Quantitative Trading Strategy Project

Introduction

This project focuses on designing a momentum-based short-selling strategy around **index rebalancing events**, particularly targeting **one-off events**. The strategy is built using quantitative analysis, technical indicators, and strict liquidity constraints to capture price distortions caused by index additions. Risk management is integrated through a **SPY hedge** to mitigate market-wide movements. The main technical indicators used in this strategy include **RSI**, **MACD**, **Bollinger Bands**, and **ATR**, which are well-suited for identifying overbought conditions and momentum shifts based on my personal equity trading experience.

Data Extraction, Cleaning, and Liquidity Filtering

Initial Data Inspection and Cleaning

I began by loading the dataset and addressing missing values using `pandas` and defining appropriate `na_values`. Initial attempts to fill missing fields (like `$MM to Trade` and `Shares to Trade`) were later abandoned as most stocks with missing data were delisted or invalid. For conciseness, I removed these tickers and cleaned the data thoroughly.

Special characters in the **Ticker** column (e.g., `*`, `-W`, `-V`) were stripped, and spaces were removed. The validity of each ticker was confirmed by fetching historical data from Yahoo Finance (`yfinance`), ensuring that only tickers with valid data were retained.

Event Classification: One-off vs. Scheduled Events

The dataset was classified into:

- **One-off Events**: Events occurring outside regularly scheduled rebalancing months.
- **Scheduled Events**: Events occurring in March, June, September, and December.

This allowed for filtering based on event type, with a focus on **one-off events** for strategy development.

Liquidity Constraints

To minimize market impact, I imposed a **liquidity constraint** based on the 20-day average trading volume, ensuring that no position size exceeded 1% of a stock's average volume, with a **minimum threshold of 50,000 shares**. This constraint was designed to ensure smooth trade execution without moving the market.

1. **Mathematical Justification**: In managing a 5,000,000 portfolio, I allocate 76% of the total capital to my aggressive short strategy, splitting this evenly across four stocks, meaning each stock receives 19% of the portfolio. The remaining 24% is allocated to a SPY hedge.To avoid creating market distortions while deploying substantial capital, I ensure that my strategy does not exceed 1% of the 20-day average trading volume for each stock. Given my capital and liquidity constraints, this balance is critical in ensuring smooth trade execution without becoming a "whale" in the market.

For context, the maximum position I could theoretically take would be 50,000 shares of a stock priced at 100 dollars. However, given that only 76% of my capital is allocated to the aggressive strategy, my total allocation per stock is limited to 19% of 5,000,000 or 950,000 per stock.

This calculation allows me to trade substantial volumes without impacting the market. For instance:

If a stock's price is 100 dollars, I can purchase 9,500 shares (950,000 / 100), which is well below the typical 50,000-share daily average.

Even if a stock’s price were lower (e.g., 50 per share), my 950,000 allocation would enable me to purchase 19,000 shares, which would still be only a fraction of the daily trading volume (about 38% of a typical 50,000-share daily average).

By adhering to these liquidity constraints, I can confidently deploy large sums of capital in stocks without exceeding 1% of the daily trading volume. This ensures that my trades don’t move the market or attract unwanted attention from larger players or algorithms, allowing me to act efficiently without becoming a "whale" and disrupting normal market behavior.

1. **Market Dynamics**: In high-frequency environments, large trades can attract unwanted attention from algorithms. The liquidity constraint ensures minimal market disruption.

Data Extraction for Trading Strategy

After filtering for liquidity, I extracted 100 days of **Open**, **Close**, **High**, **Low**, and **Volume** data starting from the event announcement date. This data formed the basis for technical analysis and signal generation.

Creation of Baseline Analysis Techniques: Leveraging Technical Analysis from Personal Equity Trading Experience

Intraday vs. Daily Data

Where available, I utilized **intraday data** for signal generation, as index rebalancing events often lead to short-term price distortions. For stocks without intraday data, I relied on **daily**

data. Intraday data provides more precise entries and exits, especially in volatile conditions.

Technical Indicators and Their Role

I chose **RSI**, **Bollinger Bands**, **MACD**, and **ATR** for detecting overbought conditions and momentum shifts. Here's how each indicator contributes to the strategy:

- **RSI**: Identifies overbought conditions, signaling when to enter a short position.
- **Bollinger Bands**: Detect price deviations from average volatility, triggering short signals when prices exceed the upper band.
- **MACD**: Confirms bearish momentum, particularly when it crosses below the signal line.
- **ATR**: Measures volatility and sets stop-loss levels, avoiding premature exits during high volatility.

Strategy Insights from Personal Trading Experience

Drawing from my trading experience, I often observed retail traders "**holding the bag**"—continuing to hold stocks after their fair value had been distorted by market events. This strategy exploits such distortions by shorting **one-off event** stocks after their prices temporarily rise due to index additions.

