In [ ]:
```
import pandas as pd
from datetime import datetime, timedelta
```

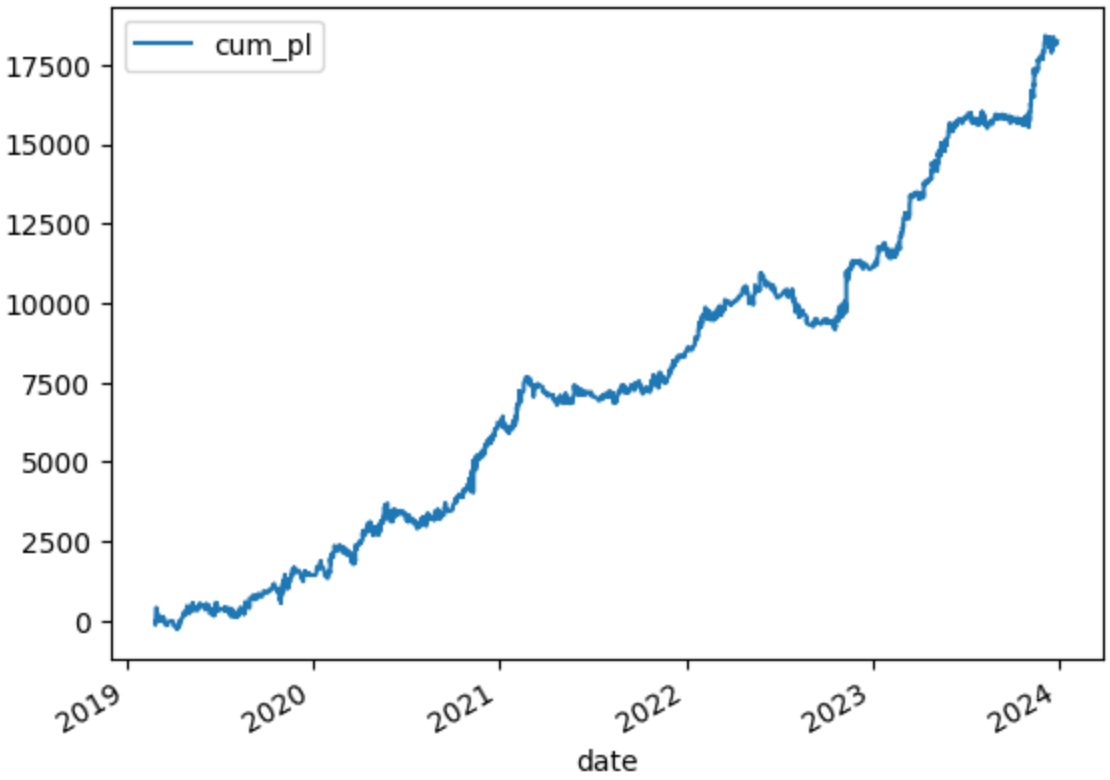
In [ ]:
```
results_file = "/Users/vsai23/Workspace/PolygonBacktest/results/ep/targets_2
df = pd.read_csv(results_file)
rs = 0
winners = 0
tradesize=50
```

In [ ]:
```
df.sort_values('date', inplace=True)
df['pl'] = None
```

In [ ]:
```
for current_index in range(len(df)):
        row = df.iloc[current_index]
        _,symbol,date,stop,targets,trades,_ = row

        # split the trades into a list of tuples
        trades = trades[2:-2].split('), (')
        trades = [trade.split(',') for trade in trades]
        trades = [(trade[0], float(trade[1]), float(trade[2])) for trade in

        r = trades[0][1] - stop
        r = max(r, 0.01)

        total = 0
        # compute the r frm the rest of the trades
        for trade in trades[1:]:
            total += (trade[1] - trades[0][1])*(-trade[2])

        if total >= 0:
            winners += 1
        rs += total/r

        pl = tradesize*(total/r)

        # append the pl to the row
        df.at[current_index, 'pl'] = pl
```

In [ ]:
```
df['date'] = pd.to_datetime(df['date'])

# cumulative pl
df['cum_pl'] = df['pl'].cumsum()

# plot the cumulative pl
df.plot(x='date', y='cum_pl')
```

