Importing required libraries

```
import sys
import os
import math
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import yfinance as yf
from zipline import run_algorithm
from zipline.finance.commission import PerTrade
from zipline.api import order_target, record, symbol, set_commission, set_slippage, set_
from zipline.finance import commission
import quantstats as qs
import warnings
warnings.filterwarnings('ignore')
warnings.filterwarnings('always')
root_dir = os.getcwd().replace('/Notebooks','')
sys.path.insert(1, root_dir)
from zipline.utils.run_algo import load_extensions
from logbook import (NestedSetup, NullHandler, Logger, StreamHandler, StderrHandler, INI
import pytz
load_extensions(
    default=True,
    extensions=[],
    strict=True,
    environ=os.environ,
)
%matplotlib inline
```

Loading trading calendar

All of the data used in this analysis follow the calender of New York Stock Exchanges.

```
from zipline.utils.calendar_utils import get_calendar
calendar_name = 'NYSE'
calendar = get_calendar(calendar_name)
print(calendar)
```

```
<exchange_calendars.exchange_calendar_xnys.XNYSExchangeCalendar object at 0x7
```

Data Collection and Preprocessing

In this task, we used multi-asset to analyze market conditions. We took stock data of Apple Inc, Microsoft, Tesla, Nvidia, and Nike from 2015 to 2022.

```
start = "2015-01-01"  #Starting day of trading
end = "2022-01-01"    #Ending day of trading

'''The function dataProcessing() is used to download data from yfinance. After download

def dataProcessing(asset, start_date = start , end_date = end):
```

```
    data = yf.download(asset, start=start_date , end=end_date)  #Downloading data from y
    data = data.drop('Adj Close', axis = 1)
    upDatedColumns = {'Open': 'open', 'High': 'high', 'Low': 'low', 'Close': 'close', 'V
    data.rename(columns = upDatedColumns, inplace = True)
    data['volume'] = data['volume'].astype(float)       #Converting datatype into float
    data['timestamp'] = pd.to_datetime(data.index)
    data = data.reindex(columns =['timestamp', 'open','high', 'low', 'close', 'volume']
    data['timestamp'] = data['timestamp'].dt.strftime('%Y-%m-%d %H:%M:%S.%f')
    data = data.sort_values(by='timestamp' , ascending = True)
    data = data.reset_index(drop = True)
    data = data.groupby('timestamp').last().reset_index()
    data.set_index('timestamp',inplace = True)
    data.to_csv(f'/home/shbmsk/Desktop/AnchorBlock Technology/Different Strategy/Single


'''Passing the asset ticker symbol through the function to download and preprocess the d

dataProcessing("AAPL")
dataProcessing("MSFT")
dataProcessing("TSLA")
dataProcessing("NVDA")
dataProcessing("NKE")
```

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

Ingesting the bundles

```
!zipline ingest -b spStocks
!zipline bundles
```

Show hidden output

Checking Stationarity of Data Series

Adfuller

Null Hypothesis (H0): Series is non-stationary, or series has a unit root.
Alternate Hypothesis(HA): Series is stationary, or series has no unit root.