Volume Dynamics in One-off Events

Index additions often trigger a **spike in volume**. This temporary volume increase allows my strategy to capitalize on price distortions. The strategy benefits from the natural price reversion that occurs once the initial excitement surrounding index inclusion fades.

Trading Constraints and Strategy Adjustments

Given the restriction against **intraday buying and selling**, I adopted a conservative approach—focusing on **shorting overbought stocks** rather than long positions. I named this strategy **"Mr. Bear"**, as it exploits overbought conditions to capture price reversals. The strategy is designed to re-enter after stop-losses if overbought conditions persist, offering flexibility.

Simplicity over Complexity

Initially, I considered more complex approaches such as **pairs trading** and **cointegration**. However, I chose to focus on a simpler, momentum-based approach first, which yielded strong results. Simplicity often proves highly effective when backed by a deep understanding of market dynamics.

Although I did not implement predictive models in this strategy, I would explore more complex techniques (e.g., **regularized regression with technical indicators** and **machine learning models**) to predict future events or test the strategy on unseen data in a real-world environment. If you're interested in my predictive models using **ML algorithms** and

Statistical Computing techniques, feel free to check out my **Quantitative Research Project** on my GitHub.

Statistical Assumptions and Validation

In this strategy, I assumed that **price reversion** would naturally occur after index inclusion due to overbought conditions. This assumption relies on **mean reversion** theory, which posits that prices will revert to their average over time. While my focus was on one-off events, further testing would involve ensuring the strategy's robustness by testing it on:

- **Different market conditions**: How does the strategy perform during a bull market or a bear market?
- **Cross-validation**: Implementing walk-forward optimization techniques to ensure that the parameters (e.g., RSI thresholds) aren't overfitted to the historical data.
- **Out-of-sample testing**: Applying the strategy to unseen data to validate its predictive power.

Robustness and Risk Management

Future iterations would include **robustness checks** by adding:

- **Monte Carlo simulations**: To evaluate the strategy's performance under different random market scenarios.
- **Backtesting on additional indices or asset classes**: To see if the strategy generalizes well beyond the S&P 500 index.
- **Hedging enhancements**: Exploring dynamic hedging strategies to adjust the SPY hedge based on market conditions.

Strategy Workflow:

1. **Data Cleaning**:
    - Handled missing values and cleaned ticker symbols.
    - Validated tickers using `yfinance`.
2. **Liquidity Filtering**:
    - Set a minimum liquidity threshold of 50,000 shares, representing 1% of the 20-day average trading volume.
3. **Data Extraction**:
    - Extracted 100 days of Open, Close, High, Low, and Volume data for **One-off** and **Scheduled Events**.
4. **Technical Indicators**:

- **RSI** identified overbought conditions for short entries.
- **Bollinger Bands** and **MACD** confirmed momentum, while **ATR** informed stop-loss levels and presence of volatility.

5. **Strategy Execution**:
    - **Signal Generation**: Short positions triggered by overbought conditions and excessive price moves.
    - **Exit Strategy**: Exited below the lower Bollinger Band or at 20% profit; stop-losses were triggered by continued momentum.
6. **Portfolio Allocation**:
    - Allocated 19% of the portfolio to each liquid stock and reserved 24% for the SPY hedge.
    - The Mathematical reasoning for this is that I wanted my aggressive bearish strategy to dominate, but also consider instances where I may be getting stopped out repeatedly and in these instances, I am able to reduce the losses or offset them with the SPY hedge.
7. **SPY Hedge**:
    - Used a long SPY position to hedge against market risk and mitigate losses during uptrends.
8. **Transaction and Overnight Costs**:
    - Included transaction costs ($0.01 per share) and overnight holding costs (based on the **Fed Funds Rate**).
9. **Performance Evaluation**:
    - Assessed profit and loss for each stock, calculated cumulative returns, and factored in all costs.

Key Takeaways:

1. **Liquidity Constraints**: Ensured smooth execution and minimal market impact.
2. **Effective Use of Indicators**: The combination of RSI and Bollinger Bands effectively captured overbought conditions for short entries.
3. **SPY Hedge**: Provided protection during market-wide uptrends, offsetting potential short strategy losses.
4. **Cost Management**: Factored in transaction and overnight costs for accurate profit calculations.
5. **Simplicity Wins**: A straightforward strategy focusing on technical analysis proved highly effective. Future iterations could include more complex models, but simplicity often works well with a strong understanding of the market.

Conclusion:

This project demonstrates how quantitative strategies can exploit price distortions caused by index rebalancing events. By focusing on **One-off Events** and using technical indicators, I developed a robust short-selling strategy with integrated risk management. While this approach was backward-looking and only applied to historical data, future improvements could involve using predictive models, regularized regression, or machine learning techniques to forecast events. Simplicity in this strategy yielded effective results, but further complexity can be explored if necessary. Check out my **Quantitative Research Project** on my GitHub for more complex statistical analysis.