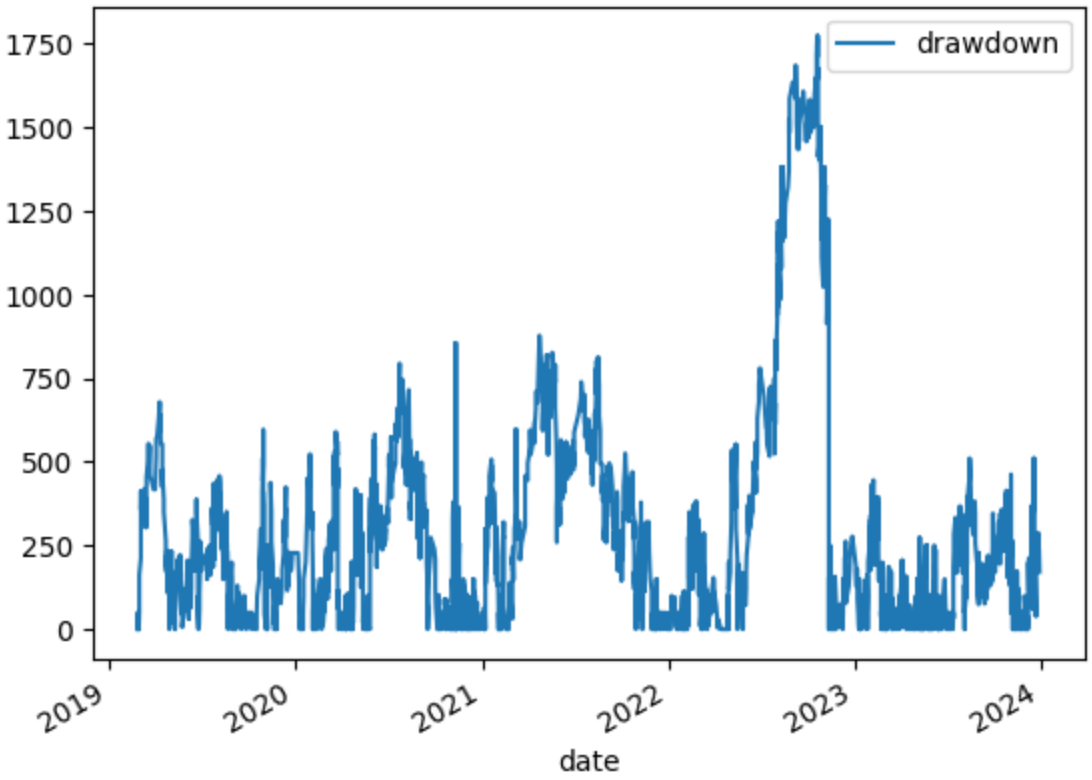
Out[ ]:
```
<Axes: xlabel='date'>
```



In [ ]:
```
# Calculate the maximum drawdown
df['drawdown'] = df['cum_pl'].cummax() - df['cum_pl']
max_drawdown = df['drawdown'].max()

print("Maximum Drawdown:", max_drawdown)

df.plot(x='date', y='drawdown')
```

```
Maximum Drawdown: 1771.5337165713663
```

Out[ ]: <Axes: xlabel='date'>



In [ ]:
```
# Calculate the average return
average_return = df['pl'].mean()

# Calculate the standard deviation of the returns
std_dev = df['pl'].std()

# Calculate the Sharpe ratio
sharpe_ratio = average_return / std_dev

print("Sharpe Ratio Per Trade:", sharpe_ratio)
```

```
Sharpe Ratio Per Trade: 0.09977412387952923
```

In [ ]:
```
weekly_pl = df.resample('W', on='date')['pl'].sum()

average_weekly_pl = weekly_pl.mean()

std_dev_weekly_pl = weekly_pl.std()

sharpe_ratio_weekly_pl = average_weekly_pl / std_dev_weekly_pl

print("Sharpe Ratio Per Week:", sharpe_ratio_weekly_pl)
```

```
Sharpe Ratio Per Week: 0.352996009158126
```

In [ ]:
```
monthly_pl = df.resample('M', on='date')['pl'].sum()

average_monthly_pl = monthly_pl.mean()

std_dev_monthly_pl = monthly_pl.std()

sharpe_ratio_monthly_pl = average_monthly_pl / std_dev_monthly_pl
```

```
print("Sharpe Ratio Per Month:", sharpe_ratio_monthly_pl)
```

```
Sharpe Ratio Per Month: 0.5868684918627847
```

