analysis

June 24, 2025

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

import matplotlib.pyplot as plt
```

1 Trade Analysis of BTC

Ilyas Ustun
June 15, 2025
Chicago, IL

1.1 Table of Contents

[2]:
```
# Read data from the data folder
# Assuming CSV file - adjust file path and read function as needed
file_path = 'data/BTC_USDT Perpetual Bybit V5, Time - Time - 15m, 1_1_2024␣
 ↪120000 AM-12_31_2024 120000 AM_3c007f81-be0d-41a8-a36c-1ed6d0b7646f.csv'  #␣
 ↪Replace with your actual file name
df = pd.read_csv(file_path, sep=";")

# Display the first few rows of the data to verify it was loaded correctly
df.head()
```

[2]:

	Close	High	Low	Median	Open	Time left
0	42557.0	42581.2	42495.5	42538.35	42531.7	2024-01-01 08:00:00.000
1	42525.5	42587.3	42515.0	42551.15	42557.0	2024-01-01 08:15:00.000

```
2  42534.5  42550.2  42492.7  42521.45  42525.5  2024-01-01 08:30:00.000
3  42596.1  42599.6  42525.3  42562.45  42534.5  2024-01-01 08:45:00.000
4  42614.7  42618.7  42574.1  42596.40  42596.1  2024-01-01 09:00:00.000

        Time right       Typical   Volume  Quote asset volume   Weighted  \
0  08.14.59.999999  42544.566667  647.627        2.755070e+07  42547.675
1  08.29.59.999999  42542.600000  185.753        7.903909e+06  42538.325
2  08.44.59.999999  42525.800000  164.984        7.015146e+06  42527.975
3  08.59.59.999999  42573.666667  276.168        1.175917e+07  42579.275
4  09.14.59.999999  42602.500000  334.340        1.424297e+07  42605.550