Code for Data Extraction, Exploration and Cleaning

In [ ]:
```
import pandas as pd
import yfinance as yf
import matplotlib.pyplot as plt
import seaborn as sns
from fredapi import Fred
import numpy as np

# Step 1: Load the Data
na_values = ["Null", "NA", "Nan", "NaN", "-", "", "N/A"]
file_path = "/Users/aidanashrafi/Downloads/Index_Add_Event_Data.xlsx"
dfs = pd.read_excel(file_path, sheet_name=None, na_values=na_values)

# Access the main data sheet
data_df = dfs.get("Data")

# Step 2: Drop rows with any missing values and clean the ticker column
def clean_ticker_column(df):
    df['Ticker'] = df['Ticker'].str.replace('*', '', regex=False)
    df['Ticker'] = df['Ticker'].str.replace('-W', '', regex=False)
    df['Ticker'] = df['Ticker'].str.replace('-V', '', regex=False)
    df['Ticker'] = df['Ticker'].str.replace('-', '', regex=False)
    df['Ticker'] = df['Ticker'].apply(lambda x: x[:-2] if x.endswith('US') else
    df['Ticker'] = df['Ticker'].str.strip().str[:4]  # Strip spaces and limit
    return df

data_df_cleaned = clean_ticker_column(data_df.dropna().copy())

# Step 3: Validate tickers by checking if they have valid data in YFinance
def check_valid_ticker(ticker):
    stock = yf.Ticker(ticker)
    hist = stock.history(period='1mo')
    return not hist.empty

# Filter for valid tickers
data_df_cleaned['Valid_Ticker'] = data_df_cleaned['Ticker'].apply(check_valid_
data_df_cleaned = data_df_cleaned[data_df_cleaned['Valid_Ticker']].drop(columns

# Step 4: Remove scheduled events (March, June, September, December)
scheduled_months = [3, 6, 9, 12]
data_df_cleaned['Month'] = pd.to_datetime(data_df_cleaned['Announced']).dt.mon
one_off_filtered_df = data_df_cleaned[~data_df_cleaned['Month'].isin(scheduled_
scheduled_df = data_df_cleaned[data_df_cleaned['Month'].isin(scheduled_months)]

# Step 5: Calculate 20-day average volume for one-off and scheduled events
```

```
def get_20_day_avg_volume(ticker, announcement_date):
    stock = yf.Ticker(ticker)
    hist = stock.history(start=announcement_date - pd.Timedelta(days=40), end=a
    return hist['Volume'].rolling(window=20).mean().iloc[-1] if not hist.empty

one_off_filtered_df['20_Day_Avg_Volume'] = one_off_filtered_df.apply(
    lambda row: get_20_day_avg_volume(row['Ticker'], row['Announced']), axis=1
one_off_filtered_df['Liquidity_Constrained_Position_Size'] = one_off_filtered_d
    lambda avg_volume: 0.01 * avg_volume if avg_volume else 0)

scheduled_df['20_Day_Avg_Volume'] = scheduled_df.apply(
    lambda row: get_20_day_avg_volume(row['Ticker'], row['Announced']), axis=1
scheduled_df['Liquidity_Constrained_Position_Size'] = scheduled_df['20_Day_Avg_
    lambda avg_volume: 0.01 * avg_volume if avg_volume else 0)

# Step 6: Filter out illiquid stocks for both one-off and scheduled events
position_threshold = 50000

# For one-off events
one_off_liquid_stocks_df = one_off_filtered_df[one_off_filtered_df['Liquidity_(

# For scheduled events
scheduled_liquid_stocks_df = scheduled_df[scheduled_df['Liquidity_Constrained_P

# Combine liquid stocks from both one-off and scheduled events
liquid_stocks_df = pd.concat([one_off_liquid_stocks_df, scheduled_liquid_stocks
print("\n Number of Liquid Stocks: ", len(liquid_stocks_df))
print("\n Number of Liquid Stocks with One-Off Events: ", len(one_off_liquid_st
# Step 7: Extract Open and Close Prices with column labels formatted with '_Cl
def extract_prices(tickers, start_dates, days=100):
    close_prices, open_prices, combined_prices = [], [], []
    for ticker, start_date in zip(tickers, start_dates):
        stock = yf.Ticker(ticker)
        hist = stock.history(start=start_date, end=start_date + pd.Timedelta(da
        if not hist.empty:
            close_series = hist['Close'].reset_index(drop=True).rename(f"{ticke
            open_series = hist['Open'].reset_index(drop=True).rename(f"{ticker}
            close_prices.append(close_series)
            open_prices.append(open_series)

            # Combined open and close prices
            combined_series = pd.DataFrame({
                f"{ticker}_Open": open_series,
                f"{ticker}_Close": close_series
            })
            combined_prices.append(combined_series)

    close_prices_df = pd.concat(close_prices, axis=1) if close_prices else pd.D
    open_prices_df = pd.concat(open_prices, axis=1) if open_prices else pd.Data
    combined_prices_df = pd.concat(combined_prices, axis=1) if combined_prices

    return close_prices_df, open_prices_df, combined_prices_df

# Extract prices for both one-off and scheduled events
tickers_one_off = one_off_liquid_stocks_df['Ticker'].tolist()
start_dates_one_off = one_off_liquid_stocks_df['Announced'].tolist()
close_prices_one_off, open_prices_one_off, combined_prices_one_off = extract_p

tickers_scheduled = scheduled_liquid_stocks_df['Ticker'].tolist()
```

```
start_dates_scheduled = scheduled_liquid_stocks_df['Announced'].tolist()
close_prices_scheduled, open_prices_scheduled, combined_prices_scheduled = ext
```

