```
pip install yfinance
```

```
Requirement already satisfied: yfinance in /usr/local/lib/python3.10/dist-packages (0.2.50)
Requirement already satisfied: pandas>=1.3.0 in /usr/local/lib/python3.10/dist-packages (from yfinance) (2.2.2)
Requirement already satisfied: numpy>=1.16.5 in /usr/local/lib/python3.10/dist-packages (from yfinance) (1.26.4)
Requirement already satisfied: requests>=2.31 in /usr/local/lib/python3.10/dist-packages (from yfinance) (2.32.3)
Requirement already satisfied: multitasking>=0.0.7 in /usr/local/lib/python3.10/dist-packages (from yfinance) (0.0.11)
Requirement already satisfied: lxml>=4.9.1 in /usr/local/lib/python3.10/dist-packages (from yfinance) (5.3.0)
Requirement already satisfied: platformdirs>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from yfinance) (4.3.6)
Requirement already satisfied: pytz>=2022.5 in /usr/local/lib/python3.10/dist-packages (from yfinance) (2024.2)
Requirement already satisfied: frozendict>=2.3.4 in /usr/local/lib/python3.10/dist-packages (from yfinance) (2.4.6)
Requirement already satisfied: peewee>=3.16.2 in /usr/local/lib/python3.10/dist-packages (from yfinance) (3.17.8)
Requirement already satisfied: beautifulsoup4>=4.11.1 in /usr/local/lib/python3.10/dist-packages (from yfinance) (4.12.3)
Requirement already satisfied: html5lib>=1.1 in /usr/local/lib/python3.10/dist-packages (from yfinance) (1.1)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-packages (from beautifulsoup4>=4.11.1->yfinance) (2.
Requirement already satisfied: six>=1.9 in /usr/local/lib/python3.10/dist-packages (from html5lib>=1.1->yfinance) (1.17.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-packages (from html5lib>=1.1->yfinance) (0.5.1)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.3.0->yfinance) (2.
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.3.0->yfinance) (2024.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests>=2.31->yfinance)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests>=2.31->yfinance) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests>=2.31->yfinance) (2.2.3
