DRL

Release 1

jj

Dec 24, 2024

CONTENTS:

CHAPTER

ONE

THE CASE OF DEUTZ

In this manual the stock value of Deutz is used, mainly because it fluctuates and has some periodicity.

I want to apply DRL (deep reinformcement learning) to support buy/sell decisions.

Upon applying DRL, I got no profit at all. But since DRL was base on sell/buy indicators, I should first focus on indicators that might make a profit.

1.1 background

- overview of technical indicators
- overview of ratio's
- overview of backtesting
- benchmarking

CHAPTER

TWO

DEUTZ AG

2.1 Overview

Deutz AG is a German manufacturer of internal combustion engines. The company was founded in 1864 by Nicolaus Otto, who invented the four-stroke engine. Deutz AG is headquartered in Cologne, Germany, and is known for its innovation and engineering excellence in the production of diesel and gas engines.

The company operates in various sectors, including agriculture, construction, material handling, and stationary equipment. Deutz engines are recognized for their reliability, efficiency, and compliance with stringent emission standards.

2.2 Share Information

Deutz AG is publicly traded on the Frankfurt Stock Exchange under the ticker symbol *DEZ*. The company's shares are included in the SDAX index, which lists small capitalization companies in Germany.

Key highlights about Deutz AG shares: - **Ticker Symbol**: DEZ - **Exchange**: Frankfurt Stock Exchange - **Index**: SDAX

Investors consider Deutz AG a stable investment due to its long history, strong market position, and continuous innovation in the engine manufacturing industry.

2.3 Useful Links

- Deutz AG Official Website
- Deutz AG Investor Relations

CHAPTER

THREE

JUPYTER VIRTUAL ENVIRONMENT

Problems in using yfinance module, can be resolved this way:

Optional: Use the Virtual Environment in Jupyter Notebook

If you are using Jupyter Notebook, you can create a kernel for your virtual environment:

> Install ipykernel within the virtual environment:
>
> pip install ipykernel

Create a new kernel for your virtual environment:

python -m ipykernel install –user –name=myenv –display-name “Python (myenv)”

> Replace myenv with the name of your virtual environment.
>
> Select the new kernel in Jupyter Notebook:
>
> Open Jupyter Notebook, go to Kernel -> Change kernel and select “Python (myenv)”.

By following these steps, you can ensure that yfinance is installed in a managed and isolated environment, preventing conflicts with the system-wide Python installation.

CHAPTER

FOUR

PREDICTING MARKET TRENDS WITH TECHNICAL INDICATORS

The most basic method to predict trends in the trading market is to analyze historical trading data. Technical indicators are mathematical calculations built on historical trading data and are more effective for traders to use to identify potential buying and selling opportunities in the trading market.

4.1 Categories of Technical Indicators

Technical indicators can be broadly classified into four categories:

1. **Trend Indicators**

   These indicators are used to identify the direction of a trend and whether it is bullish (upward) or bearish (downward). Simple Moving Average (SMA) is a popular example of trend indicators.

2. **Momentum Indicators**

   These indicators measure the speed or rate of price movements. They are used to identify overbought and oversold conditions as well as potential trend reversals. Moving Average Convergence Divergence (MACD) is an example of momentum indicators.

3. **Volatility Indicators**

   These indicators measure the volatility of the market or the degree of price fluctuations. Bollinger Bands (BB) is an example of volatility indicators.

4. **Volume Indicators**

   These indicators measure the amount of trading activity in the market. They are used to confirm price action and identify potential trend reversals. Chaikin Money Flow (CMF) is an example of volume indicators.

These categories are not mutually exclusive, and many indicators can fall into more than one category. For example, MACD is both a momentum indicator and a trend indicator.

Note

No single indicator can guarantee a successful trade, but a combination of several different types of technical indicators is still valid.

4.2 Calculate Technical Indicators with Python

We will use a combination of the four technical indicators mentioned above (SMA, MACD, BB, and CMF) to build the dataset for machine learning.

First thing first, as described in *Acquiring Data*, the first step towards using machine learning for stock trading is to download some historical trading data of Apple as an example:

```python
import yfinance as yf

stock = yf.Ticker('AAPL')
hist = stock.history(period='2y', interval='1d', actions=False)
```

4.2.1 Simple Moving Average (SMA)

The Simple Moving Average (SMA) is calculated by taking the average closing price of a stock over a specified period of time.

The SMA is a lagging indicator, meaning that it is based on past prices and does not predict future prices. It is used to smooth out price fluctuations and to identify trends. When the price of an asset is above its SMA, it is generally considered to be in an uptrend, and when the price is below its SMA, it is generally considered to be in a downtrend.

Using the trading data we obtained using *yfinance* earlier, the 10-day and 20-day SMAs can be easily calculated as:

```python
hist['SMA10'] = hist['Close'].rolling(window=10).mean()
hist['SMA20'] = hist['Close'].rolling(window=20).mean()
```

4.2.2 Moving Average Convergence Divergence (MACD)

The Moving Average Convergence Divergence (MACD) shows the relationship between two exponential moving averages (EMA) of a stock's price. The MACD is calculated by subtracting the 26-day EMA from the 12-day EMA.

The MACD includes:

- **MACD Line:** The difference between the two EMAs.
- **Signal Line:** A 9-day EMA of the MACD line.
- **Histogram:** The difference between the MACD line and the signal line.

The MACD is used to identify changes in momentum and trend direction. For example:

- A **bullish signal** occurs when the MACD line crosses above the signal line.
- A **bearish signal** occurs when the MACD line crosses below the signal line.

Calculating MACD with Python:

```python
ema_short = hist['Close'].ewm(span=12, adjust=False).mean()
ema_long = hist['Close'].ewm(span=26, adjust=False).mean()
hist['MACD_DIF'] = ema_short - ema_long
hist['MACD_SIGNAL'] = hist['MACD_DIF'].ewm(span=9, adjust=False).mean()
hist['MACD'] = hist['MACD_DIF'] - hist['MACD_SIGNAL']
```

4.2.3 Bollinger Bands (BB)

Bollinger Bands are based on a moving average of a stock's price and are designed to show the volatility and potential trading range of the stock.

Bollinger Bands include:

- **Middle Line:** A 20-day SMA.
- **Upper Band (UB):** Two standard deviations above the middle line.
- **Lower Band (LB):** Two standard deviations below the middle line.

When the price of the asset moves outside of the bands, it is considered to be overbought or oversold.

Calculate Bollinger Bands with Python:

```
sma = hist['Close'].rolling(window=20).mean()
std = hist['Close'].rolling(window=20).std()
hist['UB'] = sma + 2 * std
hist['LB'] = sma - 2 * std
```

4.2.4 Chaikin Money Flow (CMF)

Chaikin Money Flow (CMF) measures the volume-weighted average of accumulation and distribution over a specific period of time. It is used to identify buying and selling pressure of a stock.

Calculate CMF with Python:

```
mfm = ((hist['Close'] - hist['Low']) - (hist['High'] - hist['Close'])) / (hist['High'] -␣
↪hist['Low'])
mfv = mfm * hist['Volume']
hist['CMF'] = mfv.rolling(21).sum() / hist['Volume'].rolling(21).sum()
```

4.2.5 Final Dataset

Finally, check what the dataset looks like:

```
hist[['SMA10', 'SMA20', 'MACD_DIF', 'MACD_SIGNAL', 'MACD', 'UB', 'LB', 'CMF']]
```

[ ]:

```

```

[1]:

```
import yfinance as yf
import matplotlib.pyplot as plt
from datetime import datetime, timedelta
#import random
#import pandas as pd
# Download data from Yahoo Finance
today_date = datetime.today().strftime('%Y-%m-%d')
#data = yf.download('DEZ.DE', start='2013-01-01', end='2024-01-01')
data = yf.download('DEZ.DE', start='2013-01-01', end=today_date)


# Verify data download and handle potential issues
if data.empty:
    raise ValueError("No data was fetched. Verify the ticker and date range.")

# Rename the columns to remove the multi-level index
data.columns = data.columns.get_level_values(0)


data_all = data.dropna()
# add the 'PCT change' column
data_all['Return'] = 100 * (data_all['Close'].pct_change())

data_all['SMA10'] = data_all['Close'].rolling(window=10).mean()
```

(continues on next page)

(continued from previous page)