In [ ]:
```
# Check to make sure I have what I want
liquid_stocks_df
```

Simple Correlation Analysis

In [161…
```
# Generalized Correlation Analysis Function
def calculate_correlation(df, close_prices_df, event_type="Event"):
    # Get the tickers for the event type
    desired_tickers = df['Ticker'].tolist()

    # Append '_Close' to each ticker to match the columns in `close_prices_df`
    desired_columns = [ticker + '_Close' for ticker in desired_tickers]

    # Filter `close_prices_df` to only include these tickers
    filtered_close_prices_df = close_prices_df[desired_columns]

    # Calculate the correlation matrix
    correlation_matrix = filtered_close_prices_df.corr()

    # Visualize the correlation matrix using a heatmap
    plt.figure(figsize=(12, 10))
    sns.heatmap(correlation_matrix, annot=False, cmap="coolwarm", xticklabels=(
    plt.title(f"Correlation Matrix Heatmap for {event_type}")
    plt.xticks(rotation=90)
    plt.yticks(rotation=0)
    plt.show()

    # Print the correlation matrix
    print(f"Full Correlation Matrix for {event_type}:")
    print(correlation_matrix)

    return correlation_matrix
```

In [ ]:
```
# Call the generalized correlation function for one-off events
one_off_correlation_matrix = calculate_correlation(one_off_liquid_stocks_df, c
```

In [ ]:
```
# Call the generalized correlation function for scheduled events
scheduled_correlation_matrix = calculate_correlation(scheduled_liquid_stocks_d
```

FRED Data Extraction (Targeted for Stocks I selected)

In [ ]:
```
# FRED DATA EXTRACTION
fred = Fred(api_key='91ee23f35b975fbad796eab7bfe974a9')

# Get the first and last announcement dates from your cleaned dataset
first_announcement_date = one_off_liquid_stocks_df['Announced'].min()
last_announcement_date = one_off_liquid_stocks_df['Announced'].max()

# Define the extended end date (last announcement date + 100 days)
extended_end_date = last_announcement_date + pd.Timedelta(days=100)

# Fetch Fed Funds rate data from FRED for the entire date range
fed_funds_data = fred.get_series('FEDFUNDS', observation_start=first_announceme
```

```
# Convert the fetched data into a DataFrame for easier handling
fed_funds_df = pd.DataFrame(fed_funds_data, columns=['Fed Funds Rate'])
fed_funds_df['Date'] = fed_funds_df.index
fed_funds_df.reset_index(drop=True, inplace=True)

# Calculate long and short position costs
fed_funds_df['Long Position Cost'] = (fed_funds_df['Fed Funds Rate'] + 1.5)/365
fed_funds_df['Short Position Cost'] = (fed_funds_df['Fed Funds Rate'] + 1.0)/3(

# Display the first few rows to verify the data
print(fed_funds_df.head(n=10))
print("\n")
print(fed_funds_df.tail())  # Check last few rows to ensure data is complete f
```

