In [1]:

```python
# Import Modules
import pandas as pd
import numpy as np
import yfinance as yf
import matplotlib.pyplot as plt
%matplotlib inline

# Disable Warnings
import warnings
warnings.filterwarnings('ignore')

# Allow Multiple Output per Cell
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity='all'

# StatsModels for Ordinary Least Squares Regresssion
import statsmodels.api as sm

# Import the adfuller (ADF) Stationarity Test
from statsmodels.tsa.stattools import adfuller

# Import QuantStats for Trading Strategy Tear-Sheets
import quantstats as qs
```

In [2]:

```python
# Market Data: S&P 500 Index & Google
sp500 = '^GSPC'
google = 'GOOG'
tickers = [sp500, google]

start = '2020-01-01'
end= '2024-01-01'

# Create DataFrame
df = pd.DataFrame(columns=tickers)

# Download Adjusted Close Prices from Yahoo Finance!
df[tickers[0]] = yf.download(tickers[0], start, end, progress=False)['A
df[tickers[1]] = yf.download(tickers[1], start, end, progress=False)['A

# Display DataFrame
df.head()
```

Out[2]:

Date	^GSPC	GOOG
2020-01-02	3257.850098	68.368500
2020-01-03	3234.850098	68.032997
2020-01-06	3246.280029	69.710503
2020-01-07	3237.179932	69.667000
2020-01-08	3253.050049	70.216003

In [3]:
```
# Plot Market Data
fig, ax = plt.subplots(figsize=(10,5))
ax.plot(df.index, df[tickers[0]], color='blue')
plt.ylabel(tickers[0], color='blue')
ax1 = ax.twinx()
ax1.plot(df.index, df[tickers[1]], color='green')
plt.ylabel(tickers[1], color='green')
plt.grid()
plt.title(f'{tickers[0]} vs {tickers[1]}')
plt.show();
```

In [4]:
```
# Uncomment to Display Market Data
#df.head()
```

In [5]:
```
# Compute the Beta or hedge ratio between Y (Google) and X (S&P 500 Ind
Y = df[tickers[0]]
x = df[tickers[1]]

# Add Constant Intercept Term
X = sm.add_constant(x)

# Peform OLS
result = sm.OLS(Y,X).fit()

# Uncomment to View Summary
#result.summary()

# Beta or Hedge Ratio
hedge_ratio = result.params[1]
print(f'Hedge Ratio: {hedge_ratio:.4f}')
```

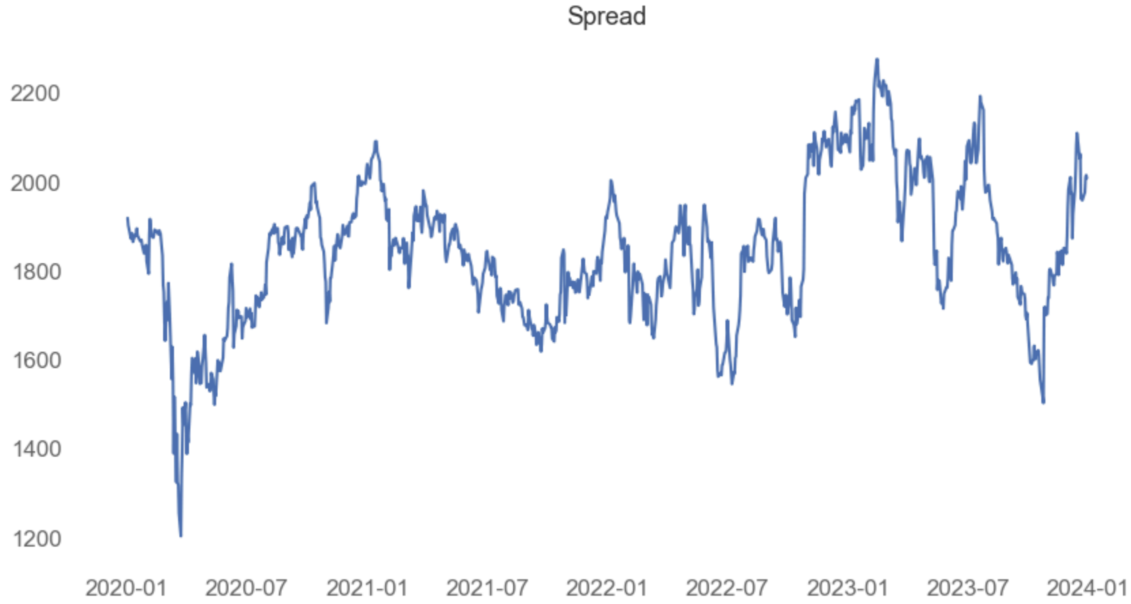
```
Hedge Ratio: 19.5973
```

In [6]:

```
# Compute the Spread
df['spread'] = Y - hedge_ratio * x

# Also Compute the Spread Log Return
df['log_return'] = np.log(df['spread']/df['spread'].shift(1))

# Uncomment to Display the Spread Data Frame
#df.head()
```