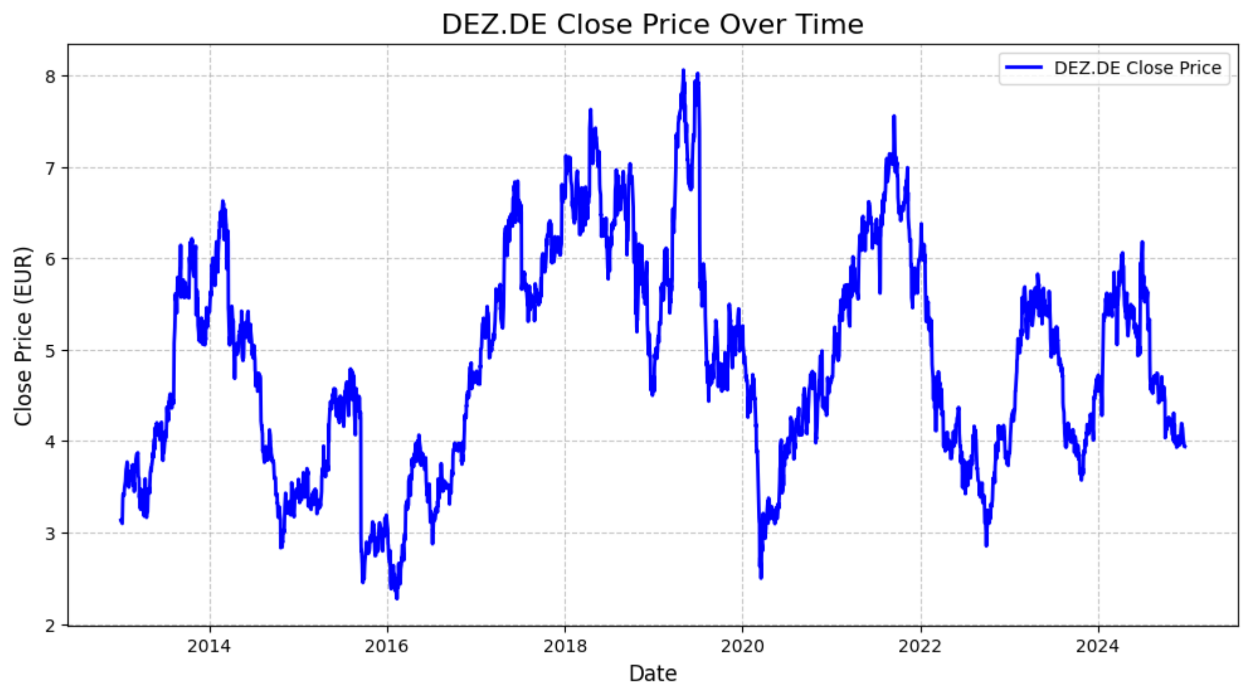
```
data_all['SMA20'] = data_all['Close'].rolling(window=20).mean()

ema_short = data_all['Close'].ewm(span=12, adjust=False).mean()
ema_long = data_all['Close'].ewm(span=26, adjust=False).mean()
data_all['MACD_DIF'] = ema_short - ema_long
data_all['MACD_SIGNAL'] = data_all['MACD_DIF'].ewm(span=9, adjust=False).mean()
data_all['MACD'] = data_all['MACD_DIF'] - data_all['MACD_SIGNAL']

#bollinger
sma = data_all['Close'].rolling(window=20).mean()
std = data_all['Close'].rolling(window=20).std()
data_all['UB'] = sma + 2 * std
data_all['LB'] = sma - 2 * std

mfm = ((data_all['Close'] - data_all['Low']) - (data_all['High'] - data_all['Close'])) /␣
↪(data_all['High'] - data_all['Low'])
mfv = mfm * data_all['Volume']
data_all['CMF'] = mfv.rolling(21).sum() / data_all['Volume'].rolling(21).sum()
```

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

[2]:
```
data_all[['SMA10', 'SMA20', 'MACD_DIF', 'MACD_SIGNAL', 'MACD', 'UB', 'LB', 'CMF']]
```

[2]:
```
Price        SMA10   SMA20  MACD_DIF  MACD_SIGNAL      MACD        UB  \
Date
2013-01-02     NaN     NaN  0.000000     0.000000  0.000000       NaN
2013-01-03     NaN     NaN -0.001065    -0.000213 -0.000852       NaN
2013-01-04     NaN     NaN -0.001688    -0.000508 -0.001180       NaN
2013-01-07     NaN     NaN -0.000293    -0.000465  0.000172       NaN
2013-01-08     NaN     NaN -0.003190    -0.001010 -0.002180       NaN
...            ...     ...       ...          ...       ...       ...
2024-12-17  4.1066  4.0487  0.000541    -0.005917  0.006458  4.210200
2024-12-18  4.1034  4.0517 -0.005862    -0.005906  0.000044  4.206800
2024-12-19  4.0964  4.0499 -0.012248    -0.007174 -0.005074  4.207463
2024-12-20  4.0806  4.0463 -0.020302    -0.009800 -0.010503  4.210468
2024-12-23  4.0552  4.0405 -0.026701    -0.013180 -0.013521  4.211574

Price             LB  CMF
Date
2013-01-02       NaN  NaN
2013-01-03       NaN  NaN
2013-01-04       NaN  NaN
2013-01-07       NaN  NaN
2013-01-08       NaN  NaN
...              ...  ...
2024-12-17  3.887200  NaN
2024-12-18  3.896600  NaN
2024-12-19  3.892337  NaN
2024-12-20  3.882132  NaN
2024-12-23  3.869426  NaN

[3044 rows x 8 columns]
```

[3]:
```
# Plot the 'Close' prices
plt.figure(figsize=(12, 6))
plt.plot(data_all['Close'], label='DEZ.DE Close Price', color='blue', linewidth=2)
plt.title('DEZ.DE Close Price Over Time', fontsize=16)
plt.xlabel('Date', fontsize=12)
plt.ylabel('Close Price (EUR)', fontsize=12)
plt.grid(True, linestyle='--', alpha=0.7)
plt.legend()


plt.show()
```

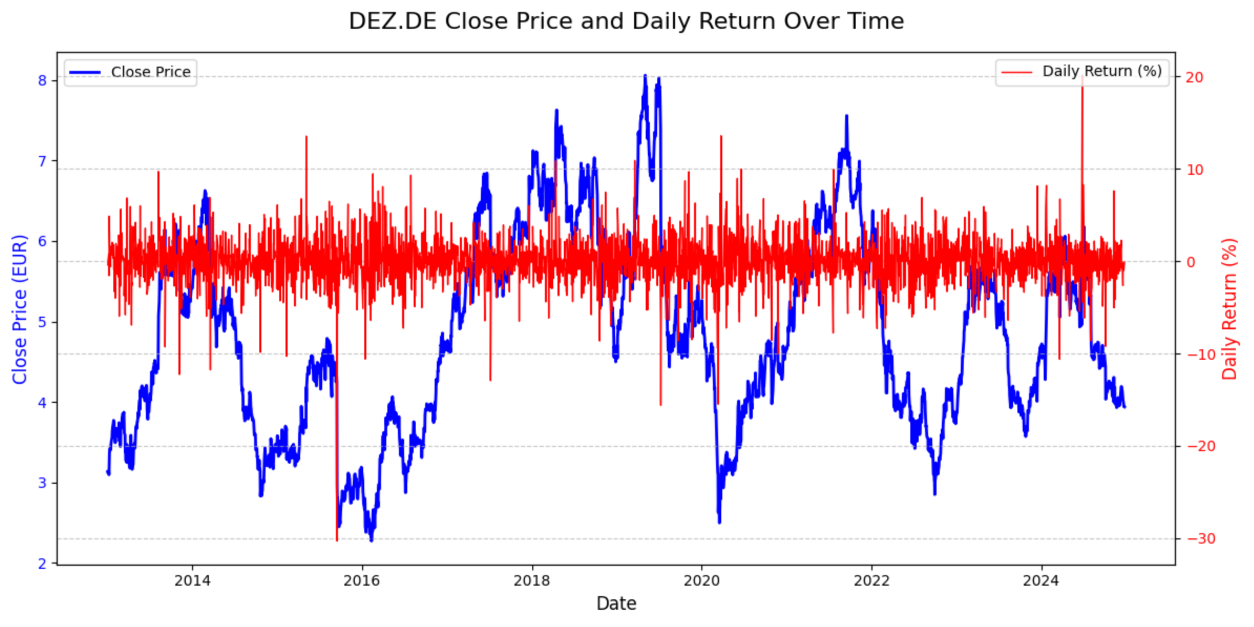


[4]:
```
# Plotting
fig, ax1 = plt.subplots(figsize=(12, 6))

# Plot Close Price
ax1.plot(data_all['Close'], label='Close Price', color='blue', linewidth=2)
ax1.set_xlabel('Date', fontsize=12)
ax1.set_ylabel('Close Price (EUR)', fontsize=12, color='blue')
ax1.tick_params(axis='y', labelcolor='blue')

# Create a second y-axis for the Return
ax2 = ax1.twinx()
ax2.plot(data_all['Return'], label='Daily Return (%)', color='red', linewidth=1)
ax2.set_ylabel('Daily Return (%)', fontsize=12, color='red')
ax2.tick_params(axis='y', labelcolor='red')

# Titles, legends, and grid
```

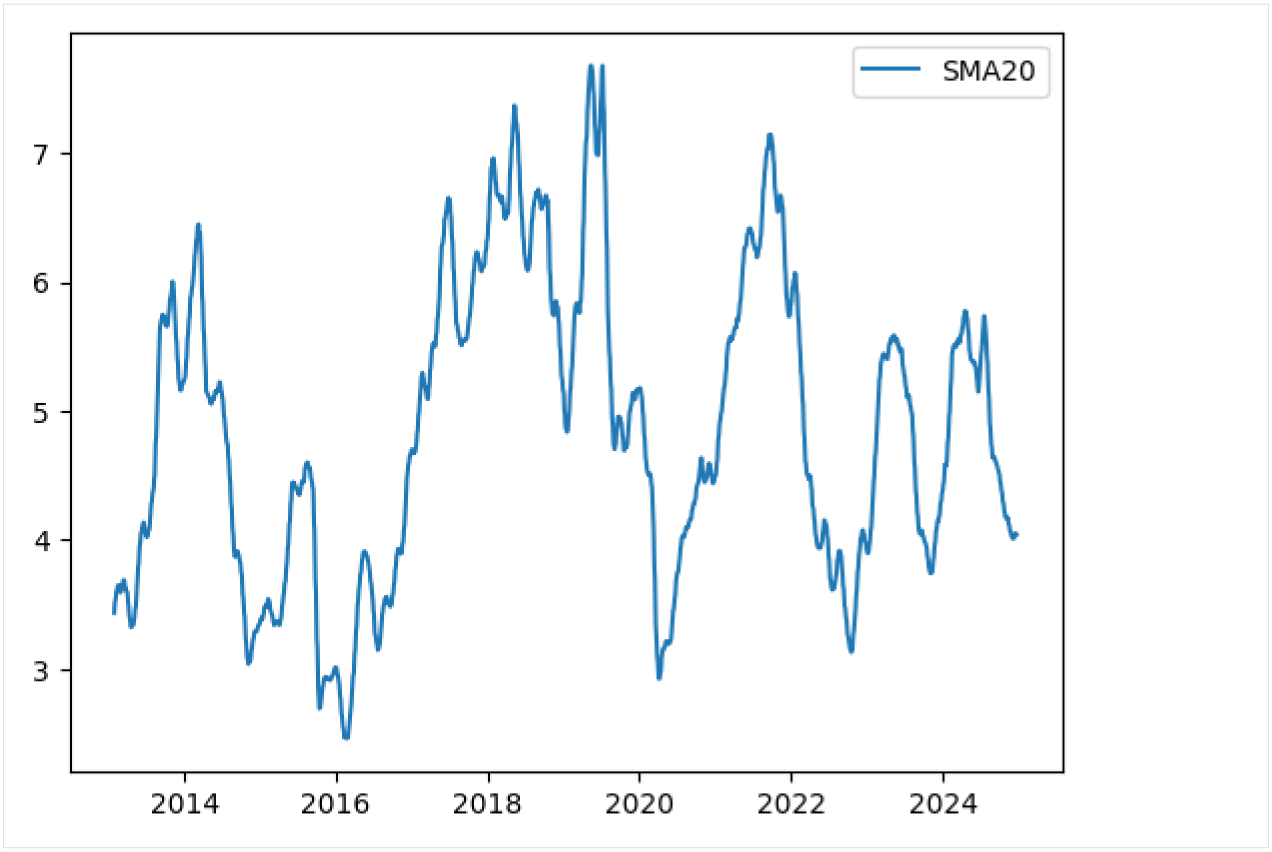
(continues on next page)