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests>=2.31->yfinance) (2024.
```

```
pip show yfinance
```

```
Name: yfinance
Version: 0.2.50
Summary: Download market data from Yahoo! Finance API
Home-page: https://github.com/ranaroussi/yfinance
Author: Ran Aroussi
Author-email: ran@aroussi.com
License: Apache
Location: /usr/local/lib/python3.10/dist-packages
Requires: beautifulsoup4, frozendict, html5lib, lxml, multitasking, numpy, pandas, peewee, platformdirs, pytz, requests
Required-by:
```

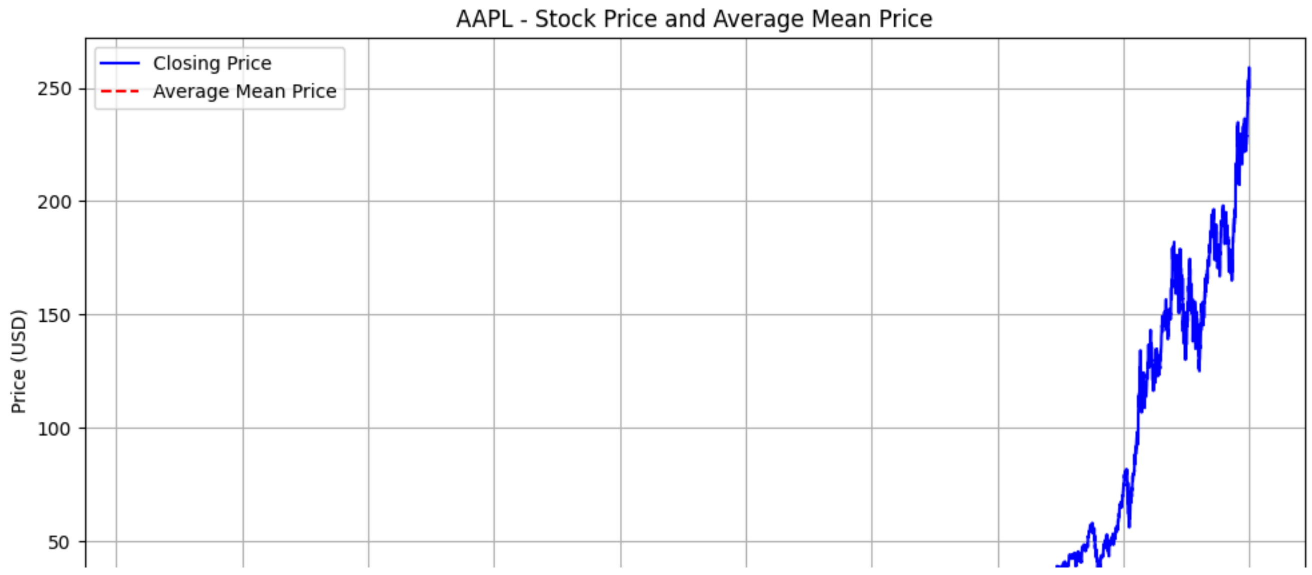
```python
import yfinance as yf
import pandas as pd
from datetime import datetime

# Function to fetch and save daily stock data
def fetch_and_save_daily_data(tickers, start_date, file_name):
    """
    Fetch daily stock data and save it to a CSV file.
    """
    try:
        # Get today's date for the end_date
        end_date = datetime.now().strftime("%Y-%m-%d")
        print(f"Fetching data for: {', '.join(tickers)} from {start_date} to {end_date}")

        # Fetch data using the 1-day interval
        data = yf.download(tickers, start=start_date, end=end_date, interval="1d")

        # Save the data to a CSV file
        data.to_csv(file_name)
        print(f"Data updated and saved to {file_name}")
    except Exception as e:
        print(f"An error occurred: {e}")

# Parameters
tickers = ["AAPL"]
start_date = "1980-01-01"
file_name = "daily_stock_data.csv"

# Fetch and save the data
fetch_and_save_daily_data(tickers, start_date, file_name)
```

```
Fetching data for: AAPL from 1980-01-01 to 2025-01-02
[*********************100%***********************]  1 of 1 completed
Data updated and saved to daily_stock_data.csv
```

```python
from google.colab import files
files.download("daily_stock_data.csv")
```

```
import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt
from datetime import datetime

# Function to fetch stock data and calculate average mean price
def fetch_and_plot_average_price(tickers, start_date, file_name):
    """
    Fetch daily stock data, calculate the average mean price, and plot the results.
    """
    try:
        # Get today's date for the end_date
