```
!pip install optuna
import yfinance as yf
import pandas as pd
import numpy as np
import optuna
import io
from contextlib import redirect_stdout
!pip install gradio --quiet
import gradio as gr
from datetime import datetime, timedelta
```

```
Collecting optuna
  Downloading optuna-4.5.0-py3-none-any.whl.metadata (17 kB)
Collecting alembic>=1.5.0 (from optuna)
  Downloading alembic-1.16.4-py3-none-any.whl.metadata (7.3 kB)
Collecting colorlog (from optuna)
  Downloading colorlog-6.9.0-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: numpy in
/usr/local/lib/python3.12/dist-packages (from optuna) (2.0.2)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.12/dist-packages (from optuna) (25.0)
Requirement already satisfied: sqlalchemy>=1.4.2 in
/usr/local/lib/python3.12/dist-packages (from optuna) (2.0.43)
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-
packages (from optuna) (4.67.1)
Requirement already satisfied: PyYAML in
/usr/local/lib/python3.12/dist-packages (from optuna) (6.0.2)
Requirement already satisfied: Mako in /usr/lib/python3/dist-packages
(from alembic>=1.5.0->optuna) (1.1.3)
Requirement already satisfied: typing-extensions>=4.12 in
/usr/local/lib/python3.12/dist-packages (from alembic>=1.5.0->optuna)
(4.14.1)
Requirement already satisfied: greenlet>=1 in
/usr/local/lib/python3.12/dist-packages (from sqlalchemy>=1.4.2-
>optuna) (3.2.4)
Downloading optuna-4.5.0-py3-none-any.whl (400 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 400.9/400.9 kB 7.1 MB/s eta
0:00:00
bic-1.16.4-py3-none-any.whl (247 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 247.0/247.0 kB 18.6 MB/s eta
0:00:00
bic, optuna
Successfully installed alembic-1.16.4 colorlog-6.9.0 optuna-4.5.0
```

```
def compute_rsi(series, period):
    delta = series.diff()
    gain = (delta.where(delta > 0, 0)).rolling(window=period).mean()
    loss = (-delta.where(delta < 0, 0)).rolling(window=period).mean()
    rs = gain / loss
    rsi = 100 - (100 / (1 + rs))
```

```
    return rsi


def backtest_strategy(data, short_ma, long_ma, rsi_period,
rsi_overbought, period, position_size=1):
    data['short_ma'] = data['Close'].rolling(short_ma).mean()
    data['long_ma'] = data['Close'].rolling(long_ma).mean()
    data['rsi'] = compute_rsi(data['Close'], rsi_period)
    data['signal'] = (data['short_ma'] > data['long_ma']) &
(data['rsi'] < rsi_overbought)
    data['position'] =
data['signal'].shift(1).astype(bool).fillna(False)

    trades = []
    in_trade = False
    entry_price = None
    entry_date = None
    open_position = None

    for i in range(1, len(data)):
        if data['position'].iloc[i] and not in_trade:
            entry_price = data['Close'].iloc[i]
            entry_date = data.index[i] if isinstance(data.index,
pd.DatetimeIndex) else data['Date'].iloc[i]
            in_trade = True
        elif (not data['position'].iloc[i] and in_trade):
        #elif (not data['position'].iloc[i] and in_trade) or (in_trade
and data['Close'].iloc[i]>=entry_price*1.02) or (in_trade and
data['Close'].iloc[i]<=entry_price*0.99):
            exit_price = data['Close'].iloc[i]
            exit_date = data.index[i] if isinstance(data.index,
pd.DatetimeIndex) else data['Date'].iloc[i]
            trades.append({
                'entry_date': entry_date.strftime('%Y-%m-%d %H:%M'),
                'exit_date': exit_date.strftime('%Y-%m-%d %H:%M'),
                'entry_price': float(entry_price),
                'exit_price': float(exit_price),
                'pnl': (float(exit_price) - float(entry_price)) *
position_size
            })
            in_trade = False

    # Check for open position at the end
    if in_trade:
        open_position = {
            'entry_date': entry_date.strftime('%Y-%m-%d %H:%M'),
            'exit_date': None,
            'entry_price': float(entry_price),
            'exit_price': None,
            'pnl': None
```

```
        }
        trades.append(open_position)
        print("\nOpen position:")
        print(open_position)
    else:
        print("\nNo open position.")

    print(f"\nNumber of Trades: {len(trades)}")
    print("Trade List:")
    for trade in trades:
        print(trade)

    # Calculate minimum equity ever needed
    running_balance = 0.0
    min_equity_needed = 0.0

    for trade in trades:
        if trade['exit_price'] is not None:  # Only closed trades
affect running balance
            running_balance -= trade['entry_price'] * position_size
            if running_balance < min_equity_needed:
                min_equity_needed = running_balance
            running_balance += trade['exit_price'] * position_size

    min_equity_needed = abs(min_equity_needed) if min_equity_needed <
0 else 0.0001
    print(f"\nMinimum Equity Ever Needed: {min_equity_needed:.2f}")

    # Calculate total P&L (only for closed trades)
    total_pnl = sum(trade['pnl'] for trade in trades if trade['pnl']
is not None)
    print(f"\nTotal P&L: {total_pnl:.2f}")
    if period == '3mo':
      a=18
    elif period == '6mo':
      a=36
    elif period == '1y':
      a=73
    print(f"End result:
{round((total_pnl+min_equity_needed)/min_equity_needed,2)} in the last
{a} days.")