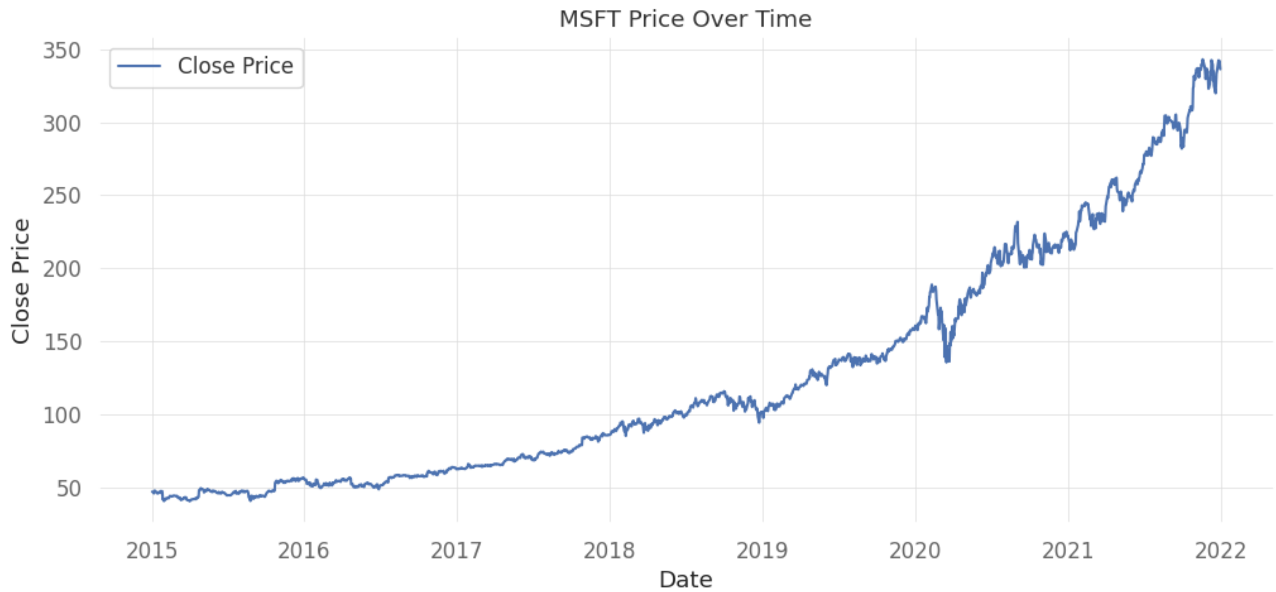
If ADF statistic < Critical Value and p-value < 0.05 – Reject Null Hypothesis(HO), i.e., time series does not have a unit root, meaning it is stationary.

```
from statsmodels.tsa.stattools import adfuller
def stationarityCheck(asset, start_date=start, end_date=end):
    for asset_name in asset:
        asset_data = yf.download(asset_name, start=start_date, end=end_date)['Close']
        adf_result = adfuller(asset_data)
        # Condition for ADF
        if adf_result[1] < 0.05 and (adf_result[0] > adf_result[4]['1%'] or adf_result[0
            print(f"{asset_name} - Data series is stationary and mean reverting.")
```

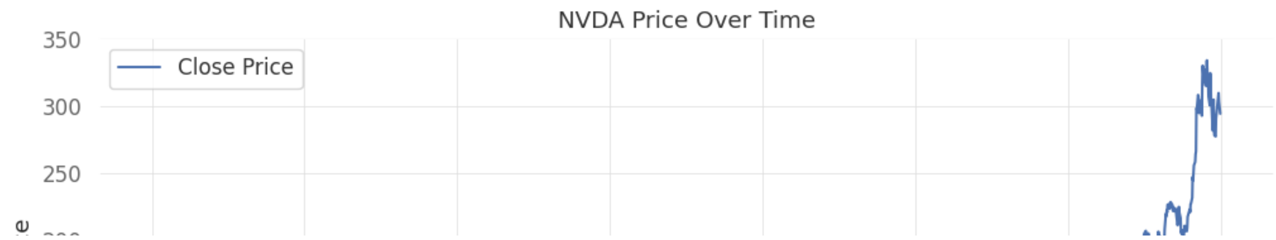
```
        else:
            print(f"{asset_name} - Data series is non-stationary and trend following.")

        # Results for ADF
        print(f"Results for {asset_name} (ADF Test):")
        print(f'ADF Statistic: {adf_result[0]}')
        print(f'p-value: {adf_result[1]}')
        print(f'Critical Values: {adf_result[4]}')

        # Plotting the graph
        plt.figure(figsize=(12, 5))
        plt.plot(asset_data, label='Close Price')
        plt.title(f'{asset_name} Price Over Time')
        plt.xlabel('Date')
        plt.ylabel('Close Price')
        plt.legend()
        plt.show()

assets = ["AAPL", "MSFT", "NVDA", "TSLA", "NKE"]
stationarityCheck(assets)
```

```
[*********************100%%**********************]  1 of 1 completed
AAPL - Data series is non-stationary and trend following.
Results for AAPL (ADF Test):
ADF Statistic: 1.8944685102826717
p-value: 0.9985190674805271
Critical Values: {'1%': -3.4341094501874854, '5%': -2.8632005876775297, '10%'
```


```
[*********************100%%**********************]  1 of 1 completed
MSFT - Data series is non-stationary and trend following.
Results for MSFT (ADF Test):
ADF Statistic: 3.618227786624649
p-value: 1.0
Critical Values: {'1%': -3.434120287918905, '5%': -2.8632053717943005, '10%':
```




```
[*********************100%%**********************]  1 of 1 completed
NVDA - Data series is non-stationary and trend following.
Results for NVDA (ADF Test):
ADF Statistic: 3.0037105665889787
p-value: 1.0
Critical Values: {'1%': -3.434120287918905, '5%': -2.8632053717943005, '10%':
```



