance)
(1.26.14)
Requirement already satisfied: pycparser in
/opt/conda/lib/python3.7/site-packages (from cffi>=1.12-
>cryptography>=3.3.2->yfinance) (2.21)
Installing collected packages: pytz, multitasking, yfinance
  Attempting uninstall: pytz
```

```
    Found existing installation: pytz 2022.1
    Uninstalling pytz-2022.1:
      Successfully uninstalled pytz-2022.1
ERROR: pip's dependency resolver does not currently take into account
all the packages that are installed. This behaviour is the source of
the following dependency conflicts.
beatrix-jupyterlab 3.1.7 requires google-cloud-bigquery-storage, which
is not installed.
pandas-profiling 3.1.0 requires markupsafe~=2.0.1, but you have
markupsafe 2.1.2 which is incompatible.
ibis-framework 2.1.1 requires importlib-metadata<5,>=4; python_version
< "3.8", but you have importlib-metadata 6.0.0 which is incompatible.
apache-beam 2.40.0 requires dill<0.3.2,>=0.3.1.1, but you have dill
0.3.6 which is incompatible.
apache-beam 2.40.0 requires pyarrow<8.0.0,>=0.15.1, but you have
pyarrow 8.0.0 which is incompatible.
Successfully installed multitasking-0.0.11 pytz-2022.7.1 yfinance-
0.2.13
WARNING: Running pip as the 'root' user can result in broken
permissions and conflicting behaviour with the system package manager.
It is recommended to use a virtual environment instead:
https://pip.pypa.io/warnings/venv
```

Quantstats

Quantstats is a Python library used for quantitative financial analysis. This library provides various tools to obtain financial data from different sources, conduct technical and fundamental analyses, and create and test different investment strategies. It is also possible to use visualization tools to analyze individual stocks and portfolios. It is a simple and easy tool for any type of quantitative finance-oriented analysis, and that's why it will be essential for this study.

This is how you can install Quantstats in your Python environment:

```
# Installing Quantstats
!pip install quantstats
```

```
Collecting quantstats
  Downloading QuantStats-0.0.59-py2.py3-none-any.whl (41 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 41.3/41.3 kB 2.0 MB/s eta
0:00:00
ent already satisfied: python-dateutil>=2.0 in
/opt/conda/lib/python3.7/site-packages (from quantstats) (2.8.2)
Requirement already satisfied: tabulate>=0.8.0 in
/opt/conda/lib/python3.7/site-packages (from quantstats) (0.9.0)
Requirement already satisfied: scipy>=1.2.0 in
/opt/conda/lib/python3.7/site-packages (from quantstats) (1.7.3)
Requirement already satisfied: seaborn>=0.9.0 in
/opt/conda/lib/python3.7/site-packages (from quantstats) (0.11.2)
Requirement already satisfied: pandas>=0.24.0 in
/opt/conda/lib/python3.7/site-packages (from quantstats) (1.3.5)
Requirement already satisfied: numpy>=1.16.5 in
/opt/conda/lib/python3.7/site-packages (from quantstats) (1.21.6)
Requirement already satisfied: yfinance>=0.1.70 in
/opt/conda/lib/python3.7/site-packages (from quantstats) (0.2.13)
Requirement already satisfied: matplotlib>=3.0.0 in
/opt/conda/lib/python3.7/site-packages (from quantstats) (3.5.2)
Requirement already satisfied: packaging>=20.0 in
/opt/conda/lib/python3.7/site-packages (from matplotlib>=3.0.0-
>quantstats) (23.0)
Requirement already satisfied: cycler>=0.10 in
/opt/conda/lib/python3.7/site-packages (from matplotlib>=3.0.0-
>quantstats) (0.11.0)
Requirement already satisfied: pyparsing>=2.2.1 in
/opt/conda/lib/python3.7/site-packages (from matplotlib>=3.0.0-
>quantstats) (3.0.9)
Requirement already satisfied: pillow>=6.2.0 in
/opt/conda/lib/python3.7/site-packages (from matplotlib>=3.0.0-
>quantstats) (9.1.1)
Requirement already satisfied: kiwisolver>=1.0.1 in
/opt/conda/lib/python3.7/site-packages (from matplotlib>=3.0.0-
>quantstats) (1.4.3)
Requirement already satisfied: fonttools>=4.22.0 in
/opt/conda/lib/python3.7/site-packages (from matplotlib>=3.0.0-
>quantstats) (4.33.3)
Requirement already satisfied: pytz>=2017.3 in
/opt/conda/lib/python3.7/site-packages (from pandas>=0.24.0-
>quantstats) (2022.7.1)
Requirement already satisfied: six>=1.5 in
/opt/conda/lib/python3.7/site-packages (from python-dateutil>=2.0-
```

```
>quantstats) (1.15.0)
Requirement already satisfied: beautifulsoup4>=4.11.1 in
/opt/conda/lib/python3.7/site-packages (from yfinance>=0.1.70-
>quantstats) (4.11.1)
Requirement already satisfied: requests>=2.26 in
/opt/conda/lib/python3.7/site-packages (from yfinance>=0.1.70-
>quantstats) (2.28.1)
Requirement already satisfied: cryptography>=3.3.2 in
/opt/conda/lib/python3.7/site-packages (from yfinance>=0.1.70-
>quantstats) (37.0.2)
Requirement already satisfied: html5lib>=1.1 in
/opt/conda/lib/python3.7/site-packages (from yfinance>=0.1.70-
>quantstats) (1.1)
Requirement already satisfied: multitasking>=0.0.7 in
/opt/conda/lib/python3.7/site-packages (from yfinance>=0.1.70-
>quantstats) (0.0.11)
Requirement already satisfied: appdirs>=1.4.4 in
/opt/conda/lib/python3.7/site-packages (from yfinance>=0.1.70-
>quantstats) (1.4.4)
Requirement already satisfied: lxml>=4.9.1 in
/opt/conda/lib/python3.7/site-packages (from yfinance>=0.1.70-
>quantstats) (4.9.2)
Requirement already satisfied: frozendict>=2.3.4 in
/opt/conda/lib/python3.7/site-packages (from yfinance>=0.1.70-
>quantstats) (2.3.4)
Requirement already satisfied: soupsieve>1.2 in
/opt/conda/lib/python3.7/site-packages (from beautifulsoup4>=4.11.1-
>yfinance>=0.1.70->quantstats) (2.3.1)
Requirement already satisfied: cffi>=1.12 in
/opt/conda/lib/python3.7/site-packages (from cryptography>=3.3.2-
>yfinance>=0.1.70->quantstats) (1.15.0)
Requirement already satisfied: webencodings in
/opt/conda/lib/python3.7/site-packages (from html5lib>=1.1-
>yfinance>=0.1.70->quantstats) (0.5.1)
Requirement already satisfied: typing-extensions in
/opt/conda/lib/python3.7/site-packages (from kiwisolver>=1.0.1-
>matplotlib>=3.0.0->quantstats) (4.1.1)
Requirement already satisfied: idna<4,>=2.5 in
/opt/conda/lib/python3.7/site-packages (from requests>=2.26-
>yfinance>=0.1.70->quantstats) (3.3)
Requirement already satisfied: certifi>=2017.4.17 in
/opt/conda/lib/python3.7/site-packages (from requests>=2.26-
>yfinance>=0.1.70->quantstats) (2022.12.7)
```

```
Requirement already satisfied: charset-normalizer<3,>=2 in
/opt/conda/lib/python3.7/site-packages (from requests>=2.26-
>yfinance>=0.1.70->quantstats) (2.1.0)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in
/opt/conda/lib/python3.7/site-packages (from requests>=2.26-
>yfinance>=0.1.70->quantstats) (1.26.14)
Requirement already satisfied: pycparser in
/opt/conda/lib/python3.7/site-packages (from cffi>=1.12-
>cryptography>=3.3.2->yfinance>=0.1.70->quantstats) (2.21)
Installing collected packages: quantstats
Successfully installed quantstats-0.0.59
WARNING: Running pip as the 'root' user can result in broken
permissions and conflicting behaviour with the system package manager.
It is recommended to use a virtual environment instead:
https://pip.pypa.io/warnings/venv
```

PyPortfolioOpt

PyPortfolioOpt is an extremely useful library for portfolio optimization and asset allocation. This library provides various tools to create optimized portfolios based on user-defined constraints, objectives, and risk preferences. It includes a range of optimization algorithms, such as the mean-variance optimization model, the Black-Litterman allocation model, and many others.

With its user-friendly interface and flexible optimization capabilities, PyPortfolioOpt is a powerful library for portfolio construction and optimization.

In the code cell below, you find how to install PyPortfolioOpt in your Python environment.

```
# installing PyPortfolioOpt
!pip install pyportfolioopt
```

```
Collecting pyportfolioopt
  Downloading pyportfolioopt-1.5.2-py3-none-any.whl (61 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 61.4/61.4 kB 2.1 MB/s eta
0:00:00
ent already satisfied: numpy<2.0,>=1.12 in
/opt/conda/lib/python3.7/site-packages (from pyportfolioopt) (1.21.6)
Requirement already satisfied: scipy<2.0,>=1.3 in
/opt/conda/lib/python3.7/site-packages (from pyportfolioopt) (1.7.3)
Requirement already satisfied: pandas>=0.19 in
/opt/conda/lib/python3.7/site-packages (from pyportfolioopt) (1.3.5)
Collecting cvxpy<2.0.0,>=1.1.10
  Downloading cvxpy-1.3.1-cp37-cp37m-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (4.0 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4.0/4.0 MB 38.7 MB/s eta
0:00:00a 0:00:01
-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (10.7 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 10.7/10.7 MB 66.7 MB/s eta
0:00:0000:0100:01
ent already satisfied: ecos>=2 in /opt/conda/lib/python3.7/site-
packages (from cvxpy<2.0.0,>=1.1.10->pyportfolioopt) (2.0.12)
Collecting osqp>=0.4.1
  Downloading osqp-0.6.2.post8-cp37-cp37m-
manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_17_x86_64.manylinux2
(296 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 296.2/296.2 kB 23.8 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.1/1.1 MB 53.0 MB/s eta 0:00:00
ent already satisfied: python-dateutil>=2.7.3 in
/opt/conda/lib/python3.7/site-packages (from pandas>=0.19-
>pyportfolioopt) (2.8.2)
Requirement already satisfied: pytz>=2017.3 in
/opt/conda/lib/python3.7/site-packages (from pandas>=0.19-
>pyportfolioopt) (2022.7.1)
Collecting qdldl
  Downloading qdldl-0.1.5.post3-cp37-cp37m-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.1 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.1/1.1 MB 56.6 MB/s eta 0:00:00
ent already satisfied: six>=1.5 in /opt/conda/lib/python3.7/site-
packages (from python-dateutil>=2.7.3->pandas>=0.19->pyportfolioopt)
(1.15.0)
Installing collected packages: setuptools, scs, qdldl, osqp, cvxpy,
pyportfolioopt
  Attempting uninstall: setuptools
```

```
    Found existing installation: setuptools 59.8.0
    Uninstalling setuptools-59.8.0:
      Successfully uninstalled setuptools-59.8.0
ERROR: pip's dependency resolver does not currently take into account
all the packages that are installed. This behaviour is the source of
the following dependency conflicts.
beatrix-jupyterlab 3.1.7 requires google-cloud-bigquery-storage, which
is not installed.
gcsfs 2022.5.0 requires fsspec==2022.5.0, but you have fsspec 2023.1.0
which is incompatible.
cloud-tpu-client 0.10 requires google-api-python-client==1.8.0, but
you have google-api-python-client 2.52.0 which is incompatible.
Successfully installed cvxpy-1.3.1 osqp-0.6.2.post8 pyportfolioopt-
1.5.2 qdldl-0.1.5.post3 scs-3.2.2 setuptools-67.6.0
WARNING: Running pip as the 'root' user can result in broken
permissions and conflicting behaviour with the system package manager.
It is recommended to use a virtual environment instead:
https://pip.pypa.io/warnings/venv
```

TA

The TA (Technical Analysis) library is a powerful tool for conducting technical analysis using Python. It provides a wide range of technical indicators, such as moving averages, Bollinger bands, MACD, and the Relative Strength Index to analyze market trends, momentum, and volatility.

The TA library is extremely easy to use and allows users to customize their analysis based on their preferred indicators and parameters. With its extensive range of technical analysis tools, it is a valuable resource for both traders and analysts looking to gain valuable insights on market behavior.

Here's how you can install the TA library in your own Python environment:

```
# Installing the TA (Technical Analysis) library
!pip install ta
```

```
Collecting ta
  Downloading ta-0.10.2.tar.gz (25 kB)
  Preparing metadata (setup.py) ... ent already satisfied: numpy in
/opt/conda/lib/python3.7/site-packages (from ta) (1.21.6)
Requirement already satisfied: pandas in
/opt/conda/lib/python3.7/site-packages (from ta) (1.3.5)
Requirement already satisfied: python-dateutil>=2.7.3 in
/opt/conda/lib/python3.7/site-packages (from pandas->ta) (2.8.2)
Requirement already satisfied: pytz>=2017.3 in
/opt/conda/lib/python3.7/site-packages (from pandas->ta) (2022.7.1)
Requirement already satisfied: six>=1.5 in
/opt/conda/lib/python3.7/site-packages (from python-dateutil>=2.7.3-
>pandas->ta) (1.15.0)
Building wheels for collected packages: ta
  Building wheel for ta (setup.py) ... e=ta-0.10.2-py3-none-any.whl
size=29088
sha256=9c1b92d61d6086f71a83f13e01c328020b7e2d8d83f5e843c13e0e8d298960ba
  Stored in directory:
/root/.cache/pip/wheels/31/31/f1/f2ff471bbc5b84a4b973698ceecdd453ae043
Successfully built ta
Installing collected packages: ta
Successfully installed ta-0.10.2
WARNING: Running pip as the 'root' user can result in broken
permissions and conflicting behaviour with the system package manager.
It is recommended to use a virtual environment instead:
https://pip.pypa.io/warnings/venv
```

Now that you've had a brief introduction to the most essential financial libraries in this notebook, we can move on to importing all the specific libraries we'll be using.

```
# Importing Libraries

# Data handling and statistical analysis
import pandas as pd
from pandas_datareader import data
import numpy as np
```

```
from scipy import stats

# Data visualization
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots

# Optimization and allocation
from pypfopt.efficient_frontier import EfficientFrontier
from pypfopt import risk_models
from pypfopt import expected_returns
from pypfopt import black_litterman, BlackLittermanModel

# Financial data
import quantstats as qs
import ta
import yfinance as yf

# Linear Regression Model
from sklearn.linear_model import LinearRegression

# Enabling Plotly offline
from plotly.offline import init_notebook_mode
init_notebook_mode(connected=True)

# Datetime and hiding warnings
import datetime as dt
import warnings
warnings.filterwarnings("ignore")
```

```
(CVXPY) Mar 21 07:42:46 PM: Encountered unexpected exception importing
solver GLOP:
RuntimeError('Unrecognized new version of ortools (9.5.2237). Expected
< 9.5.0.Please open a feature request on cvxpy to enable support for
this version.')
(CVXPY) Mar 21 07:42:46 PM: Encountered unexpected exception importing
solver PDLP:
RuntimeError('Unrecognized new version of ortools (9.5.2237). Expected
< 9.5.0.Please open a feature request on cvxpy to enable support for
this version.')
```

1 | Getting Started

Daily Returns

The first thing we're going to look at is the daily returns. A stock's daily return is the percentual change in price over a single day. You calculate it by subtracting the difference between the stock's closing price on one day and its closing price the day before, dividing the result by the closing of the day before, and multiplying it by 100.

For instance, if a stock closes at 100 dollars on Monday, and it closes at 102 dollars on Tuesday, its daily return would be calculated as:

$$((102-100)\div100)\times100 = 2\%$$

This shows that the stock increased in value by 2% over the course of one day. On the other hand, if the stock had closed at 98 dollars on Tuesday, the daily return would be calculated as:

$$((98-100)\div100)\times100 = -2\%$$

Which means that the stock has decreased in value by 2% over the course of one day.

Daily returns are relevant for investors because they provide a quick way to check the performance of a stock over a short period.

With Quantstats, it's possible to plot daily returns charts, which are graphical representations of the daily percentage changes in stocks, allowing investors to visualize the ups and downs of the stock's daily performance over time and extract information on volatility and consistency of returns.

In order to start our analysis, we're going to use Quantstats *utils.download_returns* method to download the daily returns for four different US stocks over the same period of time, for a fair comparison and analysis between them.

```
# Getting daily returns for 4 different US stocks in the same time
        window
aapl = qs.utils.download_returns('AAPL')
aapl = aapl.loc['2010-07-01':'2023-02-10']

tsla = qs.utils.download_returns('TSLA')
tsla = tsla.loc['2010-07-01':'2023-02-10']

dis = qs.utils.download_returns('DIS')
dis = dis.loc['2010-07-01':'2023-02-10']

amd = qs.utils.download_returns('AMD')
amd = amd.loc['2010-07-01':'2023-02-10']
```

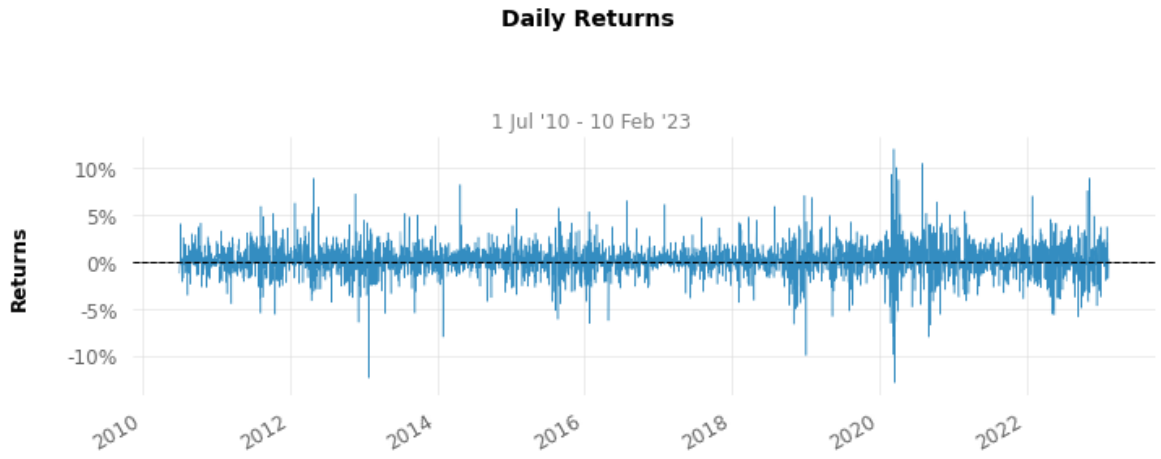
We now have the daily returns from July 1st, 2010, to February 10th, 2023, for four different US stocks from distinct industries, Apple, Tesla, The Walt Disney Company, and AMD.

We can now plot the daily returns chart for each of them using Quantstats.

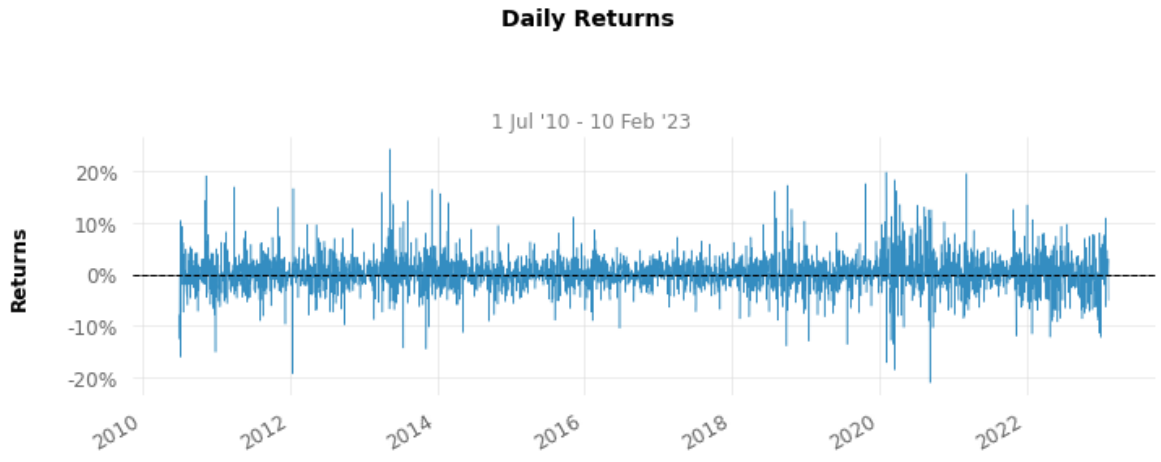
```
# Converting timezone
aapl.index = aapl.index.tz_convert(None)
tsla.index = tsla.index.tz_convert(None)
dis.index = dis.index.tz_convert(None)
amd.index = amd.index.tz_convert(None)
```

```
# Plotting Daily Returns for each stock
print('\n')
print('\nApple Daily Returns Plot:\n')
qs.plots.daily_returns(aapl)
print('\n')
print('\n')
print('\nTesla Inc. Daily Returns Plot:\n')
qs.plots.daily_returns(tsla)
print('\n')
print('\n')
print('\nThe Walt Disney Company Daily Returns Plot:\n')
qs.plots.daily_returns(dis)
print('\n')
print('\n')
print('\nAdvances Micro Devices, Inc. Daily Returns Plot:\n')
qs.plots.daily_returns(amd)
```

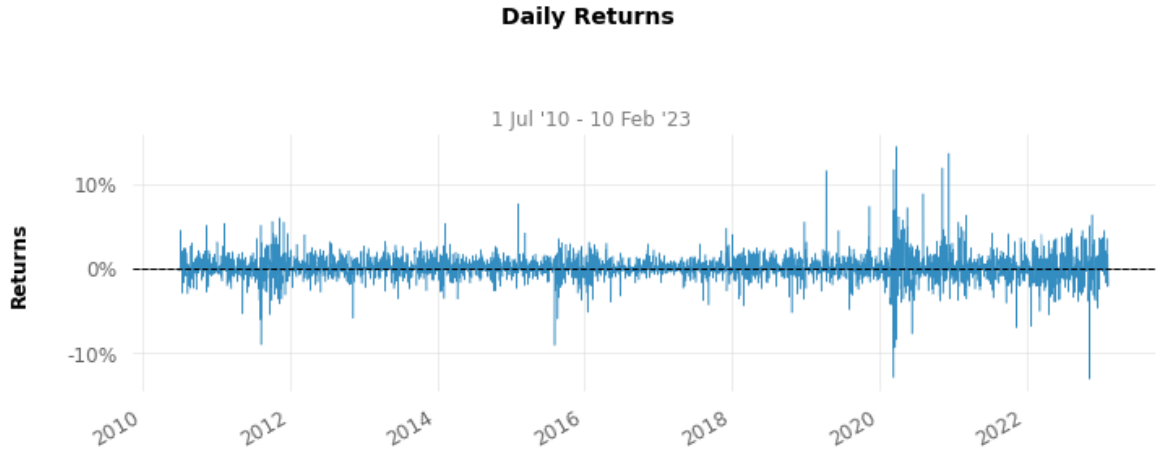
Apple Daily Returns Plot:



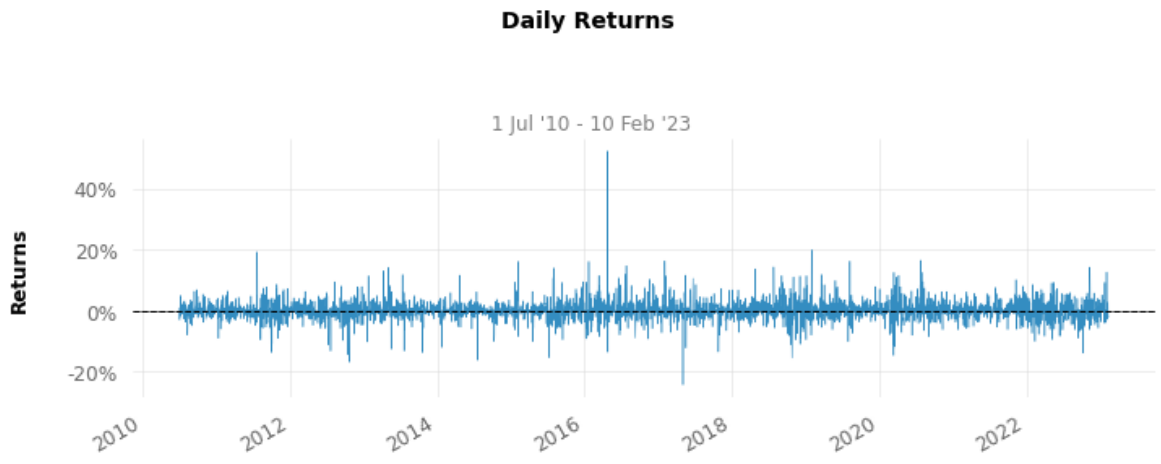
Tesla Inc. Daily Returns Plot:



The Walt Disney Company Daily Returns Plot:



Advances Micro Devices, Inc. Daily Returns Plot:



The plots above allow us to see an unusual variation in AMD stock prices, an increase of around 40% in its shares by 2016, which may have occurred for various factors, such as surprising earnings reports, increased demand for the company's products, or favorable market conditions. This behavior may indicate high volatility, thus marking it a riskier investment.

On the other hand, Disney's and Apple's stocks seem more stable and predictable investment options at first glance.

Cumulative Returns

To calculate a stock's cumulative return, the first thing to do is to determine the stock's initial price and its final price at the end of the specified period. Then subtract the initial price from the final price, add any dividends or other income received, and divide the result by the initial price. This gives us the cumulative return as a decimal, which can be multiplied by 100 to express it as a percentage.

It's important to note that cumulative return takes into account the effects of compounding, meaning that any gains from a previous period are reinvested and contribute to additional gains in future periods, which can result in a larger cumulative return than the simple average of the individual returns over the specified period.

Below, we can see line charts displaying the cumulative return for each one of the stocks we've downloaded since July, 2010.