In [7]:

```
# Plot the Spread
fig = plt.figure(figsize=(10,5))
plt.plot(df['spread'])
plt.grid()
plt.title('Spread')
plt.show();
```



Test for Co-Integration or Stationarity

In [8]:

```
# Test the Spread for Stationarity using ADF
adf = adfuller(df['spread'], maxlag=1)

# p-value
p_value = adf[1]

# Print ADF Results
print(f'ADF Result Parameters \n{adf}\n')

# Print Test Statistic
print(f'Test Statistic: {adf[0]:.4f}')

# Print Critical Value
print(f'Critical Value: {adf[4]["5%"]:.4f}')

# Print P-Value
print(f'P-Value: {adf[1]*100:.4f}%')
```

```
ADF Result Parameters
(-3.3309721139421318, 0.013549524424124 89, 0, 1005, {'1%': -3.436873463813
0847, '5%': -2.8644201518188126, '10%': -2.5683035273879358}, 9950.2491905
86503)

Test Statistic: -3.3310
Critical Value: -2.8644
P-Value: 1.3550%
```

In [9]:

```
# Method to Print Stationarity Result
def is_stationary(p_value):
    if (p_value<0.05):
        print(f'Series is Stationary (p-value {p_value*100:.4f}%)')
    else:
        print(f'Series NOT Stationary (p-value {p_value*100:.4f}%)')
    return

# Display Stationarity Result
is_stationary(p_value)
```

```
Series is Stationary (p-value 1.3550%)
```

Compute Z-Score

We compute the normalized z-score as follows:

Z-Score $= \left(\frac{x-\mu}{\sigma}\right)$

In [10]:
```
# Compute Mean and Std using window specified
window = 30
df['mean'] = df['spread'].ewm(span=window).mean()
df['std'] = df['spread'].ewm(span=window).std()

# Drop NaN Values
df.dropna(inplace=True)

# Compute the Z-Score
df['z_score'] = ( df['spread'] - df['mean'] ) / df['std']

# Z-Score Boundaries
df['z_up'] = 2.0
df['z_down'] = -2.0

# Uncomment to Display DataFrame
#df.head()
```

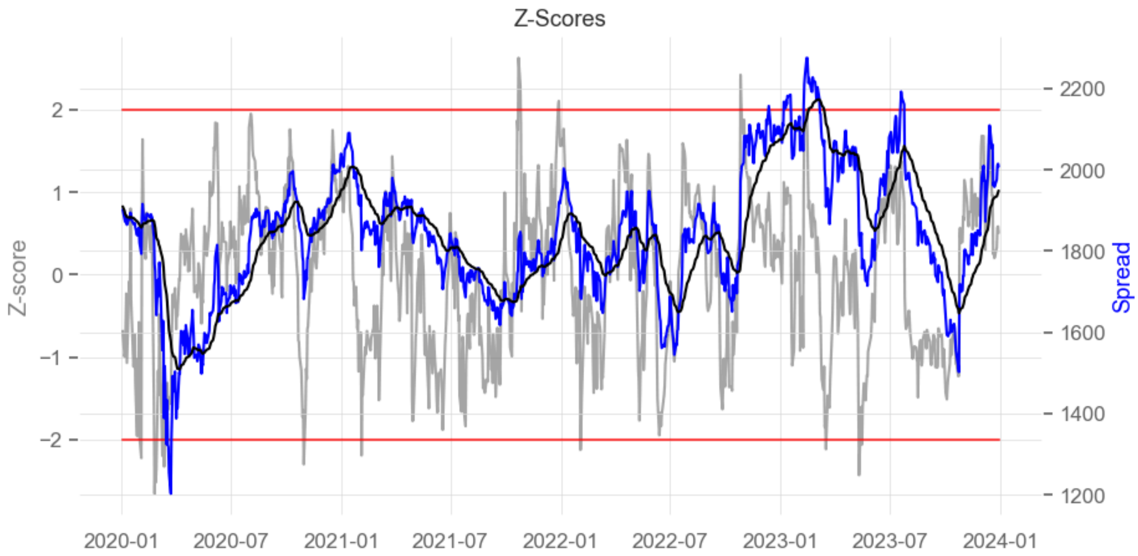
Trading Signals

In [11]:
```
# Long Trading Signals
long_entry = (df['z_score'] <= df['z_down'])
long_exit = (df['z_score'] >= 0)

# Initialize Long Position Column to NaN
df['long_pos'] = np.nan

# Apply Long Trading Signals
df.loc[long_entry, 'long_pos'] = 1
df.loc[long_exit, 'long_pos'] = 0

# Forward Fill NaN Values
df['long_pos'].fillna(method='ffill', inplace=True)

# Fill any remaining NaN values wth Zero
df['long_pos'].fillna(0, inplace=True)
```