It is clear from the test :

1. Apple Asset: Trend following
2. Microsoft Asset: Trend following
3. Nvidia Asset: Trend following
4. Tesla Asset: Trend following
5. Nike Asset: Trend following

Strategy : Bollinger Bands with Volume Filtering

Bollinger Bands

Bollinger Bands is a technical analysis tool to generate oversold or overbought signals and was developed by John Bollinger. It consist of three lines:

1. Upper Band: Calculates the moving average (typically a simple moving average) of the price data plus a certain number of standard deviations (commonly 2).
2. Middle Band: Represents the moving average itself.
3. Lower Band: Calculates the moving average minus the same number of standard deviations as the upper band.

Mathematics form of Bollinger Bands

upperBollingerBand = SMA + (SD * numSTD)
lowerBollingerBand = SMA - (SD * numSTD)

where :

- upperBollingerBand is upper Bollinger band
- lowerBollingerBand is lower Bollinger band
- SMA is a simple moving average
- SD is the standard deviation
- numSTD is the number of standard deviation

Volume filtering

I have applied volume filtering in BB. This filter acts as a simple volume filter to identify periods with relatively high trading activity. A certain condition and threshold were set.

Strategy Implementation

```
bb_window_size = 25
std_size = 2

'''
    The function bollinger_bands() takes three arguments: prices, window_size, and std_s
    The function returns the upper and lower bands.
'''

def bollinger_bands(prices, window_size=bb_window_size, std_size= std_size):
    sma = prices.rolling(window=window_size).mean()     #Calculating Simple Moving Avera
    std = prices.rolling(window=window_size).std()      #Calculating Standard Deviation
    upper_band = sma + (std * std_size)                 #Calculating upper band
    lower_band = sma - (std * std_size)                 #Calculating lower band

    return upper_band, lower_band


''' The function volumeFilter() takes two parameters data and a threshold value. The thr
    The most recent volume value (data.volume[-1]) is compared to the specified thresho
    Return:  True if the volume is greater than the threshold, denoting sufficient tradi
    Returns: False if the volume is below the threshold, denoting lower trading activity
'''

def volumeFilter(data, threshold=50000):
    return data.volume[-1] > threshold


def initialize(context):
    context.assets = [symbol("AAPL"), symbol("NVDA"), symbol("NKE"), symbol("MSFT"), sym
    context.set_commission(commission.PerShare(cost=0.0075, min_trade_cost=1.0))


def handle_data(context, data):
    for asset in context.assets:
        #Get historical prices for the asset
        prices = data.history(asset, ['price', 'volume'], bar_count=50, frequency='1d')

        # Calculate Bollinger Bands
        upper_band, lower_band = bollinger_bands(prices['price'])

        #Checking filters
        volume_condition = volumeFilter(prices)
        current_price = data.current(asset, 'price')
        if current_price > upper_band[-1] and not volume_condition:
        # Price is above the upper Bollinger Band, Volume is favorable suggesting a sell
            order(asset, -1)  # Sell signal
        elif current_price < lower_band[-1] and volume_condition:
        # Price is below the lower Bollinger Band with favorable volume suggesting a po
            order(asset, 1)   # Buy signal

        record(
            upper_band=upper_band[-1],
            lower_band=lower_band[-1],
            current_price=current_price,
            volume_condition=1 if volume_condition else 0
        )


start_date = pd.to_datetime('2018-01-01 00:00:00.0000', format='%Y-%m-%d %H:%M:%S.%f')
end_date = pd.to_datetime('2022-01-01 00:00:00.0000', format='%Y-%m-%d %H:%M:%S.%f')

'''
    Backtesting the strategy using zipline and data bundles.
'''
```

```
results = run_algorithm(
    start=start_date,
    end=end_date,
    initialize=initialize,
    handle_data=handle_data,
    capital_base=10000,
    trading_calendar=calendar,
    bundle='spStocks',
    data_frequency='daily',
)
```