(continued from previous page)

```
fig.suptitle('DEZ.DE Close Price and Daily Return Over Time', fontsize=16)
ax1.legend(loc='upper left')
ax2.legend(loc='upper right')
plt.grid(True, linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```



```
[5]: #plt.plot(data_all['Close'],label='Close')
     plt.plot(data_all['SMA20'],label='SMA20')
     plt.legend()
     plt.show()
```



get hourly data — this is only possible for last … days on yahoo

```
[6]: from datetime import timedelta
today_date = datetime.today()
five_days_ago = today_date - timedelta(days=5)
five_days_ago_formatted = five_days_ago.strftime('%Y-%m-%d')
data = yf.download('DEZ.DE', interval='1h', start=five_days_ago_formatted, end=today_
↪date)

print(data.head())
```

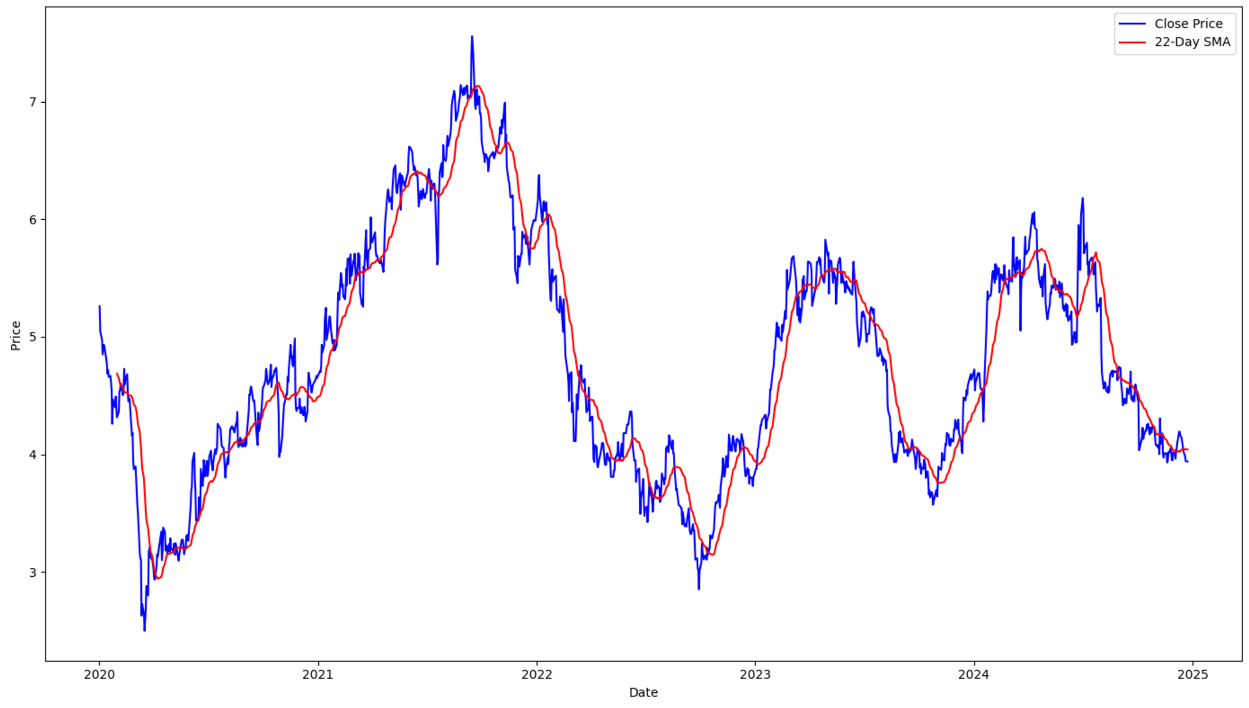
```
[*********************100%***********************]  1 of 1 completed
```

```
Price                          Close   High    Low   Open Volume
Ticker                        DEZ.DE DEZ.DE DEZ.DE DEZ.DE DEZ.DE
Datetime
2024-12-19 08:00:00+00:00  3.946  3.964  3.890  3.936      0
2024-12-19 09:00:00+00:00  3.940  3.954  3.926  3.944  79868
2024-12-19 10:00:00+00:00  3.922  3.934  3.916  3.926  38599
2024-12-19 11:00:00+00:00  3.912  3.920  3.906  3.920  21835
2024-12-19 12:00:00+00:00  3.932  3.936  3.908  3.908  59735
```

```
[ ]:
```

[1]:
```
import yfinance as yf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Haal historische gegevens op voor DEZ.DE
ticker = "DEZ.DE"
data = yf.download(ticker, start="2020-01-01", end="2024-12-31")

# Bereken voortschrijdend gemiddelde (50-daags)
data['SMA_50'] = data['Close'].rolling(window=50).mean()
data['SMA_22'] = data['Close'].rolling(window=22).mean()

plt.figure(figsize=(14, 8))
plt.plot(data['Close'], label='Close Price', color='blue')
#plt.plot(data['SMA_50'], label='50-Day SMA', color='orange')
plt.plot(data['SMA_22'], label='22-Day SMA', color='red')

plt.xlabel("Date")
plt.ylabel("Price ")
plt.legend()
plt.tight_layout()
plt.show()
```

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



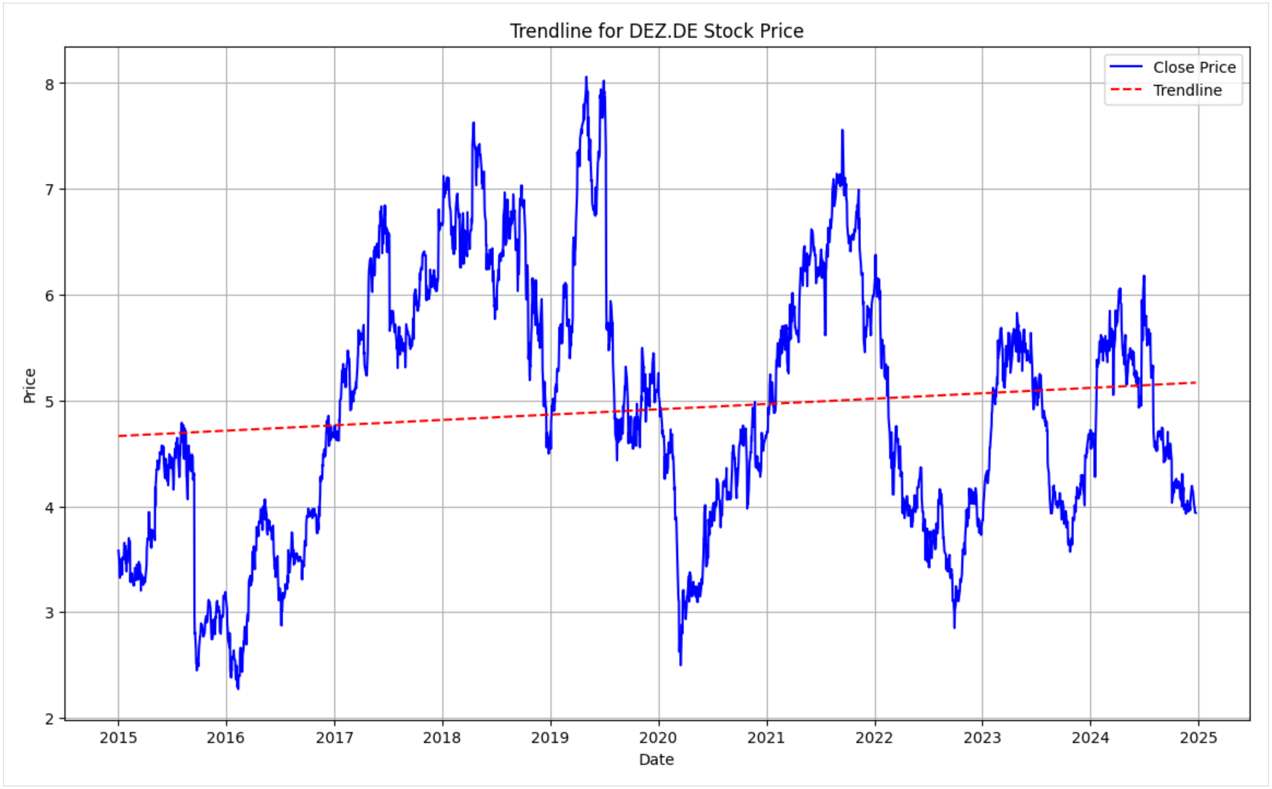
[1]:
```
import yfinance as yf
import pandas as pd
import numpy as np
```

(continues on next page)

(continued from previous page)