Technical Analysis Indicators Construction

In [ ]:

```
# Technical Analysis Indicators Construction
import yfinance as yf
import pandas as pd
import talib  # TA-Lib for technical analysis

# Function to add customized RSI, MACD, ATR, and Bollinger Bands (BB) using TA-
def add_ta_indicators(df, is_daily=False):
    # Adjust indicator time periods based on whether it's intraday or daily da
    if is_daily:
        rsi_period = 2  # Use longer periods for daily data
        macd_fast = 2
        macd_slow = 4
        macd_signal = 2
        atr_period = 2
        bb_period = 2
    else:
        rsi_period = 6  # Use shorter periods for intraday data
        macd_fast = 6
        macd_slow = 12
        macd_signal = 6
        atr_period = 12
        bb_period = 12

    df['RSI'] = talib.RSI(df['Close'], timeperiod=rsi_period)
    df['MACD'], df['MACD_Signal'], _ = talib.MACD(df['Close'], fastperiod=macd_
    df['ATR'] = talib.ATR(df['High'], df['Low'], df['Close'], timeperiod=atr_p(
    df['BB_upper'], df['BB_middle'], df['BB_lower'] = talib.BBANDS(df['Close']

    return df

# Function to fetch intraday or daily data based on available range
def fetch_stock_data(ticker, start_date, end_date, is_daily=False):
    try:
        if not is_daily:
            # Attempt to fetch 1-hour intraday data
            stock_data = yf.download(ticker, start=start_date, end=end_date, in
            if stock_data.empty:
                raise ValueError(f"No intraday data for {ticker}, switching to
        else:
            # Fetch daily data
            stock_data = yf.download(ticker, start=start_date, end=end_date, in
```

```
            print(f"Fetched daily data for {ticker}.")
    except Exception as e:
        print(e)
        # Fallback to daily data if needed
        stock_data = yf.download(ticker, start=start_date, end=end_date, interv
        print(f"\nFetched daily data for {ticker} instead.")
    return stock_data

# Iterate over each ticker and determine if it should use intraday or daily da
indicator_dfs = {}
for index, row in one_off_liquid_stocks_df.iterrows():
    ticker = row['Ticker']
    start_date = pd.to_datetime(row['Announced'])  # Announcement date
    end_date = start_date + pd.Timedelta(days=100)  # 100-day window

    # Check if the ticker requires daily data (due to the intraday limit)
    is_daily = ticker == 'AR'

    # Fetch stock data with the appropriate interval
    stock_data = fetch_stock_data(ticker, start_date, end_date, is_daily=is_da

    # Check if data is available
    if not stock_data.empty:
        # Add indicators to stock data, adjusting for intraday/daily
        stock_data_with_indicators = add_ta_indicators(stock_data, is_daily=is_

        # Save the data with indicators
        indicator_dfs[ticker] = stock_data_with_indicators

        # Display the first few rows of indicators for each ticker
        print(f"\nTechnical indicators for {ticker}:")
        print(stock_data_with_indicators.head(n=20))
    else:
        print(f"No data found for {ticker}.")
```

SPY Data Extraction for Hedge

In [ ]:

```
# Spy data extraction for hedge
import yfinance as yf
import pandas as pd

# Extract SPY data
def get_spy_data(start_date, end_date):
    spy_data = yf.download('SPY', start=start_date, end=end_date, interval='1d
    return spy_data[['Open', 'Close']]

# Example usage based on first and last announcement dates
first_announcement_date = one_off_liquid_stocks_df['Announced'].min()
last_announcement_date = one_off_liquid_stocks_df['Announced'].max()
extended_end_date = last_announcement_date + pd.Timedelta(days=100)

# Get SPY data
spy_data = get_spy_data(first_announcement_date, extended_end_date)
print(spy_data.head())

# Allocate Portfolio
portfolio_size = 5000000
spy_allocation = portfolio_size * 0.24  # 24% allocation to SPY
```

```
aggressive_allocation = portfolio_size * 0.76  # 76% to aggressive short strat

# SPY Shares held throughout
spy_shares = spy_allocation / spy_data.iloc[0]['Open']  # Buy SPY at the first

spy_data = spy_data.copy()  # Explicitly create a copy of the DataFrame

# Track SPY value over time
spy_data.loc[:, 'SPY_Value'] = spy_shares * spy_data['Close']

# Display the first few rows to verify
print(spy_data.head())
print("\n", spy_data.tail())
```

