In Summary;

In this Python project, I developed an exchange portfolio by first importing the necessary libraries and selecting the desired exchanges for analysis. I then arranged the dates and connected Python to the exchange websites to retrieve relevant data. This data was used to calculate lognormal returns, which measure the continuous compound returns of the selected exchanges. Using this data, I calculated various financial metrics such as the covariance matrix, which captures the relationship between the returns of different exchanges, as well as the portfolio standard deviation, which measures the overall risk of the portfolio. Additionally, I determined the expected return of the portfolio based on historical data, as well as the Sharpe ratio, which evaluates the risk-adjusted return, and the risk-free rate, representing the return on a theoretically risk-free asset.

Once the necessary calculations were made, I applied constraints and bounds to the optimization process, ensuring the portfolio adhered to certain limits, such as not exceeding certain allocation percentages for specific exchanges. Initial weights were also defined to establish a starting point for the optimization algorithm. Using these initial weights and constraints, the algorithm was able to optimize the portfolio's asset allocation to maximize returns while minimizing risk. The results were then visualized, providing a clear view of the portfolio's performance and allocation, helping to assess the trade-off between risk and return. This visualization also helped in making data-driven decisions for refining the portfolio to achieve optimal performance.

IMPORT REQUIRED LIBRARIES

```
import yfinance as yf
import pandas as pd
from datetime import datetime, timedelta
import numpy as np
from scipy.optimize import minimize
import matplotlib.pyplot as plt
```

DEFINE THE LIST OF TICKERS

* Select the desired exchange.

```
tickers = ['SFY', 'BND', 'GLD', 'QQQ', 'VTI']
```

* Set the end date to today.

```
end_date = datetime.today()
print(end_date)
```

* Set start date to 5 years ago.

```
start_date = end_date - timedelta(days = 10*365)
print(start_date)
```

* Create an empty Dataframe to store.

```
adj_close_df = pd.DataFrame()
```

* Import data from exchange website.

```
for ticker in tickers:
   data = yf.download(ticker, start = start_date, end = end_date)
   adj_close_df[ticker] = data['Adj Close']
```

* Print the desired exchange values between desired years.

```
print(adj_close_df)
```

* Calculate the Lognormal returns for each ticker. Lognormal returns refer to the concept that financial returns, when expressed in terms of logarithmic returns, follow a normal distribution.

* Positive Returns: Indicate a growth or appreciation in asset value.

* Negative Returns: Indicate a decline or depreciation in asset value.

```
log_returns = np.log(adj_close_df/adj_close_df.shift(1))

print(log_returns)
```

* Drop any missing values.

```
log_returns = log_returns.dropna()
```

* Calculate the covariance matrix using annualized log returns.

* A covariance matrix using annualized log returns is a key statistical tool in finance that measures the relationship between the annualized log returns of multiple assets.

* Variance (diagonal): Shows the individual risk of each asset.

* Covariance (off-diagonal): Shows how asset returns interact.

```
cov_matrix = log_returns.cov()*252

print(cov_matrix)
```

# Positive covariance: Indicates that variables tend to increase together.

# Negative covariance: Indicates that one variable tends to increase while the other decreases.

* Calculate the portfolio standard deviation.

* The portfolio standard deviation measures the overall risk of a portfolio's returns, accounting for the individual asset risks and how those assets interact with each other through correlations. It provides insight into the portfolio's total variability of returns.

* Higher Standard Deviation: Indicates higher portfolio volatility and risk. Returns are more spread out from the mean.

* Lower Standard Deviation:

Suggests more stability and lower risk.

Returns are closer to the mean, indicating less variability.

```
def standart_deviation(weights, cov_matrix):

   variance = weights.T @ cov_matrix @ weights

   return np.sqrt(variance)
```

* Calculate the expected return.

```
def expected_return(weights, log_returns, cov_matrix=None, risk_free_rate=None):
    return np.sum(log_returns.mean()*weights)*252
```

* Calculate the Sharp Ratio. The Sharpe Ratio is a measure of the risk-adjusted return of an investment or portfolio. It evaluates how well the investment compensates an investor for the level of risk taken, providing a standardized way to compare different investments.

```
def sharp_ratio(weights, log_returns, cov_matrix, risk_free_rate):
    return(expected_return(weights, log_returns, cov_matrix, risk_free_rate))
```

* Set risk free rate. The risk-free rate is the theoretical return on an investment with zero risk of financial loss. It represents the minimum return an investor would expect for any investment because it assumes no uncertainty or risk.

```
from fredapi import Fred

fred = Fred(api_key='92d2e66ede89ea6ef46a8149e6f96ad8')

ten_year_treasury_rate = fred.get_series_latest_release('GS10')/100
```

# Set the risk free rate

```
risk_free_rate = ten_year_treasury_rate.iloc[-1]

print(risk_free_rate)
```

* Define the function to minimize (negative Sharp Ratio).

```
def neg_sharp_ratio(wheights, log_returns, cov_matrix, risk_free_rate):
    return -sharp_ratio(wheights, log_returns, cov_matrix, risk_free_rate)
```