In [ ]:
```
# pull the start and end dates from the df
start_date = df['date'].iloc[0]
end_date = df['date'].iloc[-1]

# import the polygon client
from clients.polygon import PolygonClient
# create a polygon client
polygon = PolygonClient()

# get the SPY data from polygon
spy_data = polygon.convert_aggs(polygon.get_stock_data("SPY", start_date, er
```

In [ ]:
```
# Calculate the monthly returns
monthly_returns = df.resample('M', on='date')['pl'].sum()
monthly_portfolio_value = monthly_returns.cumsum() + tradesize*100

# calculate the monthly returns for the monthly portfolio value
monthly_portfolio_returns = monthly_portfolio_value.pct_change()

# calculate the monthly returns for SPY
spy_monthly_returns = spy_data.resample('M', on='time')['close'].last().pct_
```

In [ ]:
```
# Calculate the covariance between stock returns and market returns
covariance = monthly_portfolio_returns.cov(spy_monthly_returns)

# Calculate the variance of market returns
variance = spy_monthly_returns.var()

# Calculate the beta
beta = covariance / variance

print("Beta:", beta)
```

```
Beta: 0.16726292495076264
```

In [ ]:
```
from numpy import NaN


risk_free_rate = 0.0481
market_return = 0.07

# calculate the df annual returns
annual_returns = df.resample('Y', on='date')['pl'].sum()

# add an extra 0 at the front of annual returns
extra_row = pd.Series(0, index=[annual_returns.index[0] - pd.DateOffset(year
annual_returns = pd.concat([extra_row, annual_returns])

# calculate the cumusum
annual_returns = annual_returns.cumsum() + tradesize*100
```

```
# calculate the percentage annual returns
annual_returns = annual_returns.pct_change()

# calculate the alpha for the portfolio per year
alpha = annual_returns - (risk_free_rate + beta * (market_return - risk_free

for index, value in alpha.items():
    if str(value) != 'nan':
        print("Alpha for {}: {}".format(index.year, value))
```

```
Alpha for 2019: 0.2363491559172408
Alpha for 2020: 0.6936679456283172
Alpha for 2021: 0.1444218163332822
Alpha for 2022: 0.14255988428176367
Alpha for 2023: 0.39500603301064485
```

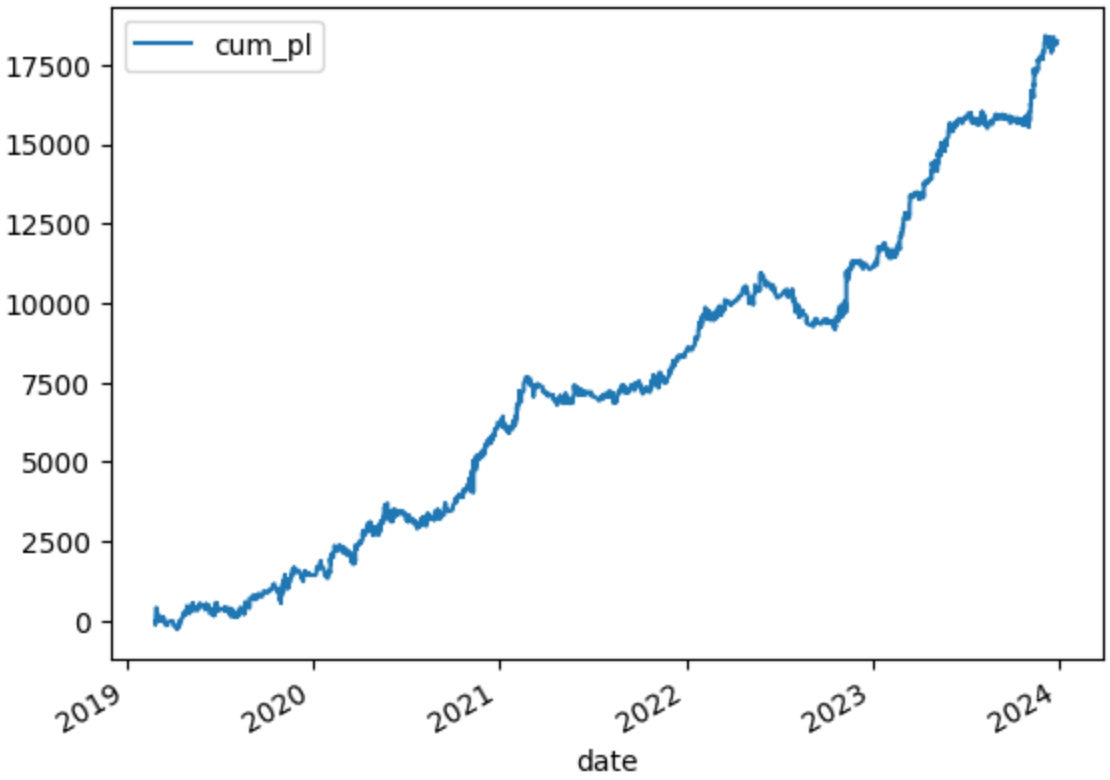
In [ ]:

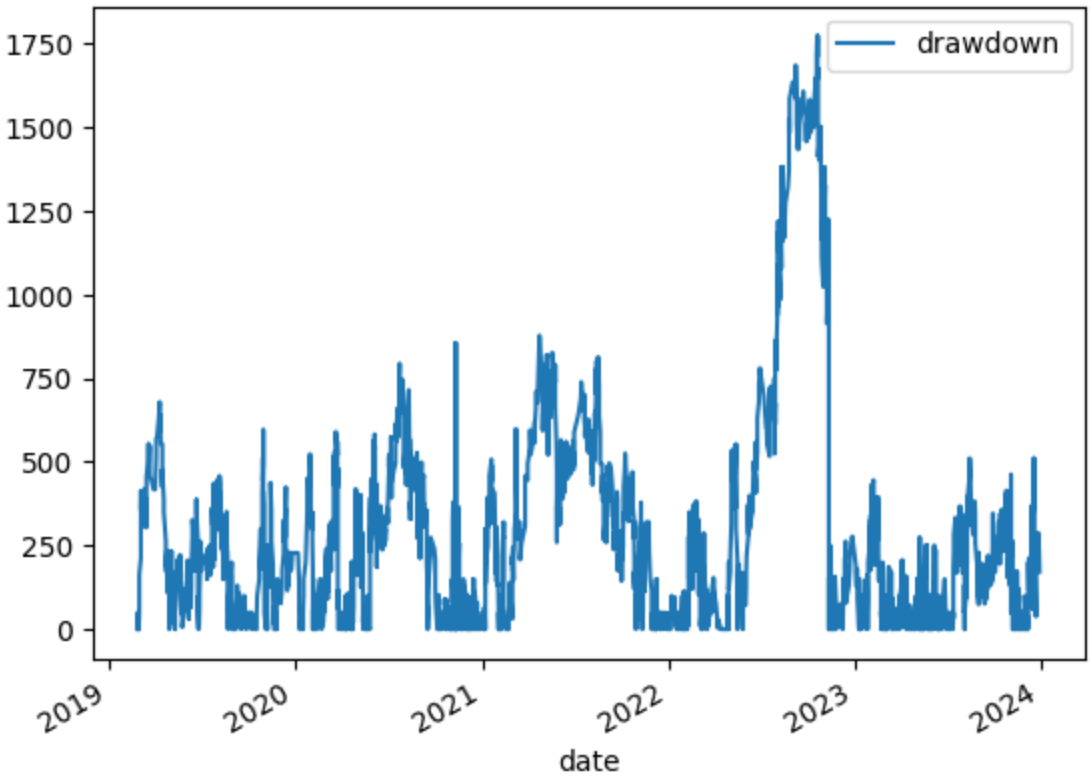
```
from io import BytesIO
from reportlab.lib.pagesizes import letter
from reportlab.pdfgen import canvas
from reportlab.lib import colors
import matplotlib.pyplot as plt
import os
from reportlab.lib import colors
from reportlab.platypus import Table, TableStyle

# Create a new PDF document
pdf_buffer = BytesIO()
pdf = canvas.Canvas(pdf_buffer, pagesize=letter)

# Set the font to Times-Roman (a similar serif font) and a standard font siz
pdf.setFont("Times-Roman", 12)

# Headings will use a larger font size
heading_font_size = 14

# Parse the strategy and start and end dates from the filename
filename_parts = results_file.split('/')[-1].split('.')[0].split('_')
exit, start_date, end_date = filename_parts[0], filename_parts[1], filename_

strategy = results_file.split('/')[-2]

# Add a title to the PDF
pdf.setFont("Times-Roman", heading_font_size)
pdf.drawString(50, 770, "{} from {} to {} using exit {} with trade risk ${}"
pdf.setFont("Times-Roman", 12)  # Reset to standard font size

# Section for Win Rate, Rs, and Total Trades
pdf.drawString(50, 740, "Win Rate: {:.2f}%".format(winners/len(df)*100 if le
pdf.drawString(50, 720, "R per trade: {:.2f}".format(rs/len(df)))
pdf.drawString(50, 700, "Total Trades: {}".format(len(df)))

# Add cum pl graph
cum_pl_graph_path = "cum_pl_graph.png"
df.plot(x='date', y='cum_pl').get_figure().savefig(cum_pl_graph_path)
pdf.drawImage(cum_pl_graph_path, 150, 530, width=350, height=150)  # Adjust
```

```
# Section for Max Drawdown, Sharpe Ratio, and Beta
pdf.drawString(50, 500, "Max Drawdown: {:.2f}".format(max_drawdown))

# Add max drawdown graph
max_drawdown_graph_path = "max_drawdown_graph.png"
df.plot(x='date', y='drawdown').get_figure().savefig(max_drawdown_graph_path
pdf.drawImage(max_drawdown_graph_path, 150, 310, width=350, height=150)  # A

# Add Sharpe Ratios
sharpe_data = [