Portfolio Construction and Returns

Trade Signal Generation, Cost Analysis, Profit/Loss Analysis

In [161…

```
# Portfolio allocation - 19% of portfolio per ticker in the aggressive short s

aggressive_allocation_per_ticker = aggressive_allocation / 4  # 19% allocation

# Transaction cost per share
transaction_cost_per_share = 0.01
total_net_pnl = 0
# Initialize variables for global totals
total_winning_trades = 0
total_losing_trades = 0
total_trades_all_tickers = 0
total_percentage_return = 0
total_average_return_per_trade = 0
win_rate = 0
# Function to calculate transaction costs
def calculate_transaction_costs(num_shares):
    return num_shares * transaction_cost_per_share

# Function to calculate overnight costs (Fed Funds Rate)
def calculate_short_overnight_costs(num_shares, days_held, fed_funds_rate):
    daily_cost_rate = (fed_funds_rate + 1.0) / 365  # Fed funds rate + 1% for s
    return num_shares * days_held * daily_cost_rate

# Function to generate entry signals based on the open price
def generate_trade_signals(df):
    df['Signal'] = 0  # Default no signal

    # Entry conditions based on the open price (only enter at the open)
    condition_1 = df['RSI'] > 70  # RSI overbought condition
    condition_2 = (df['MACD'] > df['MACD_Signal']) & (df['MACD'] > 0)  # MACD >
    condition_3 = df['Open'] > df['BB_middle']  # Price above upper Bollinger B

    # Generate short signals when all conditions are met at the open
    df.loc[condition_1 & condition_2 & condition_3, 'Signal'] = -1  # Short sig

    return df

# Function to generate exit signals (at the next open price or close)
def generate_exit_signals(df, entry_price):
```

```
    df['Exit_Signal'] = 0  # Default no exit signal

    # Profit exit condition: Open price < Bollinger Band lower or 20% return o
    profit_exit_condition = (df['Open'] < df['BB_lower']) | (df['Open'] < (ent

    # Loss exit condition
    loss_exit_condition = (df['Close'] > (entry_price * 1.10)) & (df['RSI'] > (

    # Set exit signals
    df.loc[profit_exit_condition, 'Exit_Signal'] = 1  # Exit for profit
    df.loc[loss_exit_condition, 'Exit_Signal'] = 2  # Exit for loss

    return df
```

In [161…

```
percentage_returns_all = []

spy_win = 1  # Since SPY is a winning trade
spy_trade = 1  # One trade for SPY

# Function to simulate P&L with costs (transaction and overnight costs)
def simulate_pnl_with_costs(df, fed_funds_df, ticker):
    winning_trades = 0
    losing_trades = 0
    entry_price = None
    pnl = 0
    total_costs = 0
    total_trades = 0
    percentage_returns = []
    trade_active = False
    num_shares = 0
    completed_trades = 0

    print(f"\nTicker: {ticker}")

    for i, row in df.iterrows():
        # Enter short position only at open prices
        if row['Signal'] == -1 and not trade_active:
            entry_price = row['Open']
            print(f"Entered at {entry_price} on {i}")

            num_shares = aggressive_allocation_per_ticker // entry_price
            trade_active = True
            total_trades += 1
            transaction_cost = calculate_transaction_costs(num_shares)
            total_costs += transaction_cost
            entry_date = row.name

        # Exit position based on open price for profit or loss
        if row['Exit_Signal'] in [1, 2] and trade_active:
            exit_price = row['Open']
            trade_active = False
            profit = num_shares * (entry_price - exit_price)

            pct_return = (profit / (num_shares * entry_price)) * 100
            percentage_returns.append(pct_return)
            percentage_returns_all.append(pct_return)

            if pct_return > 0:
                winning_trades += 1
```

```
            else:
                losing_trades += 1

            transaction_cost = calculate_transaction_costs(num_shares)
            total_costs += transaction_cost

            days_held = (row.name - entry_date).days
            if days_held == 0:
                days_held = 1

            trade_date = pd.to_datetime(i).floor('D').tz_localize(None)
            fed_funds_rate = fed_funds_df.loc[trade_date, 'Fed Funds Rate'] if
            overnight_cost = calculate_short_overnight_costs(num_shares, days_
            total_costs += overnight_cost
            pnl += profit

            print(f"Exited at {exit_price} on {i}, P/L: {profit}, Return: {pct_

            completed_trades += 1

    # Compute overall metrics after processing all rows
    cumulative_return = sum(percentage_returns)
    if completed_trades > 0:
        win_rate = (winning_trades / (winning_trades + losing_trades)) * 100
        avg_return_per_trade = cumulative_return / completed_trades
    else:
        win_rate = 0
        avg_return_per_trade = 0

    # Print summary of key metrics
    print(f"\n--- Summary for {ticker} ---")
    print(f"Total P/L: {pnl}")
    print(f"Total Trades: {completed_trades}")
    print(f"Winning Trades: {winning_trades}")
    print(f"Losing Trades: {losing_trades}")
    print(f"Win Rate: {win_rate:.2f}%")
    print(f"Average Return per Trade: {avg_return_per_trade:.2f}%")
    print(f"Total Costs: {total_costs}")
    print(f"Cumulative Percentage Return: {cumulative_return:.2f}%")

    return pnl, cumulative_return, completed_trades, total_costs, winning_trad
```

In [ ]:

```
# Function to calculate the last trade date from the aggressive strategy
def get_last_trade_date(indicator_dfs):
    last_trade_date = None

    # Iterate through all the tickers
    for ticker, df in indicator_dfs.items():
        # Generate signals for entry
        df_with_signals = generate_trade_signals(df)

        # Check if any entry signals exist
        if (df_with_signals['Signal'] == -1).any():
            # Take the first entry price from signals
            entry_price = df_with_signals.loc[df_with_signals['Signal'] == -1,

            # Generate exit signals based on open/close rules
            df_with_signals = generate_exit_signals(df_with_signals, entry_pri

            # Check the last date when an exit signal was generated
```

```
            last_exit_date = df_with_signals[df_with_signals['Exit_Signal'] !=

            # Ensure the last_exit_date is timezone-naive
            last_exit_date = last_exit_date.tz_localize(None)

            # If there's no existing last_trade_date, or the current ticker's e
            if last_trade_date is None or last_exit_date > last_trade_date:
                last_trade_date = last_exit_date

    return last_trade_date

# Example usage:
last_trade_date = get_last_trade_date(indicator_dfs)
print(f"Last Trade Date: {last_trade_date}")
```