        end_date = datetime.now().strftime("%Y-%m-%d")
        print(f"Fetching data for: {', '.join(tickers)} from {start_date} to {end_date}")

        # Fetch data using the 1-day interval
        data = yf.download(tickers, start=start_date, end=end_date, interval="1d")

        # Calculate the average mean price (mean of the closing prices)
        data['Average_Mean_Price'] = data['Close'].mean()

        # Save the data to a CSV file
        data.to_csv(file_name)
        print(f"Data updated and saved to {file_name}")

        # Plotting the stock's closing price and average mean price
        plt.figure(figsize=(10,6))
        plt.plot(data.index, data['Close'], label='Closing Price', color='blue')
        plt.axhline(data['Average_Mean_Price'][0], color='red', linestyle='--', label='Average Mean Price')
        plt.title(f'{", ".join(tickers)} - Stock Price and Average Mean Price')
        plt.xlabel('Date')
        plt.ylabel('Price (USD)')
        plt.legend()
        plt.xticks(rotation=45)
        plt.grid(True)
        plt.tight_layout()
        plt.show()

    except Exception as e:
        print(f"An error occurred: {e}")

# Parameters
tickers = ["AAPL"]  # List of stock ticker symbols
start_date = "1980-01-01"  # Start date
file_name = "daily_stock_data_with_average.csv"  # Output CSV file name

# Fetch data, calculate the average mean price, and plot
fetch_and_plot_average_price(tickers, start_date, file_name)
```

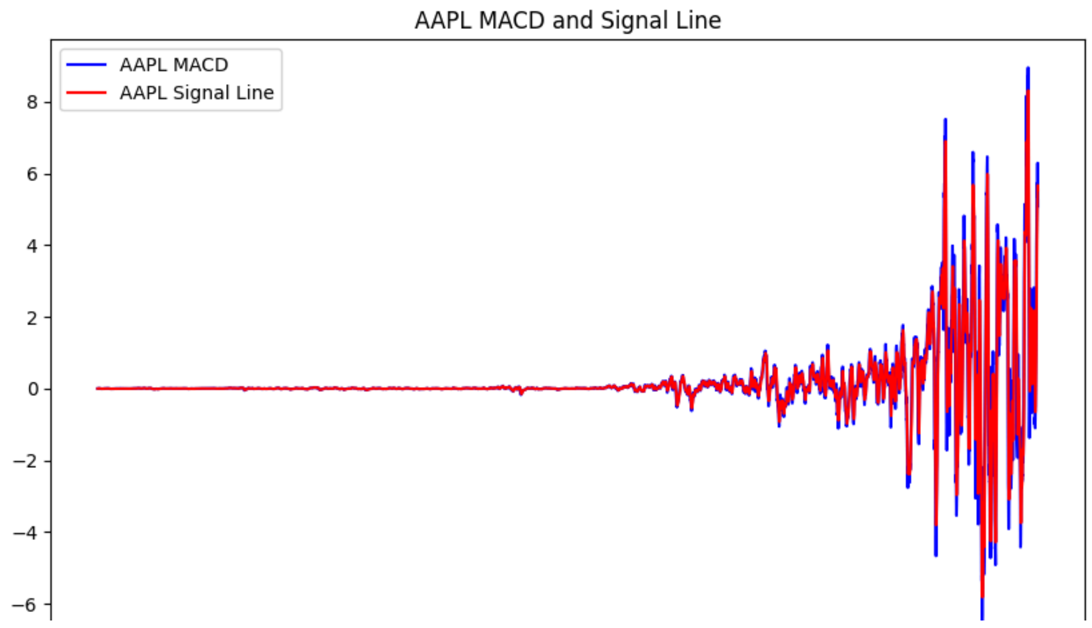
```
[*********************100%***********************]  1 of 1 completedFetching data for: AAPL from 1980-01-01 to 2025-01-02
<ipython-input-5-5e002da4ac27>:29: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version,
  plt.axhline(data['Average_Mean_Price'][0], color='red', linestyle='--', label='Average Mean Price')
Data updated and saved to daily_stock_data_with_average.csv
```



```
import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt

# Function to fetch stock data and calculate MACD
```

```
def fetch_and_add_macd(tickers, start_date):
    """
    Fetch stock data and calculate MACD for each ticker.
