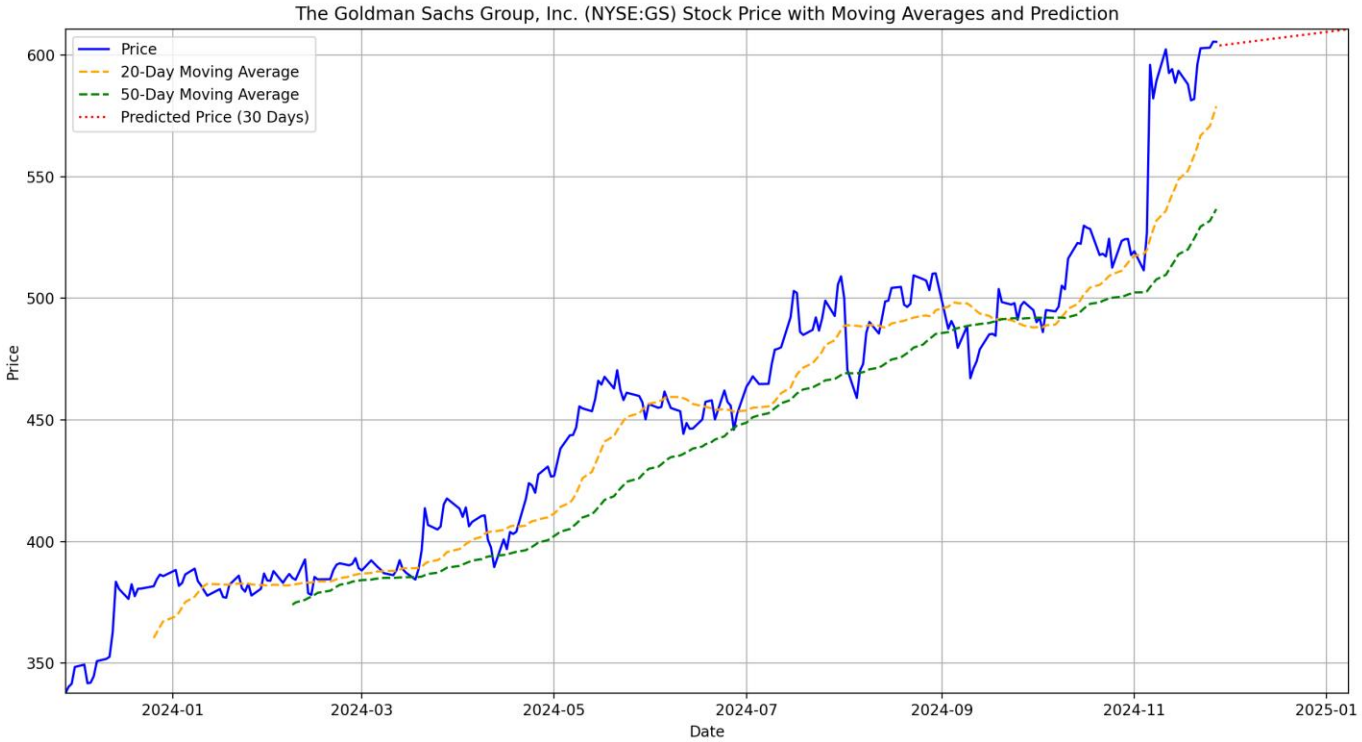
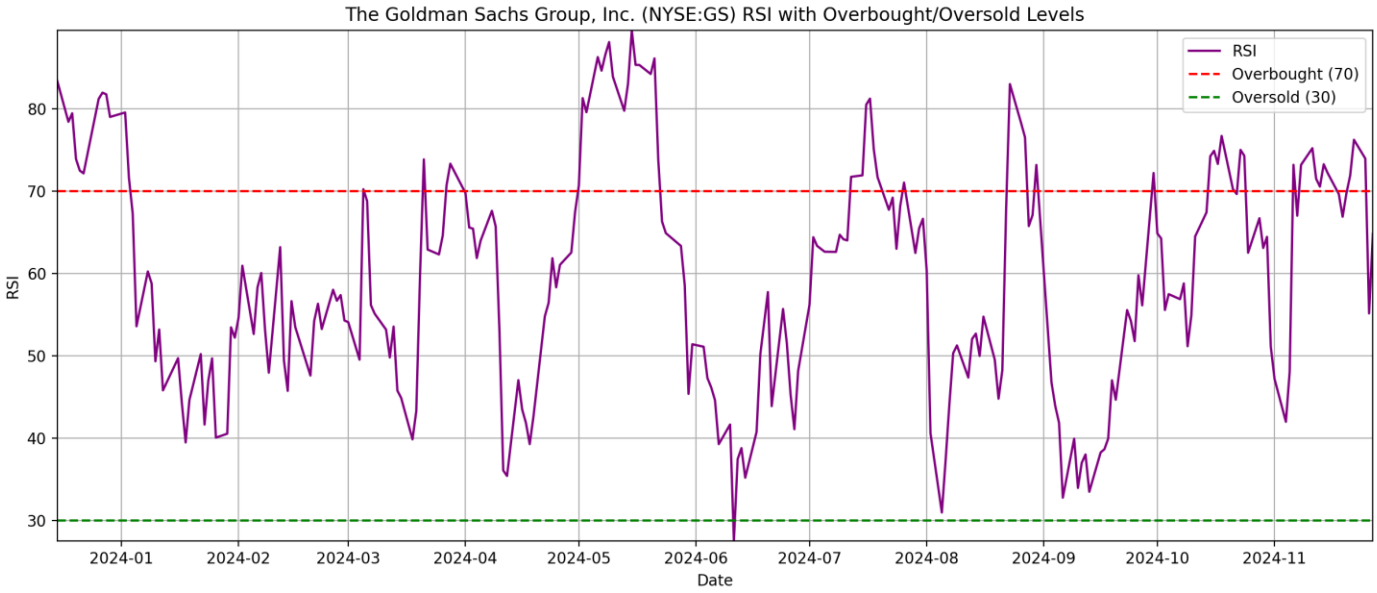
```
#Program built for historical price CSV from Capital IQ

# Import required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
import os

# Prompts the user for a valid CSV file and validates the input
# Returns:
#     str: The file path of the valid CSV
def getCsv():
    invalidFile = True
    while invalidFile:
        file = input("Enter a CSV with the data:\n")
        extension = os.path.splitext(file)[1]
        if os.path.isfile(file) and extension.lower() == ".csv":
            invalidFile = False
        else:
            print("That filepath does not lead to a valid CSV file.")
    return file

# Calculates the moving average for a given column in a DataFrame in place
# Parameters:
#     dataFrame (pd.DataFrame): The DataFrame containing the data
#     column (str): The column name for which the moving average is calculated
#     window (int): The size of the moving average window
# Returns:
#     None
def calculateMovingAverage(dataFrame, column, window):
    dataFrame[f'ma_{window}'] = dataFrame[column].rolling(window=window).mean()