In [ ]:

```
# Spy Returns
# Normalize the last_trade_date to remove any time component
last_trade_date_naive = last_trade_date.normalize()

# Ensure both SPY data and Fed Funds data have timezone-naive datetime indexes
spy_data.index = spy_data.index.tz_localize(None)

# Convert 'Date' column in fed_funds_df to datetime and set it as index
fed_funds_df['Date'] = pd.to_datetime(fed_funds_df['Date'])  # Ensure 'Date' is
fed_funds_df.set_index('Date', inplace=True)  # Set 'Date' as the index
fed_funds_df.index = fed_funds_df.index.tz_localize(None)  # Make the index tir

# Forward-fill the Fed Funds Rate data to ensure there is no missing data for a
fed_funds_df.ffill(inplace=True)

# Function to calculate SPY profit/loss based on the last trade date with trans
def calculate_spy_pnl(spy_data, last_trade_date, spy_shares, fed_funds_df):
    # Sell SPY at the close price on the last trade date
    spy_sell_price = spy_data.loc[last_trade_date, 'Close']

    # Calculate SPY profit
    spy_pnl = spy_shares * (spy_sell_price - spy_data.iloc[0]['Open'])

    # Calculate percentage return for SPY
    spy_pct_return = ((spy_sell_price - spy_data.iloc[0]['Open']) / spy_data.i

    # Transaction cost for SPY (buy + sell)
    spy_transaction_cost = spy_shares * 0.01 * 2  # Buy and sell cost

    # Accumulate overnight costs for long SPY position (Fed Funds Rate + 1.5%)
    total_overnight_cost = 0
    for date in pd.date_range(spy_data.index[0], last_trade_date):
        # Align with Fed Funds data (accessing the index, not as a column)
        if date in fed_funds_df.index:
            fed_funds_rate = fed_funds_df.loc[date, 'Fed Funds Rate']
        else:
            fed_funds_rate = fed_funds_df['Fed Funds Rate'].iloc[-1]  # Use las

        daily_cost_rate = (fed_funds_rate + 1.5) / 365  # Long position cost (l
        total_overnight_cost += spy_shares * daily_cost_rate

    # Subtract total costs (transaction + overnight) from the profit
    total_costs = spy_transaction_cost + total_overnight_cost
    net_spy_pnl = spy_pnl - total_costs
```

```
    return net_spy_pnl, spy_pct_return, spy_transaction_cost, total_overnight_

# Calculate SPY profit/loss
spy_pnl, spy_pct_return, spy_transaction_cost, total_overnight_cost = calculat

spy_net_pnl = spy_pnl - spy_transaction_cost - total_overnight_cost
# Print SPY P/L, transaction cost, overnight cost, and returns
print(f"SPY Profit/Loss: {spy_pnl:.2f}")
print(f"SPY Percentage Return: {spy_pct_return:.2f}%")
print(f"SPY Transaction Cost: {spy_transaction_cost:.2f}")
print(f"SPY Total Overnight Cost: {total_overnight_cost:.2f}")
print(f"SPY Net Profit/Loss after Costs: {(spy_pnl - spy_transaction_cost - to
```

In [ ]:
```
# Adding the SPY hedge trade
def include_spy_in_totals(spy_pct_return, spy_net_pnl):
    global total_winning_trades, total_trades_all_tickers, total_percentage_re

    # Since SPY was a winning trade, we add it as 1 winning trade and 1 total
    total_winning_trades += spy_win
    total_trades_all_tickers += spy_trade
    total_percentage_return += spy_pct_return
    total_net_pnl += spy_net_pnl

# Loop over each ticker in your data
for ticker, df in indicator_dfs.items():
    df_with_signals = generate_trade_signals(df)

    if (df_with_signals['Signal'] == -1).any():
        entry_price = df_with_signals.loc[df_with_signals['Signal'] == -1, 'Op
        df_with_signals = generate_exit_signals(df_with_signals, entry_price)
        total_pnl, cumulative_return, total_trades, total_costs, wins, losses,
        total_net_pnl += (total_pnl - total_costs)
        total_winning_trades += wins
        total_losing_trades += losses
        total_trades_all_tickers += total_trades
        total_percentage_return += cumulative_return

# Now include the SPY hedge returns
include_spy_in_totals(spy_pct_return, spy_net_pnl)

# Final calculations with SPY hedge
overall_win_rate = (total_winning_trades / total_trades_all_tickers) * 100 if
overall_avg_return_per_trade = total_percentage_return / total_trades_all_tick
```