    """
    # Download stock data from Yahoo Finance
    data = yf.download(tickers, start=start_date)

    # Calculate MACD and Signal line for each ticker
    for ticker in tickers:
        # Extract the stock's close price
        df = data['Close'][ticker]

        # Calculate the MACD
        short_ema = df.ewm(span=12, adjust=False).mean()  # 12-period EMA
        long_ema = df.ewm(span=26, adjust=False).mean()  # 26-period EMA
        macd = short_ema - long_ema  # MACD line

        # Calculate the Signal line (9-period EMA of the MACD)
        signal_line = macd.ewm(span=9, adjust=False).mean()

        # Add the MACD and Signal line to the DataFrame
        data[ticker, 'MACD'] = macd
        data[ticker, 'Signal Line'] = signal_line

        # Plot the MACD and Signal line
        plt.figure(figsize=(10, 6))
        plt.plot(data.index, macd, label=f'{ticker} MACD', color='blue')
        plt.plot(data.index, signal_line, label=f'{ticker} Signal Line', color='red')
        plt.title(f'{ticker} MACD and Signal Line')
        plt.legend(loc='best')
        plt.show()

    return data

# Parameters
tickers = ["AAPL"]
start_date = "1980-01-01"
file_name = "daily_stock_data.csv"

# Fetch stock data and add MACD
stock_data = fetch_and_add_macd(tickers, start_date)

# Save the stock data with MACD and Signal line to CSV
stock_data.to_csv('stock_data_with_macd.csv')
print("Data with MACD and Signal line saved to stock_data_with_macd.csv")
```

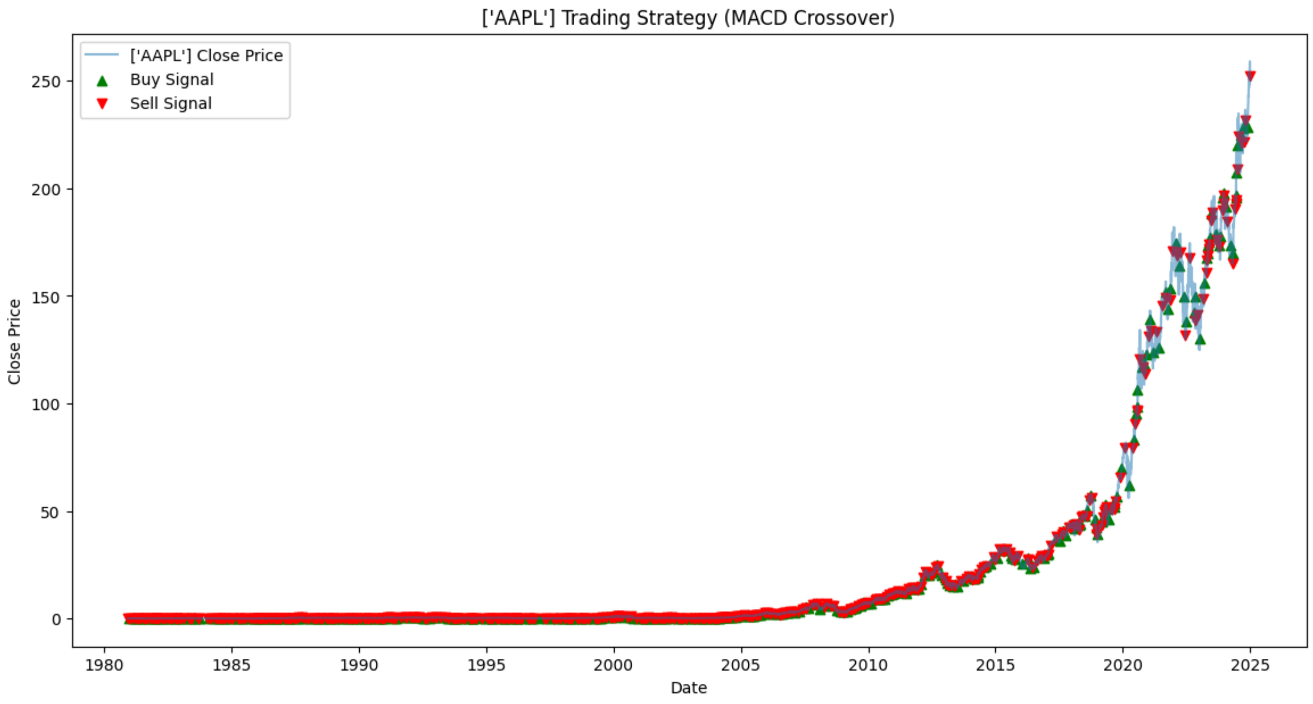
```
[*********************100%***********************]  1 of 1 completed
```

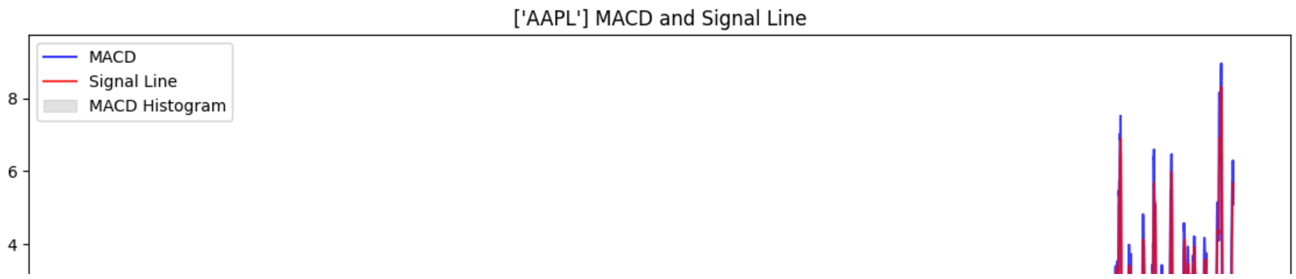


```
import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt

# Function to fetch stock data and calculate MACD
def fetch_stock_data_and_calculate_macd(tickers, start_date):
    """
    Fetch stock data, calculate MACD and Signal Line.
    """
```

```
        # Fetch stock data
        data = yf.download(tickers, start=start_date)

        # Calculate MACD and Signal Line
        short_ema = data['Close'].ewm(span=12, adjust=False).mean()  # 12-period EMA
        long_ema = data['Close'].ewm(span=26, adjust=False).mean()  # 26-period EMA