```
import matplotlib.pyplot as plt
from scipy.stats import linregress

# Download historical stock data for DEZ.DE
ticker = "DEZ.DE"
data = yf.download(ticker, start="2015-01-01", end="2024-12-31")

# Check if data was downloaded correctly
if data.empty:
    raise ValueError(f"No data found for ticker {ticker} within the specified date range.
↪")
# Rename the columns to remove the multi-level index
data.columns = data.columns.get_level_values(0)
# Ensure 'Date' is properly set and create numeric time index
data.reset_index(inplace=True)  # Convert index to a column

# Check for missing 'Close' prices
if data['Close'].isnull().any():
    raise ValueError("Missing 'Close' prices in the data. Please check the stock data.")

data['Time'] = np.arange(len(data))  # Numeric time index for regression

# Perform linear regression on 'Close' prices
slope, intercept, r_value, p_value, std_err = linregress(data['Time'], data['Close'])
data['Trendline'] = slope * data['Time'] + intercept

# Convert columns to NumPy arrays
dates = data['Date'].to_numpy()
close_prices = data['Close'].to_numpy()
trendline = data['Trendline'].to_numpy()

# Plot the stock price and its trendline
plt.figure(figsize=(14, 8))
plt.plot(dates, close_prices, label='Close Price', color='blue')
plt.plot(dates, trendline, label='Trendline', color='red', linestyle='--')
plt.title(f"Trendline for {ticker} Stock Price")
plt.xlabel("Date")
plt.ylabel("Price")
plt.legend()
plt.grid()
plt.show()
```

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



[ ]:

[1]:
```
import backtrader as bt
import datetime
import yfinance as yf
import matplotlib.pyplot as plt
import numpy as np
from backtrader.indicators import RSI, BollingerBands
#workaround to get this in sphinx notebook
from IPython.display import Image, display

# Fetch data using yfinance
data_df = yf.download('DEZ.DE', start='2010-01-01', end='2025-01-01')
data_df.columns = data_df.columns.get_level_values(0)
# Create a custom pandas feed to use with Backtrader


plt.style.use("default") #ggplot is also fine
plt.rcParams["figure.figsize"] = (11,7)


class PandasData(bt.feeds.PandasData):
    params = (
        ('fromdate', datetime.datetime(2010, 1, 1)),
        ('todate', datetime.datetime(2025, 1, 1)),
        ('open', 'Open'),
        ('high', 'High'),
```
(continues on next page)

(continued from previous page)

```
        ('low', 'Low'),
        ('close', 'Close'),
        ('volume', 'Volume'),
        ('openinterest', None),  # No open interest in Yahoo data

    )

#this is just a straight line
class MeanReversionStrategyTrend(bt.SignalStrategy):
    params = (
        ('deviation_threshold', 0.20),  # 20% deviation threshold
        ('margin_of_safety', 0.05),  # 5% margin of safety
    )

    def __init__(self):
        # Calculate the average price over the entire period
        self.average_price = np.mean(self.data.close.array)
        print(f"Average Price (Trendline): {self.average_price:.2f}")

    def next(self):
        deviation = (self.data.close[0] - self.average_price) / self.average_price
        if not self.position:  # Not in the market
            if deviation < -self.params.deviation_threshold * (1 + self.params.margin_of_
↪safety):
                # Calculate the number of shares to buy
                cash = self.broker.get_cash()
                print ("cash",cash)
                size = cash // self.data.close[0]
                print ("size",size)
                size=size-100 #otherwise problem with commission
                self.buy(size=size)
        else:  # In the market
            if deviation > self.params.deviation_threshold * (1 - self.params.margin_of_
↪safety):
                # Sell all shares
                self.sell(size=self.position.size)

    def notify_order(self, order):
        if order.status in [order.Submitted, order.Accepted]:
            # Order is submitted/accepted to/by broker - Do nothing
            return

        # Check if an order has been completed
        if order.status in [order.Completed]:
            if order.isbuy():
                print(f'BUY EXECUTED, Price: {order.executed.price:.2f}, Size: {order.
↪executed.size}')
            elif order.issell():
                print(f'SELL EXECUTED, Price: {order.executed.price:.2f}, Size: {order.
↪executed.size}')

            self.bar_executed = len(self)
```

(continues on next page)

(continued from previous page)

```
        elif order.status in [order.Canceled, order.Margin, order.Rejected]:
            print('Order Canceled/Margin/Rejected')

        # Reset orders
        self.order = None

    def notify_trade(self, trade):
        if not trade.isclosed:
            return

        print(f'OPERATION PROFIT, GROSS: {trade.pnl:.2f}, NET: {trade.pnlcomm:.2f}')


#this is based on moving average
class MeanReversionStrategy(bt.Strategy):
    params = (
        ('sma_period', 380),  # 16 months assuming 30 trading days per month
        ('deviation_threshold', 0.20),  # 20% deviation threshold
        ('margin_of_safety', 0.05),  # 5% margin of safety
    )

    def __init__(self):
        self.sma = bt.indicators.SimpleMovingAverage(self.data.close, period=self.params.
↪sma_period)
        self.deviation = (self.data.close - self.sma) / self.sma

    def next(self):
        if not self.position:  # Not in the market
            if self.deviation < -self.params.deviation_threshold * (1 + self.params.
↪margin_of_safety):
                # Calculate the number of shares to buy
                cash = self.broker.get_cash()
                size = cash // self.data.close[0]
#                self.buy(size=size-1)
                self.buy(size=1000)
        else:  # In the market
            if self.deviation > self.params.deviation_threshold * (1 - self.params.
↪margin_of_safety):
                # Sell all shares
#                self.sell(size=self.position.size)
                self.sell(size=1000)

    def notify_order(self, order):
        if order.status in [order.Submitted, order.Accepted]:
            # Order is submitted/accepted to/by broker - Do nothing
            return

        # Check if an order has been completed
        if order.status in [order.Completed]:
            if order.isbuy():
                print(f'BUY EXECUTED, Price: {order.executed.price:.2f}, Size: {order.
```

(continues on next page)

(continued from previous page)

```
↪executed.size:.2f}')
            elif order.issell():
                print(f'SELL EXECUTED, Price: {order.executed.price:.2f}, Size: {order.
↪executed.size:.2f}')

            self.bar_executed = len(self)

        elif order.status in [order.Canceled, order.Margin, order.Rejected]:
            print('Order Canceled/Margin/Rejected')

        # Reset orders
        self.order = None

    def notify_trade(self, trade):
        if not trade.isclosed:
            return




# Create an instance of Cerebro engine
cerebro = bt.Cerebro()

#cerebro.addstrategy(MeanReversionStrategy)
cerebro.addstrategy(MeanReversionStrategyTrend)
# Convert the DataFrame into a Backtrader-compatible data feed
data_feed = PandasData(dataname=data_df)

# Add the data feed to Cerebro
cerebro.adddata(data_feed)

# Set initial capital and commissions
cerebro.broker.setcash(10000)  #   starting capital
cerebro.broker.setcommission(commission=0.001)  # 0.02% commission on turnover

# Print the starting portfolio value
print('Starting Portfolio Value: %.2f' % cerebro.broker.getvalue())

# Run the backtest
results = cerebro.run()

# Print the final portfolio value
final_value = cerebro.broker.getvalue()
print('Final Portfolio Value: %.2f' % final_value)

# Calculate Profit/Loss and other metrics
profit_loss = final_value - 10000
print('Net Profit/Loss: %.2f' % profit_loss)

# Plot the results
#fig = cerebro.plot(iplot=False)[0][0]
#fig = cerebro.plot(iplot=False)
```

(continues on next page)

(continued from previous page)

```
#fig = cerebro.plot(iplot=True)
#cerebro.plot()
#plt.show()
    # Plot the results and save to a file
fig = cerebro.plot(iplot=True, figsize=(12, 8))[0][0]
fig.savefig('backtrader_plot.png')

    # Display the image in the notebook
display(Image(filename='backtrader_plot.png'))
```

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