```
# Plotting Cumulative Returns for each stock
print('\n')
print('\nApple Cumulative Returns Plot\n')
qs.plots.returns(aapl)
print('\n')
print('\n')
print('\nTesla Inc. Cumulative Returns Plot\n')
qs.plots.returns(tsla)
print('\n')
print('\n')
print('\nThe Walt Disney Company Cumulative Returns Plot\n')
qs.plots.returns(dis)
print('\n')
print('\n')
print('\nAdvances Micro Devices, Inc. Cumulative Returns Plot\n')
qs.plots.returns(amd)
```

```
Apple Cumulative Returns Plot
```

Tesla Inc. Cumulative Returns Plot

The Walt Disney Company Cumulative Returns Plot

Advances Micro Devices, Inc. Cumulative Returns Plot

The charts above shows a considerate difference between Tesla's and Disney's returns. At the peak of its returns, Tesla surpassed the mark of over 25,000%, an extraordinary investment for those who bought the

company's shares by the beginning of the decade. On the other hand, Disney's shares had some modest returns, peaking at around 650%.

Of course, when analyzing stocks data, we don't make an investment merely looking only at the cumulative returns. It's crucial to look at other indicators and evaluate the risks of the investment. Besides, 650% returns are still significant, and in the stock market, slow but steady growth can be just as valuable as explosive returns.

A variety of strategies must be taken into account in order to build a robust portfolio.

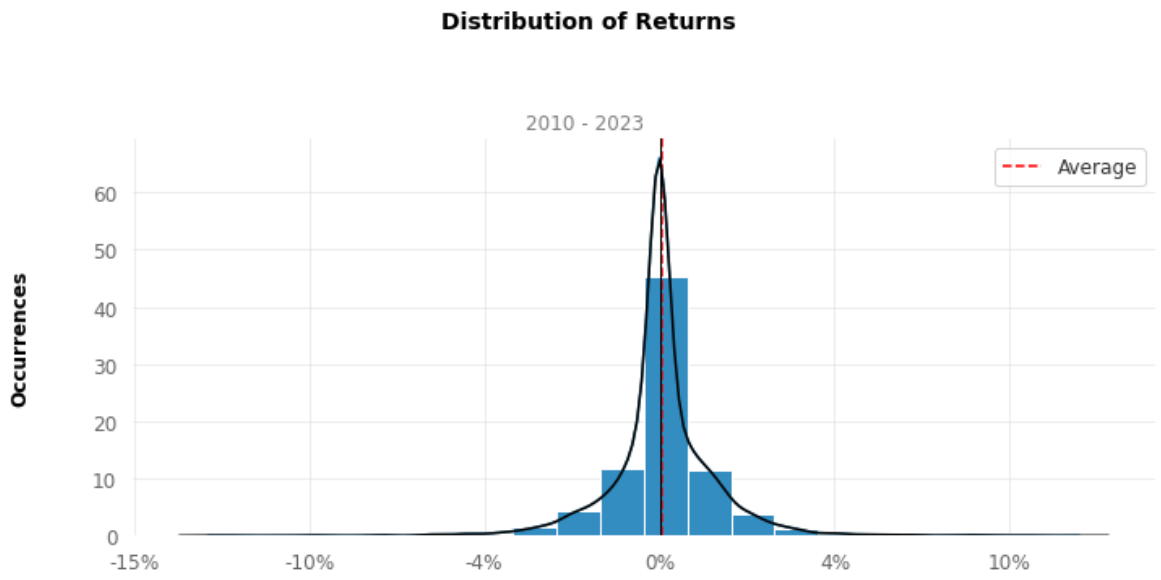
Histograms

Histograms are a graphical representation of the distribution of values, displaying how frequent they are in a dataset.

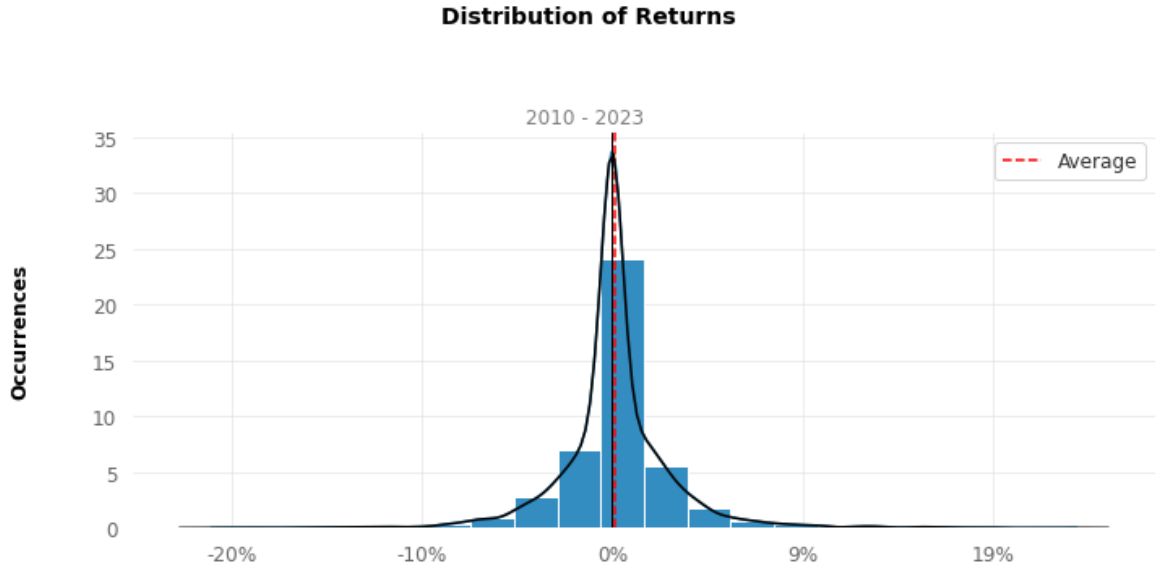
Histograms of daily returns are valuable to help investors to identify patterns, such as the range of daily returns of an asset over a certain period, indicating its level of stability and volatility.

```
# Plotting histograms for daily returns
print('\n')
print('\nApple Daily Returns Histogram')
qs.plots.histogram(aapl, resample = 'D')
print('\n')
print('\nTesla Inc. Daily Returns Histogram')
qs.plots.histogram(tsla, resample = 'D')
print('\n')
print('\nThe Walt Disney Company Daily Returns Histogram')
qs.plots.histogram(dis, resample = 'D')
print('\n')
print('\nAdvances Micro Devices, Inc. Daily Returns Histogram')
qs.plots.histogram(amd, resample = 'D')
```

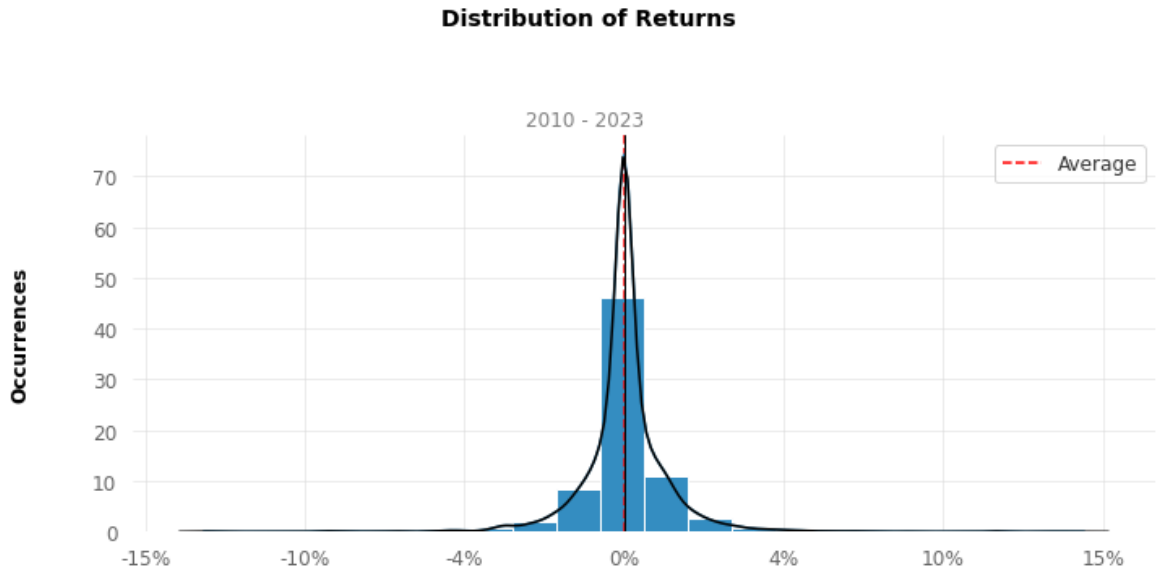
Apple Daily Returns Histogram



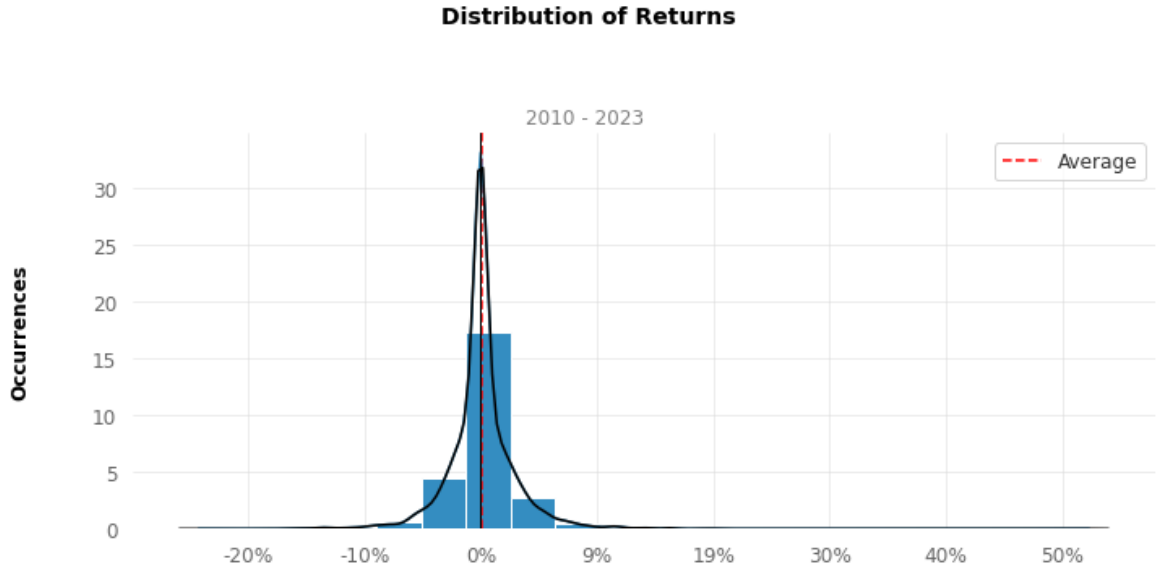
Tesla Inc. Daily Returns Histogram



The Walt Disney Company Daily Returns Histogram



Advances Micro Devices, Inc. Daily Returns Histogram



Through the analysis of the histograms, we can observe that most daily returns are close to zero in the center of the distribution. However, it's easy to see some extreme values that are distant from the mean, which is the case of AMD, with daily returns of around 50%, indicating the presence of outliers in the positive range of the distribution, in contrast with the negative field where it seems to limit at about -20%.

Disney's stocks have more balanced returns with values ranging from -15% to 15%, while most returns are closer to the mean.

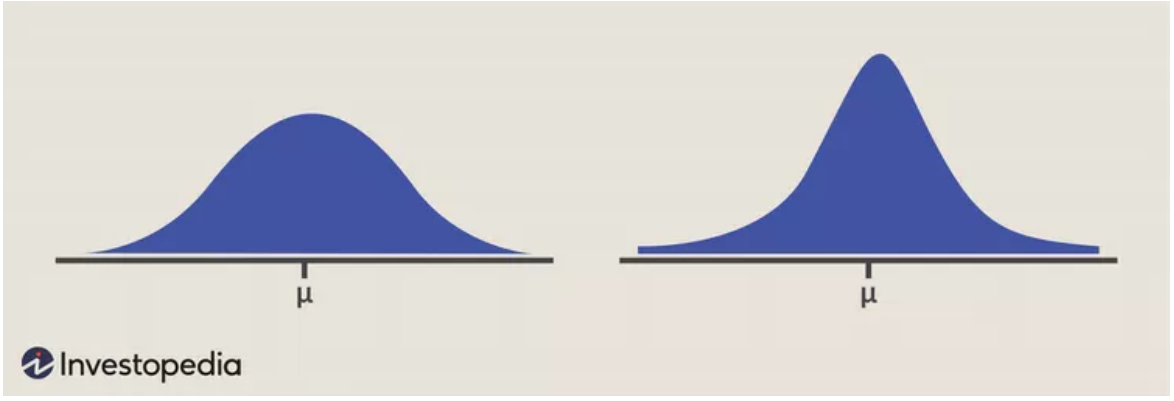
Through histograms, we can extract some valuable statistics such as kurtosis and skewness.

Kurtosis

A high kurtosis value for daily returns may indicate frequent fluctuations in price that deviate significantly from the average returns of that investment, which can lead to increased volatility and risk associated with the stock.

A kurtosis value above 3.0 defines a leptokurtic distribution, characterized by outliers and more values that are distant from the average, which reflects in the histogram as stretching of the horizontal axis. Stocks with a leptokurtic distribution are generally associated with a higher level of risk but also offer the potential for higher returns due to the substantial price movements that have occurred in the past.

In the image below, it's possible to see the difference between a negative kurtosis on the left and a positive kurtosis on the right. The distribution on the left displays a lower probability of extreme values and a lower concentration of values around the mean, while the distribution on the right shows a higher concentration of values near the mean, but also the existence (and thus a higher probability of occurrence) of extreme values.



Kurtosis measures the concentration of observations in the tails versus the center of a distribution. In finance, a high level of excess kurtosis, or "tail risk," represents the chance of a loss occurring as a result of a rare event. This type of risk is important for investors to consider when making investment decisions, as it may impact the potential returns and stability of a particular stock.

```
# Using quantstats to measure kurtosis
print('\n')
print("Apple's kurtosis: ", qs.stats.kurtosis(aapl).round(2))
print('\n')
print("Tesla's kurtosis: ", qs.stats.kurtosis(tsla).round(2))
print('\n')
print("Walt Disney's kurtosis: ", qs.stats.kurtosis(dis).round(3))
print('\n')
print("Advances Micro Devices' kurtosis: ",
        qs.stats.kurtosis(amd).round(3))
```

```
Apple's kurtosis:  5.26
```

```
Tesla's kurtosis:  5.04
```

```
Walt Disney's kurtosis:  11.033
```

```
Advances Micro Devices' kurtosis:  17.125
```

The kurtosis values above show that all four stocks, Apple, Tesla, Walt Disney, and Advanced Micro Devices, have high levels of kurtosis, indicating a high concentration of observations in the tails of their daily returns distributions, which suggests that all four stocks are subject to high levels of volatility and risk, with considerate price fluctuations that deviate significantly from their average returns.

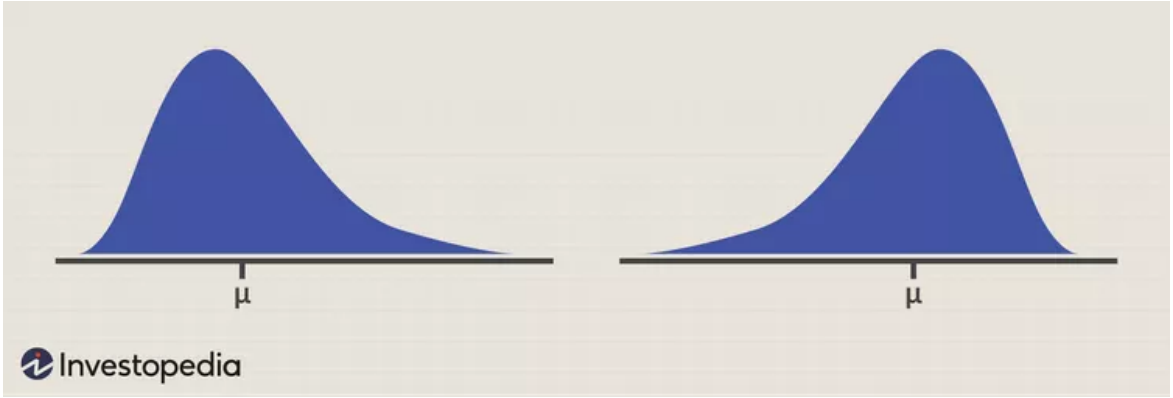
However, AMD has the highest kurtosis, with a value of 17.125, which indicates that AMD is subject to an extremely high level of volatility and tail risk, with a large concentration of extreme price movements. On the

other hand, Disney has a kurtosis of 11.033, which is still higher than a typical value for a normal distribution, but not as extreme as AMD's.

Skewness

Skewness is a metric that quantifies the asymmetry of returns. It reflects the shape of the distribution and determines if it is symmetrical, skewed to the left, or skewed to the right.

Below, it is possible to see two different asymmetrical distributions. On the left, it shows an example of a positively skewed distribution, with a long right tail, indicating a substantial probability of extremely positive daily returns compared to a normal distribution. On the other hand, a negatively skewed distribution would most likely resemble the distribution on the right, with a long tail representing more frequency of outliers on the negative side of returns.



A skewness value of zero indicates a symmetrical distribution, in which observations are evenly distributed on both sides of the mean, and the right and left tails are of approximately equal size.

The skewness is calculated with the following formula:

$$\text{skewness} = \frac{\mu_{3}(x) - 3 \times \mu(x) \times \sigma^{2}(x) - \mu^{3}(x)}{\sigma^{3}(x)}$$

Where $\boldsymbol{x}$ represents the set of returns data, $\boldsymbol{\mu}$ represents the mean of the returns, and $\boldsymbol{\sigma}$ represents the standard deviation of the returns. This formula results in a single numerical value that summarizes the skewness of returns.

```
# Measuring skewness with quantstats
print('\n')
print("Apple's skewness: ", qs.stats.skew(aapl).round(2))
print('\n')
print("Tesla's skewness: ", qs.stats.skew(tsla).round(2))
print('\n')
print("Walt Disney's skewness: ", qs.stats.skew(dis).round(3))
print('\n')
print("Advances Micro Devices' skewness: ",
        qs.stats.skew(amd).round(3))
```

```
Apple's skewness:  -0.07


Tesla's skewness:  0.33


Walt Disney's skewness:  0.199


Advances Micro Devices' skewness:  1.043
```

Generally, a value between -0.5 and 0.5 indicates a slight level of skewness, while values below -1 and above 1 are indications of strong asymmetry.

Apple, Tesla, and Disney are just slightly skewed, and Disney's slight skewness can be seen by looking at the range of the x-axis of its histogram, where it is pretty much balanced between -15% and 15%.

AMD stocks are strongly skewed, which can also be easily identified by looking at the range between -20% and 50% in its histogram. AMD has a lot of outliers on the positive tail, which could've been a good thing for those who bought its shares but it also suggests higher volatility and risk to this investment.

Standard Deviation

Standard deviation is a widely used statistical metric that quantifies the variability of the dataset. When applied to a stock's daily returns, it can indicate the risk level associated with investing in that particular stock. A stock exhibiting high daily return volatility, characterized by a high standard deviation, is considered riskier when compared to one with low daily return volatility, represented by a low standard deviation.

The formula for standard deviation is given by:

$$\sigma = \sqrt{\frac{1}{N-1} \sum_{i=1}^{N} (x_i - \bar{x})^2}$$

Where $\boldsymbol{x}$ represents the set of returns data, $\bar{x}$ is the mean of the returns data, and $\boldsymbol{N}$ is the number of observations. Standard deviation enables investors to assess the risk level and to compare the volatility of different stocks. For instance, if two assets have similar average returns, but one has a higher standard deviation, it is usually considered a riskier investment. Hence, standard deviation serves as a useful tool in helping investors to make informed decisions regarding their investment choices and portfolio management.

```
# Calculating Standard Deviations
print('\n')
```

```
print("Apple's Standard Deviation from 2010 to 2023: ",
        aapl.std().round(3))
print('\n')
print("\nTesla's Standard Deviation from 2010 to 2023: ",
        tsla.std().round(3))
print('\n')
print("\nDisney's Standard Deviation from 2010 to 2023: ",
        dis.std().round(3))
print('\n')
print("\nAMD's Standard Deviation from 2010 to 2023: ",
        amd.std().round(3))
```

```
Apple's Standard Deviation from 2010 to 2023:  0.018
```

```
Tesla's Standard Deviation from 2010 to 2023:  0.036
```

```
Disney's Standard Deviation from 2010 to 2023:  0.016
```

```
AMD's Standard Deviation from 2010 to 2023:  0.036
```

Based on the values above, we can say that Apple and Disney are less volatile than Tesla and AMD, suggesting that Apple and Disney are safer investment options, exhibiting more stable price fluctuations in the market.

Pairplots and Correlation Matrix

Correlation analysis in the stock market allows us for interesting investment strategies. A widely known strategy in the market is called Long-Short, which is the act of buying shares of a company, while selling shares of another company, believing that both assets will have opposite directions in the market. That is, when one goes up, the other goes down. To develop Long-Short strategies, investors rely on correlation analysis between stocks.

Correlation analysis is not only useful for Long-Short strategies, but it's also crucial to avoid systemic risk, which is described as the risk of the breakdown of an entire system rather than simply the failure of individual parts. To make it simple, if your portfolio has stocks that are highly correlated, or are all in the same industry, if something happens to that specific industry, all of your stocks may lose market value and it can cause greater financial losses.

Pairplots and correlation matrices are useful tools to visualize correlation among assets. In the correlation matrix, values range between -1 and 1, where -1 represents a perfect negative correlation and 1 represents a perfect positive correlation. Keep in mind that, when assets are positively correlated, they tend to go up and down simultaneously in the market, while the opposite is true for those that are negatively correlated.

```
# Merging daily returns into one dataframe
merged_df = pd.concat([aapl, tsla, dis, amd], join = 'outer', axis =
        1)
merged_df.columns = ['aapl', 'tsla', 'dis', 'amd']
merged_df # Displaying dataframe
```

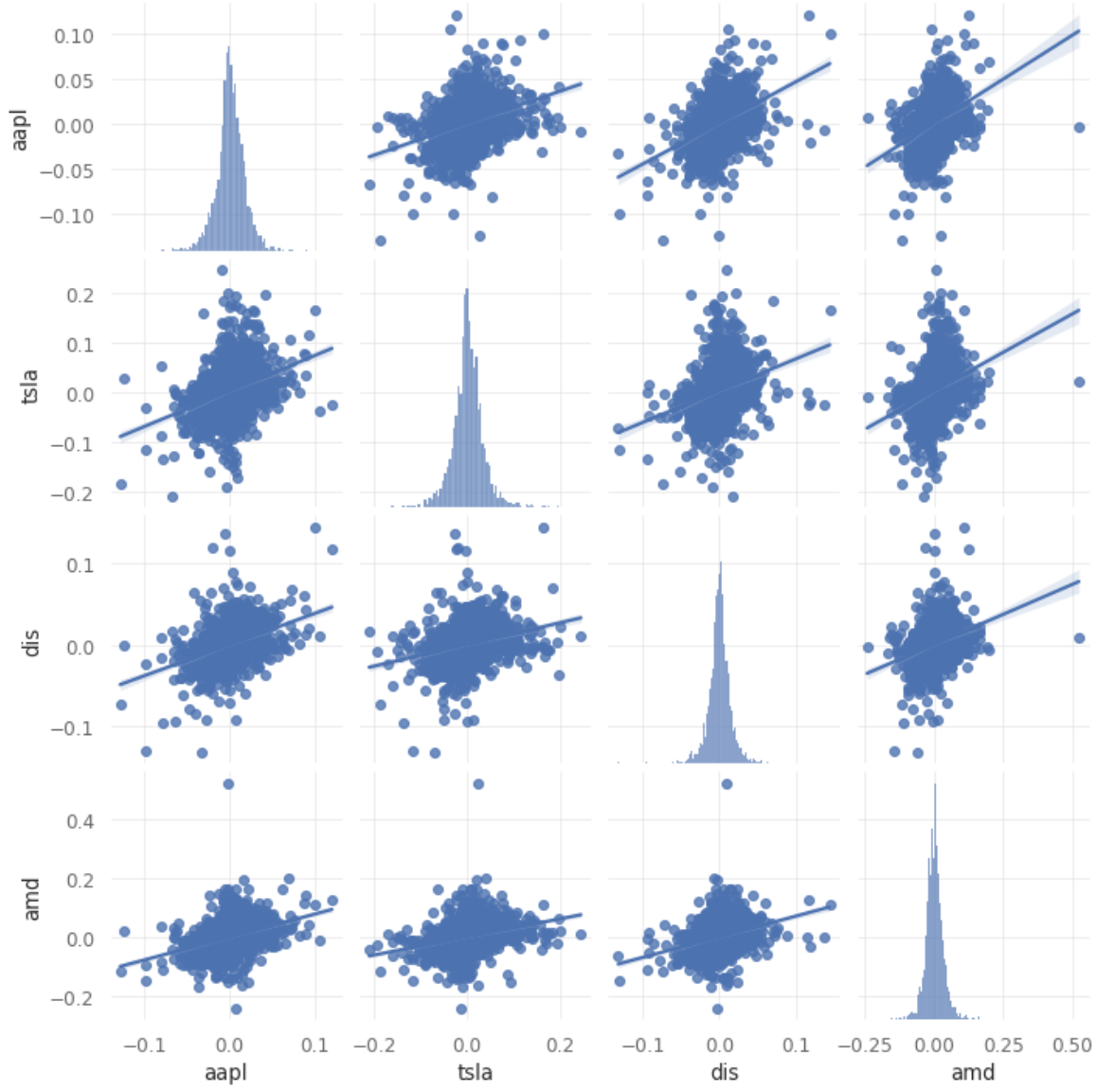
	aapl	tsla	dis	amd
Date				
2010-07-01 04:00:00	-0.012126	-0.078473	-0.000317	0.009563
2010-07-02 04:00:00	-0.006197	-0.125683	-0.003493	-0.029770
2010-07-06 04:00:00	0.006844	-0.160937	0.010835	-0.018131
2010-07-07 04:00:00	0.040381	-0.019243	0.044767	0.049716
2010-07-08 04:00:00	-0.002242	0.105064	0.006035	-0.002706
...	...	...	...	...
2023-02-06 05:00:00	-0.017929	0.025161	-0.007587	-0.027994
2023-02-07 05:00:00	0.019245	0.010526	0.016019	0.026649
2023-02-08 05:00:00	-0.017653	0.022763	0.001344	-0.014201
2023-02-09 05:00:00	-0.006912	0.029957	-0.012704	-0.017476

	aapl	tsla	dis	amd
Date				
2023-02-10 05:00:00	0.002456	-0.050309	-0.020841	-0.020791

3176 rows × 4 columns

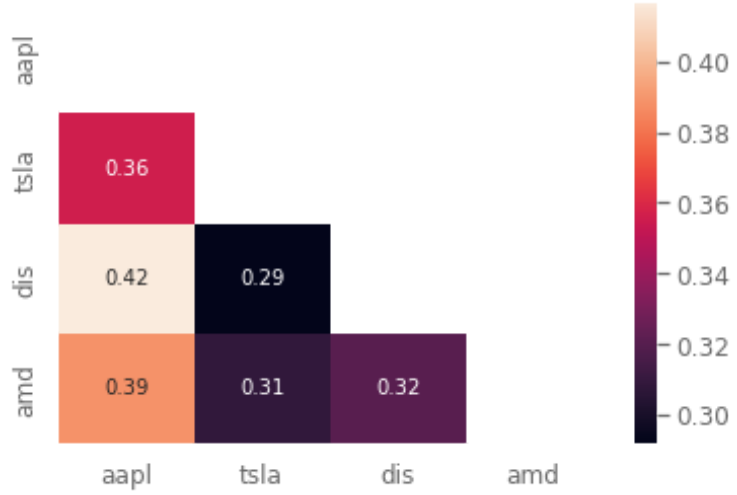
The dataframe above has dates serving as the index and each stock is represented as a column, displaying their respective returns for each specific day. This dataframe will be used to calculate the correlation between these stocks and to create a pairplot visualization.