# Calculates the Relative Strength Index (RSI) for a given column in a DataFrame in place
# Parameters:
#     dataFrame (pd.DataFrame): The DataFrame containing the data
#     column (str): The column name for which the RSI is calculated
#     window (int): The size of the RSI window (default is 14 for industry standard)
# Returns:
#     None
def calculateRsi(dataFrame, column, window=14):
    delta = dataFrame[column].diff(1)
    gain = np.where(delta > 0, delta, 0)
    loss = np.where(delta < 0, -delta, 0)

    avgGain = pd.Series(gain).rolling(window=window).mean()
    avgLoss = pd.Series(loss).rolling(window=window).mean()

    rs = avgGain / avgLoss
    dataFrame['rsi'] = 100 - (100 / (1 + rs))

# Automatically finds the date and price columns in a DataFrame
# Parameters:
#     dataFrame (pd.DataFrame): The DataFrame containing the data
# Returns:
#     tuple: A tuple of strings (dateColumn, priceColumn) representing the found column names
# Raises:
#     ValueError: If the required columns cannot be found
def findColumnNames(dataFrame):
    dateColumn = None
    priceColumn = None
    for colName in dataFrame.columns:
        if dateColumn and priceColumn:
            break
        if "Date" in colName or "Dates" in colName:
            dateColumn = colName
        elif "Price" in colName or "Share Pricing" in colName:
            priceColumn = colName
    if not dateColumn or not priceColumn:
        raise ValueError("Could not find the required columns (Date and Price) in the data.")