```
Starting Portfolio Value: 10000.00
Average Price (Trendline): 4.68
cash 10000.0
size 3452.0
BUY EXECUTED, Price: 2.82, Size: 3352.0
SELL EXECUTED, Price: 5.71, Size: -3352.0
OPERATION PROFIT, GROSS: 9679.59, NET: 9651.00
cash 19650.996299421196
size 5351.0
BUY EXECUTED, Price: 3.63, Size: 5251.0
SELL EXECUTED, Price: 5.58, Size: -5251.0
OPERATION PROFIT, GROSS: 10211.15, NET: 10162.81
cash 29813.80776825101
size 8149.0
BUY EXECUTED, Price: 3.66, Size: 8049.0
SELL EXECUTED, Price: 5.61, Size: -8049.0
OPERATION PROFIT, GROSS: 15644.73, NET: 15570.10
cash 45383.90966176748
size 12322.0
Order Canceled/Margin/Rejected
cash 45383.90966176748
size 12738.0
BUY EXECUTED, Price: 3.51, Size: 12638.0
SELL EXECUTED, Price: 5.58, Size: -12638.0
OPERATION PROFIT, GROSS: 26210.33, NET: 26095.40
cash 71479.31409280212
size 20467.0
Order Canceled/Margin/Rejected
cash 71479.31409280212
size 19701.0
Order Canceled/Margin/Rejected
cash 71479.31409280212
size 20003.0
BUY EXECUTED, Price: 3.54, Size: 19903.0
SELL EXECUTED, Price: 5.56, Size: -19903.0
OPERATION PROFIT, GROSS: 40294.91, NET: 40113.73
cash 111593.04117381517
size 30505.0
BUY EXECUTED, Price: 3.63, Size: 30405.0
```

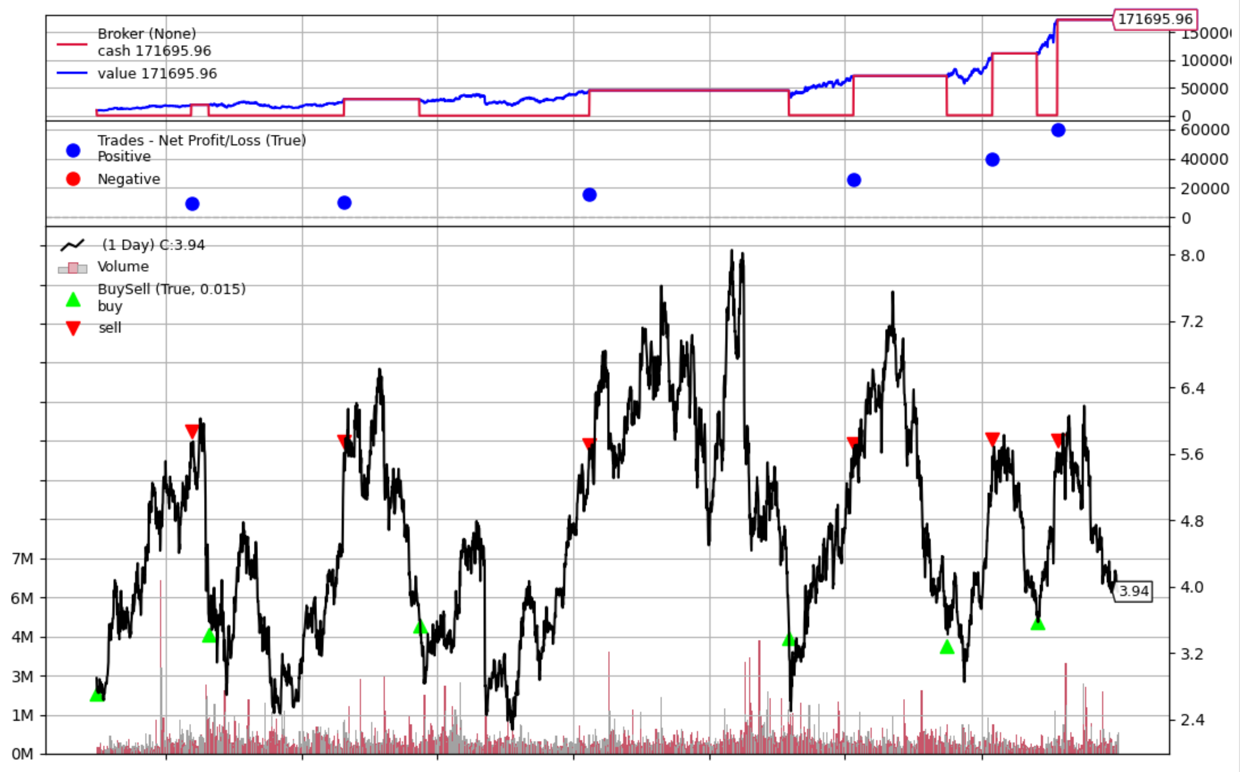
(continues on next page)

(continued from previous page)

```
SELL EXECUTED, Price: 5.62, Size: -30405.0
OPERATION PROFIT, GROSS: 60384.22, NET: 60102.92
Final Portfolio Value: 171695.96
Net Profit/Loss: 161695.96
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```



```
[ ]:
```

```
[1]: import backtrader as bt
import datetime
import yfinance as yf
import matplotlib.pyplot as plt
import numpy as np
from backtrader.indicators import RSI, BollingerBands
# Fetch data using yfinance
#workaround to get this in sphinx notebook
from IPython.display import Image, display


# Enable inline plotting
%matplotlib inline

fromdate = datetime.datetime(2020, 3, 1)
todate = datetime.datetime(2021, 9, 1)
```

(continues on next page)

(continued from previous page)

```python
startbalans = 10000
commissie = 0.001

data_df = yf.download('DEZ.DE', start=fromdate, end=todate)
data_df.columns = data_df.columns.get_level_values(0)
# Create a custom pandas feed to use with Backtrader


plt.style.use("default") #ggplot is also fine
plt.rcParams["figure.figsize"] = (12,8)


class PandasData(bt.feeds.PandasData):
    params = (
        ('fromdate', fromdate),
        ('todate', todate),
        ('open', 'Open'),
        ('high', 'High'),
        ('low', 'Low'),
        ('close', 'Close'),
        ('volume', 'Volume'),
        ('openinterest', None),  # No open interest in Yahoo data

    )
#Chaikin
class CMF(bt.Indicator):
    lines = ('money_flow',)
    params = (('len', 20),)

    def __init__(self):
        # Calculate the AD line
        c = self.data.close
        h = self.data.high
        l = self.data.low
        v = self.data.volume

        # Prevent division by zero
        self.data.ad = bt.If(bt.Or(bt.And(c == h, c == l), h == l), 0, ((2*c - l - h) /␣
↪bt.If(h == l, 1, h - l)) * v)

        # Calculate the Chaikin Money Flow (CMF)
        sum_volume = bt.indicators.SumN(self.data.volume, period=self.p.len)
        self.lines.money_flow = bt.If(sum_volume == 0, 0, bt.indicators.SumN(self.data.
↪ad, period=self.p.len) / sum_volume)

class CMFStrategy(bt.Strategy):
    params = (('period', 20),)

    def __init__(self):
        self.cmf = CMF(self.data, len=self.params.period)

    def next(self):
```

(continues on next page)

(continued from previous page)

```
        cash = self.broker.get_cash()
        #print("cash", cash)

        if self.cmf.money_flow > 0:
            if not self.position:
                size = cash // self.data.close[0]
                #print("size", size)
                size = size - 100
                self.buy(size=size)
        elif self.cmf.money_flow < 0:
            if self.position:
                self.sell(size=self.position.size)
               # self.sell()
    def notify_order(self, order):
        if order.status in [order.Submitted, order.Accepted]:
            # Order is submitted/accepted to/by broker - Do nothing
            return

        # Check if an order has been completed
        if order.status in [order.Completed]:
            if order.isbuy():
                print(f'BUY EXECUTED, Price: {order.executed.price:.2f}, Size: {order.
↪executed.size}')
            elif order.issell():
                print(f'SELL EXECUTED, Price: {order.executed.price:.2f}, Size: {order.
↪executed.size}')

            self.bar_executed = len(self)

        elif order.status in [order.Canceled, order.Margin, order.Rejected]:
            print('Order Canceled/Margin/Rejected')

        # Reset orders
        self.order = None

    def notify_trade(self, trade):
        if not trade.isclosed:
            return

        print(f'OPERATION PROFIT, GROSS: {trade.pnl:.2f}, NET: {trade.pnlcomm:.2f}')



class testStrategy(bt.Strategy):

    def __init__(self):
        self.rsi = MoneyFlow(self.data)



# Create an instance of Cerebro engine
cerebro = bt.Cerebro()
```

(continues on next page)

(continued from previous page)

```

# Add the strategy to Cerebro

cerebro.addstrategy(CMFStrategy)

# Convert the DataFrame into a Backtrader-compatible data feed
data_feed = PandasData(dataname=data_df)

# Add the data feed to Cerebro
cerebro.adddata(data_feed)

# Set initial capital and commissions
cerebro.broker.setcash(startbalans)  # Rs 3,00,000 starting capital
cerebro.broker.setcommission(commission=commissie)  # 0.02% commission on turnover

# Print the starting portfolio value
print('Starting Portfolio Value: %.2f' % cerebro.broker.getvalue())

# Run the backtest
results = cerebro.run()

# Print the final portfolio value
final_value = cerebro.broker.getvalue()
print('Final Portfolio Value: %.2f' % final_value)

# Calculate Profit/Loss and other metrics
profit_loss = final_value - startbalans
print('Net Profit/Loss: %.2f' % profit_loss)

# Plot the results
#fig = cerebro.plot(iplot=False)[0][0]
#fig = cerebro.plot(iplot=True, figsize=(10, 6))

#fig = cerebro.plot(iplot=True, figsize=(10, 6))
#cerebro.plot(iplot=True, style='bar', figsize=(10, 6))
#plt.show()


    # Plot the results and save to a file
fig = cerebro.plot(iplot=True, figsize=(12, 8))[0][0]
fig.savefig('backtrader_plot.png')

    # Display the image in the notebook
display(Image(filename='backtrader_plot.png'))
```

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

```
Starting Portfolio Value: 10000.00
BUY EXECUTED, Price: 3.16, Size: 3096.0
SELL EXECUTED, Price: 3.15, Size: -3096.0
OPERATION PROFIT, GROSS: -50.80, NET: -70.33
BUY EXECUTED, Price: 3.22, Size: 3029.0
```