* Set the constraints and bounds. Constraints are the specific conditions or rules that a solution must satisfy in a mathematical model or optimization problem. They can be equations or inequalities that define relationships between variables. Bounds are simple restrictions on the range of individual variables in a problem. They define the minimum and maximum values that a variable can take.

```
constraints = {'type': 'eq', 'fun': lambda weights: np.sum(weights) -1}

bounds = [(0, 0.5) for _ in range(len(tickers))]
```

* Set the initial weights. In the context of machine learning and neural networks, initial weight refers to the starting values assigned to the weights of the connections between neurons in the model. Weights are the parameters that determine how input data is transformed and propagated through the network to make predictions.

```
initial_weights = np.array([1/len(tickers)]*len(tickers))
```

* Optimize the weights to maximize the Sharp Ratio.

```
optimized_results = minimize(
   neg_sharp_ratio,
   initial_weights,
   args=(log_returns, cov_matrix, risk_free_rate),
   method='SLSQP',
   constraints=constraints,
   bounds=bounds
)
```

* Get the opitmal weights.

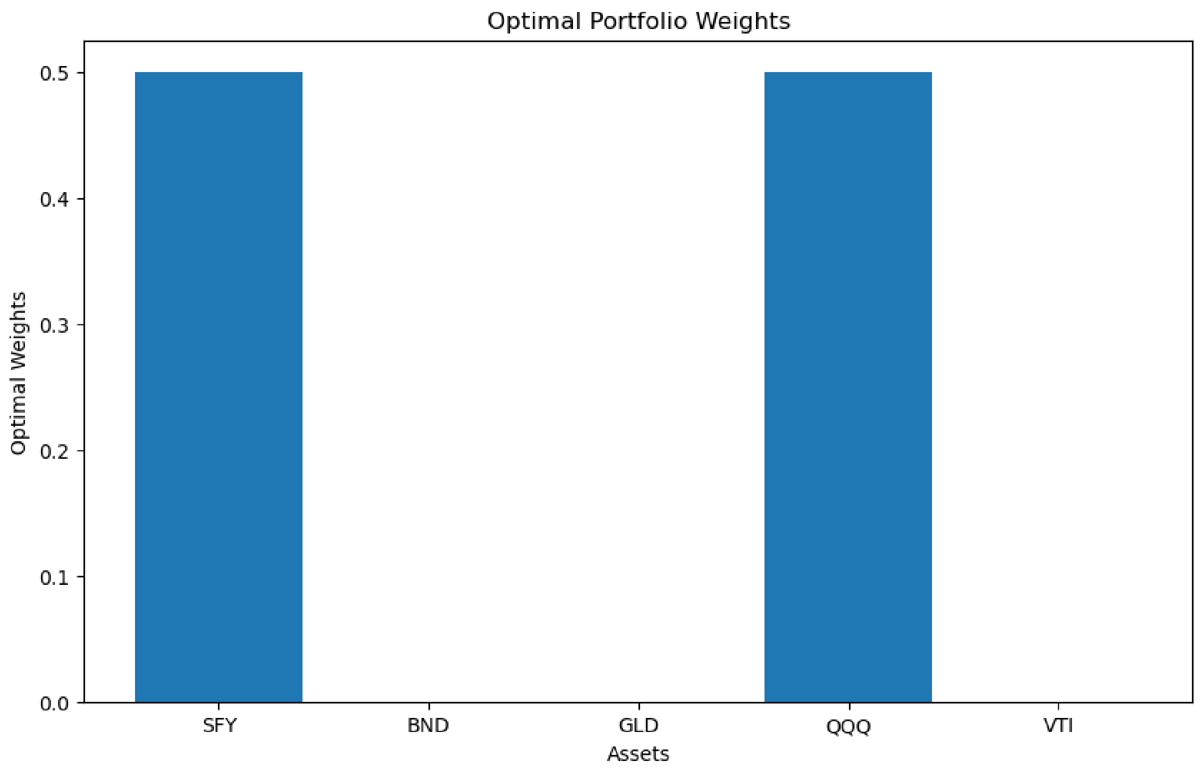
```
optimal_weights = optimized_results.x
```

* Analyze the Optimal Portfolio.

```
print('Optimal Weights:')
for ticker, weight in zip(tickers, optimal_weights):
   print(f"{ticker}: {weight:4f}")
   print()
   optimal_portfolio_return = expected_return(optimal_weights, log_returns)
   optimal_portfolio_volatility = standart_deviation(optimal_weights, cov_matrix)
   optimal_sharpe_ratio = sharp_ratio(optimal_weights, log_returns, cov_matrix,
risk_free_rate)


   print(f"Expected Annual Return: {optimal_portfolio_return: 4f}")
   print(f"Expected Volatility: {optimal_portfolio_volatility: 4f}")
   print(f"Sharpe Ratio: {optimal_sharpe_ratio: 4f}")
```

* Display the Final Portfolio.

```
plt.figure(figsize=(10,6))
plt.bar(tickers, optimal_weights)


plt.xlabel('Assets')
plt.ylabel('Optimal Weights')
plt.title('Optimal Portfolio Weights')


plt.show()
```


Optimal Portfolio Weights
Optimal Weights
SFY
BND
GLD
QQQ
VTI
Assets