Show hidden output

```
start = results.index[0]
end = results.index[-1]
benchmark = yf.download("^OEX", start = start , end =  end )["Adj Close"].pct_change()
results.index = pd.to_datetime(results.index).tz_localize(None)
results.index = benchmark.index
qs.reports.full(
    results["returns"],
    benchmark = benchmark,
    match_dates = True,
    figsize =(8,4),
    df = results,
)
```

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

Performance Metrics

```
/home/shbmsk/anaconda3/lib/python3.10/site-packages/quantstats/reports.py:133
  _stats.expected_return(
/home/shbmsk/.local/lib/python3.10/site-packages/numpy/core/fromnumeric.py:86
  return reduction(axis=axis, out=out, **passkwargs)
/home/shbmsk/anaconda3/lib/python3.10/site-packages/quantstats/reports.py:133
  _stats.expected_return(
/home/shbmsk/.local/lib/python3.10/site-packages/numpy/core/fromnumeric.py:86
  return reduction(axis=axis, out=out, **passkwargs)
/home/shbmsk/anaconda3/lib/python3.10/site-packages/quantstats/reports.py:134
  _stats.expected_return(
/home/shbmsk/.local/lib/python3.10/site-packages/numpy/core/fromnumeric.py:86
  return reduction(axis=axis, out=out, **passkwargs)
                           Benchmark    Strategy
-------------------------  -----------  ----------
Start Period               2018-01-31   2018-01-31
End Period                 2021-12-31   2021-12-31
Risk-Free Rate             0.0%         0.0%
Time in Market             100.0%       100.0%

Cumulative Return          75.3%        309.27%
Mean Active Return         0.0%         0.09%
CAGR%                     10.4%        28.19%

Sharpe                     0.77         1.57
Prob. Sharpe Ratio         93.42%       99.88%
Smart Sharpe               0.69         1.39
Sortino                    1.07         2.32
Smart Sortino              0.95         2.06
Sortino/√2                 0.76         1.64
Smart Sortino/√2           0.67         1.46
Omega                      1.38         1.38

Max Drawdown               -31.53%      -27.1%
Longest DD Days            204          197
Volatility (ann.)          21.47%       24.97%
R^2                        0.55         0.55
Calmar                     0.33         1.04
Skew                       -0.53        -0.26
Kurtosis                   15.21        8.0

Expected Daily %           0.06%        0.14%
Expected Monthly %         1.18%        2.98%
Expected Yearly %          15.06%       42.23%
Kelly Criterion            5.15%        11.25%
Risk of Ruin               0.0%         0.0%
Daily Value-at-Risk        -2.16%       -2.43%
Expected Shortfall (cVaR)  -2.16%       -2.43%

Max Consecutive Wins       9            13
Max Consecutive Losses     7            7
Gain/Pain Ratio            0.17         0.38
Gain/Pain (1M)             1.07         3.62

Payoff Ratio               0.84         1.0
Profit Factor              1.17         1.38
Common Sense Ratio         0.94         1.73
CPC Index                  0.56         0.77
```