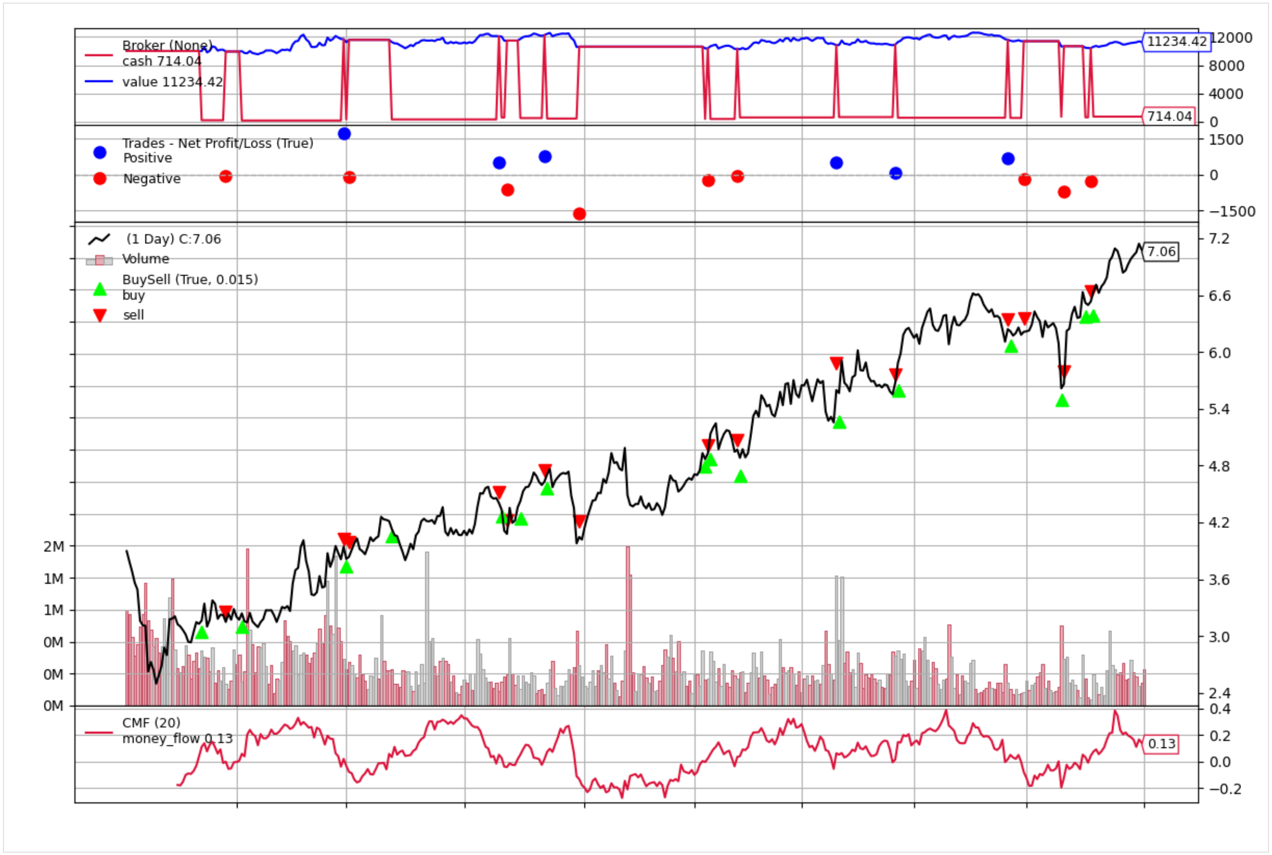
(continues on next page)

(continued from previous page)

```
SELL EXECUTED, Price: 3.81, Size: -3029.0
OPERATION PROFIT, GROSS: 1772.75, NET: 1751.46
BUY EXECUTED, Price: 3.96, Size: 2862.0
SELL EXECUTED, Price: 3.94, Size: -2862.0
OPERATION PROFIT, GROSS: -78.27, NET: -100.88
BUY EXECUTED, Price: 4.25, Size: 2648.0
SELL EXECUTED, Price: 4.45, Size: -2648.0
OPERATION PROFIT, GROSS: 526.25, NET: 503.22
BUY EXECUTED, Price: 4.34, Size: 2643.0
SELL EXECUTED, Price: 4.12, Size: -2643.0
OPERATION PROFIT, GROSS: -587.89, NET: -610.27
BUY EXECUTED, Price: 4.33, Size: 2531.0
SELL EXECUTED, Price: 4.64, Size: -2531.0
OPERATION PROFIT, GROSS: 800.63, NET: 777.92
BUY EXECUTED, Price: 4.64, Size: 2545.0
SELL EXECUTED, Price: 4.00, Size: -2545.0
OPERATION PROFIT, GROSS: -1617.09, NET: -1639.07
BUY EXECUTED, Price: 4.98, Size: 2051.0
SELL EXECUTED, Price: 4.87, Size: -2051.0
OPERATION PROFIT, GROSS: -224.37, NET: -244.58
BUY EXECUTED, Price: 4.98, Size: 2004.0
SELL EXECUTED, Price: 4.96, Size: -2004.0
OPERATION PROFIT, GROSS: -27.40, NET: -47.33
BUY EXECUTED, Price: 4.91, Size: 1977.0
SELL EXECUTED, Price: 5.18, Size: -1977.0
OPERATION PROFIT, GROSS: 522.66, NET: 502.71
BUY EXECUTED, Price: 5.55, Size: 1833.0
SELL EXECUTED, Price: 5.61, Size: -1833.0
OPERATION PROFIT, GROSS: 108.61, NET: 88.17
BUY EXECUTED, Price: 5.68, Size: 1822.0
SELL EXECUTED, Price: 6.07, Size: -1822.0
OPERATION PROFIT, GROSS: 705.91, NET: 684.51
BUY EXECUTED, Price: 6.29, Size: 1758.0
SELL EXECUTED, Price: 6.19, Size: -1758.0
OPERATION PROFIT, GROSS: -184.30, NET: -206.23
BUY EXECUTED, Price: 6.04, Size: 1770.0
SELL EXECUTED, Price: 5.65, Size: -1770.0
OPERATION PROFIT, GROSS: -685.76, NET: -706.46
BUY EXECUTED, Price: 6.66, Size: 1510.0
SELL EXECUTED, Price: 6.48, Size: -1510.0
OPERATION PROFIT, GROSS: -268.43, NET: -288.27
BUY EXECUTED, Price: 6.49, Size: 1491.0
Final Portfolio Value: 11234.42
Net Profit/Loss: 1234.42
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```


Broker (None)
cash 714.04
value 11234.42
11234.42
714.04
Trades - Net Profit/Loss (True)
Positive
Negative
(1 Day) C:7.06
Volume
BuySell (True, 0.015)
buy
sell
7.06
CMF (20)
money_flow 0.13
0.13