```
# Pairplots
sns.pairplot(merged_df, kind = 'reg')
plt.show()
```




```
# Correlation Matrix
corr = merged_df.corr()
```

```
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True
sns.heatmap(corr, annot=True, mask = mask)
plt.show()
```



The stronger correlation among the assets above is between Disney and Apple. However, a correlation of 0.42 is not a strong one.

It's important to note that there is not any negative correlation among the assets above, which indicates that none of them acts to limit losses. In the financial market, a hedge is an investment position intended to offset potential losses by investing in assets that may have a negative correlation with the others in a portfolio. Many investors buy gold to serve as protection for riskier investments, such as stocks, and when the market as a whole goes into a bear market, the gold tends to increase in value, limiting potential losses for the overall portfolio.

Beta and Alpha

Beta and Alpha are two key metrics used in finance to evaluate the performance of a stock relative to the overall market. Beta is a measure of a stock's volatility compared to the market. A Beta of 1 means that the stock is as volatile as the market, a Beta greater than 1 indicates higher volatility than the market, and a Beta less than 1 suggests lower volatility.

Alpha, on the other hand, is a measurement of a stock's excess return relative to its expected performance based on its Beta. A positive Alpha indicates that a stock has outperformed its expected performance based on its Beta, while a negative Alpha suggests underperformance. By analyzing the Beta and Alpha values of stocks, investors can get a better understanding of the risk and potential returns of the stock compared to the market, and make informed investment decisions accordingly.

To determine Beta and Alpha, we require data from the SP500, which acts as the benchmark, to fit a linear regression model between the stocks and the index. This will enable us to extract the Beta and Alpha values of the stocks.

Let's then load data on the SP500:

```
# Loading data from the SP500, the american benchmark
sp500 = qs.utils.download_returns('^GSPC')
sp500 = sp500.loc['2010-07-01':'2023-02-10']
sp500.index = sp500.index.tz_convert(None)
sp500
```

```
Date
2010-07-01 04:00:00   -0.003240
2010-07-02 04:00:00   -0.004662
2010-07-06 04:00:00    0.005359
2010-07-07 04:00:00    0.031331
2010-07-08 04:00:00    0.009413
                           ...
2023-02-06 05:00:00   -0.006140
2023-02-07 05:00:00    0.012873
2023-02-08 05:00:00   -0.011081
2023-02-09 05:00:00   -0.008830
2023-02-10 05:00:00    0.002195
Name: Close, Length: 3176, dtype: float64
```

```
# Removing indexes
sp500_no_index = sp500.reset_index(drop = True)
aapl_no_index = aapl.reset_index(drop = True)
tsla_no_index = tsla.reset_index(drop = True)
dis_no_index = dis.reset_index(drop = True)
amd_no_index = amd.reset_index(drop = True)
```

```
sp500_no_index # Daily returns for the SP500
```

```
0      -0.003240
1      -0.004662
2       0.005359
3       0.031331
4       0.009413
           ...
3171   -0.006140
3172    0.012873
3173   -0.011081
3174   -0.008830
3175    0.002195
Name: Close, Length: 3176, dtype: float64
```

```
aapl_no_index # Daily returns for Apple stocks without index
```

```
0      -0.012126
1      -0.006197
2       0.006844
3       0.040381
4      -0.002242
           ...
3171   -0.017929
3172    0.019245
3173   -0.017653
3174   -0.006912
3175    0.002456
Name: Close, Length: 3176, dtype: float64
```

We can use the Scikit-Learn's Linear Regression model to extract Beta and Alpha from the analyzed stocks.

```
# Fitting linear relation among Apple's returns and Benchmark
X = sp500_no_index.values.reshape(-1,1)
y = aapl_no_index.values.reshape(-1,1)

linreg = LinearRegression().fit(X, y)

beta = linreg.coef_[0]
alpha = linreg.intercept_
print('\n')
print('AAPL beta: ', beta.round(3))
print('\nAAPL alpha: ', alpha.round(3))
```

```
AAPL beta:  [1.111]

AAPL alpha:  [0.001]
```

```
# Fitting linear relation among Tesla's returns and Benchmark
X = sp500_no_index.values.reshape(-1,1)
y = tsla_no_index.values.reshape(-1,1)

linreg = LinearRegression().fit(X, y)

beta = linreg.coef_[0]
alpha = linreg.intercept_
print('\n')
print('TSLA beta: ', beta.round(3))
print('\nTSLA alpha: ', alpha.round(3))
```

```
TSLA beta:  [1.377]

TSLA alpha:  [0.001]
```

```
# Fitting linear relation among Walt Disney's returns and Benchmark
X = sp500_no_index.values.reshape(-1,1)
y = dis_no_index.values.reshape(-1,1)

linreg = LinearRegression().fit(X, y)

beta = linreg.coef_[0]
alpha = linreg.intercept_
print('\n')
print('Walt Disney Company beta: ', beta.round(3))
print('\nWalt Disney Company alpha: ', alpha.round(4))
```

```
Walt Disney Company beta:  [1.024]

Walt Disney Company alpha:  [0.0001]
```

```
# Fitting linear relation among AMD's returns and Benchmark
X = sp500_no_index.values.reshape(-1,1)
```

```
y = amd_no_index.values.reshape(-1,1)

linreg = LinearRegression().fit(X, y)

beta = linreg.coef_[0]
alpha = linreg.intercept_
print('\n')
print('AMD beta: ', beta.round(3))
print('\nAMD alpha: ', alpha.round(4))
```

```
AMD beta:  [1.603]

AMD alpha:  [0.0006]
```

Beta values for all the stocks are greater than 1, meaning that they are more volatile than the benchmark and may offer higher returns, but also come with greater risks. On the other hand, the alpha values for all the stocks are small, close to zero, suggesting that there is little difference between the expected returns and the risk-adjusted returns.

Sharpe Ratio

The Sharpe ratio is a measure of the risk-adjusted return of an investment. It is calculated by dividing the average excess return of the investment over the standard deviation of the returns, as shown by the following equation:

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

where $\boldsymbol{R_p}$ is the average return of the investment, $\boldsymbol{R_f}$ is the risk-free rate of return, and $\boldsymbol{\sigma_p}$ is the standard deviation of the returns. The average excess return is the difference between the average return of the

investment and the risk-free rate of return, typically represented by a government bond. The standard deviation is a measurement of the volatility of returns.

A higher Sharpe ratio indicates that an investment provides higher returns for a given level of risk compared to other investments with a lower Sharpe ratio. In general, a Sharpe ratio greater than 1 is considered good, while a Sharpe ratio less than 1 is considered poor. A Sharpe ratio of 1 means that the investment's average return is equal to the risk-free rate of return.

In general, a Sharpe ratio under 1.0 is considered bad, equal to 1.0 is considered acceptable or good, 2.0 or higher is rated as very good, and 3.0 or higher is considered excellent.

```
# Calculating Sharpe ratio
print('\n')
print("Sharpe Ratio for AAPL: ", qs.stats.sharpe(aapl).round(2))
print('\n')
print("Sharpe Ratio for TSLA: ", qs.stats.sharpe(tsla).round(2))
print('\n')
print("Sharpe Ratio for DIS: ", qs.stats.sharpe(dis).round(2))
print('\n')
print("Sharpe Ratio for AMD: ", qs.stats.sharpe(amd).round(2))
```

```
Sharpe Ratio for AAPL:  0.97


Sharpe Ratio for TSLA:  0.95


Sharpe Ratio for DIS:  0.55


Sharpe Ratio for AMD:  0.62
```

Apple and Tesla have the highest Sharpe ratios among the stocks analyzed, 0.97 and 0.95, respectively, indicating that these investments offer a better risk-return relationship. However, none of the stocks have a Sharp ratio above 1, which may suggest that these investments' average returns are beneath the risk-free rate of return.

It's important to note that the Sharpe ratio is an annual metric and, since the beginning of 2022, the market, in general, has been bearish, with prices going down over the past year.

Initial Conclusions

Some initial conclusions can be drawn via the analysis of the metrics above:

. Apple and Tesla have the best Sharpe ratios, which indicates a better risk-return relationship;

. Tesla has the highest returns of them all, but it's also more volatile than Apple and Disney;

. Apple has higher returns and low volatility compared to the other assets. It has the best Sharpe ratio, low beta, low standard deviation, and low asymmetry of returns;

. AMD is the riskier and more volatile investment option of the four. Its returns distribution is highly asymmetric, it has a high standard deviation value and high beta;

. Disney stocks may be a good option for investors that are sensitive to risk, considering they had a steady and stable return over the period.

It's possible to say that, from all the assets analyzed, Apple offers the best risk-return relationship, with high rentability and lower risk than the other options.

2 | Building and Optimizing Portfolios

What is a Portfolio?

A portfolio in financial markets is a collection of financial assets, such as stocks, bonds, commodities, and other investments, held by an individual or institution. Portfolios provide investors with a way to diversify their investments, manage risk, and increase returns.

To build a portfolio, investors must select a combination of assets that are expected to perform well under different economic and market conditions. The allocation of funds to each asset is determined by the investor's risk tolerance and investment goals. This process involves analyzing the investor's financial situation, objectives, time horizon, and risk tolerance, as well as researching and analyzing individual securities and market trends. Portfolios are dynamic and should be reviewed and adjusted periodically to reflect changes in market conditions and in the investor's financial situation, or goals.

The weights in a portfolio refer to the percentage of the total value allocated to each individual asset. Allocating weights is a critical aspect of portfolio building because it determines the level of risk and return characteristics of the portfolio. The weight assigned to an asset reflects

the investor's confidence in the asset's ability to generate returns and their willingness to accept its associated risk. Weights can be determined by analyzing an asset's historical performance, future growth prospects, sector exposure, and diversification benefits. Portfolio managers may use various techniques, such as modern portfolio theory and factor-based investing, to determine optimal weightings. Getting the weightings right is crucial to achieving the desired outcomes and is a key factor in the success of any investment strategy.

To start exploring portfolio construction and optimization, we ought to build a portfolio consisting of the four stocks that have been analyzed so far, with an initial weighting of 25% each.

```
weights = [0.25, 0.25, 0.25, 0.25] # Defining weights for each stock
portfolio = aapl*weights[0] + tsla*weights[1] + dis*weights[2] +
        amd*weights[3] # Creating portfolio multiplying each stock
        for its respective weight
portfolio # Displaying portfolio's daily returns
```

```
Date
2010-07-01 04:00:00   -0.020338
2010-07-02 04:00:00   -0.041286
2010-07-06 04:00:00   -0.040348
2010-07-07 04:00:00    0.028905
2010-07-08 04:00:00    0.026538
                           ...
2023-02-06 05:00:00   -0.007087
2023-02-07 05:00:00    0.018110
2023-02-08 05:00:00   -0.001937
2023-02-09 05:00:00   -0.001783
2023-02-10 05:00:00   -0.022371
Name: Close, Length: 3176, dtype: float64
```

With Quantstats you can easily create a report to compare the portfolio's performance and its level of risk with a benchmark, which in this case is the SP500. The platform provides various metrics and useful visualizations to analyze the portfolio's performance and risk.

```
# Generating report on portfolio performance from July 1st, 2010 to
        February 10th, 2023
qs.reports.full(portfolio, benchmark = sp500)
```

Performance Metrics

	Strategy	Benchmark
Start Period	2010-07-01	2010-07-01
End Period	2023-02-10	2023-02-10
Risk-Free Rate	0.0%	0.0%
Time in Market	100.0%	100.0%
Cumulative Return	3,429.90%	296.86%
CAGR%	32.62%	11.54%
Sharpe	1.08	0.71
Prob. Sharpe Ratio	99.99%	99.33%
Smart Sharpe	1.04	0.68
Sortino	1.58	0.99
Smart Sortino	1.53	0.95
Sortino/√2	1.12	0.7
Smart Sortino/√2	1.08	0.68
Omega	1.21	1.21
Max Drawdown	-52.21%	-33.92%
Longest DD Days	404	416
Volatility (ann.)	30.59%	17.69%
R^2	0.55	0.55
Information Ratio	0.06	0.06
Calmar	0.62	0.34
Skew	-0.08	-0.5
Kurtosis	3.9	12.87
Expected Daily %	0.11%	0.04%
Expected Monthly %	2.37%	0.91%
Expected Yearly %	28.99%	10.35%
Kelly Criterion	7.01%	3.22%
Risk of Ruin	0.0%	0.0%
Daily Value-at-Risk	-3.04%	-1.78%
Expected Shortfall (cVaR)	-3.04%	-1.78%
Max Consecutive Wins	13	8
Max Consecutive Losses	8	9
Gain/Pain Ratio	0.21	0.15
Gain/Pain (1M)	1.35	0.9

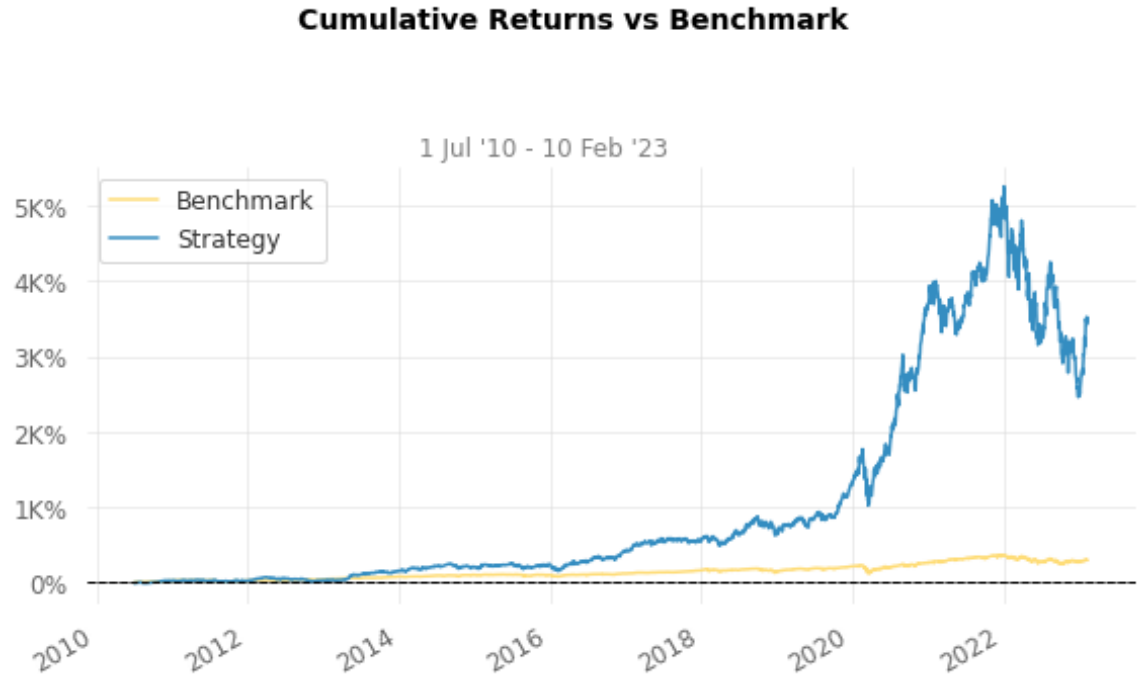
Payoff Ratio	0.95	0.9
Profit Factor	1.21	1.15
Common Sense Ratio	1.26	1.09
CPC Index	0.63	0.56
Tail Ratio	1.04	0.95
Outlier Win Ratio	2.81	5.46
Outlier Loss Ratio	2.91	5.48
MTD	6.71%	0.34%
3M	22.42%	9.12%
6M	-14.88%	-2.84%
YTD	31.62%	6.54%
1Y	-26.29%	-10.83%
3Y (ann.)	35.01%	12.74%
5Y (ann.)	39.74%	8.11%
10Y (ann.)	38.55%	10.42%
All-time (ann.)	32.62%	11.54%
Best Day	13.76%	9.38%
Worst Day	-12.65%	-11.98%
Best Month	30.34%	12.68%
Worst Month	-19.49%	-12.51%
Best Year	172.25%	29.6%
Worst Year	-47.22%	-19.44%
Avg. Drawdown	-4.46%	-1.75%
Avg. Drawdown Days	27	18
Recovery Factor	65.7	8.75
Ulcer Index	0.13	0.07
Serenity Index	19.5	3.8
Avg. Up Month	8.42%	3.5%
Avg. Down Month	-6.95%	-4.26%
Win Days %	54.72%	54.24%
Win Month %	61.18%	66.45%
Win Quarter %	66.67%	76.47%
Win Year %	92.86%	71.43%
Beta	1.28	-
Alpha	0.17	-
Correlation	73.94%	-
Treynor Ratio	2682.16%	-

None

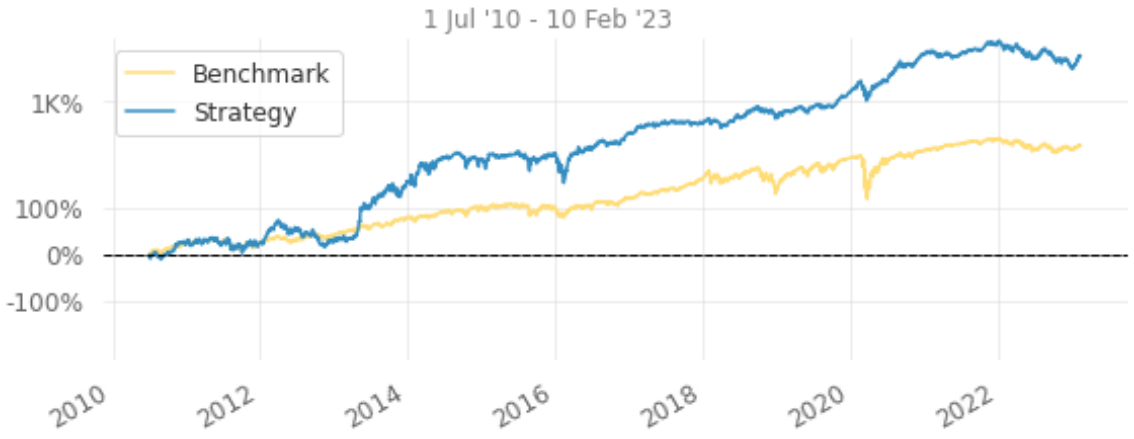
5 Worst Drawdowns

	Start	Valley	End	Days	Max Drawdown	99% Max Drawdown
1	2022-01-04	2022-12-28	2023-02-10	402	-52.208458	-51.448122
2	2020-02-20	2020-03-18	2020-06-09	109	-39.974767	-38.637689
3	2012-03-28	2012-11-15	2013-05-06	404	-33.340372	-31.760458
4	2015-07-06	2016-02-10	2016-04-22	291	-28.967480	-27.003148
5	2018-10-02	2018-12-24	2019-07-03	274	-25.400777	-22.377614

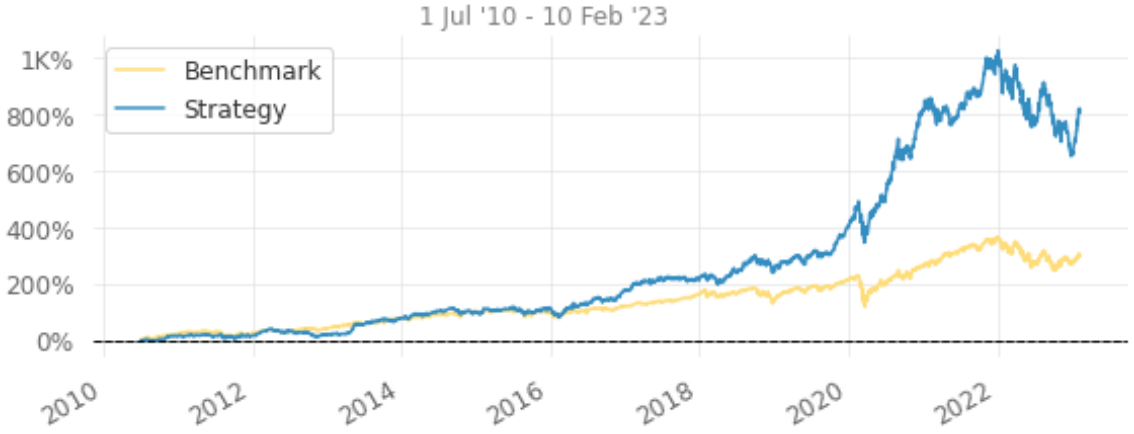
Strategy Visualization



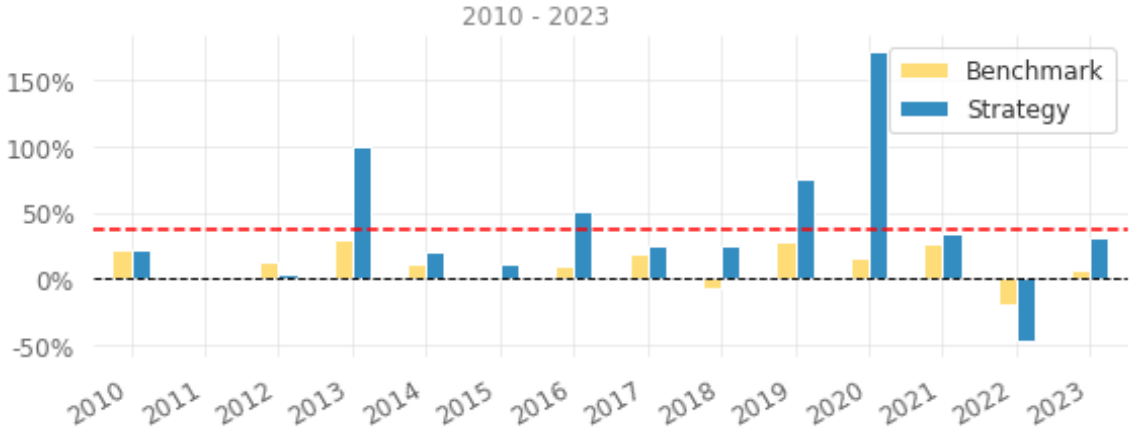
Cumulative Returns vs Benchmark (Log Scaled)



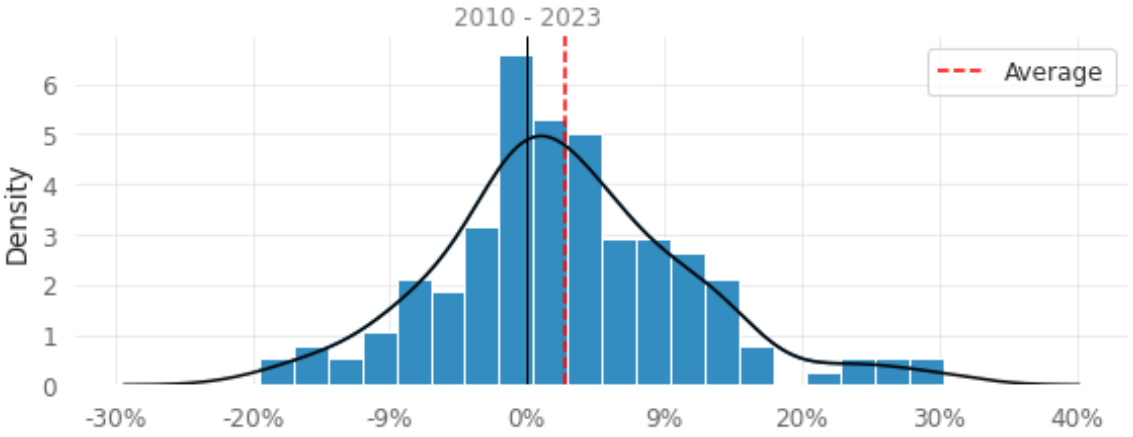
Cumulative Returns vs Benchmark (Volatility Matched)



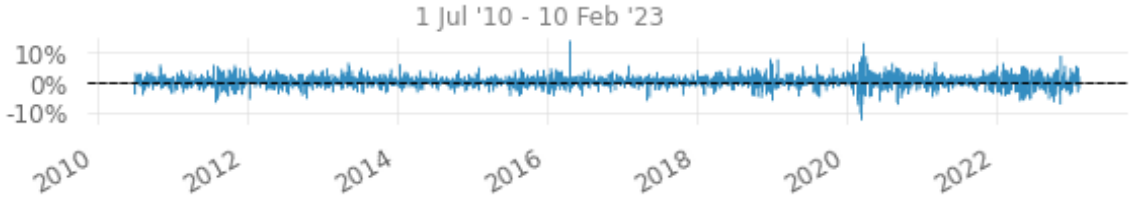
EOY Returns vs Benchmark



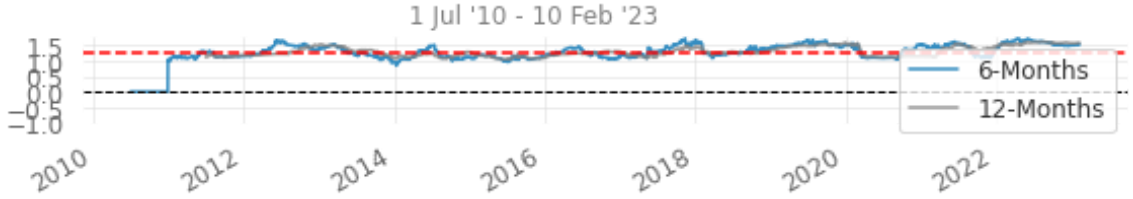
Distribution of Monthly Returns



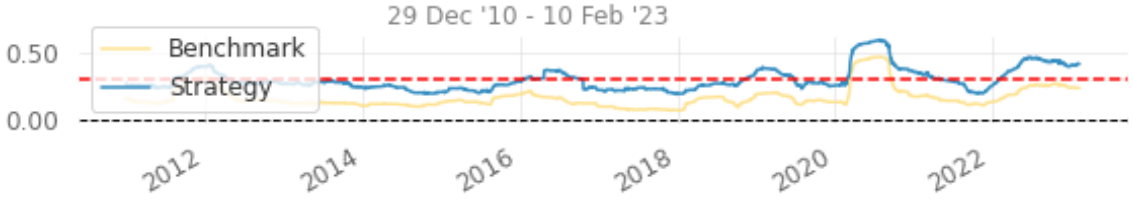
Daily Returns



Rolling Beta to Benchmark

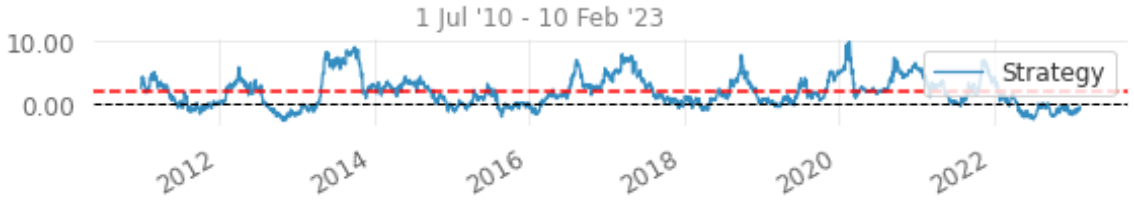


Rolling Volatility (6-Months)

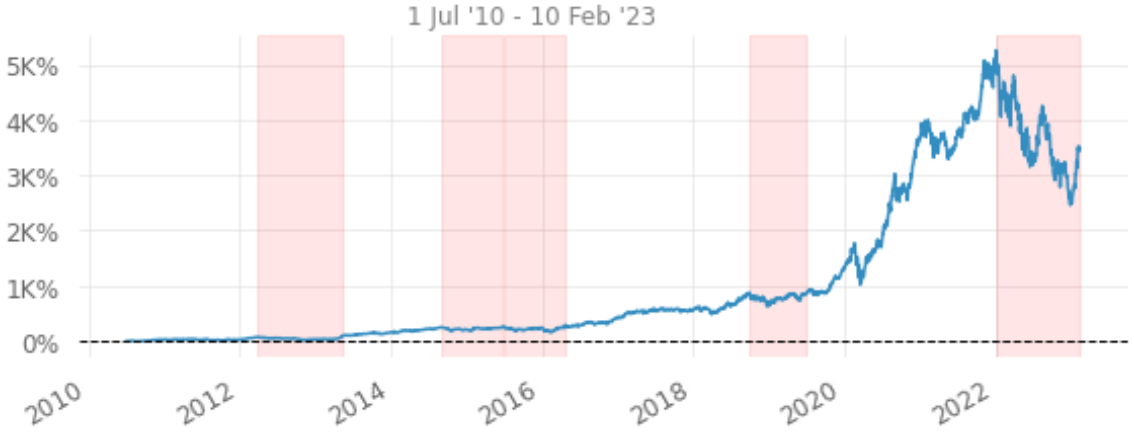


Rolling Sharpe (6-Months)

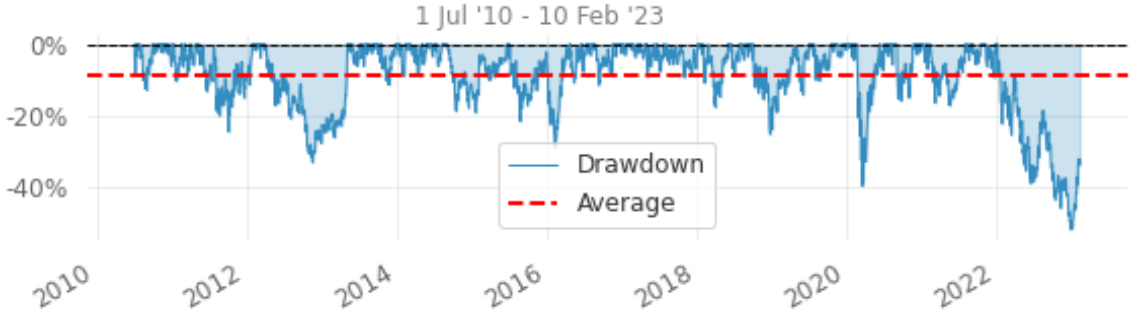
Rolling Sortino (6-Months)

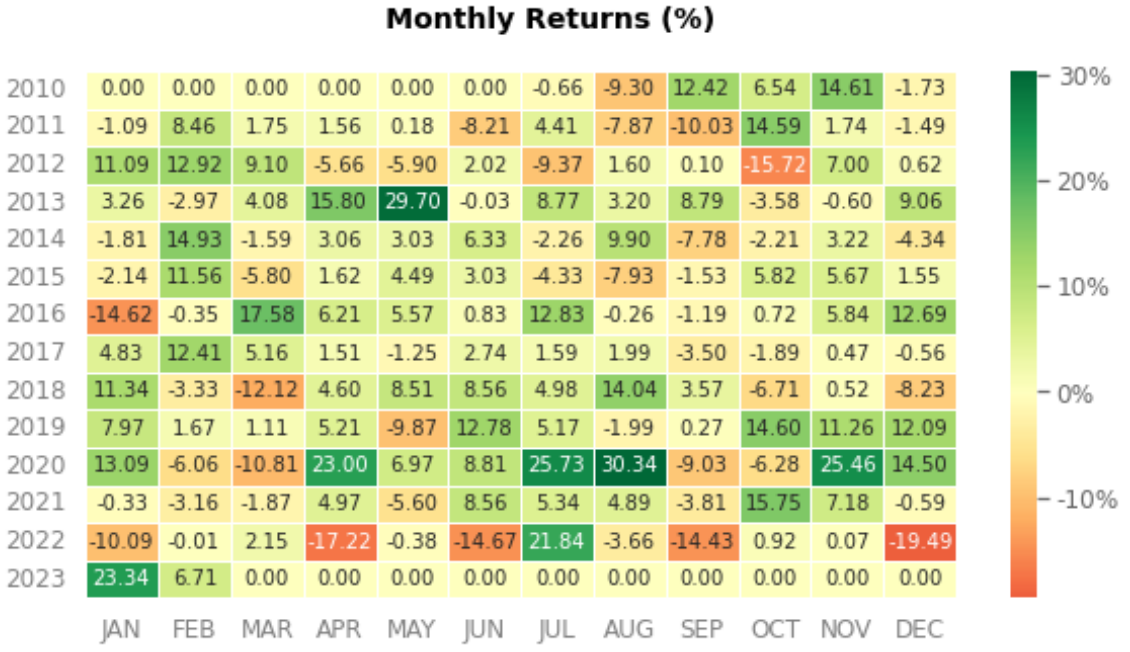


Worst 5 Drawdown Periods



Underwater Plot





We have a range of metrics and plots available to look at. Firstly, the Cumulative Return of the portfolio is higher than the benchmark, at 3,429.9% compared to 296.86% for the SP500. The Sharpe Ratio and Sortino Ratio of the portfolio are also higher, indicating that it generates better returns for the level of risk taken. In addition, the portfolio has higher expected daily, monthly and annual returns than the SP500, and its best day, month, and year outperforms the benchmark's best day, month, and year.

However, the portfolio's maximum drawdown is greater than the benchmark, at -52.21% compared to -33.92%. This indicates that the portfolio has experienced larger losses at times than the benchmark. The

annualized volatility of the portfolio is also higher, at around 30.59% compared to the benchmark's 17.96%.

While the portfolio has higher returns on its best day, month, and year, it also has bigger losses on its worst day, month, and year compared to the benchmark. The beta of 1.28 shows that the portfolio is about 28% more volatile than the overall market, and its 73.94% correlation indicates a strong positive relationship among the four stocks, suggesting that they tend to move in the same direction, which could increase the systemic risk of the portfolio.

Overall, the portfolio has generated impressive returns, but it also comes with a higher degree of risk and volatility. This prompts the question of whether or not it's possible to optimize the portfolio to reduce risk and volatility while also increasing returns.

Optimizing Portfolio

Portfolio optimization is the process of selecting the optimal combination of assets and weights to maximize returns and minimize risk. This process involves selecting the most appropriate weights for each asset,by taking into account the historical performance of the assets, their correlations with each other, and other relevant factors such as market conditions and economic outlook. The main goal is to create a well-diversified portfolio that balances risk and returns, and that aligns with the investor's risk tolerance.

In Python, PyPortfolioOpt is a very useful library for portfolio optimization, providing an efficient and easy-to-use toolset for constructing optimal portfolios, with a wide range of optimization algorithms and methods available. These include modern portfolio theory, minimum variance portfolio, maximum diversification portfolio, Black-Litterman optimization, and many more.

To start the optimization process, we must have a pandas dataframe containing the adjusted closing prices of the stocks, with dates as index, and each columns representing each stock. This dataframe will serve as input to optimize the weighting of the stocks in the portfolio