        data['MACD'] = short_ema - long_ema  # MACD line
        data['Signal Line'] = data['MACD'].ewm(span=9, adjust=False).mean()  # Signal line

        return data

    # Function to generate Buy and Sell signals based on MACD crossover
    def generate_trading_signals(data):
        """
        Generate Buy/Sell signals based on MACD crossover strategy.
        """
        data['Buy Signal'] = (data['MACD'] > data['Signal Line']) & (data['MACD'].shift(1) <= data['Signal Line'].shift(1))
        data['Sell Signal'] = (data['MACD'] < data['Signal Line']) & (data['MACD'].shift(1) >= data['Signal Line'].shift(1))

        return data

    # Function to visualize the strategy
    def plot_trading_strategy(data, ticker):
        """
        Plot Buy/Sell signals on the stock price chart along with MACD.
        """
        # Plot stock price with Buy/Sell signals
        plt.figure(figsize=(14, 7))
        plt.plot(data['Close'], label=f"{ticker} Close Price", alpha=0.5)
        plt.scatter(data[data['Buy Signal']].index, data[data['Buy Signal']]['Close'], label='Buy Signal', marker='^', color='green', alpha=
        plt.scatter(data[data['Sell Signal']].index, data[data['Sell Signal']]['Close'], label='Sell Signal', marker='v', color='red', alpha
        plt.title(f"{ticker} Trading Strategy (MACD Crossover)")
        plt.xlabel("Date")
        plt.ylabel("Close Price")
        plt.legend()
        plt.show()

        # Plot MACD and Signal Line with crossover points
        plt.figure(figsize=(14, 7))
        plt.plot(data['MACD'], label='MACD', color='blue', alpha=0.75)
        plt.plot(data['Signal Line'], label='Signal Line', color='red', alpha=0.75)
        plt.fill_between(data.index, data['MACD'] - data['Signal Line'], color='gray', alpha=0.2, label='MACD Histogram')
        plt.title(f"{ticker} MACD and Signal Line")
        plt.legend()
        plt.show()

    # Main function to execute the trading strategy
    def execute_trading_strategy(tickers, start_date):
        """
        Execute the trading strategy: Fetch data, calculate MACD, and apply strategy.