```
[ ]:
```

CHAPTER

FIVE

WALK FORWARD OPTIMIZATION

Walk forward optimization is a method used in quantitative trading and algorithmic trading to optimize trading strategies. It involves dividing the historical data into multiple segments to iteratively optimize and validate the trading strategy. This helps to ensure that the strategy is robust and performs well on unseen data.

5.1 Steps Involved in Walk Forward Optimization

1. **Data Partitioning:**
    - Divide the historical data into two main parts: the in-sample (training) data and the out-of-sample (testing) data.
    - The in-sample data is used to optimize the strategy, while the out-of-sample data is used to validate its performance.
2. **Optimization:**
    - Optimize the trading strategy parameters using the in-sample data.
    - This involves finding the best set of parameters that maximize the performance metric (e.g., profit, Sharpe ratio) on the in-sample data.
3. **Walk Forward Testing:**
    - Apply the optimized parameters to the out-of-sample data to test the strategy's performance.
    - This step checks if the strategy performs well on unseen data, indicating its robustness.
4. **Sliding Window:**
    - Slide the window forward by a fixed period and repeat the optimization and testing process.
    - This creates a new in-sample and out-of-sample dataset for each iteration.
5. **Aggregate Results:**
    - Aggregate the results of all the walk forward tests to evaluate the overall performance of the strategy.
    - Analyze the consistency and robustness of the strategy across different market conditions.

5.2 Example

Let's say you have 10 years of historical data. You could divide it into 5 segments of 2 years each:

1. **First Iteration:**
    - In-sample data: Year 1 to Year 4
    - Out-of-sample data: Year 5 to Year 6

2. **Second Iteration:**
    - Slide the window forward
    - In-sample data: Year 3 to Year 6
    - Out-of-sample data: Year 7 to Year 8
3. **Third Iteration:**
    - Slide the window forward again
    - In-sample data: Year 5 to Year 8
    - Out-of-sample data: Year 9 to Year 10

By repeating this process, you can optimize and test your strategy multiple times, ensuring that it performs well across different periods.

5.3 Benefits

- **Robustness**: Ensures that the strategy is not overfitted to a specific period of data.
- **Adaptability**: Helps in adapting the strategy to changing market conditions.
- **Validation**: Provides a more realistic assessment of the strategy's performance on unseen data.

5.4 Conclusion

Walk forward optimization is a powerful technique for developing robust trading strategies. It helps traders and quantitative analysts to optimize and validate their strategies across multiple periods, ensuring that they perform well in different market conditions.

CHAPTER

SIX

WHAT IS THE SHARPE RATIO?

The **Sharpe Ratio** is a financial metric used to evaluate the risk-adjusted return of an investment or portfolio. It helps investors understand how much excess return they are earning per unit of risk. The ratio is especially useful for comparing different investments or portfolios, as it accounts for both return and volatility.

The Bigger, the better!!

6.1 Formula

The Sharpe Ratio is calculated as:

$$\text{Sharpe Ratio} = \frac{R_p - R_f}{\sigma_p}$$

Where: - R_p: The average return of the portfolio or investment. - R_f: The risk-free rate of return (often represented by government bond yields). - σ_p: The standard deviation of the portfolio's return (a measure of risk or volatility).

6.2 Interpretation

- **Higher Sharpe Ratio**: Indicates a better risk-adjusted return. The investment generates higher returns for the level of risk taken.
- **Lower Sharpe Ratio**: Suggests the investment's returns are not sufficiently compensating for the risk.

6.3 Example

If a portfolio has an average return of 10%, a risk-free rate of 3%, and a standard deviation of 8%:

$$\text{Sharpe Ratio} = \frac{10\% - 3\%}{8\%} = 0.875$$

This means the portfolio earns 0.875 units of return for every unit of risk.

6.4 Applications

- **Performance Comparison**: Investors use the Sharpe Ratio to compare mutual funds, stocks, or portfolios.
- **Risk Management**: A low Sharpe Ratio may prompt reevaluation of an investment's risk-reward profile.

6.5 Limitations

- Assumes returns are normally distributed, which may not always hold true in financial markets.
- Does not account for skewness or kurtosis (asymmetric or fat-tailed distributions).
- Sensitive to the choice of the risk-free rate.

CHAPTER

SEVEN

SORTINO RATIO

The Sortino Ratio is a variation of the Sharpe Ratio that differentiates between harmful volatility and overall volatility. It measures the risk-adjusted return of an investment, considering only downside risk, which is more relevant to investors concerned with negative returns.

The Sortino Ratio is calculated as:

$$\text{Sortino Ratio} = \frac{R_p - R_f}{\sigma_d}$$

Where: - (R_p) is the expected return of the portfolio. - (R_f) is the risk-free rate of return. - (sigma_d) is the downside deviation of the portfolio returns.

Key Components: - **Expected Return ((R_p))**: The average return of the investment or portfolio. - **Risk-Free Rate ((R_f))**: The return of a theoretically risk-free investment, often represented by government bonds. - **Downside Deviation ((sigma_d))**: A measure of downside risk, focusing on returns that fall below a minimum threshold (often the risk-free rate or zero).

The Sortino Ratio provides a more accurate measure of an investment's risk-adjusted performance by penalizing only negative volatility, making it a preferred metric for investors who are primarily concerned with downside risk.

Example:

Suppose an investment portfolio has an expected return of 10%, a risk-free rate of 2%, and a downside deviation of 5%. The Sortino Ratio would be calculated as follows:

$$\text{Sortino Ratio} = \frac{0.10 - 0.02}{0.05} = 1.6$$

A higher Sortino Ratio indicates a more desirable risk-adjusted return, focusing on minimizing downside risk.

CHAPTER

EIGHT

WHAT IS REVERSE DISCOUNTED CASH FLOW (DCF)?

The **Reverse Discounted Cash Flow (DCF)** analysis is a valuation method that works backward from the current market price of a stock to infer the assumptions the market is making about the company's future cash flows. It helps investors determine whether the stock's current price is justified based on the expected growth rates and discount rates.

8.1 Traditional DCF vs. Reverse DCF

- **Traditional DCF**: Calculates the intrinsic value of a company by forecasting future cash flows and discounting them to the present value.
- **Reverse DCF**: Starts with the company's current market price and works backward to derive the implied growth assumptions in cash flows or the discount rate.

8.2 Steps in Reverse DCF

1. **Input Current Market Price**: Use the stock's current market capitalization or share price as the starting point.
2. **Forecast Cash Flows**: Assume a baseline projection of free cash flows (FCF) for future years.
3. **Apply a Discount Rate**: Choose a discount rate (cost of capital) to reflect the time value of money and risk.
4. **Solve for Growth Rate**: Adjust the cash flow growth rates iteratively until the discounted cash flows equal the current market price.

8.3 Formula

Reverse DCF uses the standard DCF formula:

$$\text{Market Price} = \sum_{t=1}^{n} \frac{FCF_t}{(1+r)^t} + \frac{TV}{(1+r)^n}$$

Where: - FCF_t: Free cash flow in year t. - r: Discount rate (cost of capital). - TV: Terminal value at the end of the forecast period. - n: Number of years in the forecast period.

In a Reverse DCF, you solve for either the implied FCF_t growth rate or the discount rate r.

8.4 Use Cases

- **Investor Insight**: Understand the growth assumptions implied by the market price.
- **Valuation Check**: Compare implied assumptions to your own analysis to decide if the stock is overvalued or undervalued.

8.5 Example

Suppose a company's stock is priced at $100, and the following inputs are used: - Projected free cash flow (FCF) for Year 1 = $10 million. - Discount rate = 8%. - Terminal value is calculated as a perpetuity.

Through Reverse DCF, you iteratively solve for the growth rate that justifies the $100 stock price.

8.6 Advantages

- Highlights the market's assumptions about growth or risk.
- Simplifies investment decisions by focusing on key inputs like growth and discount rate.

8.7 Limitations

- Relies on accurate input data, such as current cash flows and discount rates.
- Sensitive to assumptions, especially terminal value and growth rates.
- May oversimplify complex valuation scenarios.

8.8 Conclusion

Reverse DCF is a powerful tool for evaluating whether a stock's price is supported by realistic growth and risk assumptions. By working backward, investors can align market expectations with their own valuation models.

CHAPTER

NINE

REVERSE DCF ANALYSIS FOR DEUTZ AG (DEZ.DE)

9.1 Objective

This analysis works backward from the current market price of €4.03 per share to determine the implied growth rate in free cash flows (FCF) based on a risk-free rate, WACC, and other assumptions.

9.2 Assumptions

- **Current Share Price**: €4.03
- **Free Cash Flow (FCF) per Share**: €0.27/year
- **Weighted Average Cost of Capital (WACC)**: 6%
- **Terminal Growth Rate**: 2%
- **Forecast Period**: 10 years

9.3 Calculation

1. **Discounted Cash Flows**: The sum of discounted cash flows over 10 years is calculated as:

$$\text{DCF} = \sum_{t=1}^{10} \frac{FCF_t}{(1 + WACC)^t} = 5.83$$

2. **Terminal Value**: The discounted terminal value at the end of 10 years is calculated using:

$$TV = \frac{FCF_{10} \times (1 + g)}{WACC - g}$$

   The discounted terminal value for the assumed inputs contributes significantly to the final valuation.

3. **Implied Growth Rate**: Solving for the growth rate that aligns with the current share price of €4.03 yields:

$$g \approx -0.98\%$$

9.4 Results

The implied growth rate in FCF is approximately **-0.98%**, indicating that the market anticipates slightly negative growth in DEUTZ AG's cash flows over the forecast period.

9.5 Advantages

- Highlights market expectations embedded in the stock price.
- Useful for assessing whether the stock is overvalued or undervalued.

9.6 Limitations

- Sensitive to input assumptions, especially WACC and terminal growth rate.
- Assumes consistent growth in FCF, which may not reflect real-world volatility.

9.7 Conclusion

The Reverse DCF analysis suggests the current market price reflects slightly negative growth expectations for DEUTZ AG. Investors may wish to evaluate whether these assumptions are overly conservative or align with their own forecasts.

CHAPTER

TEN

STOCK TRADING AND DEEP REINFORCEMENT LEARNING

Stock trading is a complex economic behavior, and no one can give a formula to predict it accurately. This is where **Deep Reinforcement Learning (DRL)**, which has been growing rapidly in recent years, can come into play.

10.1 Deep Reinforcement Learning (DRL)

DRL combines reinforcement learning algorithms with deep neural networks to learn how to make decisions in complex environments. In DRL, an agent learns to interact with an environment by taking actions and receiving feedback in the form of rewards. The goal of the agent is to learn a policy that maximizes its cumulative reward over time.

10.2 OpenAI Gym

OpenAI Gym is a popular toolkit for developing reinforcement learning algorithms. It provides a collection of environments that simulate different scenarios for agents to learn in, such as:

- Playing Atari games,
- Controlling robots, or
- Navigating mazes.

The environments are designed to be easy to use and provide a standardized interface for agents to interact with.

Install Gym

Before starting, install the Gym library:

```
pip install gym
```

10.3 Custom Gym.Env Class

The *Gym Env* class provides methods for the agent to take actions, observe status, and receive rewards. A simple custom *Gym Env* class includes the *step* and *reset* methods:

```
import gym

class DRL4TEnv(gym.Env):
    metadata = {'render.modes': ['human']}

    def __init__(self):
        super(DRL4TEnv, self).__init__()

```

(continues on next page)

(continued from previous page)

```
    def step(self, action):
        pass

    def reset(self):
        pass
```

10.4 Class Constructor

The first step is to transfer data and other parameters into the *DRL4TEnv* class and initialize settings. **Data** is used to train the model and includes thousands of stocks, each with historical trading data and technical indicators. Each episode uses one stock to train the model.

Parameters include:

- **indicator_columns**: Columns of data used for training.
- **price_column**: Column representing stock prices.
- **sample_days**: Number of previous trading days' data observed for each step.
- **starting_balance**: Initial cash for each episode.
- **commission_rate**: Commission rate for transactions.
- **random_on_reset**: Whether a random stock is selected for training in each episode.

Example:

```
import numpy as np
import gym
from gym import spaces
import random

class DRL4TEnv(gym.Env):
    metadata = {'render.modes': ['human']}

    def __init__(self,
                 data,
                 indicator_columns=['SMARatio10', 'SMARatio20', 'MACD', 'BBP', 'CMF'],
                 price_column='Close',
                 sample_days=30,
                 starting_balance=100000,
                 commission_rate=0.001,
                 random_on_reset=True):
        super(DRL4TEnv, self).__init__()

        self.indicator_columns = indicator_columns
        self.price_column = price_column
        self.sample_days = sample_days
        self.starting_balance = starting_balance
        self.commission_rate = commission_rate
        self.random_on_reset = random_on_reset

        self.data = dict(filter(lambda item: len(item[1]) > sample_days, data.items()))
```

10.5 Reset and Step Methods

Reset Method

The *reset()* method initializes the environment for a new episode.