```
# Getting dataframes info for Stocks using yfinance
aapl_df = yf.download('AAPL', start = '2010-07-01', end = '2023-02-
        11')
tsla_df = yf.download('TSLA', start = '2010-07-01', end = '2023-02-
        11')
dis_df = yf.download('DIS', start = '2010-07-01', end = '2023-02-11')
amd_df = yf.download('AMD', start = '2010-07-01', end = '2023-02-11')
```

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

```
# Extracting Adjusted Close for each stock
aapl_df = aapl_df['Adj Close']
tsla_df = tsla_df['Adj Close']
dis_df = dis_df['Adj Close']
amd_df = amd_df['Adj Close']
```

```
# Merging and creating an Adj Close dataframe for stocks
df = pd.concat([aapl_df, tsla_df, dis_df, amd_df], join = 'outer',
        axis = 1)
df.columns = ['aapl', 'tsla', 'dis', 'amd']
df # Visualizing dataframe for input
```

	aapl	tsla	dis	amd
Date				
2010-07-01	7.553068	1.464000	27.428722	7.390000
2010-07-02	7.506257	1.280000	27.332911	7.170000
2010-07-06	7.557627	1.074000	27.629063	7.040000
2010-07-07	7.862813	1.053333	28.865921	7.390000
2010-07-08	7.845182	1.164000	29.040131	7.370000
...	...	...	...	...
2023-02-06	151.498688	194.759995	109.870003	83.680000
2023-02-07	154.414230	196.809998	111.629997	85.910004
2023-02-08	151.688400	201.289993	111.779999	84.690002

	aapl	tsla	dis	amd
Date				
2023-02-09	150.639999	207.320007	110.360001	83.209999
2023-02-10	151.009995	196.889999	108.059998	81.480003

3176 rows × 4 columns

The dataframe above will be used as input for the algorithms to optimize the portfolio

```
# Importing libraries for portfolio optimization
from pypfopt.efficient_frontier import EfficientFrontier
from pypfopt import risk_models
from pypfopt import expected_returns
```

Markowitz Mean-Variance Optimization Model

The Markowitz Mean-Variance Optimization Model is a widely-used framework for constructing portfolios with the best risk-return relationship. It is based on the idea that investors should maximize the expected return of a portfolio while minimizing its risk.

PyPortfolioOpt simplifies the implementation of the Markowitz Mean-Variance Optimization Model, allowing investors to easily determine how to allocate weights across a portfolio for maximal Sharpe ratio, besides many other different objectives according to each investor's risk tolerance and goals.

There are two key requirements for mean-variance optimization:

First, we need to have expected returns for each of the assets in the portfolio. PyPortfolioOpt provides the *expected_returns* module, which calculates expected returns for the assets by computing the arithmetic

mean of their daily percentage changes. The module assumes that daily prices are available as input and produces expected annual returns as output. More information on this topic is available here.

Secondly, we need to choose a risk model that quantifies the level of risk in each asset. The most commonly used risk model is the covariance matrix, which describes the volatilities of assets and the degree to which they are co-dependent. Choosing an appropriate risk model is critical, because it can help to reduce risk by making many uncorrelated bets. PyPortfolioOpt offers a range of risk models to choose from, including the annualized sample covariance matrix of daily returns, semicovariance matrix, and exponentially-weighted covariance matrix. Further information on risk models can be found here

.

```
# Calculating the annualized expected returns and the annualized
        sample covariance matrix
mu = expected_returns.mean_historical_return(df) #expected returns
S = risk_models.sample_cov(df) #Covariance matrix

# Visualizing the annualized expected returns
mu
```

```
aapl    0.268385
tsla    0.475549
dis     0.114966
amd     0.209862
dtype: float64
```

```
# Visualizing the covariance matrix
S
```

	aapl	tsla	dis	amd
aapl	0.081699	0.058321	0.031096	0.063529
tsla	0.058321	0.329160	0.043762	0.101222
dis	0.031096	0.043762	0.068223	0.047868
amd	0.063529	0.101222	0.047868	0.326817

Now that we have estimated the expected returns and the covariance matrix, we can use these inputs for portfolio optimization.

The PyPortfolioOpt library provides the EfficientFrontier class, which takes the covariance matrix and expected returns as inputs. The weights variable stores the optimized weights for each asset based on the specified objective, which in this case is the maximization of the Sharpe ratio, achieved by using the *max_sharpe* method.

PyPortfolioOpt offers various other optimization objectives, such as weights optimized for minimum volatility, maximum returns for a given target risk, maximum quadratic utility, and many others. To read more on optimization objectives, click here.

```
# Optimizing for maximal Sharpe ratio
ef = EfficientFrontier(mu, S) # Providing expected returns and
        covariance matrix as input
weights = ef.max_sharpe() # Optimizing weights for Sharpe ratio
        maximization

clean_weights = ef.clean_weights() # clean_weights rounds the weights
        and clips near-zeros

# Printing optimized weights and expected performance for portfolio
clean_weights
```

```
OrderedDict([('aapl', 0.70828), ('tsla', 0.29172), ('dis', 0.0),
('amd', 0.0)])
```

After running the optimizer, it resulted in an optimized weighting for a portfolio where 70.83% of its allocation is invested in Apple stocks, and the remaining 29.17% invested in Tesla stocks. No allocation was made to Disney or AMD.

With the optimized weights in hand, we can construct a new portfolio and use Quantstats to compare its performance to that of the previously constructed portfolio.

```
# Creating new portfolio with optimized weights
new_weights = [0.70828, 0.29172]
optimized_portfolio = aapl*new_weights[0] + tsla*new_weights[1]
optimized_portfolio # Visualizing daily returns

Date
2010-07-01 04:00:00   -0.031481
2010-07-02 04:00:00   -0.041054
2010-07-06 04:00:00   -0.042102
2010-07-07 04:00:00    0.022988
2010-07-08 04:00:00    0.029061
                          ...
2023-02-06 05:00:00   -0.005359
2023-02-07 05:00:00    0.016701
2023-02-08 05:00:00   -0.005863
2023-02-09 05:00:00    0.003844
2023-02-10 05:00:00   -0.012936
Name: Close, Length: 3176, dtype: float64

# Displaying new reports comparing the optimized portfolio to the
        first portfolio constructed
qs.reports.full(optimized_portfolio, benchmark = portfolio)
```

Performance Metrics

	Strategy	Benchmark
Start Period	2010-07-01	2010-07-01
End Period	2023-02-10	2023-02-10
Risk-Free Rate	0.0%	0.0%
Time in Market	100.0%	100.0%
Cumulative Return	4,830.76%	3,429.90%
CAGR%	36.18%	32.62%
Sharpe	1.17	1.08
Prob. Sharpe Ratio	100.0%	99.99%
Smart Sharpe	1.12	1.04
Sortino	1.72	1.58
Smart Sortino	1.65	1.52
Sortino/√2	1.21	1.12
Smart Sortino/√2	1.17	1.08
Omega	1.23	1.23

Max Drawdown	-45.96%	-52.21%
Longest DD Days	566	404
Volatility (ann.)	30.52%	30.59%
R^2	0.74	0.74
Information Ratio	0.01	0.01
Calmar	0.79	0.62
Skew	-0.16	-0.08
Kurtosis	3.79	3.9
Expected Daily %	0.12%	0.11%
Expected Monthly %	2.6%	2.37%
Expected Yearly %	32.11%	28.99%
Kelly Criterion	7.94%	8.07%
Risk of Ruin	0.0%	0.0%
Daily Value-at-Risk	-3.02%	-3.04%
Expected Shortfall (cVaR)	-3.02%	-3.04%
Max Consecutive Wins	12	13
Max Consecutive Losses	9	8
Gain/Pain Ratio	0.23	0.21
Gain/Pain (1M)	1.42	1.35
Payoff Ratio	0.99	0.97
Profit Factor	1.23	1.21
Common Sense Ratio	1.28	1.26
CPC Index	0.66	0.64
Tail Ratio	1.05	1.04
Outlier Win Ratio	3.75	3.8
Outlier Loss Ratio	3.55	3.46
MTD	7.43%	6.71%
3M	13.07%	22.42%
6M	-16.35%	-14.88%
YTD	28.36%	31.62%
1Y	-18.92%	-26.29%
3Y (ann.)	45.48%	35.01%
5Y (ann.)	41.49%	39.74%
10Y (ann.)	37.81%	38.55%
All-time (ann.)	36.18%	32.62%
Best Day	11.86%	13.76%
Worst Day	-14.53%	-12.65%
Best Month	35.64%	30.34%

Worst Month	-19.82%	-19.49%
Best Year	203.82%	172.25%
Worst Year	-39.08%	-47.22%
Avg. Drawdown	-4.32%	-4.46%
Avg. Drawdown Days	24	27
Recovery Factor	105.12	65.7
Ulcer Index	0.13	0.13
Serenity Index	31.72	19.5
Avg. Up Month	9.77%	9.15%
Avg. Down Month	-5.9%	-6.47%
Win Days %	54.09%	54.72%
Win Month %	59.21%	61.18%
Win Quarter %	72.55%	66.67%
Win Year %	92.86%	92.86%
Beta	0.86	-
Alpha	0.07	-
Correlation	86.12%	-
Treynor Ratio	5623.07%	-

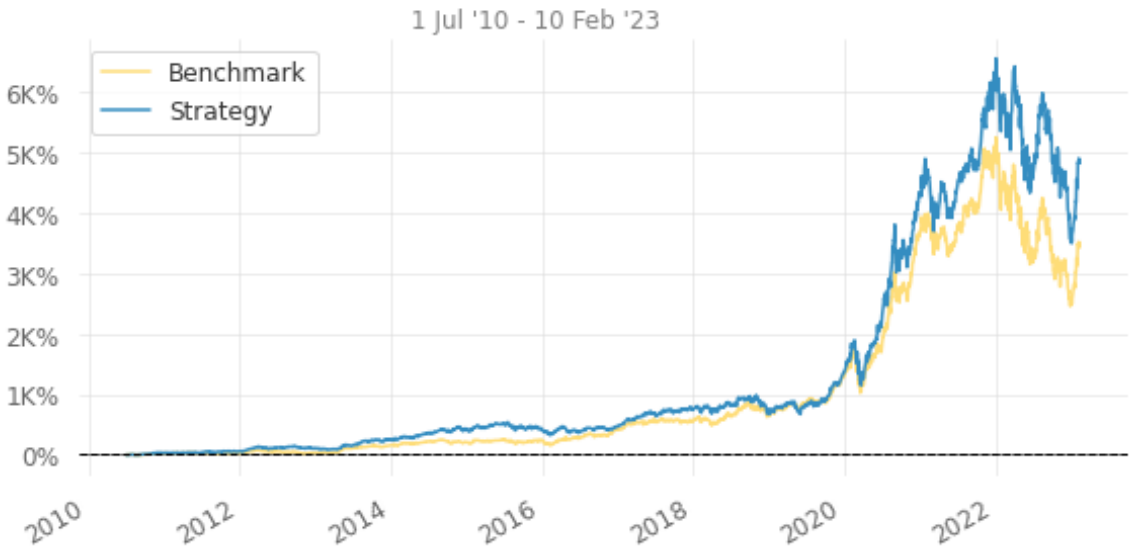
None

5 Worst Drawdowns

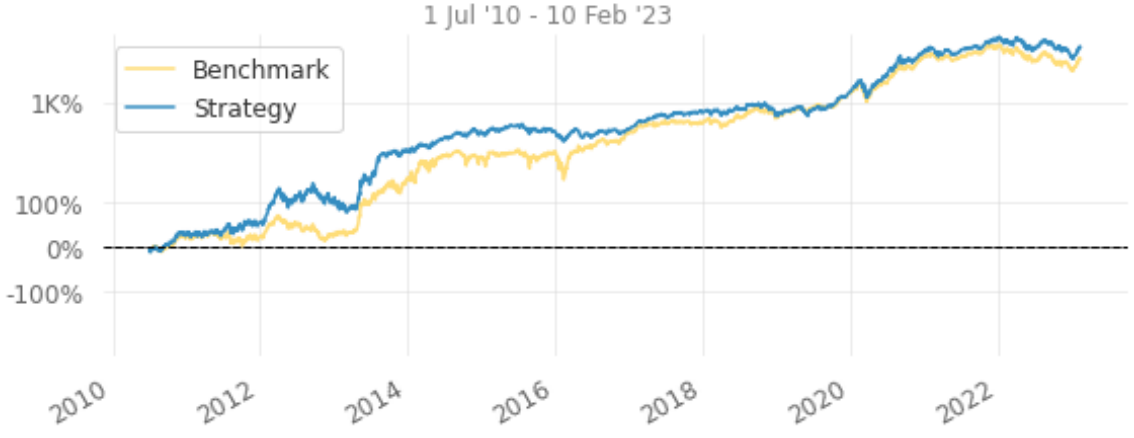
	Start	Valley	End	Days	Max Drawdown	99% Max Drawdown
1	2022-01-04	2023-01-03	2023-02-10	402	-45.956182	-44.753398
2	2020-02-20	2020-03-23	2020-06-01	101	-37.293181	-36.695389
3	2015-07-21	2016-02-10	2017-02-06	566	-33.810390	-32.844632
4	2018-11-02	2019-01-03	2019-10-22	354	-29.444364	-27.889058
5	2012-09-18	2013-03-04	2013-05-13	237	-27.126556	-25.178782

Strategy Visualization

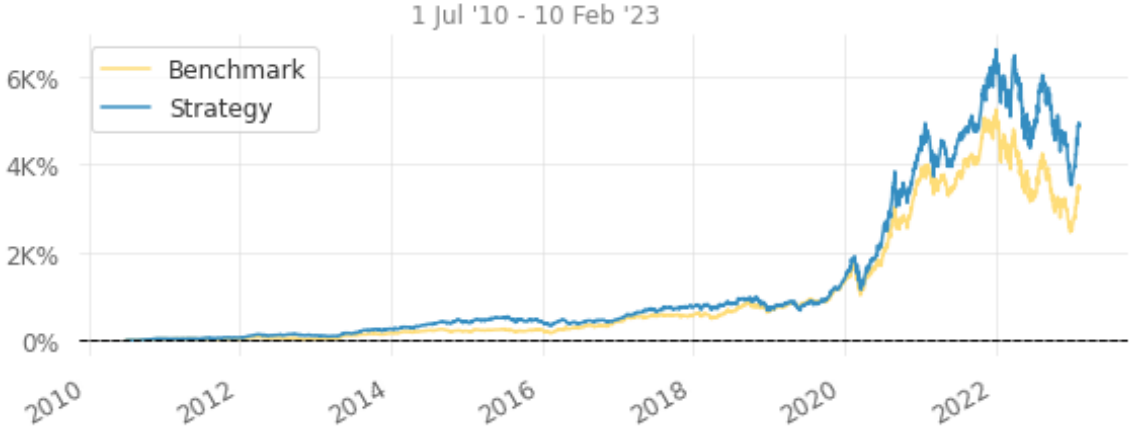
Cumulative Returns vs Benchmark



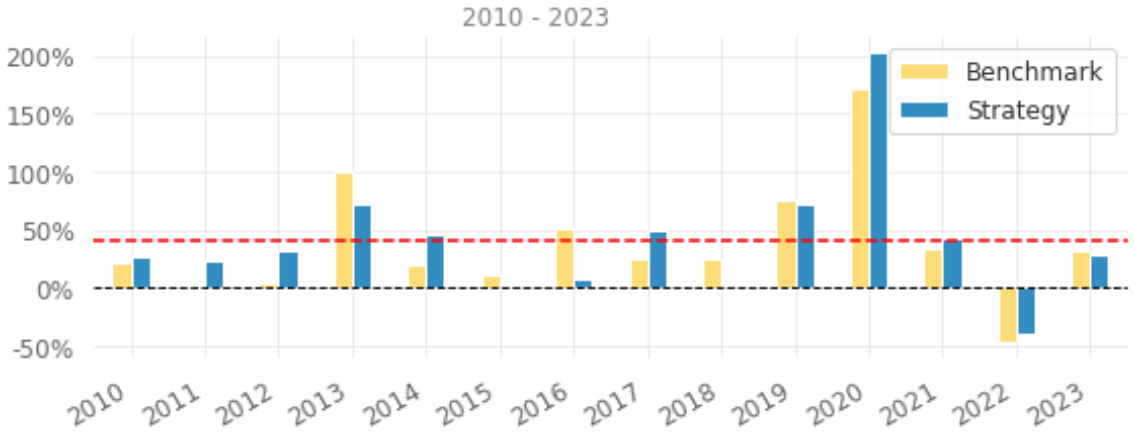
Cumulative Returns vs Benchmark (Log Scaled)



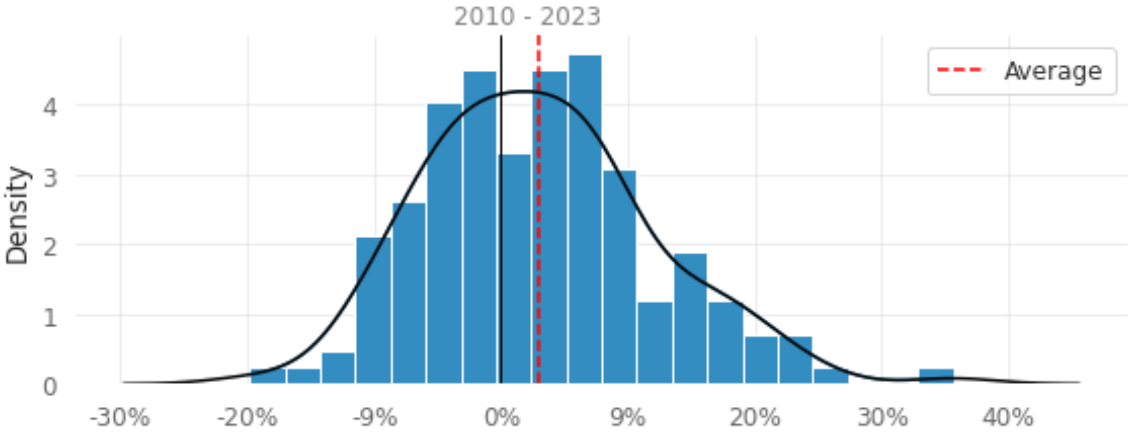
Cumulative Returns vs Benchmark (Volatility Matched)



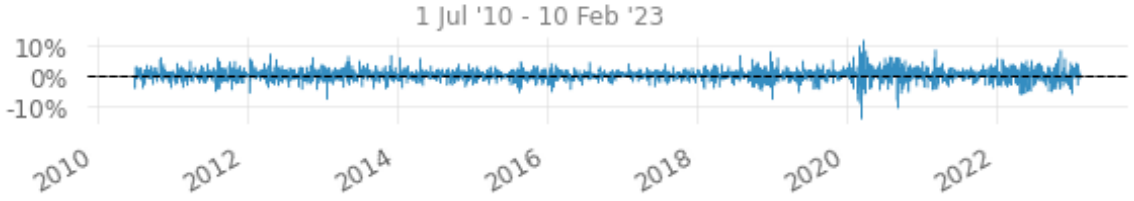
EOY Returns vs Benchmark



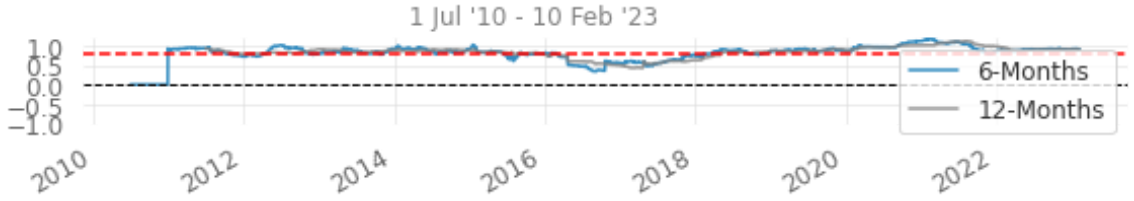
Distribution of Monthly Returns



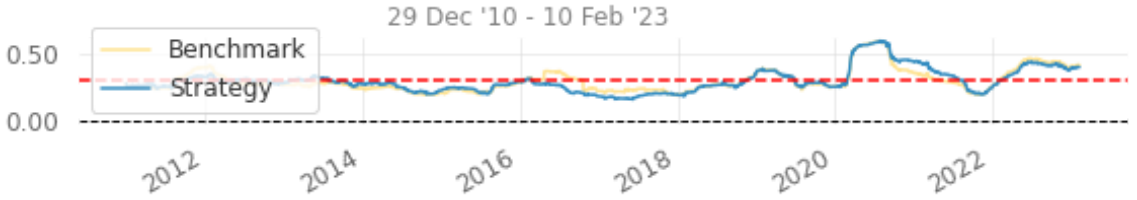
Daily Returns



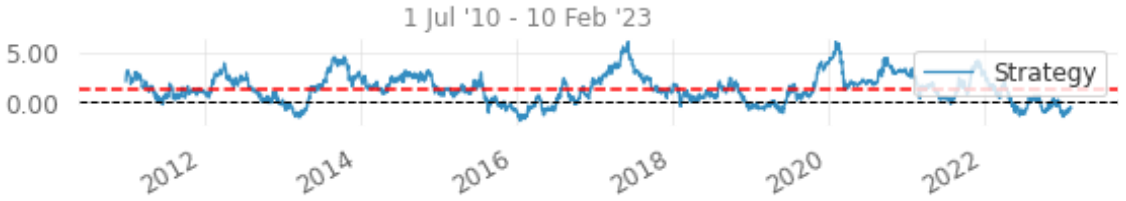
Rolling Beta to Benchmark



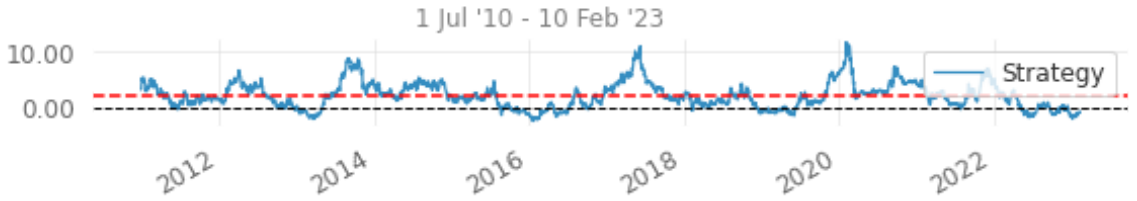
Rolling Volatility (6-Months)



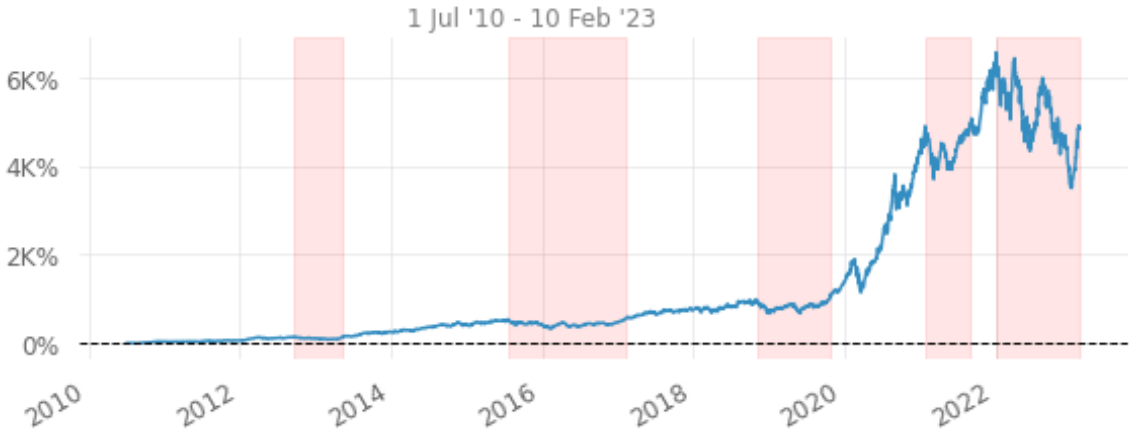
Rolling Sharpe (6-Months)

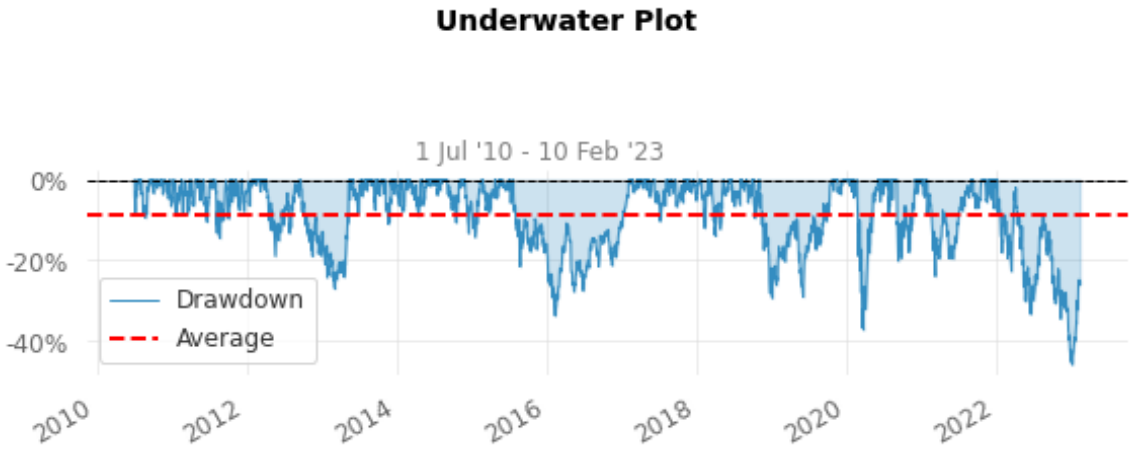


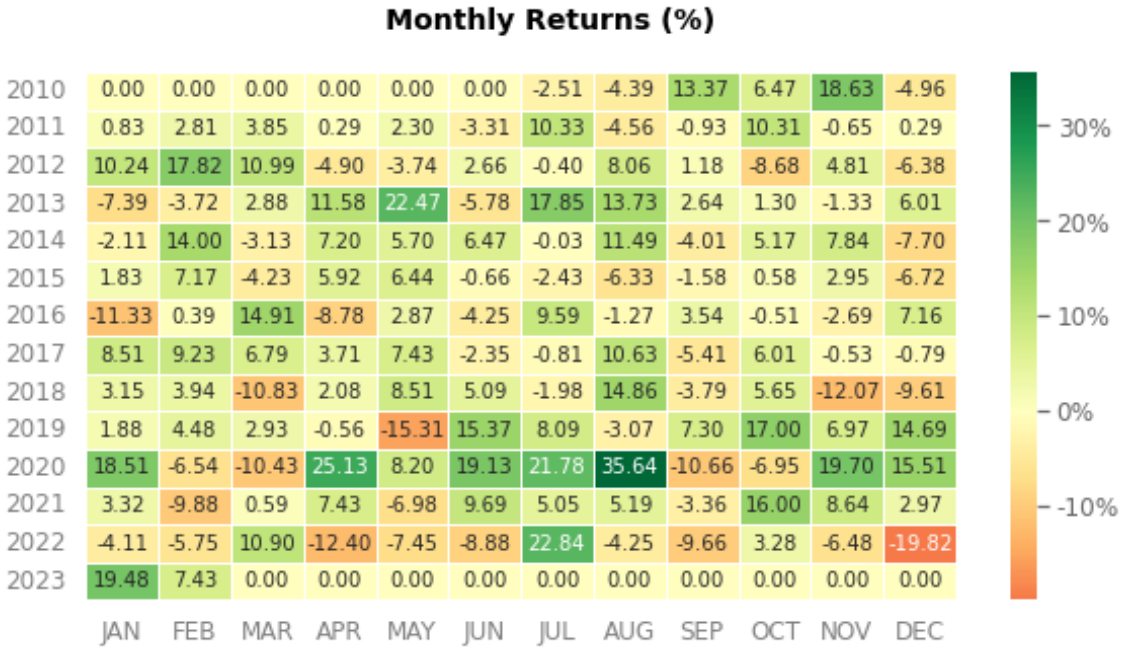
Rolling Sortino (6-Months)



Worst 5 Drawdown Periods







Based on the report above, the optimized portfolio appears to have performed better than the original portfolio. Here are some key conclusions that can be drawn by looking at the metrics and plots in the report:

. **Cumulative Return:** The optimized portfolio has generated a significantly higher cumulative return of 4,830.76% compared to 3,429.90% for the original portfolio. This means that the an investment in the optimized portfolio would have provided higher earnings for investors compared to the original portfolio.

. **CAGR:** The compounded annual growth rate (CAGR) of the optimized portfolio is higher at 36.18% compared to 32.62% for the original portfolio. This suggests that the optimized portfolio has generated a higher rate of return per year over the entire investment period.

. **Sharpe Ratio:** The optimized portfolio has a slightly higher Sharpe ratio of 1.17 compared to 1.08 for the original portfolio, indicating that it has generated a better risk-adjusted return.

. **Drawdown:** The maximum drawdown for the optimized portfolio is lower at -45.96% compared to -52.21% for the original portfolio. This means that the optimized portfolio experienced lower losses during the worst period of performance.

. **Recovery Factor:** The recovery factor for the optimized portfolio is much higher at 105.12 compared to 65.7 for the original portfolio, which suggests that the optimized portfolio was able to recover from drawdowns more quickly and generate higher returns after experiencing losses.

. **Win Rates:** The optimized portfolio has slightly higher win rates for win days, win months, win quarters, and win years, indicating that it had a higher probability of generating positive returns over these periods.

. **Beta:** The optimized portfolio's beta of 0.86 indicates that the optimized portfolio is less volatile than the overall market, and much less volatile than the previously built portfolio.

. **Annual Volatility:** The optimized portfolio has a slightly lower annual volatility than the original portfolio, with 30.52% compared to 30.59%, respectively.

Black-Litterman Allocation Model

In 1992, Fischer Black and Robert Litterman introduced the Black-Litterman Allocation Model, which takes a Bayesian approach to asset allocation. It combines a prior estimate of returns with the investor's particular views on his/her expected returns to generate an optimal allocation. Multiple sources of information can be used to establish the prior estimate of returns, and the model allows investors to provide a confidence level for their views, which is then used to optimize allocation.

The Black-Litterman formula calculates a weighted average between the prior estimate of returns and the views, with the weighting determined by the level of confidence for each view.

Prior

A commonly used approach for determining a prior estimate of returns involves relying on the market's expectations, which are reflected in the asset's market capitalization.

To do this, we first need to estimate the level of risk aversion among market participants, represented by a parameter known as ***delta***, which we calculate using the closing prices of the SP500. The higher the value for **delta**, the greater the market's risk aversion.

With this information, we can calculate the prior expected returns for each stock based on its market capitalization,the ***delta***, and the covariance matrix ***S***, which we've obtained before optimizing our portfolio with the Markowitz Mean-Variance Model. These prior expected returns gives us a starting point for the expected returns before we incoporate any of our views as investors.

Views

In the Black-Litterman model, investors can express their views as either absolute or relative. Absolute views involve statements like "APPL will return 10%", while relative views are represented by statements such as "AMZN will outperform AMD by 10%".

These views must be specified in the vector ***Q*** and mapped into each asset via the picking matrix ***P***.

For instance, let's consider a portfolio with 10 assets:

1. TSLA

2. AAPL

3. NVDA

4. MSFT

5. META

6. AMZN

7. AMD

8. HD

9. GOOGL

10. BRKa

And then consider two absolute views and two relative views, such as:

1. TSLA will raise by 20%

2. APPL will drop by 15%

3. HD will outperform META by 10%

4. GOOGL and BRKa will outperform MSTF and AMZN by 5%

The views vector would be formed by taking the numbers above and specifying them as below:

```
Q = np.array([0.20, -0.15, 0.10,
0.05]).reshape(-1,1)
```

The picking matrix would then be used to link the views of the 8 mentioned assets above to the universe of 10 assets, allowing us to propagate our expectations into the model:

```
P = np.array([
[1,0,0,0,0,0,0,0,0,0], [0,1,0,0,0,0,0,0,0,0], [0,0,0,0,-1,0,0,1,0,0],
[0,0,0,-0.5,0,-0.5,0,0,0.5,0.5], ])
```

Absolute views have a single 1 in the column corresponding to the asset's order in the asset universe, while relative views have a positive number in the outperforming asset column, and a negative number in the underperforming asset column. Each row for relative views in $\boldsymbol{P}$ must sum up to 0, and the order of views in $\boldsymbol{Q}$ must correspond to the order of rows in $\boldsymbol{P}$.

Confidences

The confidence matrix is used to help to define the allocations in each stock. It can be implemented using the Idzorek's method, allowing investors to specify their confidence level in each of their views as a

percentage. The values in the confidence matrix range from 0 to 1, where 0 indicates a low level of confidence in the view, and 1 indicates a high level of confidence.

By using the confidence matrix, investors can better understand the potential impact of their views on their allocations. For example, if an investor has a high level of confidence in their view on a particular asset, they may choose to allocate a larger portion of their portfolio to that asset. On the other hand, if an investor has a low level of confidence in their view, they may choose to allocate a smaller portion of their portfolio or avoid the asset altogether.

For more information on the Black-Litterman Allocation Model, I highly suggest you read this session on the PyPortfolioOpt documentation.

```
# Mapping assets
assets = ['AAPL', 'TSLA', 'DIS', 'AMD']

# Obtaining market cap for stocks
market_caps = data.get_quote_yahoo(assets)['marketCap']
market_caps # Visualizing market caps for stocks

AAPL    2518055124992
TSLA     624150380544
DIS      176707338240
AMD      155483013120
Name: marketCap, dtype: int64

# Obtaining closing prices for the SP500
market_prices = yf.download("^GSPC",start = '2010-07-01', end = '2023-
        02-11')['Adj Close']
market_prices # Visualizing closing prices for the SP500

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

Date
2010-07-01    1027.369995
2010-07-02    1022.580017
2010-07-06    1028.060059
2010-07-07    1060.270020
2010-07-08    1070.250000
                  ...
2023-02-06    4111.080078
2023-02-07    4164.000000
```

```
2023-02-08    4117.859863
2023-02-09    4081.500000
2023-02-10    4090.459961
Name: Adj Close, Length: 3176, dtype: float64
```

```
# Obtaining market-implied risk aversion, the delta
delta = black_litterman.market_implied_risk_aversion(market_prices)
delta # Visualizing delta
```

```
3.3668161617990653
```

```
# Visualizing Covariance Matrix
S
```

	aapl	tsla	dis	amd
aapl	0.081699	0.058321	0.031096	0.063529
tsla	0.058321	0.329160	0.043762	0.101222
dis	0.031096	0.043762	0.068223	0.047868
amd	0.063529	0.101222	0.047868	0.326817

```
# Changing columns and index to uppercase so it matches market_caps
S.index = S.index.str.upper()
S.columns = S.columns.str.upper()
S
```

	AAPL	TSLA	DIS	AMD
AAPL	0.081699	0.058321	0.031096	0.063529
TSLA	0.058321	0.329160	0.043762	0.101222
DIS	0.031096	0.043762	0.068223	0.047868
AMD	0.063529	0.101222	0.047868	0.326817

```
# Obtaining Prior estimates
prior = black_litterman.market_implied_prior_returns(market_caps,
        delta, S)
prior # Visualizing prior estimates
```

```
AAPL    0.269523
TSLA    0.384137
DIS     0.141240
AMD     0.293677
dtype: float64
```

Now that we have our prior estimates for each stock, we can now provide the model our views on these stocks and our confidence levels in our views.

```
# APPL will raise by 5%
# TSLA will raise by 10%
# AMD will outperform Disney by 15%

Q = np.array([0.05, 0.10, 0.15])

# Linking views to assets
P = np.array([
    [1,0,0,0], # APPL = 0.05
    [0,1,0,0], # TSLA = 0.10
    [0,0,-1,1] # AMD > DIS by 0.15
])

# Providing confidence levels
# Closer to 0.0 = Low confidence
# Closer to 1.0 = High confidence
confidences = [0.5,
               0.4,
               0.8]

# Creating model
bl = BlackLittermanModel(S, # Covariance Matrix
                         pi = prior, # Prior expected returns
                         Q = Q, # Vector of views
                         P = P, # Matrix mapping the views
                         omega = 'idzorek', # Method to estimate
        uncertainty level of the views based on historical data
                         view_confidences = confidences) # Confidences

rets = bl.bl_returns() # Calculating Expected returns
ef = EfficientFrontier(rets, S) # Optimizing asset allocation

ef.max_sharpe() # Optimizing weights for maximal Sharpe ratio
weights = ef.clean_weights() # Cleaning weights
weights # Printing weights

OrderedDict([('AAPL', 0.63718),
             ('TSLA', 0.18636),
             ('DIS', 0.01442),
             ('AMD', 0.16204)])
```

```
# Building Black-Litterman portfolio
black_litterman_weights = [0.62588,
                           0.19951,
                           0.016,
                           0.15861]
black_litterman_portfolio = aapl*black_litterman_weights[0] +
        tsla*black_litterman_weights[1] +
        dis*black_litterman_weights[2] +
        amd*black_litterman_weights[3]
```

After obtaining prior expected returns and providing our views, as well as our confidence levels, we have an optimized portfolio with the following weights for each asset:

Apple: 62.59%

Tesla: 19.95%

Disney: 1.6%

AMD: 15.86%

```
# Black-Litterman Portfolio daily returns
black_litterman_portfolio
```

```
Date
2010-07-01 04:00:00   -0.021734
2010-07-02 04:00:00   -0.033732
2010-07-06 04:00:00   -0.030528
2010-07-07 04:00:00    0.030036
2010-07-08 04:00:00    0.019225
                          ...
2023-02-06 05:00:00   -0.010763
2023-02-07 05:00:00    0.018628
2023-02-08 05:00:00   -0.008738
2023-02-09 05:00:00   -0.001324
2023-02-10 05:00:00   -0.012131
Name: Close, Length: 3176, dtype: float64
```

We might now go on and compare the Black-Litterman optimized portfolio to the original portfolio