        """
        # Fetch stock data and calculate MACD
        data = fetch_stock_data_and_calculate_macd(tickers, start_date)

        # Generate trading signals
        data = generate_trading_signals(data)

        # Save the data with Buy/Sell signals to a CSV file
        file_name = f"{tickers}_trading_strategy_phase3.csv"
        data.to_csv(file_name)
        print(f"Data with trading signals saved to {file_name}")

        # Visualize the strategy
        plot_trading_strategy(data, tickers)

    # Parameters
    tickers = ["AAPL"]
    start_date = "1980-01-01"
    file_name = "daily_stock_data.csv"

    # Execute the trading strategy
    execute_trading_strategy(tickers, start_date)
```

```
[*********************100%***********************]  1 of 1 completed
Data with trading signals saved to ['AAPL']_trading_strategy_phase3.csv
```





```
import yfinance as yf
import pandas as pd
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
from datetime import datetime

# Function to fetch stock data and calculate MACD
def fetch_stock_data_and_calculate_macd(tickers, start_date):
    """
    Fetch stock data and calculate MACD and Signal Line.
    """
    data = yf.download(tickers, start=start_date)
    short_ema = data['Close'].ewm(span=12, adjust=False).mean()  # 12-period EMA
    long_ema = data['Close'].ewm(span=26, adjust=False).mean()  # 26-period EMA
    data['MACD'] = short_ema - long_ema
    data['Signal Line'] = data['MACD'].ewm(span=9, adjust=False).mean()
    return data

# Function to filter stocks based on MACD crossover
def filter_stocks(tickers, start_date):
    """
    Filter stocks where MACD has crossed above the Signal Line.
    """
    selected_stocks = []

    for ticker in tickers:
        data = fetch_stock_data_and_calculate_macd(tickers, start_date)

        # Check if the latest MACD crosses above the Signal Line
        if data['MACD'].iloc[-1] > data['Signal Line'].iloc[-1] and data['MACD'].iloc[-2] <= data['Signal Line'].iloc[-2]:
            selected_stocks.append(ticker)

    return selected_stocks

# Function to send email alerts
def send_email_alert(selected_stocks):
    """
    Send email alerts for selected stocks.
    """
    if not selected_stocks:
        print("No stocks meet the criteria. No email will be sent.")
        return

    # Email configuration
    sender_email = "shagunjain410@.com"
    sender_password = "9166070972"
    recipient_email = "sumanjain6434@gmail.com.com"  # Replace with recipient email

    # Create email content
    subject = "Stock Alert: MACD Crossover Strategy"
    body = f"The following stocks meet the criteria for MACD crossover:\n\n{', '.join(selected_stocks)}\n\nGenerated on: {datetime.now(

    # Create email message
    msg = MIMEMultipart()
    msg['From'] = sender_email
    msg['To'] = recipient_email
    msg['Subject'] = subject
    msg.attach(MIMEText(body, 'plain'))

    try:
        # Connect to the SMTP server and send the email
        with smtplib.SMTP('smtp.gmail.com', 587) as server:
            server.starttls()
            server.login(sender_email, sender_password)
            server.send_message(msg)

        print("Email alert sent successfully!")
    except Exception as e:
        print(f"An error occurred while sending email: {e}")

# Main function
def execute_stock_filtering_and_alerts(tickers, start_date):
    """
    Execute the stock filtering and send alerts for selected stocks.
    """
    # Filter stocks based on the strategy
    selected_stocks = filter_stocks(tickers, start_date)

    # Send email alerts for the selected stocks
    send_email_alert(selected_stocks)

# Parameters
tickers = ["AAPL"]
start_date = "1980-01-01"
file_name = "daily_stock_data.csv"

# Execute
execute_stock_filtering_and_alerts(tickers, start_date)
```

```
[*********************100%***********************]  1 of 1 completedNo stocks meet the criteria. No email will be sent.
```