    return round(total_pnl,2)



def backtest(data, short_ma, long_ma, rsi_period, rsi_overbought,
position_size=1):
    data['short_ma'] = data['Close'].rolling(short_ma).mean()
    data['long_ma'] = data['Close'].rolling(long_ma).mean()
    data['rsi'] = compute_rsi(data['Close'], rsi_period)
```

```
    data['signal'] = (data['short_ma'] > data['long_ma']) &
(data['rsi'] < rsi_overbought)
    data['position'] =
data['signal'].shift(1).astype(bool).fillna(False)

    trades = []
    in_trade = False
    entry_price = None
    entry_date = None
    open_position = None

    for i in range(1, len(data)):
        if data['position'].iloc[i] and not in_trade:
            entry_price = data['Close'].iloc[i]
            entry_date = data.index[i] if isinstance(data.index,
pd.DatetimeIndex) else data['Date'].iloc[i]
            in_trade = True
        elif (not data['position'].iloc[i] and in_trade):
        #elif (not data['position'].iloc[i] and in_trade) or (in_trade
and data['Close'].iloc[i]>=entry_price*1.02) or (in_trade and
data['Close'].iloc[i]<=entry_price*0.99):
            exit_price = data['Close'].iloc[i]
            exit_date = data.index[i] if isinstance(data.index,
pd.DatetimeIndex) else data['Date'].iloc[i]
            trades.append({
                'entry_date': entry_date,
                'exit_date': exit_date,
                'entry_price': entry_price,
                'exit_price': exit_price,
                'pnl': (exit_price - entry_price) * position_size
            })
            in_trade = False

    # Check for open position at the end
    if in_trade:
        open_position = {
            'entry_date': entry_date,
            'exit_date': None,
            'entry_price': entry_price,
            'exit_price': None,
            'pnl': None
        }
        trades.append(open_position)

    # Calculate minimum equity ever needed
    running_balance = 0.0
    min_equity_needed = 0.0

    for trade in trades:
        if trade['exit_price'] is not None:  # Only closed trades
```

```
affect running balance
            running_balance -= trade['entry_price'] * position_size
            if running_balance < min_equity_needed:
                min_equity_needed = running_balance
            running_balance += trade['exit_price'] * position_size

    # Calculate total P&L (only for closed trades)
    total_pnl = sum(trade['pnl'] for trade in trades if trade['pnl']
is not None)
    return round(total_pnl,2)


def objective(trial, data):
    short_ma = trial.suggest_int("short_ma", 3, 50)  #1 100
    long_ma = trial.suggest_int("long_ma", 50, 300)     #1 500
    rsi_period = trial.suggest_int("rsi_period", 10, 30)   #1 200
    rsi_overbought = trial.suggest_int("rsi_overbought", 60, 90)   #1
100
    return backtest(data.copy(), short_ma, long_ma, rsi_period,
rsi_overbought)


def operation(period, trial, interval='1h'):
    period = period

    data = yf.download(ticker, period=period, interval=interval,
prepost=True)

    data.index = data.index.tz_convert('America/Toronto')
    data = data['Close'].round(4)


    data = data.reset_index() # Move the datetime index to a column
    data.columns = ['Date', 'Close'] # Rename the columns


    data['Close'] = pd.to_numeric(data['Close'], errors='coerce')


    split_ratio = 0.8  # 80% train, 20% test
    split_index = int(len(data) * split_ratio)
    train_data = data.iloc[:split_index].copy()
    test_data = data.iloc[split_index:].copy()


    study = optuna.create_study(direction="maximize",
pruner=optuna.pruners.MedianPruner())
    study.optimize(lambda x: objective(x,train_data), n_trials=trial)

    f = io.StringIO()
    with redirect_stdout(f):
```

```
        print('\n'+ticker)
        print(period)
        print(interval)
        print(study.best_params)

        a = backtest_strategy(
            test_data.copy(),
            study.best_params['short_ma'],
            study.best_params['long_ma'],
            study.best_params['rsi_period'],
            study.best_params['rsi_overbought'],
            period
        )

    output = f.getvalue()

    return output

def greet(ticker_input):

    tickers = [tick.strip().upper() for tick in
ticker_input.split(',')]
    reports = []


    for ticker in tickers:
      globals()['ticker'] = ticker

      report1 = operation(period='3mo', trial=75)
      report2 = operation(period='6mo', trial=125)
      report3 = operation(period='1y', trial=200)


      report = report1 + '\n' + report2 + '\n' + report3

      reports.append(report)

    return '\n\n'.join(reports)

import gradio as gr

# Your custom logic for generating reports
def greet(ticker_input):
    tickers = [tick.strip().upper() for tick in
ticker_input.split(',')]
    reports = []

    for ticker in tickers:
        globals()['ticker'] = ticker

        report1 = operation(period='3mo', trial=75)
```

```
        report2 = operation(period='6mo', trial=125)
        report3 = operation(period='1y', trial=200)

        report = report1 + '\n' + report2 + '\n' + report3
        reports.append(report)

    return '\n\n'.join(reports)

# Gradio UI
with gr.Blocks() as demo:
    gr.Markdown("## Ticker Report Generator")
    ticker_input = gr.Textbox(label="Enter Tickers (comma-separated)")
    ticker_output = gr.TextArea(label="Report", lines=10,
max_lines=30)
    ticker_input.submit(greet, inputs=ticker_input,
outputs=ticker_output)

demo.launch()
```

```
It looks like you are running Gradio on a hosted Jupyter notebook,
which requires `share=True`. Automatically setting `share=True` (you
can turn this off by setting `share=False` in `launch()` explicitly).

Colab notebook detected. To show errors in colab notebook, set
debug=True in launch()
* Running on public URL: https://df1bf921795ee8b323.gradio.live

This share link expires in 1 week. For free permanent hosting and GPU
upgrades, run `gradio deploy` from the terminal in the working
directory to deploy to Hugging Face Spaces
(https://huggingface.co/spaces)

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