```
# Comparing Black-Litterman portfolio to the original portfolio
qs.reports.full(black_litterman_portfolio, benchmark = portfolio)
```

Performance Metrics

	Strategy	Benchmark
Start Period	2010-07-01	2010-07-01
End Period	2023-02-10	2023-02-10
Risk-Free Rate	0.0%	0.0%
Time in Market	100.0%	100.0%
Cumulative Return	4,105.09%	3,429.90%
CAGR%	34.48%	32.62%
Sharpe	1.15	1.08
Prob. Sharpe Ratio	100.0%	99.99%
Smart Sharpe	1.09	1.03
Sortino	1.68	1.58
Smart Sortino	1.6	1.51
Sortino/√2	1.19	1.12
Smart Sortino/√2	1.13	1.06
Omega	1.22	1.22
Max Drawdown	-45.36%	-52.21%
Longest DD Days	481	404
Volatility (ann.)	29.68%	30.59%
R^2	0.89	0.89
Information Ratio	0.01	0.01
Calmar	0.76	0.62
Skew	-0.18	-0.08
Kurtosis	3.71	3.9
Expected Daily %	0.12%	0.11%
Expected Monthly %	2.49%	2.37%
Expected Yearly %	30.61%	28.99%
Kelly Criterion	9.35%	7.94%
Risk of Ruin	0.0%	0.0%
Daily Value-at-Risk	-2.94%	-3.04%
Expected Shortfall (cVaR)	-2.94%	-3.04%

Max Consecutive Wins	13	13
Max Consecutive Losses	8	8
Gain/Pain Ratio	0.22	0.21
Gain/Pain (1M)	1.44	1.35
Payoff Ratio	0.97	0.97
Profit Factor	1.22	1.21
Common Sense Ratio	1.28	1.26
CPC Index	0.65	0.64
Tail Ratio	1.05	1.04
Outlier Win Ratio	3.72	3.6
Outlier Loss Ratio	3.5	3.44
MTD	7.19%	6.71%
3M	16.89%	22.42%
6M	-15.01%	-14.88%
YTD	26.4%	31.62%
1Y	-20.82%	-26.29%
3Y (ann.)	40.43%	35.01%
5Y (ann.)	42.34%	39.74%
10Y (ann.)	38.88%	38.55%
All-time (ann.)	34.48%	32.62%
Best Day	11.5%	13.76%
Worst Day	-13.75%	-12.65%
Best Month	30.35%	30.34%
Worst Month	-18.08%	-19.49%
Best Year	165.75%	172.25%
Worst Year	-39.67%	-47.22%
Avg. Drawdown	-4.22%	-4.46%
Avg. Drawdown Days	25	27
Recovery Factor	90.49	65.7
Ulcer Index	0.12	0.13
Serenity Index	26.63	19.5
Avg. Up Month	8.96%	8.86%
Avg. Down Month	-6.18%	-6.51%
Win Days %	55.38%	54.72%
Win Month %	61.84%	61.18%
Win Quarter %	74.51%	66.67%
Win Year %	92.86%	92.86%

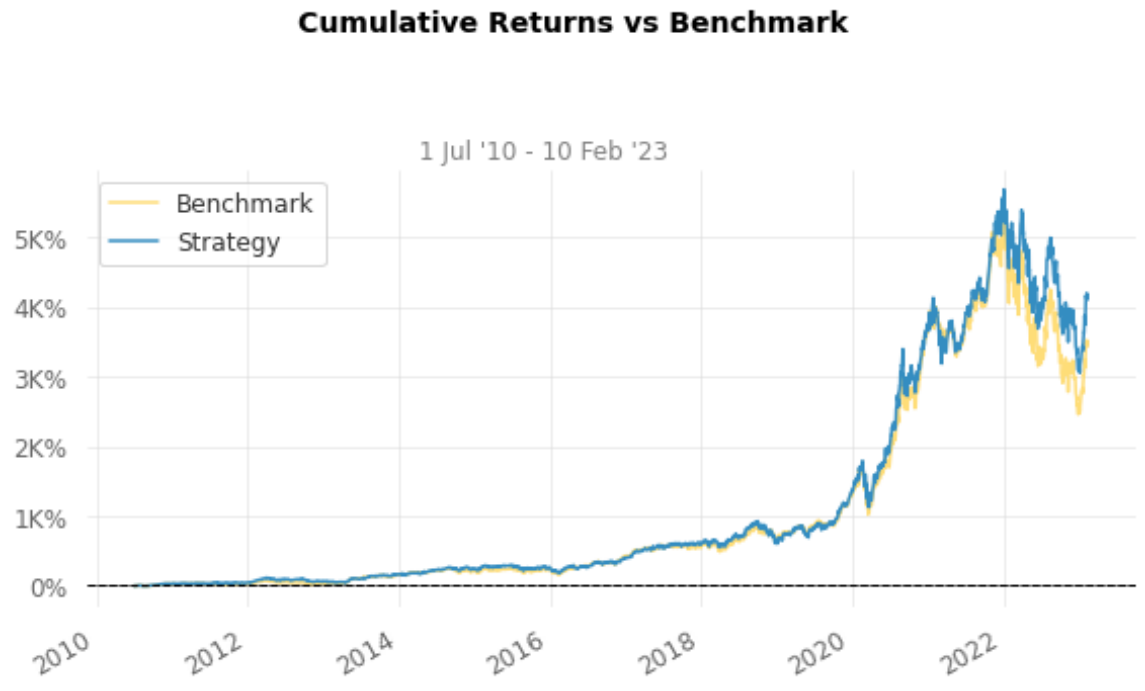
Beta	0.91	-
Alpha	0.04	-
Correlation	94.09%	-
Treynor Ratio	4497.23%	-

None

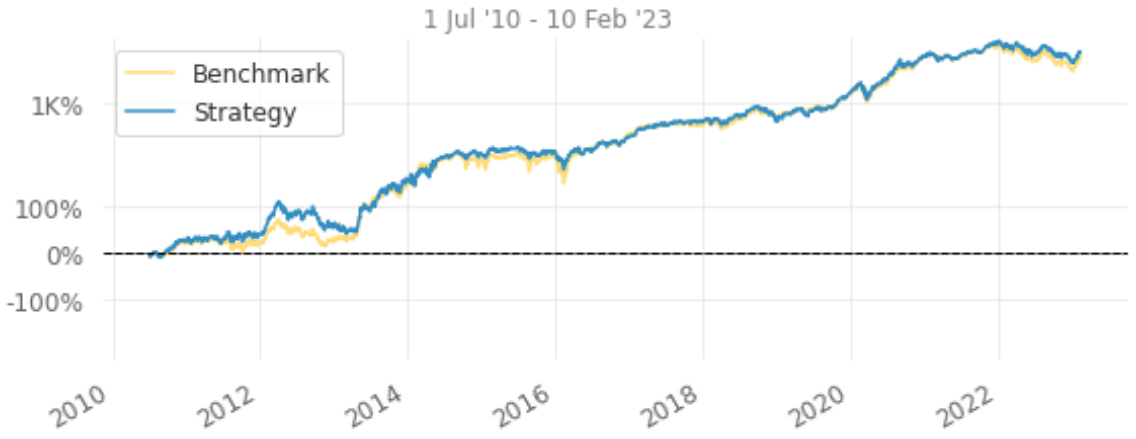
5 Worst Drawdowns

	Start	Valley	End	Days	Max Drawdown	99% Max Drawdown
1	2022-01-04	2023-01-03	2023-02-10	402	-45.364855	-44.337327
2	2020-02-20	2020-03-23	2020-06-05	105	-35.049884	-34.916210
3	2012-04-04	2013-03-04	2013-07-29	481	-33.317405	-31.487018
4	2018-10-02	2019-01-03	2019-10-11	374	-31.590421	-27.700415
5	2015-06-25	2016-02-10	2016-07-25	396	-28.549292	-27.349262

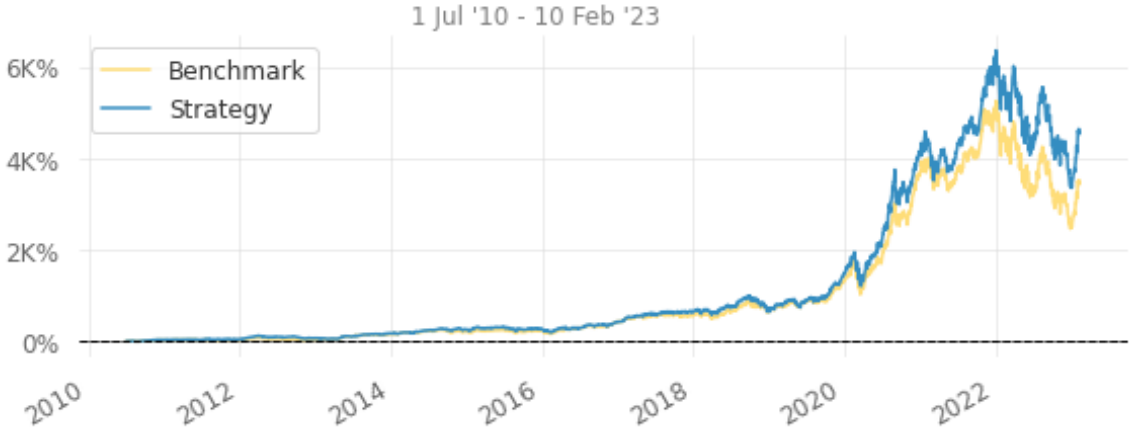
Strategy Visualization



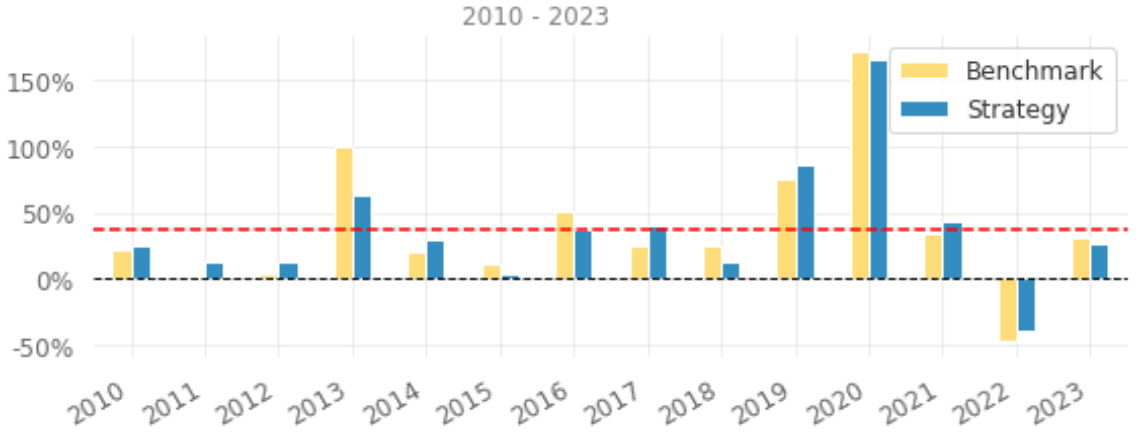
Cumulative Returns vs Benchmark (Log Scaled)



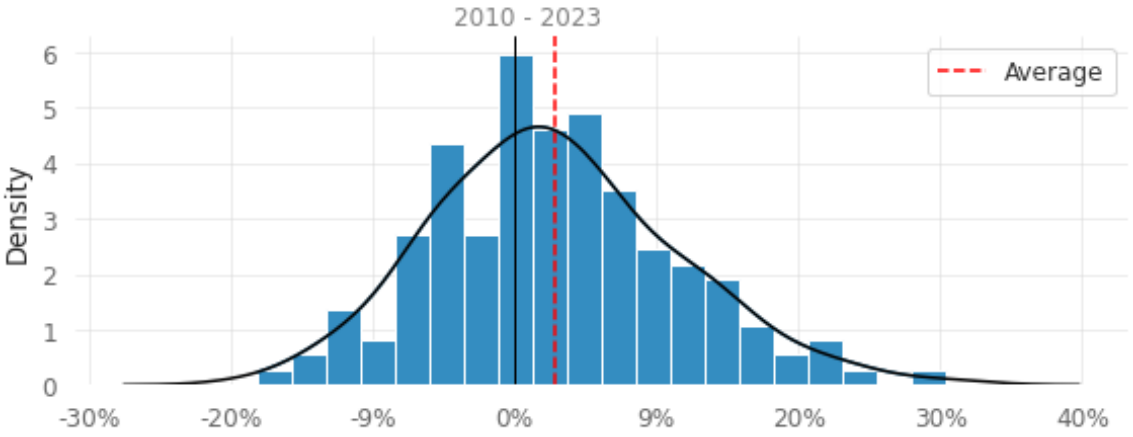
Cumulative Returns vs Benchmark (Volatility Matched)



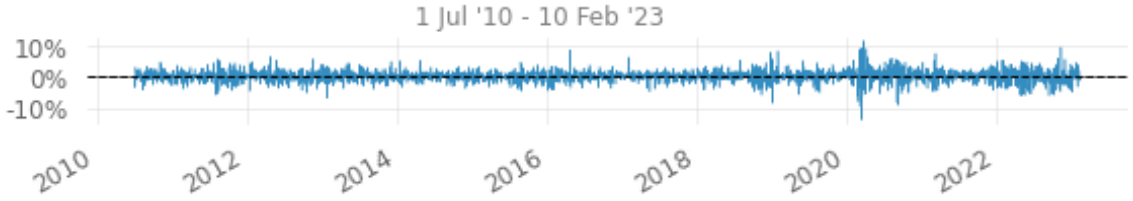
EOY Returns vs Benchmark



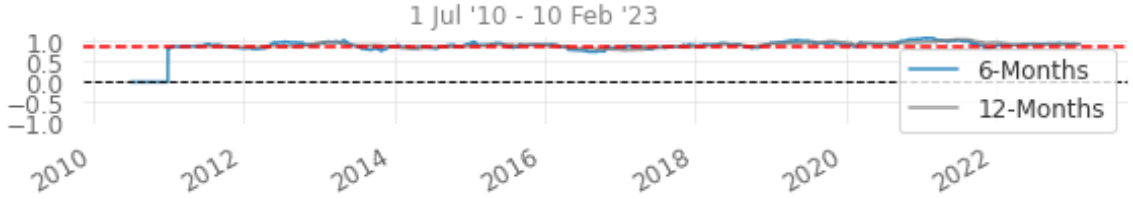
Distribution of Monthly Returns



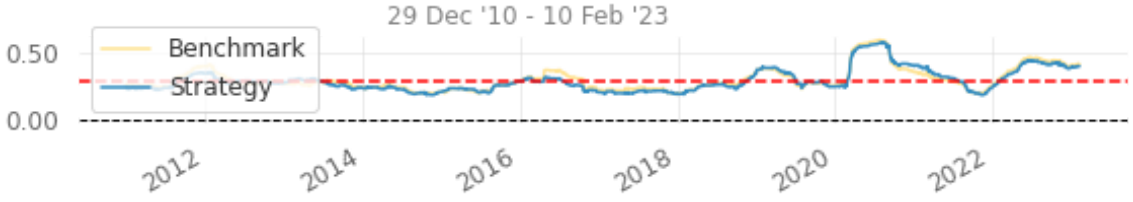
Daily Returns



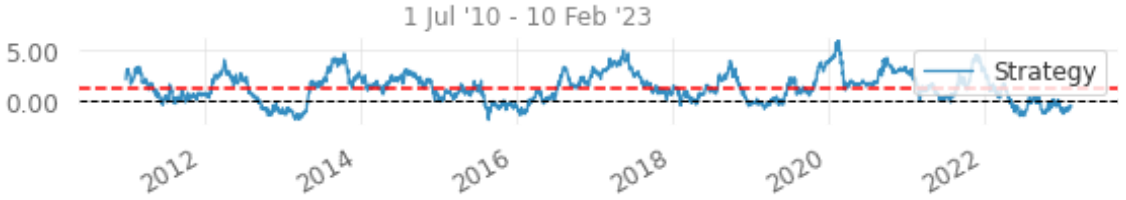
Rolling Beta to Benchmark



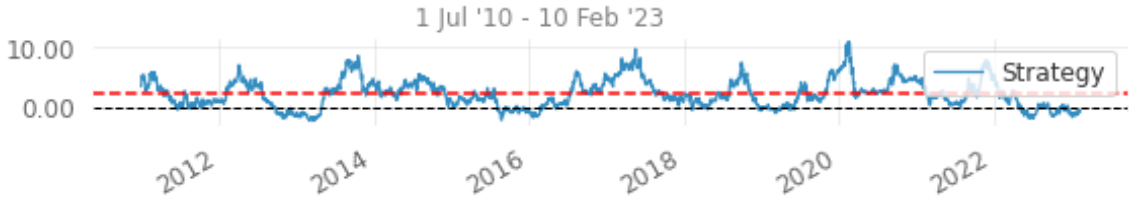
Rolling Volatility (6-Months)



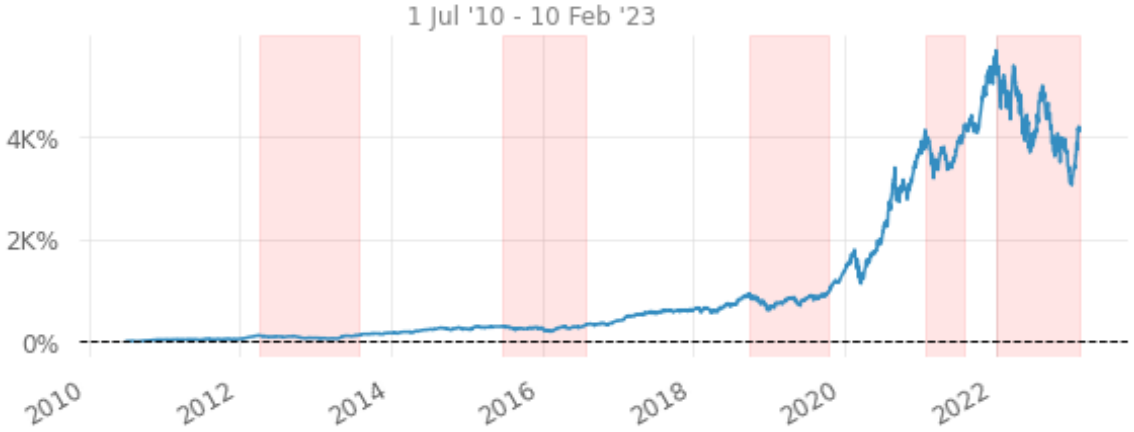
Rolling Sharpe (6-Months)



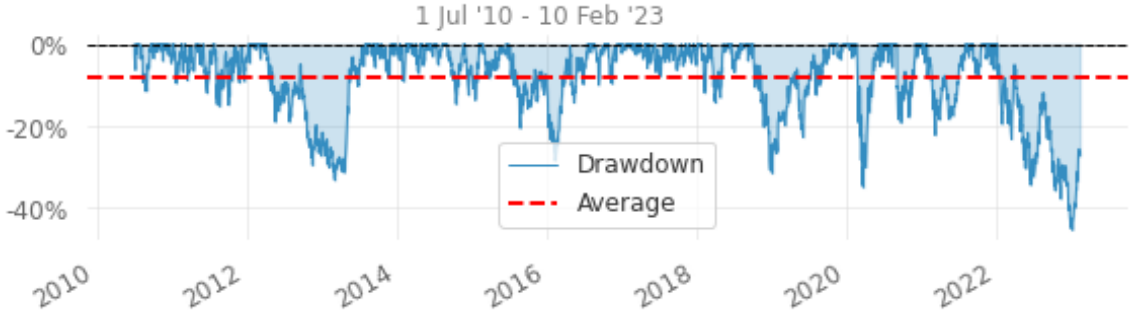
Rolling Sortino (6-Months)

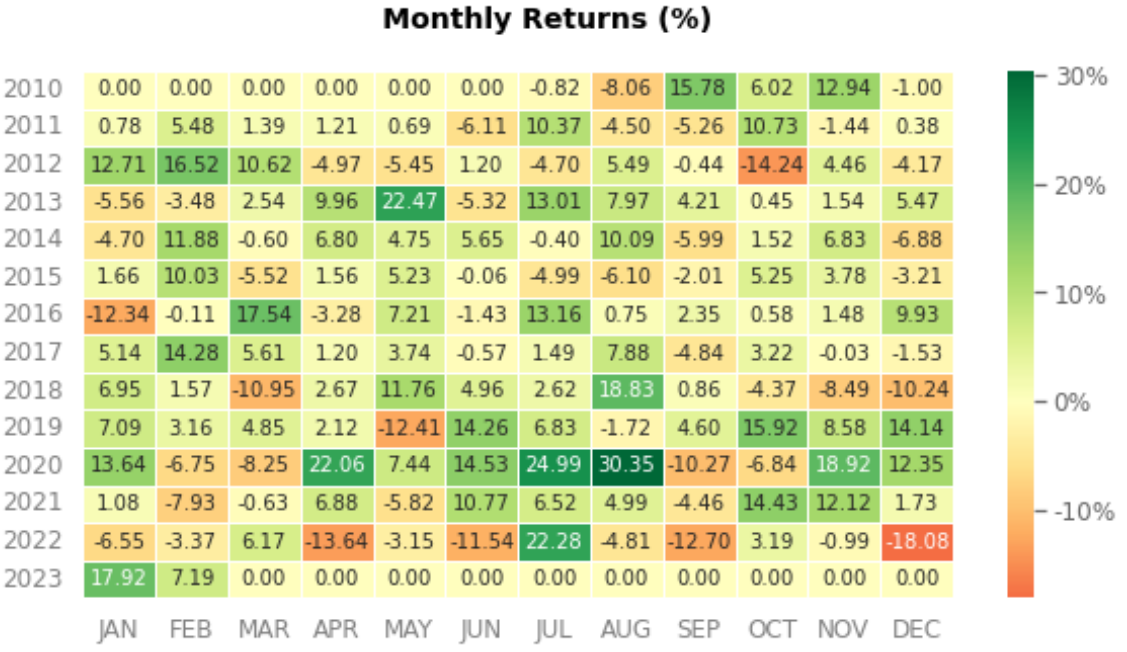


Worst 5 Drawdown Periods



Underwater Plot





By using the Black-Litterman Allocation Model, we were able to improve our investment portfolio's performance metrics compared to the original portfolio, where each asset was allocated a uniform weight of 25%. The Black-Litterman optimized portfolio outperformed the original portfolio in several key metrics. First, it generated higher cumulative return and CAGR, indicating a stronger overall performance. Additionally, the Sharpe and Sortino ratios were higher, demonstrating greater risk-adjusted returns.

Moreover, the Black-Litterman portfolio had a lower maximum drawdown and annual volatility compared to the original portfolio, implying less downside risk and more stability to the optimized

portfolio. In terms of expected returns, the Black-Litterman portfolio had higher daily, monthly, and yearly returns, and the Daily Value-at-Risk was lower, indicating a lower risk of significant losses in a day.

The Black-Litterman portfolio had a lower averaged drawdown and higher recovery factor, meaning that it can bounce back faster from losses, and the beta of the optimized portfolio was much lower than that of the original portfolio, indicating lower overall market risk. Overall, the Black-Litterman optimized portfolio achieved higher returns at lower risks.

Both the Markowitz Mean-Variance Model and the Black-Litterman Allocation Model effectively enhanced the performance and reduced the risk of our original portfolio by optimizing the allocation weights of Apple, Tesla, Disney, and AMD stocks.

The Markowitz optimization resulted in a portfolio that primarily invested in Apple and a smaller portion in Tesla, with no allocation in Disney and AMD. On the other hand, the Black-Litterman optimization allocated funds into all four stocks, but still favored Apple with the majority of the allocation.

The preference for Apple in both optimizations is not coincidental. Our initial analysis did reveal that Apple had the highest Sharpe ratio, lowest beta, and demonstrated superior performance with lower risk compared to the other stocks.

It's also interesting to compare the two optimized portfolios. The Markowitz optimized portfolio outperformed the Black-Litterman portfolio in terms of cumulative returns, CAGR, Sharpe ratio, Profit Factor, Recovery Factor, and overall performance. On the other hand, the Black-Litterman portfolio demonstrated some advantages, such as lower maximum and average drawdowns, lower annual volatility, and

better performance on its worst day and worst month, although the Markowitz optimization still had lower losses than the Black-Litterman optimization did on its worst year.

In conclusion, portfolio optimization is a very important step to improve the risk-return relationship of a portfolio by adjusting its asset allocation. By using various mathematical models and optimization techniques, it's possible to efficiently improve performance and reduce the exposure to risk.

While there are different approaches to portfolio optimization, including the Markowitz Mean-Variance model and the Black-Litterman allocation model, there is no one-size-fits-all solution. The choice of the model to use depends on the investor's risk tolerance, investment goals, and overall market conditions.

To gain a deeper understanding of portfolio optimization, you can explore various research papers, journals, and books on the subject. Some sources of research to consider include books such as Modern Portfolio Theory and Investment Analysis by Edwin J. Elton and Martin Jay Gruber, Portfolio Selection by Harry Markowitz, and Active Portfolio Management: A Quantitative Approach for Producing Superior Returns and Controlling Risk by Richard C. Grinold and Ronald N. Kahn, which can provide valuable insights into the theory and practice of portfolio optimization.

3 | Fundamental vs. Technical Analysis

Fundamental vs. Technical Analysis: Which Approach is Better?

Technical analysis is a very popular method used for many traders and investors to evaluate stocks and other assets based on historical price and volume data. It is an approach used to identify trends, or lack of trends, and help traders and investors to make decisions based on what they believe the future price movements will be. The underlying assumption of technical analysis is that past patterns and price movements tend to repeat themselves, so those can be used to predict future movements. Therefore, technical analysts examine charts and look for opportunities in patterns and indicators.

In technical analysis, price charts can display a variety of data, including open, high, low, and close prices, as well as various timeframes such as daily, weekly, and monthly charts. Volume is another important factor in technical analysis because it can provide insight into the strength of a trend or an indicator.

Technical analysts use a variety of indicators, which are basically statistical calculations based on price and volume that can provide additional information about trends and potential opportunities. Some of the most commonly used indicators in technical analysis include moving averages, relative strength index (RSI), Bollinger Bands, and many others.

Despite its popularity among traders, the use of technical analysis may be controversial. Some critics argue that technical analysis relies too heavily on subjective interpretations of chart patterns and that it lacks a clear theoretical foundation. They also argue that technical analysis is prone to false signals and that traders who rely on technical analysis may miss out on important fundamental factors that can influence the price of stocks.

Fundamental analysis, on the other hand, focuses on analyzing a company's financial and economic fundamentals, such as revenue, earnings, and market share. Unlike technical analysis, fundamental analysis is based on the belief that a company's intrinsic value can be determined by analyzing these factors. Fundamental analysts look at financial statements, industry trends, and other relevant data to make investment decisions.

It can be said that technical analysis tend to be more appropriate for short-term trading, whereas fundamental analysis may be better suited for long-term investing. Fundamental analysis provides investors with a more comprehensive understanding of a company's financial health and long-term growth prospects.

It can also be said that, while humans tend to operate better with fundamental analysis, as it requires a deep understanding of the underlying factors that drive a company's value, computers may operate better with technical analysis, as it relies heavily on quantitative data that can be analyzed quickly and efficiently. An evidence to that is the fact that the use of automated trading bots that trade based on technical analysis has become increasingly popular in recent years. These bots use algorithms to identify patterns and trends in price data and make trades based on technical signals.

In Python, there are quite a few libraries that are suited for both technical and fundamental analysis, and we're going to explore how to use these indicators, not only in conventional ways but also to feed machine learning models.

Technical Indicators

There is a wide range of technical indicators used by traders and investors alike, each with its own unique set of benefits and drawbacks. Some of the most commonly used indicators include moving averages,

which can be either simple or exponential, as well as Bollinger Bands, RSI, ATR, historic volatility, and many others.

To help you better understand these indicators, I will briefly introduce some of the most commonly used ones and display them on a candlestick chart.

Moving Averages

Moving averages are commonly used to smooth out price fluctuations and identify trends in the price action of stocks, commodities, forex, crypto, and many others.

A moving average is calculated by taking the average price of an asset over a certain period of time. This period can be as nine days or as long as 200 days, depending on the trader's preference and the asset being analyzed. The resulting line that represents the moving average is then plotted on a chart, and traders use this line to gauge the direction and strength of the trend.

There are two types of moving averages: simple moving averages (SMA) and exponential moving averages (EMA). SMA calculates the average price of the asset over the specified period of time and gives equal weight to each data point.So, for instance, a 20-day SMA would take the sum of the closing prices over the past 20 days and divide by 20 to get the average price.

EMA, on the other hand, gives more weight to recent price action. The formula for calculating EMA involves using a multiplier that gives more weight to the most recent price data. The formula for calculating a 9-day EMA, for example, would be:

$$EMA = (\text{Close - EMA(previous day)}) \times Multiplier + \text{EMA(previous day)}$$

The multiplier used depends on the length of the period used. For a 9-day EMA, the multiplier would be 2/(9+1) = 0.2 (rounded to one decimal place).

Traders often use moving averages to identify dynamic support and resistance levels, as well as to spot potential trend changes. When the price of an asset is above its moving average, it is considered to be in an uptrend, while a price below the moving average is considered to be in a downtrend. Traders also look for crossovers between different moving averages, which can signal a change in trend direction, such as the Golden Cross (when the 50-day moving average crosses above the 200-day moving average) and Death Cross (when the 50-day moving average crosse below the 200-day moving average).

Moving averages can also be used to set stop loss and take profit levels for trades. For example, a trader may place a stop loss order below the moving average to limit their losses if the trend reverses.

Bollinger Bands

Bollinger Bands are an exteremely popular tool used by traders and analysts to measure volatility and to identify potential buy and sell opportunities.

The Bollinger Bands consist of three lines on a chart: a simple moving average (SMA) in the middle,typically a 20-day moving average, and two bands that are set at a distance of two standard deviations away from the SMA. When the market is volatile, the bands widen, and when the market is less volatile, the bands contract. The distance between the bands can therefore be used as an indicator of volatility.

Usually, traders use these bands to identify possible buying and selling entries. When the price touches or crosses below the lower band, it may be considered oversold and a potential buy signal. Conversely, when the price touches or crosses above the upper band, it may be considered overbought and a potential sell signal.

To obtain the Bollinger Bands values, there are three important steps:

1. Calculate the 20-day SMA of the stock's price.

2. Calculate the standard deviation of the stocks's price over the past 20 periods.

3.Calculate the upper and lower bands by adding or subtracting two standard deviations from the 20-day SMA.

The formula to calculate the upper band is:

$$Upper\ Band = 20\text{day}\ SMA + (2 \times \sigma)$$

Whereas the formula for the lower band is:

$$Lower\ Band = 20\text{day}\ SMA - (2 \times \sigma)$$

Relative Strenght Index (RSI)

The RSI is another popular technical analysis tool used by traders and analysts in financial markets. It measures the strength of price action and can be used to identify possible buy and sell opportunities.

The RSI is calculated by comparing the average gains and losses of a stock over a specific period of time. The formula for calculating the RSI involves two steps:

First, you calculate the Relative Strength (RS) of the stock, which is the ratio of the average gains to the average losses over time, which is typically the last 14 days.

$$\text{RS} = \frac{\text{Average Gain}}{\text{Average Loss}}$$

In order to obtain the average gain or loss, the difference between the closing price of the current day and the previous day is taken. If that difference is positive, then it is considered a gain, but if that difference is negative, it is considered a loss.

The next step is to obtain the RSI using the RS value. The RSI ranges from 0 to 100 and is plotted on a chart.

$$\text{RSI} = 100 - \frac{100}{1 + \text{RS}}$$

When the RSI is above 70, it is considered an overbought region and could signal a potential opportunity for selling. When it is below the value of 30, it is considered to be on an oversold region and could signal a potential opportunity for buying low.

The RSI is also popularly used to identify divergences between the RSI and price, which can indicate the loss of *momentum*, i.e. strenght, in the current trend and suggest a possible reversal.

Average True Range (ATR)

The Average True Range (ATR) is yet another commong indicator used to measure the volatility of price action and help traders to identify potential buy and sell opportunities.

It is calculated by comparing the true range of a stock over a specific period of time. The true range is the maximum value of the three calculations below:

1.The difference between the current high and the previous close.

2. The difference between the current low and the previous close.

The difference between the current high and the current low.

Before obtaining the ATR, we need to first obtain the **true range** through the following formula:

$$TR = \max(high - low, abs(high - close\_prev), abs(low - close\_prev))$$

Where:

High is the current day's highest price

Low is the current day's lowest price

close_prev is the previous day's closing price

The ATR ranges from 0 to infinity, and the higher its value, the higher may be the volatility in price action, while the opposite is true for a lower ATR value.

Traders and analysts may use the ATR to set stop-loss and take-profit levels, as well as to identify potential trend reversals. For example, if the ATR value is increasing, it could indicate that a trend reversal is likely to occur.

```
# Downloading Apple Stocks
aapl = yf.download('AAPL', start = '2010-07-01', end = '2023-02-11')
```

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

```
aapl # Displaying data
```

	Open	High	Low	Close	Adj Close
Date					
2010-07-01	9.082143	9.100000	8.686429	8.874286	7.553066
2010-07-02	8.946071	8.961786	8.685714	8.819286	7.506257
2010-07-06	8.964286	9.028571	8.791429	8.879643	7.557625
2010-07-07	8.946071	9.241786	8.919643	9.238214	7.862811
2010-07-08	9.374286	9.389286	9.103214	9.217500	7.845183
...	...	...	...	...	...
2023-02-06	152.570007	153.100006	150.779999	151.729996	151.498688
2023-02-07	150.639999	155.229996	150.639999	154.649994	154.414230
2023-02-08	153.880005	154.580002	151.169998	151.919998	151.688400
2023-02-09	153.779999	154.330002	150.419998	150.869995	150.639999
2023-02-10	149.460007	151.339996	149.220001	151.009995	151.009995

3176 rows × 6 columns

```
aapl.dtypes # Printing data types
```

```
Open         float64
High         float64
Low          float64
```

```
Close        float64
Adj Close    float64
Volume         int64
dtype: object
```

After obtaining historical data of Apple stocks with yfinance, we may use Plotly to plot a Candlestick chart containing information on price and volume.

```
# Plotting candlestick chart without indicators
fig = make_subplots(rows=2, cols=1, shared_xaxes=True,
        vertical_spacing=0.05, row_heights = [0.7, 0.3])
fig.add_trace(go.Candlestick(x=aapl.index,
                             open=aapl['Open'],
                             high=aapl['High'],
                             low=aapl['Low'],
                             close=aapl['Adj Close'],
                             name='AAPL'),
              row=1, col=1)


# Plotting volume chart on the second row
fig.add_trace(go.Bar(x=aapl.index,
                     y=aapl['Volume'],
                     name='Volume',
                     marker=dict(color='orange', opacity=1.0)),
              row=2, col=1)

# Plotting annotation
fig.add_annotation(text='Apple (AAPL)',
                   font=dict(color='white', size=40),
                   xref='paper', yref='paper',
                   x=0.5, y=0.65,
                   showarrow=False,
                   opacity=0.2)

# Configuring layout
```

```
fig.update_layout(title='AAPL Candlestick Chart From July 1st, 2010 to
        February 10th, 2023',
                  yaxis=dict(title='Price (USD)'),
                  height=1000,
                 template = 'plotly_dark')

# Configuring axes and subplots
fig.update_xaxes(rangeslider_visible=False, row=1, col=1)
fig.update_xaxes(rangeslider_visible=False, row=2, col=1)
fig.update_yaxes(title_text='Price (USD)', row=1, col=1)
fig.update_yaxes(title_text='Volume', row=2, col=1)

fig.show()
```

Above, it's possible to see a daily candlestick chart containing price and volume of Apple stocks traded from July 1st, 2010 to February 10th, 2023. The candlestick chart makes it easy to see information on the opening, closing, high, and low prices of each trading day, as well as the overall trend in the last 13 years.

The use of the 'Adj Close' column instead of the 'Close' column to plot the chart above is due to the adjustment of historical prices for dividends and stock splits. The 'Adj Close' value represents the closing price adjusted for these factors, which allows for a more accurate representation of the stock's true price over time.

The volume bars below the candlestick chart display the financial volume of shares traded for each day.

After taking a brief look at a candlestick chart without indicators, we can go on and add a few indicators to our Apple stocks dataframe and display them on the candlestick chart.