```
def reset(self):
    self.cash = self.starting_balance
    self.shares = 0
    return self.next_observation(Actions.Hold.value)
```

Step Method

The *step()* method executes an action, calculates the reward, and moves to the next state.

```
def step(self, action):
    balance = self.cur_balance

    self.cur_step += 1
    if self.cur_step == self.total_steps:
        self.cur_episode = self.next_episode()
        self.cur_step = self.sample_days

    self.take_action(action)

    obs = self.next_observation(action)
    reward = self.cur_balance - balance
    done = self.cur_step == self.total_steps - 1
    info = { 'Date': self.cur_data.index[self.cur_step].strftime('%Y-%m-%d'),
             'Reward': round(reward, 2),
             'Symbol': self.cur_symbol,
             'Action': Actions(action).name,
             'Shares': self.shares,
             'Close': round(self.cur_close_price, 2),
             'Cash': round(self.cash, 2),
             'Total': round(self.cur_balance, 2) }

    if done:
        self.reset()

    return obs, reward, done, info
```

10.6 Additional Methods and Usage

Actions

```
import enum

class Actions(enum.Enum):
    Hold = 0
    Buy = 1
    Sell = 2
```

Take Action

```
def take_action(self, action):
    if action == Actions.Buy.value:
        if self.shares == 0:
            price = self.cur_close_price * (1 + self.commission_rate)
            self.shares = int(self.cash / price / 100) * 100
            self.cash -= self.shares * price
    elif action == Actions.Sell.value:
        if self.shares > 0:
            price = self.cur_close_price * (1 - self.commission_rate)
            self.cash += self.shares * price
            self.shares = 0
```

Next Observation

```
def next_observation(self, action):
    observation = []
    for i in range(self.sample_days, 0, -1):
        observation = np.append(observation, self.cur_indicators.values[self.cur_step -␣
↪i + 1])
    return np.append(observation, [self.cash, self.shares * self.cur_close_price,␣
↪action])
```

Usage

To create an instance of the trading environment:

```
from drl4t_data import download

train_data, test_data = download('nyse.csv')
env = DRL4TEnv(train_data)
```

10.7 Next Steps

We will use this trading environment to train a deep reinforcement learning model.

CHAPTER

ELEVEN

CREATE CUSTOM OPENAI GYM ENVIRONMENT FOR DEEP REINFORCEMENT LEARNING (DRL4T-04)

11.1 Xiaoguang Li

Deep Reinforcement Learning (DRL) combines reinforcement learning algorithms with deep neural networks to learn how to make decisions in complex environments. In DRL, an agent learns to interact with an environment by taking actions and receiving feedback in the form of rewards. The goal of the agent is to learn a policy that maximizes its cumulative reward over time.

11.2 Install Gym

https://github.com/openai/gym

11.3 Important Notice

The team that has been maintaining Gym since 2021 has moved all future development to Gymnasium, a drop in replacement for Gym (import gymnasium as gym), and Gym will not be receiving any future updates. Please switch over to Gymnasium as soon as you're able to do so. If you'd like to read more about the story behind this switch, please check out this blog post.

11.4 https://github.com/Farama-Foundation/Gymnasium

CHAPTER

TWELVE

RELATIONSHIP BETWEEN DQN AND MDP

A **Deep Q-Network (DQN)** is a reinforcement learning algorithm designed to solve decision-making problems that are formally modeled as a **Markov Decision Process (MDP)**. Below, we detail the relationship between these two concepts.

12.1 MDP Framework

An MDP provides the theoretical structure for decision-making problems and consists of:

- **States ((S))**: A set of all possible situations in the environment.
- **Actions ((A))**: A set of possible actions an agent can take in a given state.
- **Transition Probabilities ((P(s' mid s, a)))**: The probability of transitioning to state s' from state s after taking action a.
- **Rewards ((R(s, a, s')))**: The immediate reward received for transitioning between states due to a specific action.
- **Policy ((pi(s)))**: A mapping from states to actions that guides decision-making.

The goal in an MDP is to find an optimal policy $\pi(s)$ that maximizes the expected cumulative rewards.

12.2 Role of Q-Learning in MDPs

Q-learning is a model-free algorithm used to solve MDPs by learning the **Q-function**: [Q(s, a) = text{Expected cumulative reward starting from state } s text{, taking action } a text{, and following the optimal policy thereafter.}]

The **Bellman equation** governs the update of Q-values: [Q(s, a) leftarrow Q(s, a) + alpha left[R(s, a, s') + gamma max_{a'} Q(s', a') - Q(s, a) right]] Where: - α is the learning rate. - γ is the discount factor.

12.3 Deep Q-Networks (DQNs)

A DQN extends Q-learning to handle large or high-dimensional state spaces by using a **neural network** to approximate the Q-function, $Q(s, a; \theta)$, where θ represents the network parameters.

Instead of maintaining a Q-table (which is impractical for large MDPs), DQNs use a deep learning approach to predict Q-values. The Bellman equation is used as the loss function during training: [mathcal{L}(theta) = mathbb{E} left[left(R + gamma max_{a'} Q(s', a'; theta^{-}) - Q(s, a; theta) right)^2 right]] Here: - θ^- represents the parameters of a target network to stabilize learning.

12.4 MDP Assumptions in DQNs

DQNs assume the problem adheres to the MDP framework: - The environment satisfies the **Markov property** (future states depend only on the current state and action). - Rewards and transitions guide learning toward optimal policies.

12.5 Summary

1. **MDP** provides the theoretical foundation for modeling decision-making problems.
2. **Q-learning** solves MDPs by learning the Q-function via the Bellman equation.
3. **DQN** extends Q-learning by using neural networks to approximate the Q-function, enabling the application of reinforcement learning to complex, high-dimensional MDPs.

12.6 Applications

- **Gaming**: Atari games, AlphaGo.
- **Robotics**: Navigation and control.
- **Decision Systems**: Resource optimization, dynamic pricing.

PROXIMAL POLICY OPTIMIZATION

PPO is classified as a policy gradient method for training an agent's policy network. This network approximates a policy function, used by the agent to make decisions. Essentially, to train the policy function, PPO takes a small policy update step, so the agent can reliably reach the optimal solution.

Consequently, PPO implements a clip function that constrains the policy update of an agent from being too large, so that larger step sizes may be used without negatively affecting the gradient ascent process.

https://en.wikipedia.org/wiki/Proximal_policy_optimization

CHAPTER

FOURTEEN

MARKOV DECISION PROCESS

A **Markov Decision Process (MDP)** is a mathematical framework used for modeling decision-making in scenarios where outcomes are influenced by both randomness and a decision-maker's actions.

14.1 Components

An MDP consists of the following elements:

- **States (S):** A finite set of states representing all possible situations in the system.
- **Actions (A):** A finite set of actions available to the decision-maker in each state.
- **Transition Probabilities (P):** The probability $P(s' \mid s, a)$ of transitioning to state s' from state s after taking action a.
- **Rewards (R):** The immediate reward $R(s, a, s')$ received after transitioning from state s to s' due to action a.
- **Discount Factor:** γ is a factor between 0 and 1 that determines the importance of future rewards.

14.2 Goal

The objective of solving an MDP is to determine a **policy** $\pi(s)$, which maps states to actions, to maximize the expected cumulative reward over time.

14.3 Example

Consider a robot navigating a grid:

- **States:** Each grid cell is a state.
- **Actions:** The robot can move up, down, left, or right.
- **Rewards:** +10 for reaching the goal cell, -1 for each step taken.

By solving the MDP, the robot identifies the optimal policy π to maximize its rewards.

CHAPTER FIFTEEN

DEEP Q-NETWORK (DQN)

Deep Q-Network (DQN) is a reinforcement learning algorithm that uses a deep neural network to approximate the Q-value function. The Q-value function is a function that receives a status-action pair and outputs a value representing the expected cumulative reward for taking that action from that status, and continues playing following the optimal policy.

The basic idea behind DQN is to use a deep neural network to estimate the Q-value function, rather than using a table, which is not feasible for large and continuous status space. DQN also uses techniques such as experience replay and separate target network to optimize and stabilize the learning process.

CHAPTER SIXTEEN

LINKS

16.1 youtube links:

- The Most Important Algorithm in Machine Learning (Artem Kirsanov)

https://www.youtube.com/watch?v=SmZmBKc7Lrs&t=1898s

- Reinforcement Learning from scratch (Graphics in 5 Minutes)

https://www.youtube.com/watch?v=vXtfdGphr3c

- L1 MDPs, Exact Solution Methods, Max-ent RL (Foundations of Deep RL Series) (Pieter Abbeel)

https://www.youtube.com/watch?v=2GwBez0D20A

- The spelled-out intro to neural networks and backpropagation: building micrograd (Andrej Karpathy)

https://www.youtube.com/watch?v=VMj-3S1tku0&t=3502s

16.2 control theory

PID Control - A brief introduction (Brian Douglas)

https://www.youtube.com/watch?v=UR0hOmjaHp0&t=14s

16.3 algorithmic trading

https://github.com/ThibautTheate/An-Application-of-Deep-Reinforcement-Learning-to-Algorithmic-Trading

CHAPTER SEVENTEEN

FURTHER READING :

There is a paper on DRL, which I found readable and understandable :

- An Application of Deep Reinforcement Learning to Algorithmic Trading

Thibaut Théate, Damien Ernst, Montefiore Institute, University of Liège (Allée de la découverte 10, 4000 Liège, Belgium)

CHAPTER EIGHTEEN

REMARKS:

- You can feed a few technical indicators into a DRL model, but … there is of course much more that determines the price-formation of a single stock.
- trading costs is something that impacts me as an individual investor, too much trading may affect the end result.

CHAPTER

NINETEEN

INDICES AND TABLES

- genindex
- modindex
- search