    return dateColumn, priceColumn

# Loads the data, finds necessary columns, and preprocesses it for analysis
# Parameters:
#     filePath (str): The path to the CSV file
# Returns:
#     tuple: A tuple containing the processed DataFrame, dateColumn, priceColumn, and companyName
def preprocessData(filePath):
    data = pd.read_csv(filePath)
    dateColumn, priceColumn = findColumnNames(data)
    data[dateColumn] = pd.to_datetime(data[dateColumn])
    data = data.sort_values(by=dateColumn)
    companyName = priceColumn.split(' - Share Pricing')[0].strip() if ' - Share Pricing' in priceColumn else
priceColumn
```

```
    return data, dateColumn, priceColumn, companyName

# Trains a linear regression model and predicts future prices
# Parameters:
#     dataFrame (pd.DataFrame): The DataFrame containing historical data
#     dateColumn (str): The name of the date column
#     priceColumn (str): The name of the price column
# Returns:
#     pd.DataFrame: A DataFrame with future dates and predicted prices
def trainModelAndPredict(dataFrame, dateColumn, priceColumn):
    dataFrame['days'] = (dataFrame[dateColumn] - dataFrame[dateColumn].min()).dt.days
    features = dataFrame[['days', 'ma_20', 'ma_50', 'rsi']].dropna()
    target = dataFrame.loc[features.index, priceColumn]

    model = LinearRegression()
    model.fit(features, target)

    futureDays = 30
    futureDates = pd.date_range(dataFrame[dateColumn].max(), periods=futureDays + 1, freq='B')[1:]
    futureDaysArray = np.array([(date - dataFrame[dateColumn].min()).days for date in futureDates])

    lastMa20 = dataFrame['ma_20'].iloc[-1]
    lastMa50 = dataFrame['ma_50'].iloc[-1]
    lastRsi = dataFrame['rsi'].iloc[-1]
    futureFeatures = pd.DataFrame({
        'days': futureDaysArray,
        'ma_20': [lastMa20] * len(futureDaysArray),
        'ma_50': [lastMa50] * len(futureDaysArray),
        'rsi': [lastRsi] * len(futureDaysArray)
    })

    futurePrices = model.predict(futureFeatures)
    return pd.DataFrame({'dates': futureDates, 'predicted_price': futurePrices})

# Plots historical stock prices, moving averages, RSI, and predicted prices
# Parameters:
#     dataFrame (pd.DataFrame): The DataFrame containing historical data
#     dateColumn (str): The name of the date column
#     priceColumn (str): The name of the price column
#     futureData (pd.DataFrame): The DataFrame containing future predictions
#     companyName (str): The name of the company for the plot titles
# Returns:
#     None
def plotData(dataFrame, dateColumn, priceColumn, futureData, companyName):
    plt.figure(figsize=(16, 8))
    plt.plot(dataFrame[dateColumn], dataFrame[priceColumn], label='Price', color='blue')
    plt.plot(dataFrame[dateColumn], dataFrame['ma_20'], label='20-Day Moving Average', color='orange',
linestyle='--')
    plt.plot(dataFrame[dateColumn], dataFrame['ma_50'], label='50-Day Moving Average', color='green',
linestyle='--')
    plt.plot(futureData['dates'], futureData['predicted_price'], label='Predicted Price (30 Days)', color='red',
linestyle='dotted')
    plt.margins(x=0, y=0)  # Remove unnecessary spacing
    plt.title(f'{companyName} Stock Price with Moving Averages and Prediction')
    plt.xlabel('Date')
    plt.ylabel('Price')
    plt.legend()
    plt.grid()
    plt.show()

    plt.figure(figsize=(16, 6))
    plt.plot(dataFrame[dateColumn], dataFrame['rsi'], label='RSI', color='purple')
    plt.axhline(70, color='red', linestyle='--', label='Overbought (70)')
    plt.axhline(30, color='green', linestyle='--', label='Oversold (30)')
    plt.margins(x=0, y=0)  # Remove unnecessary spacing
    plt.title(f'{companyName} RSI with Overbought/Oversold Levels')
    plt.xlabel('Date')
    plt.ylabel('RSI')
    plt.legend()
    plt.grid()
    plt.show()

# Main function to run the stock analysis and prediction pipeline
def main():
    filePath = getCsv()
    data, dateColumn, priceColumn, companyName = preprocessData(filePath)
    calculateMovingAverage(data, priceColumn, 20)
    calculateMovingAverage(data, priceColumn, 50)
    calculateRsi(data, priceColumn)
    futureData = trainModelAndPredict(data, dateColumn, priceColumn)
    plotData(data, dateColumn, priceColumn, futureData, companyName)

if __name__ == "__main__":
    main()
```


The Goldman Sachs Group, Inc. (NYSE:GS) Stock Price with Moving Averages and Prediction
Price
20-Day Moving Average
50-Day Moving Average
Predicted Price (30 Days)
Price
Date
600
550
500
450
400
350
2024-01
2024-03
2024-05
2024-07
2024-09
2024-11
2025-01


The Goldman Sachs Group, Inc. (NYSE:GS) RSI with Overbought/Oversold Levels
RSI
Overbought (70)
Oversold (30)
RSI
Date
80
70
60
50
40
30
2024-01
2024-02
2024-03
2024-04
2024-05
2024-06
2024-07
2024-08
2024-09
2024-10
2024-11