```
# Adding Moving Averages
aapl['EMA9'] = aapl['Adj Close'].ewm(span = 9, adjust = False).mean()
        # Exponential 9-Period Moving Average
```

```
aapl['SMA20'] = aapl['Adj Close'].rolling(window=20).mean() # Simple
        20-Period Moving Average
aapl['SMA50'] = aapl['Adj Close'].rolling(window=50).mean() # Simple
        50-Period Moving Average
aapl['SMA100'] = aapl['Adj Close'].rolling(window=100).mean() # Simple
        100-Period Moving Average
aapl['SMA200'] = aapl['Adj Close'].rolling(window=200).mean() # Simple
        200-Period Moving Average


# Adding RSI for 14-periods
delta = aapl['Adj Close'].diff() # Calculating delta
gain = delta.where(delta > 0,0) # Obtaining gain values
loss = -delta.where(delta < 0,0) # Obtaining loss values
avg_gain = gain.rolling(window=14).mean() # Measuring the 14-period
        average gain value
avg_loss = loss.rolling(window=14).mean() # Measuring the 14-period
        average loss value
rs = avg_gain/avg_loss # Calculating the RS
aapl['RSI'] = 100 - (100 / (1 + rs)) # Creating an RSI column to the
        Data Frame


# Adding Bollinger Bands 20-periods
aapl['BB_UPPER'] = aapl['SMA20'] + 2*aapl['Adj
        Close'].rolling(window=20).std() # Upper Band
aapl['BB_LOWER'] = aapl['SMA20'] - 2*aapl['Adj
        Close'].rolling(window=20).std() # Lower Band


# Adding ATR 14-periods
aapl['TR'] = pd.DataFrame(np.maximum(np.maximum(aapl['High'] -
        aapl['Low'], abs(aapl['High'] - aapl['Adj Close'].shift())),
        abs(aapl['Low'] - aapl['Adj Close'].shift())), index =
        aapl.index)
aapl['ATR'] = aapl['TR'].rolling(window = 14).mean() # Creating an ART
        column to the Data Frame

aapl.tail(50) # Plotting last 50 trading days with indicators
```

	Open	High	Low	Close	Adj Close
Date					
2022-11-30	141.399994	148.720001	140.550003	148.029999	147.804321
2022-12-01	148.210007	149.130005	146.610001	148.309998	148.083893
2022-12-02	145.960007	148.000000	145.649994	147.809998	147.584656
2022-12-05	147.770004	150.919998	145.770004	146.630005	146.406464

	Open	High	Low	Close	Adj Close
Date					
2022-12-06	147.070007	147.300003	141.919998	142.910004	142.692139
2022-12-07	142.190002	143.369995	140.000000	140.940002	140.725143
2022-12-08	142.360001	143.520004	141.100006	142.649994	142.432526
2022-12-09	142.339996	145.570007	140.899994	142.160004	141.943283
2022-12-12	142.699997	144.500000	141.059998	144.490005	144.269730
2022-12-13	149.500000	149.970001	144.240005	145.470001	145.248230
2022-12-14	145.350006	146.660004	141.160004	143.210007	142.991684
2022-12-15	141.110001	141.800003	136.029999	136.500000	136.291901
2022-12-16	136.690002	137.649994	133.729996	134.509995	134.304932
2022-12-19	135.110001	135.199997	131.320007	132.369995	132.168198
2022-12-20	131.389999	133.250000	129.889999	132.300003	132.098312
2022-12-21	132.979996	136.809998	132.750000	135.449997	135.243500
2022-12-22	134.350006	134.559998	130.300003	132.229996	132.028412
2022-12-23	130.919998	132.419998	129.639999	131.860001	131.658981
2022-12-27	131.380005	131.410004	128.720001	130.029999	129.831772
2022-12-28	129.669998	131.029999	125.870003	126.040001	125.847855
2022-	127.989998	130.479996	127.730003	129.610001	129.412415

	Open	High	Low	Close	Adj Close
Date					
12-29					
2022-12-30	128.410004	129.949997	127.430000	129.929993	129.731918
2023-01-03	130.279999	130.899994	124.169998	125.070000	124.879326
2023-01-04	126.889999	128.660004	125.080002	126.360001	126.167366
2023-01-05	127.129997	127.769997	124.760002	125.019997	124.829399
2023-01-06	126.010002	130.289993	124.889999	129.619995	129.422394
2023-01-09	130.470001	133.410004	129.889999	130.149994	129.951584
2023-01-10	130.259995	131.259995	128.119995	130.729996	130.530701
2023-01-11	131.250000	133.509995	130.460007	133.490005	133.286499
2023-01-12	133.880005	134.259995	131.440002	133.410004	133.206619
2023-01-13	132.029999	134.919998	131.660004	134.759995	134.554550
2023-01-17	134.830002	137.289993	134.130005	135.940002	135.732758
2023-01-18	136.820007	138.610001	135.029999	135.210007	135.003876
2023-01-19	134.080002	136.250000	133.770004	135.270004	135.063782
2023-01-20	135.279999	138.020004	134.220001	137.869995	137.659805
2023-01-23	138.119995	143.320007	137.899994	141.110001	140.894882
2023-01-24	140.309998	143.160004	140.300003	142.529999	142.312714

	Open	High	Low	Close	Adj Close
Date					
2023-01-25	140.889999	142.429993	138.809998	141.860001	141.643738
2023-01-26	143.169998	144.250000	141.899994	143.960007	143.740540
2023-01-27	143.160004	147.229996	143.080002	145.929993	145.707520
2023-01-30	144.960007	145.550003	142.850006	143.000000	142.781998
2023-01-31	142.699997	144.339996	142.279999	144.289993	144.070023
2023-02-01	143.970001	146.610001	141.320007	145.429993	145.208282
2023-02-02	148.899994	151.179993	148.169998	150.820007	150.590088
2023-02-03	148.029999	157.380005	147.830002	154.500000	154.264465
2023-02-06	152.570007	153.100006	150.779999	151.729996	151.498688
2023-02-07	150.639999	155.229996	150.639999	154.649994	154.414230
2023-02-08	153.880005	154.580002	151.169998	151.919998	151.688400
2023-02-09	153.779999	154.330002	150.419998	150.869995	150.639999
2023-02-10	149.460007	151.339996	149.220001	151.009995	151.009995

```
# Plotting Candlestick charts with indicators
fig = make_subplots(rows=4, cols=1, shared_xaxes=True,
        vertical_spacing=0.05,row_heights=[0.6, 0.10, 0.10, 0.20])

# Candlestick
fig.add_trace(go.Candlestick(x=aapl.index,
                             open=aapl['Open'],
                             high=aapl['High'],
```

```
                                low=aapl['Low'],
                                close=aapl['Adj Close'],
                                name='AAPL'),
                  row=1, col=1)

    # Moving Averages
    fig.add_trace(go.Scatter(x=aapl.index,
                             y=aapl['EMA9'],
                             mode='lines',
                             line=dict(color='#90EE90'),
                             name='EMA9'),
                  row=1, col=1)

    fig.add_trace(go.Scatter(x=aapl.index,
                             y=aapl['SMA20'],
                             mode='lines',
                             line=dict(color='yellow'),
                             name='SMA20'),
                  row=1, col=1)

    fig.add_trace(go.Scatter(x=aapl.index,
                             y=aapl['SMA50'],
                             mode='lines',
                             line=dict(color='orange'),
                             name='SMA50'),
                  row=1, col=1)

    fig.add_trace(go.Scatter(x=aapl.index,
                             y=aapl['SMA100'],
                             mode='lines',
                             line=dict(color='purple'),
                             name='SMA100'),
                  row=1, col=1)

    fig.add_trace(go.Scatter(x=aapl.index,
                             y=aapl['SMA200'],
                             mode='lines',
                             line=dict(color='red'),
                             name='SMA200'),
                  row=1, col=1)

    # Bollinger Bands
    fig.add_trace(go.Scatter(x=aapl.index,
```

```
                         y=aapl['BB_UPPER'],
                         mode='lines',
                         line=dict(color='#00BFFF'),
                         name='Upper Band'),
              row=1, col=1)

fig.add_trace(go.Scatter(x=aapl.index,
                         y=aapl['BB_LOWER'],
                         mode='lines',
                         line=dict(color='#00BFFF'),
                         name='Lower Band'),
              row=1, col=1)

fig.add_annotation(text='Apple (AAPL)',
                   font=dict(color='white', size=40),
                   xref='paper', yref='paper',
                   x=0.5, y=0.65,
                   showarrow=False,
                   opacity=0.2)


# Relative Strengh Index (RSI)
fig.add_trace(go.Scatter(x=aapl.index,
                         y=aapl['RSI'],
                         mode='lines',
                         line=dict(color='#CBC3E3'),
                         name='RSI'),
              row=2, col=1)

# Adding marking lines at 70 and 30 levels
fig.add_shape(type="line",
              x0=aapl.index[0], y0=70, x1=aapl.index[-1], y1=70,
              line=dict(color="red", width=2, dash="dot"),
              row=2, col=1)
fig.add_shape(type="line",
              x0=aapl.index[0], y0=30, x1=aapl.index[-1], y1=30,
              line=dict(color="#90EE90", width=2, dash="dot"),
              row=2, col=1)

# Average True Range (ATR)
fig.add_trace(go.Scatter(x=aapl.index,
                         y=aapl['ATR'],
                         mode='lines',
```

```
                        line=dict(color='#00BFFF'),
                        name='ATR'),
              row=3, col=1)


# Volume
fig.add_trace(go.Bar(x=aapl.index,
                     y=aapl['Volume'],
                     name='Volume',
                     marker=dict(color='orange', opacity=1.0)),
              row=4, col=1)


# Layout
fig.update_layout(title='AAPL Candlestick Chart From July 1st, 2010 to
        February 10th, 2023',
                  yaxis=dict(title='Price (USD)'),
                  height=1000,
                 template = 'plotly_dark')

# Axes and subplots
fig.update_xaxes(rangeslider_visible=False, row=1, col=1)
fig.update_xaxes(rangeslider_visible=False, row=4, col=1)
fig.update_yaxes(title_text='Price (USD)', row=1, col=1)
fig.update_yaxes(title_text='RSI', row=2, col=1)
fig.update_yaxes(title_text='ATR', row=3, col=1)
fig.update_yaxes(title_text='Volume', row=4, col=1)

fig.show()
```

Finally, the Candlestick chart above is a full representation of the price and volume over time, along with additional indicators such as moving averages, Bollinger Bands, Relative Strength Index, and ATR.

The amount of indicators on screen may appear overwhelming at first glance, but this is done on purpose! Don't forget that with **Plotly's** interactive features, you have the flexibility to customize the way you visualize the chart to suit your preferences by selecting the time period and indicators you wish to view.

In the first chart below, we have displayed only the 9-period exponential moving average and the 20-period simple moving average on the price row. This simplifies the visual representation of the data and allows you to focus only on these two indicators.

In the next chart, we have chosen to display only the Bollinger Bands with the 20-period simple moving average on the price row.

Feel free to experiment with the Candlestick chart as you like!

You've seen that, with Python, it is extremely easy not only to calculate, but also to visualize a wide variety of technical indicators to analyze stocks, as well as other assets.

Beyond that, these indicators can be used as features for machine learning models, and Python's variety makes it easy to backtest strategies based on technical analysis and see how well we could perform buying and selling stocks based on these strategies, which is something we will approach later on in this notebook.

Fundamental Indicators

Just as we have a broad range of technical indicators, the same applies to fundamental indicators used by investors when they're looking for relevant information when deciding whether to invest in a company or not.

The most commonly used fundamental indicators include earnings per share (EPS), price-to-earnings ratio (P/E ratio), return on equity (ROE), debt-to-equity ratio, and dividend yield. These indicators provide insights into the company's profitability, valuation, efficiency, financial leverage, and dividend payments. Overall, fundamental analysis involves examining a company's financial and economic data to evaluate its financial health and potential for growth.

I'll also briefly present to you some fundamental indicators, so we can make comparison on the stocks we've been working on for now.

Earnings per Share (EPS)

Earnings per share (EPS) is an indicator that measures a company's net profit per the number of stocks available in the stock market. It is calculated by dividing the net profit by the total number of shares. The EPS provides a way for investors to evaluate a company's profitability on a per-share basis. A higher EPS indicates a company that is more profitable, which may lead to increased demand for its shares and a higher stock price in the future.

Price-to-Earnings Ratio (P/E)

The P/E ratio indicator serves to compare a company's stock price to its earnings per share (EPS). It is calculated by dividing the current stock price by the EPS. The P/E ratio provides investors with insight into how much they are paying for each dollar of earnings. A higher P/E ratio indicates that investors are willing to pay more for each dollar of earnings, which may suggest that they expect the company to grow in the future.

Return on Equity (ROE)

The ROE indicator measures a company's profitability by calculating the amount of net income generated as a percentage of the company's shareholder equity. It is calculated by dividing the net income by the shareholder equity. The ROE provides a way for investors to evaluate how effectively a company is using its equity to generate profits. A higher ROE indicates a more efficient use of shareholder equity, which can lead to increased demand for shares and higher stock price, as well as increase in company's profits in the future.

Debt-to-Equity Ratio

The debt-to-equity ratio is an indicator that measures a company's leverage by comparing its total liabilities to its shareholder equity. It is calculated by dividing the total liabilities by the shareholder equity. The debt-to-equity ratio provides investors with insight into how much debt a company is using to finance its operations. A higher debt-to-equity ratio suggests that a company is relying more on debt financing, which increase its financial risk.

Dividend Yield

The dividend yield is an indicator that measures the annual dividend income generated by a company's stock relative to its current market price. It is calculated by dividing the annual dividend per share by the current market price per share. The dividend yield provides a way for investors to evaluate the income potential of a company's stock. A higher dividend yield indicates that the company is paying out a larger portion of its profits as dividends, which can be attractive to investors looking for passive income.

Important Note: As of March 2023 it seems that yfinance, as well as other libraries, are having the **Exception: yfinance failed to decrypt Yahoo data response** issue. I've tried other methods to obtain these indicators, but none of them worked, or the data obtained didn't seem to be reliable at all. I'll manually add the indicators and print them below, however, I'll leave a code cell demonstrating how you would usually execute this extraction using yfinance, so you can reproduce it when this issue gets solved.

```
# Getting AAPL data
# aapl = yf.Ticker("AAPL")
# aapl_eps = aapl.info['trailingEps']
# aapl_pe_ratio = aapl.info['trailingPE']
# aapl_roe = aapl.info['returnOnEquity']*100
# aapl_dy = aapl.info['dividendYield']*100

# Getting AMD data
# amd = yf.Ticker("AMD")
# amd_eps = amd.info['trailingEps']
# amd_pe_ratio = amd.info['trailingPE']
# amd_roe = amd.info['returnOnEquity']*100
# amd_dy = amd.info['dividendYield']*100

# AAPL indicators (obtained from
        https://finance.yahoo.com/quote/AAPL/key-statistics?p=AAPL)

aapl_eps = 5.89
aapl_pe_ratio = 26.12
aapl_roe = 147.94
```

```
aapl_dy = 0.60

# AMD indicators (obtained from
        https://finance.yahoo.com/quote/AMD/key-statistics?p=AMD)
amd_eps = 0.82
amd_pe_ratio = 96.62
amd_roe = 4.24
amd_dy = 0.00

# Printing data
print('\n')
print('Apple (AAPL) Fundamental Indicators: ')
print('\n')
print('Earnings per Share (EPS): ',aapl_eps)
print('Price-to-Earnings Ratio (P/E): ', aapl_pe_ratio)
print('Return on Equity (ROE): ', aapl_roe,"%")
print('Dividend Yield: ', aapl_dy,"%")
print('\n')
print('AMD Fundamental Indicators: ')
print('\n')
print('Earnings per Share (EPS): ',amd_eps)
print('Price-to-Earnings Ratio (P/E): ', amd_pe_ratio)
print('Return on Equity (ROE): ',amd_roe,"%")
print('Dividend Yield: ', amd_dy,"%")
print('\n')
```

```
Apple (AAPL) Fundamental Indicators:


Earnings per Share (EPS):  5.89
Price-to-Earnings Ratio (P/E):  26.12
Return on Equity (ROE):  147.94 %
Dividend Yield:  0.6 %


AMD Fundamental Indicators:


Earnings per Share (EPS):  0.82
Price-to-Earnings Ratio (P/E):  96.62
Return on Equity (ROE):  4.24 %
```

```
Dividend Yield:  0.0 %
```

To briefly draw some conclusion on the indicators above, it can be stated that, compared to AMD, AAPL has a higher EPS, indicating that they are generating a higher net profit per share. AAPL also has a lower P/E ratio, meaning that the stock is less expensive per unit of earnings. It also has an incredibly high ROE, which suggests that they are doing a good job of managing shareholders' equity to generate profits. On the other hand, AMD has a relatively low ROE, indicating that the company is not as profitable when compared to AAPL. AAPL also has a modest dividend yield of 0.6%, indicating that they pay dividends to shareholders, while AMD does not pay any dividends at all.

Overall, it can be inferred that the indicators above suggest that AAPL is a more profitable company with a more mature business model than AMD.

4 | Backtesting

Backtesting is the process of evaluating the performance of a trading strategy based on historical market data. In other words, it involves testing a trading strategy using historical data to see how it would have performed if it had been applied in the past. This allows traders and investors to assess the profitability and effectiveness of their trading strategies before applying them to live trading.

With Python, it is extremely easy to perform backtesting on historic data to se how a portfolio performs on a single strategy, or multiple different strategies applied to many financial securities.

Below, we're going to backtest the RSI strategy and the moving average crossover strategy, which are two different types of trading strategies, on the EUR/USD currency pair, one of the most traded pair in Forex, in three different timeframes.

RSI Backtesting

The Relative Strength Index (RSI), as mentioned before, can be used to identify overbought and oversold regions, which allows traders to either sell or buy when it is above 70 or below 30.

This kind of strategy can be seen as countertrend strategy, as traders are looking for good entry points to open a position that goes against the current trend.

For this backtesting, we will buy whenever the RSI goes below 30 and sell whenever it goes above 70.

Consider that this backtesting is very simple, just to give you a direction to look for when you perform your own backtests. Usually, it is also important to consider the taxes, slippage, and other costs you'll have while trading. Here, we're roughly talking about gross profits and losses, so keep this in mind.

For this backtesting, let's consider an initial capital of 100.00 dollars. Then, we're going to print the initial capital, the number of trades, and the final capital after the backtest.

Hourly Data

```
# Loading hourly data for the EUR/USD pair
end_date = dt.datetime.now() # Defining the datetime for March 21st
start_date = end_date - dt.timedelta(days=729) # Loading hourly data
        for the last 729 days

hourly_eur_usd = yf.download('EURUSD=X', start=start_date,
        end=end_date, interval='1h')
hourly_eur_usd
```

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

	Open	High	Low	Close	Adj Close
Datetime					
2021-03-22 19:00:00+00:00	1.194458	1.194600	1.193887	1.194030	1.194030
2021-03-22 20:00:00+00:00	1.194030	1.194458	1.193602	1.193602	1.193602
2021-03-22 21:00:00+00:00	1.193602	1.194315	1.193460	1.193887	1.193887
2021-03-22 22:00:00+00:00	1.193745	1.194172	1.193460	1.193745	1.193745
2021-03-22 23:00:00+00:00	1.193745	1.194172	1.193317	1.193602	1.193602
...	...	...	...	...	...
2023-03-21 15:00:00+00:00	1.077935	1.078516	1.076890	1.077470	1.077470
2023-03-21 16:00:00+00:00	1.077354	1.077935	1.076542	1.077122	1.077122
2023-03-21 17:00:00+00:00	1.077238	1.077470	1.076426	1.076658	1.076658
2023-03-21 18:00:00+00:00	1.076658	1.077006	1.076079	1.077006	1.077006
2023-03-21 19:00:00+00:00	1.076774	1.077354	1.076542	1.076890	1.076890

12392 rows × 6 columns

```
# Calculating the RSI with the TA library
hourly_eur_usd['rsi'] = ta.momentum.RSIIndicator(hourly_eur_usd['Adj
        Close'], window = 14).rsi()
```

```
hourly_eur_usd
```

	Open	High	Low	Close	Adj Close
Datetime					
2021-03-22 19:00:00+00:00	1.194458	1.194600	1.193887	1.194030	1.194030
2021-03-22 20:00:00+00:00	1.194030	1.194458	1.193602	1.193602	1.193602
2021-03-22 21:00:00+00:00	1.193602	1.194315	1.193460	1.193887	1.193887
2021-03-22 22:00:00+00:00	1.193745	1.194172	1.193460	1.193745	1.193745
2021-03-22 23:00:00+00:00	1.193745	1.194172	1.193317	1.193602	1.193602
...	...	...	...	...	...
2023-03-21 15:00:00+00:00	1.077935	1.078516	1.076890	1.077470	1.077470
2023-03-21 16:00:00+00:00	1.077354	1.077935	1.076542	1.077122	1.077122
2023-03-21 17:00:00+00:00	1.077238	1.077470	1.076426	1.076658	1.076658
2023-03-21 18:00:00+00:00	1.076658	1.077006	1.076079	1.077006	1.077006
2023-03-21 19:00:00+00:00	1.076774	1.077354	1.076542	1.076890	1.076890

12392 rows × 7 columns

```
# Plotting candlestick chart for hourly EUR/USD
fig = make_subplots(rows=2, cols=1, shared_xaxes=True,
        vertical_spacing=0.05,row_heights=[0.7, 0.3])
fig.add_trace(go.Candlestick(x=hourly_eur_usd.index,
                             open=hourly_eur_usd['Open'],
                             high=hourly_eur_usd['High'],
                             low=hourly_eur_usd['Low'],
```

```
                              close=hourly_eur_usd['Adj Close'],
                              name='EUR/USD'), row=1, col=1)

# Adding annotation
fig.add_annotation(text='EUR/USD 1HR',
                   font=dict(color='white', size=40),
                   xref='paper', yref='paper',
                   x=0.5, y=0.65,
                   showarrow=False,
                   opacity=0.2)


# Relative Strengh Index (RSI) Indicator
fig.add_trace(go.Scatter(x=hourly_eur_usd.index,
                         y=hourly_eur_usd['rsi'],
                         mode='lines',
                         line=dict(color='#CBC3E3'),
                         name='RSI'),
              row=2, col=1)


# Adding marking lines at 70 and 30 levels
fig.add_shape(type="line",
              x0=hourly_eur_usd.index[0], y0=70,
        x1=hourly_eur_usd.index[-1], y1=70,
              line=dict(color="red", width=2, dash="dot"),
              row=2, col=1)
fig.add_shape(type="line",
              x0=hourly_eur_usd.index[0], y0=30,
        x1=hourly_eur_usd.index[-1], y1=30,
              line=dict(color="#90EE90", width=2, dash="dot"),
              row=2, col=1)

# Layout
fig.update_layout(title='EUR/USD Hourly Candlestick Chart from 2021 to
        2023',
                  yaxis=dict(title='Price'),
                  height=1000,
                 template = 'plotly_dark')

# Configuring subplots
fig.update_xaxes(rangeslider_visible=False)
fig.update_yaxes(title_text='Price', row = 1)
fig.update_yaxes(title_text='RSI', row = 2)
```

```
fig.show()
```

```
# Defining the parameters for the RSI strategy
rsi_period = 14
overbought = 70
oversold = 30

# Creating a new column to hold the signals
hourly_eur_usd['signal'] = 0 # When we're not positioned, 'signal' = 0

# Generating the entries
for i in range(rsi_period, len(hourly_eur_usd)):
    if hourly_eur_usd['rsi'][i] > overbought and hourly_eur_usd['rsi']
        [i - 1] <= overbought:
        hourly_eur_usd['signal'][i] = -1 # We sell when 'signal' = -1
    elif hourly_eur_usd['rsi'][i] < oversold and hourly_eur_usd['rsi']
        [i - 1] >= oversold:
        hourly_eur_usd['signal'][i] = 1 # We buy when 'signal' = 1

# Calculating the pair's daily returns
hourly_eur_usd['returns'] = hourly_eur_usd['Adj Close'].pct_change()
hourly_eur_usd['cumulative_returns'] = (1 +
        hourly_eur_usd['returns']).cumprod() - 1 # Total returns of
        the period

# Applying the signals to the returns
hourly_eur_usd['strategy_returns'] = hourly_eur_usd['signal'].shift(1)
        * hourly_eur_usd['returns']

# Calculating the cumulative returns of the strategy
hourly_eur_usd['cumulative_strategy_returns'] = (1 +
        hourly_eur_usd['strategy_returns']).cumprod() - 1

# Setting the initial capital to $100
initial_capital = 100

# Calculating total portfolio value
hourly_eur_usd['portfolio_value'] = (1 +
        hourly_eur_usd['strategy_returns']).cumprod() *
        initial_capital

# Printing the number of trades, initial capital, and final capital
num_trades = hourly_eur_usd['signal'].abs().sum()
final_capital = hourly_eur_usd['portfolio_value'].iloc[-1]
total_return = (final_capital - initial_capital) / initial_capital *
        100
```

```
print('\n')
print(f"Number of trades: {num_trades}")
print(f"Initial capital: ${initial_capital}")
print(f"Final capital: ${final_capital:.2f}")
print(f"Total return: {total_return:.2f}%")
print('\n')

# Plotting the portfolio total value
fig = go.Figure()

fig.add_trace(go.Scatter(x=hourly_eur_usd.index,
                         y=hourly_eur_usd['portfolio_value'].round(2),
                         mode='lines',
                         line=dict(color='#00BFFF'),
                         name='Portfolio Value'))

fig.update_layout(title='Portfolio - RSI Strategy on Hourly Data',
                  xaxis_title='Date',
                  yaxis_title='Value ($)',
                  template='plotly_dark',
                 height = 600)

fig.show()
```

```
Number of trades: 359
Initial capital: $100
Final capital: $100.70
Total return: 0.70%
```

Daily Data

```
# Loading daily EUR/USD pair data for the last eight years
daily_eur_usd = yf.download('EURUSD=X', start='2015-03-13', end='2023-
        03-13', interval='1d')
```

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

```
daily_eur_usd # Daily dataframe
```

	Open	High	Low	Close	Adj Close	Volume
Date						
2015-03-13	1.062598	1.063000	1.048630	1.062631	1.062631	0
2015-03-16	1.048449	1.061400	1.048449	1.048163	1.048163	0
2015-03-17	1.057284	1.064900	1.055298	1.057295	1.057295	0
2015-03-18	1.059591	1.064600	1.058050	1.059805	1.059805	0
2015-03-19	1.084399	1.091405	1.062141	1.084999	1.084999	0
...	...	...	...	...	...	...
2023-03-06	1.062620	1.069381	1.062304	1.062620	1.062620	0

	Open	High	Low	Close	Adj Close	Volume
Date						
2023-03-07	1.068833	1.069633	1.057161	1.068833	1.068833	0
2023-03-08	1.055075	1.057328	1.052665	1.055075	1.055075	0
2023-03-09	1.054953	1.058649	1.053852	1.054953	1.054953	0
2023-03-10	1.058470	1.069862	1.057574	1.058470	1.058470	0

2083 rows × 6 columns

```
# Calculating the RSI with the TA library
daily_eur_usd["rsi"] = ta.momentum.RSIIndicator(daily_eur_usd["Adj
        Close"], window=14).rsi()
daily_eur_usd # Displaying dataframe
```

	Open	High	Low	Close	Adj Close	Volume
Date						
2015-03-13	1.062598	1.063000	1.048630	1.062631	1.062631	0
2015-03-16	1.048449	1.061400	1.048449	1.048163	1.048163	0
2015-03-17	1.057284	1.064900	1.055298	1.057295	1.057295	0
2015-03-18	1.059591	1.064600	1.058050	1.059805	1.059805	0
2015-03-19	1.084399	1.091405	1.062141	1.084999	1.084999	0
...	...	...	...	...	...	...
2023-03-06	1.062620	1.069381	1.062304	1.062620	1.062620	0
2023-03-07	1.068833	1.069633	1.057161	1.068833	1.068833	0

	Open	High	Low	Close	Adj Close	Volume
Date						
2023-03-08	1.055075	1.057328	1.052665	1.055075	1.055075	0
2023-03-09	1.054953	1.058649	1.053852	1.054953	1.054953	0
2023-03-10	1.058470	1.069862	1.057574	1.058470	1.058470	0

2083 rows × 7 columns

```
# Plotting candlestick chart for daily EUR/USD
fig = make_subplots(rows=2, cols=1, shared_xaxes=True,
        vertical_spacing=0.05,row_heights=[0.7, 0.3])
fig.add_trace(go.Candlestick(x=daily_eur_usd.index,
                             open=daily_eur_usd['Open'],
                             high=daily_eur_usd['High'],
                             low=daily_eur_usd['Low'],
                             close=daily_eur_usd['Adj Close'],
                             name='EUR/USD'), row=1, col=1)

# Annotation
fig.add_annotation(text='EUR/USD 1D',
                   font=dict(color='white', size=40),
                   xref='paper', yref='paper',
                   x=0.5, y=0.65,
                   showarrow=False,
                   opacity=0.2)


# Relative Strengh Index (RSI)
fig.add_trace(go.Scatter(x=daily_eur_usd.index,
                         y=daily_eur_usd['rsi'],
                         mode='lines',
                         line=dict(color='#CBC3E3'),
                         name='RSI'),
              row=2, col=1)


# Adding marking lines at 70 and 30 levels
```

```
fig.add_shape(type="line",
              x0=daily_eur_usd.index[0], y0=70,
        x1=daily_eur_usd.index[-1], y1=70,
              line=dict(color="red", width=2, dash="dot"),
              row=2, col=1)
fig.add_shape(type="line",
              x0=daily_eur_usd.index[0], y0=30,
        x1=daily_eur_usd.index[-1], y1=30,
              line=dict(color="#90EE90", width=2, dash="dot"),
              row=2, col=1)

fig.update_layout(title='EUR/USD Daily Candlestick Chart from 2015 to
        2023',
                  yaxis=dict(title='Price'),
                  height=1000,
                 template = 'plotly_dark')

# Configuring subplots
fig.update_xaxes(rangeslider_visible=False)
fig.update_yaxes(title_text='Price', row = 1)
fig.update_yaxes(title_text='RSI', row = 2)

fig.show()
```

```
# Defining the parameters for the RSI strategy
rsi_period = 14
overbought = 70
oversold = 30

# Creating a new column to hold the signals
daily_eur_usd['signal'] = 0

# Generating entry signals
for i in range(rsi_period, len(daily_eur_usd)):
    if daily_eur_usd['rsi'][i] > overbought and daily_eur_usd['rsi'][i
        - 1] <= overbought:
        daily_eur_usd['signal'][i] = -1 # sell signal
    elif daily_eur_usd['rsi'][i] < oversold and daily_eur_usd['rsi'][i
        - 1] >= oversold:
        daily_eur_usd['signal'][i] = 1 # buy signal