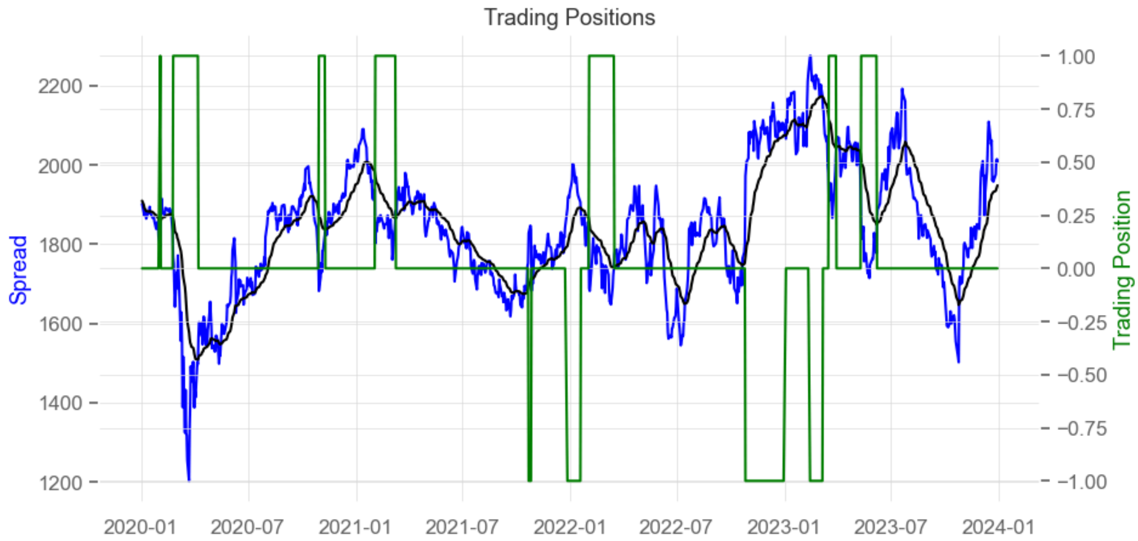
In [12]:
```
# Short Trading Signals
short_entry = (df['z_score'] >= df['z_up'])
short_exit = (df['z_score'] <= 0)

# Initialize Short Position Column to NaN
df['short_pos'] = np.nan

# Apply Long Trading Signals
df.loc[short_entry, 'short_pos'] = -1
df.loc[short_exit, 'short_pos'] = 0

# Forward Fill NaN Values
df['short_pos'].fillna(method='ffill', inplace=True)

# Fill any remaining NaN values wth Zero
df['short_pos'].fillna(0, inplace=True)
```

In [13]:
```
# Total Trading Position
df['total_pos'] = df['long_pos'] + df['short_pos']

# Uncomment to Display DataFrame
#df.head()
```

In [14]:
```
# Plot the Z-Scores
fig, ax = plt.subplots(figsize=(10,5))
ax.plot(df.index, df['z_score'], color='grey', alpha=0.7)
ax.plot(df.index, df['z_up'], color='red', alpha=0.7)
ax.plot(df.index, df['z_down'], color='red', alpha=0.7)
ax.set_ylabel('Z-score', color='grey')
ax1 = ax.twinx()
ax1.plot(df.index, df['spread'], color='blue')
ax1.plot(df.index, df['mean'], color='black')
ax1.set_ylabel('Spread', color='blue')
plt.title('Z-Scores')
plt.show();
```



In [15]:
```
# Plot Trading Positions
fig, ax = plt.subplots(figsize=(10,5))
plt.plot(df.index, df['spread'], color='blue')
plt.plot(df.index, df['mean'], color='black')
plt.ylabel('Spread', color='blue')
ax1 = ax.twinx()
ax1.plot(df.index, df['total_pos'], color='green')
ax1.set_ylabel('Trading Position', color='green')
plt.title('Trading Positions')
plt.show();
```



Performance

In [16]:
```
# Compute Strategy Returns - Shift Position by 1 Day to mitigate Look-a
df['strategy_returns'] = df['total_pos'].shift(1) * df['log_return']
strategy_mean = df['strategy_returns'].mean()
strategy_std = df['strategy_returns'].std()

# Compute the Sharpe Ratio
risk_free_rate = 0.05 # assume 5.0% per year

# Daily Sharpe Ratio
# Note: we convert risk-free rate to a daily rate i.e. divide by 252
sharpe_daily = (strategy_mean - (risk_free_rate/252) ) / strategy_std
print(f'Daily Sharpe Ratio: {sharpe_daily:.4f}')

# Annualized Sharpe Ratio
# Scale daily by t/sqrt(t) = sqrt(t) i.e. sqrt(252)
sharpe_annual = sharpe_daily * np.sqrt(252)
print(f'Annual Sharpe Ratio: {sharpe_annual:.4f}')
```

```
Daily Sharpe Ratio: 0.0083
Annual Sharpe Ratio: 0.1323
```

In [17]:
```
# QuantStats Tearsheet
qs.reports.basic(df['strategy_returns'], rf = risk_free_rate)
```

Performance Metrics

```
                    Strategy
------------------  ----------
Start Period        2020-01-06
End Period          2023-12-29
Risk-Free Rate      5.0%
Time in Market      20.0%

Cumulative Return   25.16%
CAGR﹪              3.97%

Sharpe              0.14
Prob. Sharpe Ratio  20.87%
Sortino             0.2
Sortino/√2          0.14
Omega               1.06

Max Drawdown        -34.63%
```