    ["Period", "Sharpe"],
    ["Daily", "{:.2f}".format(sharpe_ratio)],
    ["Weekly", "{:.2f}".format(sharpe_ratio_weekly_pl)],
    ["Monthly", "{:.2f}".format(sharpe_ratio_monthly_pl)]
]

sharpe_table = Table(sharpe_data, colWidths=[100, 100])
sharpe_table.setStyle(TableStyle([
    ('BACKGROUND', (0, 0), (-1, 0), colors.grey),
    ('TEXTCOLOR', (0, 0), (-1, 0), colors.whitesmoke),
    ('ALIGN', (0, 0), (-1, -1), 'CENTER'),
    ('FONTNAME', (0, 0), (-1, -1), 'Times-Roman'),
    ('FONTSIZE', (0, 0), (-1, -1), 12),
    ('BOTTOMPADDING', (0, 0), (-1, -1), 12),
    ('BACKGROUND', (0, 1), (-1, -1), colors.beige),
]))
sharpe_table.wrapOn(pdf, 500, 400)
sharpe_table.drawOn(pdf, 50, 100)

# Section for Alpha in a table
alpha_data = [["Year", "Alpha"]]
for index, value in alpha.items():
    if str(value) != 'nan':
        alpha_data.append([index.year, "{:.2f}".format(value)])

alpha_table = Table(alpha_data, colWidths=[100, 100])
alpha_table.setStyle(TableStyle([
    ('BACKGROUND', (0, 0), (-1, 0), colors.grey),
    ('TEXTCOLOR', (0, 0), (-1, 0), colors.whitesmoke),
    ('ALIGN', (0, 0), (-1, -1), 'CENTER'),
    ('FONTNAME', (0, 0), (-1, -1), 'Times-Roman'),
    ('FONTSIZE', (0, 0), (-1, -1), 12),
    ('BOTTOMPADDING', (0, 0), (-1, -1), 12),
    ('BACKGROUND', (0, 1), (-1, -1), colors.beige),
]))
alpha_table.wrapOn(pdf, 500, 400)
alpha_table.drawOn(pdf, 300, 100)

# Save the PDF document
pdf.save()

# Retrieve the PDF bytes
pdf_bytes = pdf_buffer.getvalue()

# Close the buffer
pdf_buffer.close()
```

```
# Write the PDF to a file
final_report_path = "final_report.pdf"
with open(final_report_path, "wb") as f:
    f.write(pdf_bytes)

# Clean up temporary files
os.remove(cum_pl_graph_path)
os.remove(max_drawdown_graph_path)

# Final report path
final_report_path
```

Out[ ]: 'final_report.pdf'




drawdown
1750
1500
1250
1000
750
500
250
0
2019
2020
2021
2022
2023
2024
date