# Calculating total returns for the EUR/USD
daily_eur_usd['returns'] = daily_eur_usd['Adj Close'].pct_change()
daily_eur_usd['cumulative_returns'] = (1 +
        daily_eur_usd['returns']).cumprod() - 1

# Applying the signals to the returns
```

```
daily_eur_usd['strategy_returns'] = daily_eur_usd['signal'].shift(1) *
        daily_eur_usd['returns']


# Calculating the cumulative returns for the strategy
daily_eur_usd['cumulative_strategy_returns'] = (1 +
        daily_eur_usd['strategy_returns']).cumprod() - 1


# Setting the initial capital
initial_capital = 100


# Calculating portfolio value
daily_eur_usd['portfolio_value'] = (1 +
        daily_eur_usd['strategy_returns']).cumprod() *
        initial_capital


# Printing the number of trades, initial capital, and final capital
num_trades = daily_eur_usd['signal'].abs().sum()
final_capital = daily_eur_usd['portfolio_value'].iloc[-1]
total_return = (final_capital - initial_capital) / initial_capital *
        100


print('\n')
print(f"Number of trades: {num_trades}")
print(f"Initial capital: ${initial_capital}")
print(f"Final capital: ${final_capital:.2f}")
print(f"Total return: {total_return:.2f}%")
print('\n')


# Plotting portfolio evolution
fig = go.Figure()


fig.add_trace(go.Scatter(x=daily_eur_usd.index,
                         y=daily_eur_usd['portfolio_value'].round(2),
                         mode='lines',
                         line=dict(color='#00BFFF'),
                         name='Portfolio Value'))


fig.update_layout(title='Portfolio - RSI Strategy on Daily Data',
                  xaxis_title='Date',
                  yaxis_title='Value ($)',
                  template='plotly_dark',
                 height = 600)


fig.show()
```

```
Number of trades: 55
Initial capital: $100
Final capital: $97.99
Total return: -2.01%
```

Weekly Data

```
# Weekly time frame
weekly_eur_usd = yf.download('EURUSD=X', start='2015-03-13',
        end='2023-03-13', interval='1wk')
```

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

```
# Calculating the RSI with the TA library
weekly_eur_usd["rsi"] = ta.momentum.RSIIndicator(weekly_eur_usd["Adj
        Close"], window=14).rsi()
weekly_eur_usd
```

	Open	High	Low	Close	Adj Close	Volume
Date						
2015-03-09	1.062598	1.063000	1.046003	1.048163	1.048163	0
2015-03-16	1.048449	1.091405	1.048449	1.082661	1.082661	0
2015-03-23	1.082603	1.104520	1.076950	1.088495	1.088495	0
2015-03-30	1.088198	1.102196	1.071765	1.100134	1.100134	0
2015-04-06	1.100146	1.103594	1.057150	1.060985	1.060985	0
...	...	...	...	...	...	...
2023-02-06	1.079086	1.080030	1.066928	1.070320	1.070320	0
2023-02-13	1.067771	1.079844	1.061380	1.072156	1.072156	0
2023-02-20	1.068444	1.070549	1.053774	1.058649	1.058649	0
2023-02-27	1.055476	1.069107	1.053352	1.065190	1.065190	0
2023-03-06	1.062620	1.069862	1.052665	1.065871	1.065871	0

418 rows × 7 columns

```
# Plotting candlestick chart for weekly EUR/USD
fig = make_subplots(rows=2, cols=1, shared_xaxes=True,
        vertical_spacing=0.05,row_heights=[0.7, 0.3])
fig.add_trace(go.Candlestick(x=weekly_eur_usd.index,
                             open=weekly_eur_usd['Open'],
                             high=weekly_eur_usd['High'],
                             low=weekly_eur_usd['Low'],
```

```
                              close=weekly_eur_usd['Adj Close'],
                              name='EUR/USD'), row=1, col=1)

# Annotations
fig.add_annotation(text='EUR/USD 1W',
                   font=dict(color='white', size=40),
                   xref='paper', yref='paper',
                   x=0.5, y=0.65,
                   showarrow=False,
                   opacity=0.2)


# Relative Strengh Index (RSI)
fig.add_trace(go.Scatter(x=weekly_eur_usd.index,
                         y=weekly_eur_usd['rsi'],
                         mode='lines',
                         line=dict(color='#CBC3E3'),
                         name='RSI'),
              row=2, col=1)


# Adding marking lines at 70 and 30 levels
fig.add_shape(type="line",
              x0=weekly_eur_usd.index[0], y0=70,
        x1=weekly_eur_usd.index[-1], y1=70,
              line=dict(color="red", width=2, dash="dot"),
              row=2, col=1)
fig.add_shape(type="line",
              x0=weekly_eur_usd.index[0], y0=30,
        x1=weekly_eur_usd.index[-1], y1=30,
              line=dict(color="#90EE90", width=2, dash="dot"),
              row=2, col=1)

fig.update_layout(title='EUR/USD Weekly Candlestick Chart from 2015 to
        2023',
                  yaxis=dict(title='Price'),
                  height=1000,
                 template = 'plotly_dark')

# Configuring subplots
fig.update_xaxes(rangeslider_visible=False)
fig.update_yaxes(title_text='Price', row = 1)
fig.update_yaxes(title_text='RSI', row = 2)
```

```
fig.show()
```

```
# Defining the parameters for the RSI strategy
rsi_period = 14
overbought = 70
oversold = 30

# Creating a new column to hold the signals
weekly_eur_usd['signal'] = 0

# Generating entry signals
for i in range(rsi_period, len(weekly_eur_usd)):
    if weekly_eur_usd['rsi'][i] > overbought and weekly_eur_usd['rsi']
        [i - 1] <= overbought:
        weekly_eur_usd['signal'][i] = -1 # sell signal
    elif weekly_eur_usd['rsi'][i] < oversold and weekly_eur_usd['rsi']
        [i - 1] >= oversold:
        weekly_eur_usd['signal'][i] = 1 # buy signal

# Calculating total returns
weekly_eur_usd['returns'] = weekly_eur_usd['Adj Close'].pct_change()
weekly_eur_usd['cumulative_returns'] = (1 +
        weekly_eur_usd['returns']).cumprod() - 1

# Applying the signals to the returns
weekly_eur_usd['strategy_returns'] = weekly_eur_usd['signal'].shift(1)
        * weekly_eur_usd['returns']

# Calculating the cumulative returns
weekly_eur_usd['cumulative_strategy_returns'] = (1 +
        weekly_eur_usd['strategy_returns']).cumprod() - 1

# Setting the initial capital
initial_capital = 100

# Calculating total portfolio value
weekly_eur_usd['portfolio_value'] = (1 +
        weekly_eur_usd['strategy_returns']).cumprod() *
        initial_capital

# Printing the number of trades, initial capital, and final capital
num_trades = weekly_eur_usd['signal'].abs().sum()
final_capital = weekly_eur_usd['portfolio_value'].iloc[-1]
total_return = (final_capital - initial_capital) / initial_capital *
        100

print('\n')
```

```
print(f"Number of trades: {num_trades}")
print(f"Initial capital: ${initial_capital}")
print(f"Final capital: ${final_capital:.2f}")
print(f"Total return: {total_return:.2f}%")
print('\n')

# Plotting strategy returns
fig = go.Figure()

fig.add_trace(go.Scatter(x=weekly_eur_usd.index,
                         y=weekly_eur_usd['portfolio_value'].round(2),
                         mode='lines',
                         line=dict(color='#00BFFF'),
                         name='Portfolio Value'))

fig.update_layout(title='Portfolio - RSI Strategy on Weekly Data',
                  xaxis_title='Date',
                  yaxis_title='Value ($)',
                  template='plotly_dark',
                 height = 600)

fig.show()
```

```
Number of trades: 8
Initial capital: $100
Final capital: $100.32
Total return: 0.32%
```

Several conclusions can be drawn from the results above. First, the hourly timeframe is characterized by high volatility, leading to frequent swings in the RSI between 30 and 70, and resulting in a significantly higher number of trades compared to the daily and weekly timeframes.

Overall, the backtesting results indicate that the strategy was not very successful, with the portfolio losing 2.01% of its initial value on the daily timeframe and generating only small returns of 0.32% and 0.31% on the weekly and hourly timeframes, respectively. It's important to note, however, that this backtesting did not account for slippage and other additional costs, which could have a significant impact on the overall profitability of the strategy. Therefore, it's important to carefully consider these factors when applying this kind of strategy in real-world scenarios.

Moving Average Crossover Backtesting

The moving average crossover strategy, in contrast to the RSI strategy, is a trend-following strategy. This strategy is based on two different moving averages, one representing a shorter period and another representing a longer period.

When the shorter moving average crosses above the longer moving average, it's a signal to buy, and when it crosses below the longer moving average, it's a signal to sell.

For this backtesting, I used an exponential moving average of 9 periods for the shorter moving average and a simple moving average of 20 periods for the longer moving average. The choice of an exponential moving average for the shorter period is because it gives more weight to recent prices, making it more responsive to market changes compared to the simple moving average.

Once again, we'll consider an initial capital of 100.00 dollars for this backtesting.

Hourly Data

```
# Creating the 9-period exponential moving average with the TA library
hourly_eur_usd['ema9'] = ta.trend.ema_indicator(hourly_eur_usd['Adj
        Close'], window=9)

# Creating the 20-period exponential moving average with the TA
        library
hourly_eur_usd['sma20'] = ta.trend.sma_indicator(hourly_eur_usd['Adj
        Close'], window=20)
hourly_eur_usd[['Open', 'High', 'Low', 'Adj Close', 'ema9', 'sma20']]
        # Displaying dataframe with the moving averages
```

	Open	High	Low	Adj Close	ema9
Datetime					
2021-03-22 19:00:00+00:00	1.194458	1.194600	1.193887	1.194030	NaN
2021-03-22 20:00:00+00:00	1.194030	1.194458	1.193602	1.193602	NaN
2021-03-22 21:00:00+00:00	1.193602	1.194315	1.193460	1.193887	NaN

	Open	High	Low	Adj Close	ema9
Datetime					
2021-03-22 22:00:00+00:00	1.193745	1.194172	1.193460	1.193745	NaN
2021-03-22 23:00:00+00:00	1.193745	1.194172	1.193317	1.193602	NaN
...	...	...	...	...	...
2023-03-21 15:00:00+00:00	1.077935	1.078516	1.076890	1.077470	1.076459
2023-03-21 16:00:00+00:00	1.077354	1.077935	1.076542	1.077122	1.076591
2023-03-21 17:00:00+00:00	1.077238	1.077470	1.076426	1.076658	1.076605
2023-03-21 18:00:00+00:00	1.076658	1.077006	1.076079	1.077006	1.076685
2023-03-21 19:00:00+00:00	1.076774	1.077354	1.076542	1.076890	1.076726

12392 rows × 6 columns

```
# Plotting Candlestick chart with the moving averages
fig = make_subplots(rows=1, cols=1, shared_xaxes=True,
        vertical_spacing=0.05)
fig.add_trace(go.Candlestick(x=hourly_eur_usd.index,
                             open=hourly_eur_usd['Open'],
                             high=hourly_eur_usd['High'],
                             low=hourly_eur_usd['Low'],
                             close=hourly_eur_usd['Adj Close'],
                             name='EUR/USD'))
```

```
# 9 EMA
fig.add_trace(go.Scatter(x=hourly_eur_usd.index,
                         y=hourly_eur_usd['ema9'],
                         mode='lines',
                         line=dict(color='yellow'),
                         name='EMA 9'))

# 20 SMA
```

```
fig.add_trace(go.Scatter(x=hourly_eur_usd.index,
                         y=hourly_eur_usd['sma20'],
                         mode='lines',
                         line=dict(color='green'),
                         name='SMA 20'))

# Annotation
fig.add_annotation(text='EUR/USD 1HR',
                   font=dict(color='white', size=40),
                   xref='paper', yref='paper',
                   x=0.5, y=0.65,
                   showarrow=False,
                   opacity=0.2)

# Layout
fig.update_layout(title='EUR/USD Hourly Candlestick Chart from 2021 to
        2023',
                  yaxis=dict(title='Price'),
                  height=1000,
                 template = 'plotly_dark')

fig.update_xaxes(rangeslider_visible=False)
fig.show()
```

```
# Defining the parameters for the moving average crossover strategy
short_ma = 'ema9'
long_ma = 'sma20'

# Creating a new column to hold the signals
hourly_eur_usd['signal'] = 0

# Generating the entry signals
for i in range(1, len(hourly_eur_usd)):
    if hourly_eur_usd[short_ma][i] > hourly_eur_usd[long_ma][i] and
        hourly_eur_usd[short_ma][i - 1] <= hourly_eur_usd[long_ma][i
        - 1]:
        hourly_eur_usd['signal'][i] = 1 # buy signal
    elif hourly_eur_usd[short_ma][i] < hourly_eur_usd[long_ma][i] and
        hourly_eur_usd[short_ma][i - 1] >= hourly_eur_usd[long_ma][i
        - 1]:
        hourly_eur_usd['signal'][i] = -1 # sell signal

# Calculating total returns
hourly_eur_usd['returns'] = hourly_eur_usd['Adj Close'].pct_change()
hourly_eur_usd['cumulative_returns'] = (1 +
        hourly_eur_usd['returns']).cumprod() - 1
```

```
# Applying the signals to the returns
hourly_eur_usd['strategy_returns'] = hourly_eur_usd['signal'].shift(1)
        * hourly_eur_usd['returns']


# Calculating the cumulative returns
hourly_eur_usd['cumulative_strategy_returns'] = (1 +
        hourly_eur_usd['strategy_returns']).cumprod() - 1


# Setting the initial capital
initial_capital = 100


# Calculating total portfolio value
hourly_eur_usd['portfolio_value'] = (1 +
        hourly_eur_usd['strategy_returns']).cumprod() *
        initial_capital


# Printing the number of trades, initial capital, and final capital
num_trades = hourly_eur_usd['signal'].abs().sum()
final_capital = hourly_eur_usd['portfolio_value'].iloc[-1]
total_return = (final_capital - initial_capital) / initial_capital *
        100


print('\n')
print(f"Number of trades: {num_trades}")
print(f"Initial capital: ${initial_capital}")
print(f"Final capital: ${final_capital:.2f}")
print(f"Total return: {total_return:.2f}%")
print('\n')


# Plotting strategy value
fig = go.Figure()


fig.add_trace(go.Scatter(x=hourly_eur_usd.index,
                         y=hourly_eur_usd['portfolio_value'].round(2),
                         mode='lines',
                         line=dict(color='#00BFFF'),
                         name='Portfolio Value'))


fig.update_layout(title='Portfolio - Moving Average Crossover Strategy
        on Hourly Data',
                  xaxis_title='Date',
                  yaxis_title='Value ($)',
                  template='plotly_dark',
                 height = 600)
```

```
fig.show()
```

```
Number of trades: 777
Initial capital: $100
Final capital: $101.48
Total return: 1.48%
```

Daily Data

```
daily_eur_usd['ema9'] = ta.trend.ema_indicator(daily_eur_usd['Adj
        Close'], window=9)
```

```
daily_eur_usd['sma20'] = ta.trend.sma_indicator(daily_eur_usd['Adj
        Close'], window=20)
daily_eur_usd[['Open', 'High', 'Low', 'Adj Close', 'ema9', 'sma20']]
```

	Open	High	Low	Adj Close	ema9	sma20
Date						
2015-03-13	1.062598	1.063000	1.048630	1.062631	NaN	NaN
2015-03-16	1.048449	1.061400	1.048449	1.048163	NaN	NaN
2015-03-17	1.057284	1.064900	1.055298	1.057295	NaN	NaN
2015-03-18	1.059591	1.064600	1.058050	1.059805	NaN	NaN
2015-03-19	1.084399	1.091405	1.062141	1.084999	NaN	NaN
...	...	...	...	...	...	...
2023-03-06	1.062620	1.069381	1.062304	1.062620	1.062645	1.066405
2023-03-07	1.068833	1.069633	1.057161	1.068833	1.063883	1.066193
2023-03-08	1.055075	1.057328	1.052665	1.055075	1.062121	1.065298
2023-03-09	1.054953	1.058649	1.053852	1.054953	1.060687	1.064467
2023-03-10	1.058470	1.069862	1.057574	1.058470	1.060244	1.063679

2083 rows × 6 columns

```
fig = make_subplots(rows=1, cols=1, shared_xaxes=True,
        vertical_spacing=0.05)
fig.add_trace(go.Candlestick(x=daily_eur_usd.index,
                             open=daily_eur_usd['Open'],
                             high=daily_eur_usd['High'],
                             low=daily_eur_usd['Low'],
                             close=daily_eur_usd['Adj Close'],
                             name='EUR/USD'))
```

```
fig.add_trace(go.Scatter(x=daily_eur_usd.index,
                         y=daily_eur_usd['ema9'],
                         mode='lines',
                         line=dict(color='yellow'),
                         name='EMA 9'))

fig.add_trace(go.Scatter(x=daily_eur_usd.index,
                         y=daily_eur_usd['sma20'],
                         mode='lines',
                         line=dict(color='green'),
                         name='SMA 20'))

fig.add_annotation(text='EUR/USD 1D',
                   font=dict(color='white', size=40),
                   xref='paper', yref='paper',
                   x=0.5, y=0.65,
                   showarrow=False,
                   opacity=0.2)


fig.update_layout(title='EUR/USD Daily Candlestick Chart from 2015 to
        2023',
                  yaxis=dict(title='Price'),
                  height=1000,
                 template = 'plotly_dark')

fig.update_xaxes(rangeslider_visible=False)
fig.show()
```

```
# Defining the parameters for the moving average crossover strategy
short_ma = 'ema9'
long_ma = 'sma20'

# Creating a new column to hold the signals
daily_eur_usd['signal'] = 0

# Generating the entry signals
for i in range(1, len(daily_eur_usd)):
    if daily_eur_usd[short_ma][i] > daily_eur_usd[long_ma][i] and
        daily_eur_usd[short_ma][i - 1] <= daily_eur_usd[long_ma][i -
        1]:
        daily_eur_usd['signal'][i] = 1 # buy signal
    elif daily_eur_usd[short_ma][i] < daily_eur_usd[long_ma][i] and
        daily_eur_usd[short_ma][i - 1] >= daily_eur_usd[long_ma][i -
        1]:
```

```
        daily_eur_usd['signal'][i] = -1 # sell signal

# Calculating the total returns
daily_eur_usd['returns'] = daily_eur_usd['Adj Close'].pct_change()
daily_eur_usd['cumulative_returns'] = (1 +
        daily_eur_usd['returns']).cumprod() - 1


# Applying the signals to the returns
daily_eur_usd['strategy_returns'] = daily_eur_usd['signal'].shift(1) *
        daily_eur_usd['returns']


# Calculate the cumulative returns
daily_eur_usd['cumulative_strategy_returns'] = (1 +
        daily_eur_usd['strategy_returns']).cumprod() - 1


# Setting the initial capital
initial_capital = 100


# Calculating the total portfolio value
daily_eur_usd['portfolio_value'] = (1 +
        daily_eur_usd['strategy_returns']).cumprod() *
        initial_capital


# Printing the number of trades, initial capital, and final capital
num_trades = daily_eur_usd['signal'].abs().sum()
final_capital = daily_eur_usd['portfolio_value'].iloc[-1]
total_return = (final_capital - initial_capital) / initial_capital *
        100


print('\n')
print(f"Number of trades: {num_trades}")
print(f"Initial capital: ${initial_capital}")
print(f"Final capital: ${final_capital:.2f}")
print(f"Total return: {total_return:.2f}%")
print('\n')


# Plotting strategy portfolio
fig = go.Figure()


fig.add_trace(go.Scatter(x=daily_eur_usd.index,
                         y=daily_eur_usd['portfolio_value'].round(2),
                         mode='lines',
                         line=dict(color='#00BFFF'),
                         name='Portfolio Value'))
```

```
fig.update_layout(title='Portfolio - Moving Average Crossover Strategy
        on Daily Data',
                  xaxis_title='Date',
                  yaxis_title='Value ($)',
                  template='plotly_dark',
                 height = 600)

fig.show()
```

```
Number of trades: 131
Initial capital: $100
Final capital: $93.95
Total return: -6.05%
```

Weekly Data

```
weekly_eur_usd['ema9'] = ta.trend.ema_indicator(weekly_eur_usd['Adj
        Close'], window=9)


weekly_eur_usd['sma20'] = ta.trend.sma_indicator(weekly_eur_usd['Adj
        Close'], window=20)
weekly_eur_usd[['Open', 'High', 'Low', 'Adj Close', 'ema9', 'sma20']]
```

	Open	High	Low	Adj Close	ema9	sma20
Date						
2015-03-09	1.062598	1.063000	1.046003	1.048163	NaN	NaN
2015-03-16	1.048449	1.091405	1.048449	1.082661	NaN	NaN
2015-03-23	1.082603	1.104520	1.076950	1.088495	NaN	NaN
2015-03-30	1.088198	1.102196	1.071765	1.100134	NaN	NaN
2015-04-06	1.100146	1.103594	1.057150	1.060985	NaN	NaN
...	...	...	...	...	...	...
2023-02-06	1.079086	1.080030	1.066928	1.070320	1.071372	1.040531
2023-02-13	1.067771	1.079844	1.061380	1.072156	1.071529	1.045115
2023-02-20	1.068444	1.070549	1.053774	1.058649	1.068953	1.049345
2023-02-27	1.055476	1.069107	1.053352	1.065190	1.068200	1.053981
2023-03-06	1.062620	1.069862	1.052665	1.065871	1.067734	1.057960

418 rows × 6 columns

```
fig = make_subplots(rows=1, cols=1, shared_xaxes=True,
        vertical_spacing=0.05)
fig.add_trace(go.Candlestick(x=weekly_eur_usd.index,
                             open=weekly_eur_usd['Open'],
```

```
                                high=weekly_eur_usd['High'],
                                low=weekly_eur_usd['Low'],
                                close=weekly_eur_usd['Adj Close'],
                                name='EUR/USD'))


fig.add_trace(go.Scatter(x=weekly_eur_usd.index,
                         y=weekly_eur_usd['ema9'],
                         mode='lines',
                         line=dict(color='yellow'),
                         name='EMA 9'))

fig.add_trace(go.Scatter(x=weekly_eur_usd.index,
                         y=weekly_eur_usd['sma20'],
                         mode='lines',
                         line=dict(color='green'),
                         name='SMA 20'))

fig.add_annotation(text='EUR/USD 1WK',
                   font=dict(color='white', size=40),
                   xref='paper', yref='paper',
                   x=0.5, y=0.65,
                   showarrow=False,
                   opacity=0.2)


fig.update_layout(title='EUR/USD Weekly Candlestick Chart from 2015 to
        2023',
                  yaxis=dict(title='Price'),
                  height=1000,
                 template = 'plotly_dark')

fig.update_xaxes(rangeslider_visible=False)
fig.show()
```

```
# Defining the parameters for the moving average crossover strategy
short_ma = 'ema9'
long_ma = 'sma20'

# Creating a new column to hold the signals
weekly_eur_usd['signal'] = 0

# Generating the entry signals
for i in range(1, len(weekly_eur_usd)):
```

```
    if weekly_eur_usd[short_ma][i] > weekly_eur_usd[long_ma][i] and
        weekly_eur_usd[short_ma][i - 1] <= weekly_eur_usd[long_ma][i
        - 1]:
        weekly_eur_usd['signal'][i] = 1 # buy signal
    elif weekly_eur_usd[short_ma][i] < weekly_eur_usd[long_ma][i] and
        weekly_eur_usd[short_ma][i - 1] >= weekly_eur_usd[long_ma][i
        - 1]:
        weekly_eur_usd['signal'][i] = -1 # sell signal


# Calculating the total returns
weekly_eur_usd['returns'] = weekly_eur_usd['Adj Close'].pct_change()
weekly_eur_usd['cumulative_returns'] = (1 +
        weekly_eur_usd['returns']).cumprod() - 1


# Applying the signals to the returns
weekly_eur_usd['strategy_returns'] = weekly_eur_usd['signal'].shift(1)
        * weekly_eur_usd['returns']


# Calculating the cumulative returns
weekly_eur_usd['cumulative_strategy_returns'] = (1 +
        weekly_eur_usd['strategy_returns']).cumprod() - 1


# Setting the initial capital
initial_capital = 100


# Calculating the portfolio value
weekly_eur_usd['portfolio_value'] = (1 +
        weekly_eur_usd['strategy_returns']).cumprod() *
        initial_capital


# Printing the number of trades, initial capital, and final capital
num_trades = weekly_eur_usd['signal'].abs().sum()
final_capital = weekly_eur_usd['portfolio_value'].iloc[-1]
total_return = (final_capital - initial_capital) / initial_capital *
        100


print('\n')
print(f"Number of trades: {num_trades}")
print(f"Initial capital: ${initial_capital}")
print(f"Final capital: ${final_capital:.2f}")
print(f"Total return: {total_return:.2f}%")
print('\n')


# Plotting the strategy total returns
fig = go.Figure()


fig.add_trace(go.Scatter(x=weekly_eur_usd.index,
```

```
                         y=weekly_eur_usd['portfolio_value'].round(2),
                         mode='lines',
                         line=dict(color='#00BFFF'),
                         name='Portfolio Value'))

fig.update_layout(title='Portfolio - Moving Average Crossover Strategy
        on Weekly Data',
                  xaxis_title='Date',
                  yaxis_title='Value ($)',
                  template='plotly_dark',
                 height = 600)

fig.show()
```

```
Number of trades: 24
Initial capital: $100
Final capital: $93.20
Total return: -6.80%
```

Although the plots of the total portfolio values for the RSI strategies appeared to be stationary, the portfolio evolution for the moving average crossover strategy exhibits some sort of trend, especially in the hourly data. As can be seen, the portfolio experienced gradual losses from April 2021 to October 2021, which then turned to exponential losses. However, since May 2022, the strategy has been generating gains again and has been performing quite well in recent months, at least on the 1 hour timeframe.

Despite this, the strategy generated only a 1.37% return after 777 trades on the hourly chart, without accounting for taxes, slippage, and other costs. On the daily timeframe, the strategy resulted in 131 trades with a total loss of 6.05%, closing at 93.95 dollars. Similarly, on the weekly timeframe, the strategy presents a total loss of 6.80%. And, of course, it is important to mention that additional costs and factors like slippage could further impact the final returns.

Overall, the backtests presented here are intended to encourage you to perform your own tests. I highly suggest that you include additional factors such as slippage, broker costs, and taxes in your analysis, and try combining multiple strategies simultaneously.

In this notebook, we have only tested the RSI and the moving average crossover strategies individually. You can, however, combine these strategies, or use the RSI in conjunction with other indicators such as Bollinger Bands, to develop more complex strategies.

Ultimately, the key to successful backtesting is to continuously refine and optimize your strategies based on your own experiences and conditions.

Conclusion

This notebook provided an introduction to using Python and Data Science for Financial Markets. You learned how to analyze financial assets, build portfolios, and optimize them for the best risk-adjusted returns. You also learned how to evaluate portfolios and strategies, calculate technical indicators, and plot them on a candlestick chart with Plotly. Furthermore, you learned how to obtain some of the most widely used fundamental indicators and backtest technical strategies on historic data.

However, this is just the beginning of your journey into the world of financial data analysis. To avoid overwhelming you with information, I've decided to conclude this introductory notebook here.

But don't worry! Soon, I'll post the second part of this notebook, which will be entirely dedicated to using Machine Learning algorithms for financial markets. So, stay tuned for that!

Thank you so much for reading!.

Follow me on LinkedIn. I'm always posting data science and machine learning content, as well as the use of financial metrics and strategies based on data science.

Best regards,

Luís Fernando Torres