Summary of Returns

In [ ]:
```
print(f"Hedge: SPY Net P/L after Costs: {spy_net_pnl:.2f}")
print(f"\nTotal Net P/L after Costs for Aggressive Bearish Strategy: {total_ne
print(f"\nTotal Net P/L after Costs: {total_net_pnl:.2f}")
print(f"\nTotal Net Percentage Return without compounding II: {100*(total_net_
# Print the overall metrics
print(f"\nTotal Winning Trades: {total_winning_trades}")
print(f"\nTotal Losing Trades: {total_losing_trades}")
print(f"\nOverall Win Rate: {overall_win_rate:.2f}%")
print(f"\nOverall Average Return per Trade: {overall_avg_return_per_trade:.2f}

import numpy as np
# Convert percentage_returns_all to a numpy array
```

```
returns_array = np.array(percentage_returns_all)

print(f"\nWorst Trade: {returns_array.min():.2f}%")
print(f"\nBest Trade: {returns_array.max():.2f}%")

# Calculate the standard deviation of returns
std_dev_returns = np.std(returns_array)
risk_free_rate = 0.0  # Assumed zero for simplicity
sharpe_ratio = (overall_avg_return_per_trade - risk_free_rate) / std_dev_returı
print(f"\nSharpe Ratio: {sharpe_ratio:.2f}")
```

Results Summary and Performance Evaluation

Key Results:

- **Total Net Profit Across All Trades**: $1,294,957.06
- **Number of Trades**: 23
- **Cumulative Return**: 25.90% cumulative return on the $5,000,000 portfolio.
- **Average Return per Trade**: On average, each trade generated a 5.84% return.
- **Max Drawdown**: The largest observed drawdown was -13.59%, representing the largest portfolio drop during testing.
- **Sharpe Ratio**: The strategy achieved a Sharpe ratio of 0.63, which is decent, but still below the ideal level of 1.0 for a professional quantitative strategy.

SPY Hedge Performance:

- **Hedge Allocation**: 24% of the portfolio was allocated to SPY as a hedge.
- **Hedge Contribution**: The SPY hedge helped to offset losses during broad market uptrends, contributing $631,976.74 to the total portfolio return.
- **Net P&L from Hedge**: The long SPY position contributed 48.82% of the total portfolio return.

Trade Metrics:

- **Win Rate**: 56.52% of trades were profitable, highlighting a balanced strategy with a slight edge in favor of winning trades.
- **Best Trade**: The most profitable trade yielded a 35.53% return.
- **Worst Trade**: The largest loss incurred was -13.59%, signaling the need for better stop-loss measures to mitigate large drawdowns.

Risk and Volatility Management:

- **SPY Hedge Efficiency**: The 24% allocation to SPY as a hedge reduced portfolio volatility and provided consistent protection during market uptrends. It played a significant role in cushioning losses when the market moved against short positions.
- **Risk-Adjusted Returns**: While the Sharpe ratio of 0.63 demonstrates that the strategy provides positive risk-adjusted returns, refining risk management measures such as

stop-losses and dynamic hedging could improve this significantly.

Strategy Refinement Considerations:

- **Stop-Loss Improvements**: The -13.59% loss highlights the need for improved stop-loss rules. Incorporating a **dynamic ATR-based stop-loss** would minimize the impact of such large losses, however for such a simplistic strategy it could also deteriorate the effectiveness given the purposeful risk tolerance.
- **Future Enhancements**: Adding more complex strategies such as **cointegration** or **pairs trading**, as well as **predictive machine learning models**, could improve signal generation and the robustness of the strategy. These enhancements could improve the Sharpe ratio and reduce drawdowns in future iterations.

Final Conclusion:

The momentum-based short strategy demonstrated a solid performance, with a **25.90% cumulative return** and an effective **SPY hedge** that reduced overall portfolio risk. The Sharpe ratio of 0.63 suggests a reasonable risk-adjusted return, though there is room for improvement.

To make this strategy more competitive in professional quantitative settings, enhancing **stop-loss mechanisms** and introducing more sophisticated statistical techniques such as **cointegration** or **machine learning models** for predictive analysis would further optimize results. This could potentially push the Sharpe ratio above 1.0, positioning the strategy for use in more demanding environments.

Further out-of-sample testing and backtesting on different asset classes would also be important steps toward verifying the robustness of the model and ensuring its viability in live market conditions.

For more advanced modeling techniques and quantitative research, feel free to explore my **Quantitative Research Project** on GitHub, where I delve deeper into **predictive modeling** and **machine learning algorithms**.