   Unnamed: 11
0          NaN
1          NaN
2          NaN
3          NaN
4          NaN
```

2 Data Exploration and Preprocessing

This notebook analyzes Bitcoin (BTC/USDT) perpetual futures data from Bybit for the entire year 2024, using 15-minute intervals.

2.1 Dataset Overview

- **Time Period:** January 1, 2024 to December 31, 2024
- **Interval:** 15-minute candlesticks
- **Data Points:** 35,059 entries
- **Key Features:** Open, High, Low, Close prices, Volume, and derived technical indicators

2.2 Technical Indicators

- **EMA (10):** 10-period Exponential Moving Average of closing prices
- **ATR (10):** 10-period Average True Range, measuring market volatility

The dataset provides a comprehensive view of Bitcoin price movements throughout the year, allowing for detailed technical analysis and pattern identification.

2.3 Data Cleaning and Initial Preprocessing

Before proceeding with analysis, I'll clean the dataset by:

1. **Removing unused columns**: The "Unnamed: 11" column contains only missing values and will be dropped.
2. **Handling time-related columns**: The dataset includes both "Time left" (now the index) and "Time right" columns. The "Time right" column contains only the time portion of the timestamp and is redundant since we're using "Time left" as our datetime index.

The cleaned dataset will include: - **Price data**: Open, High, Low, Close, Median, Typical, Weighted - **Volume data**: Volume and Quote asset volume - **Technical indicators**: EMA_10 (10-period Exponential Moving Average) and ATR_10 (10-period Average True Range)

This cleaning ensures we work with only relevant data for our analysis and visualization in subsequent steps.

[3]:
```
# Duplicate rows in the data
df.loc[df.duplicated(subset=['Time left'], keep=False)]
```

[3]:

	Close	High	Low	Median	Open	Time left
999	46528.4	46587.4	46208.0	46397.70	46208.1	2024-01-11 17:45:00.000
1000	46528.4	46587.4	46208.0	46397.70	46208.1	2024-01-11 17:45:00.000
1999	41152.8	41225.4	41121.0	41173.20	41190.4	2024-01-22 03:30:00.000
2000	41152.8	41225.4	41121.0	41173.20	41190.4	2024-01-22 03:30:00.000
2999	42159.9	42173.4	41965.1	42069.25	42002.1	2024-02-01 13:15:00.000
3000	42159.9	42173.4	41965.1	42069.25	42002.1	2024-02-01 13:15:00.000
3999	48052.0	48183.7	48044.9	48114.30	48154.5	2024-02-11 23:00:00.000
4000	48052.0	48183.7	48044.9	48114.30	48154.5	2024-02-11 23:00:00.000
4999	51911.7	51953.5	51815.7	51884.60	51833.8	2024-02-22 08:45:00.000
5000	51911.7	51953.5	51815.7	51884.60	51833.8	2024-02-22 08:45:00.000
5999	62882.4	62944.8	62654.6	62799.70	62763.1	2024-03-03 18:30:00.000
6000	62882.4	62944.8	62654.6	62799.70	62763.1	2024-03-03 18:30:00.000
6999	73384.4	73639.9	73365.3	73502.60	73623.6	2024-03-14 04:15:00.000
7000	73384.4	73639.9	73365.3	73502.60	73623.6	2024-03-14 04:15:00.000
7999	65564.5	65564.5	65319.8	65442.15	65369.4	2024-03-24 14:00:00.000
8000	65564.5	65564.5	65319.8	65442.15	65369.4	2024-03-24 14:00:00.000
8999	65971.5	66185.0	65947.7	66066.35	66174.7	2024-04-03 23:45:00.000
9000	65971.5	66185.0	65947.7	66066.35	66174.7	2024-04-03 23:45:00.000
9999	64059.9	64430.0	63875.3	64152.65	64320.1	2024-04-14 09:30:00.000
10000	64059.9	64430.0	63875.3	64152.65	64320.1	2024-04-14 09:30:00.000
27982	68856.1	68900.6	68734.1	68817.35	68788.8	2024-10-18 17:00:00.000
27983	68856.1	68900.6	68734.1	68817.35	68788.8	2024-10-18 17:00:00.000
28982	71192.0	71435.6	71140.9	71288.25	71378.5	2024-10-29 02:45:00.000
28983	71192.0	71435.6	71140.9	71288.25	71378.5	2024-10-29 02:45:00.000
29982	76108.5	76210.0	75994.5	76102.25	76035.0	2024-11-08 12:30:00.000
29983	76108.5	76210.0	75994.5	76102.25	76035.0	2024-11-08 12:30:00.000
30982	91344.6	91364.9	91176.0	91270.45	91296.3	2024-11-18 22:15:00.000
30983	91344.6	91364.9	91176.0	91270.45	91296.3	2024-11-18 22:15:00.000
31982	95864.4	95983.8	95700.1	95841.95	95702.4	2024-11-29 08:00:00.000
31983	95864.4	95983.8	95700.1	95841.95	95702.4	2024-11-29 08:00:00.000
32982	97398.7	97908.5	97350.6	97629.55	97661.5	2024-12-09 17:45:00.000
32983	97398.7	97908.5	97350.6	97629.55	97661.5	2024-12-09 17:45:00.000
33982	96985.6	97509.5	96961.9	97235.70	97471.3	2024-12-20 03:30:00.000
33983	96985.6	97509.5	96961.9	97235.70	97471.3	2024-12-20 03:30:00.000
34982	93262.7	93454.4	93165.8	93310.10	93342.4	2024-12-30 13:15:00.000
34983	93262.7	93454.4	93165.8	93310.10	93342.4	2024-12-30 13:15:00.000

	Time right	Typical	Volume	Quote asset volume	Weighted	\
999	05.59.59.999999	46441.266667	3922.904	1.819599e+08	46463.050	
1000	05.59.59.999999	46441.266667	3922.904	1.819599e+08	46463.050	
1999	03.44.59.999999	41166.400000	519.074	2.136900e+07	41163.000	
2000	03.44.59.999999	41166.400000	519.074	2.136900e+07	41163.000	
2999	01.29.59.999999	42099.466667	1335.922	5.621038e+07	42114.575	
3000	01.29.59.999999	42099.466667	1335.922	5.621038e+07	42114.575	
3999	11.14.59.999999	48093.533333	533.988	2.568420e+07	48083.150	
4000	11.14.59.999999	48093.533333	533.988	2.568420e+07	48083.150	
4999	08.59.59.999999	51893.633333	758.704	3.937959e+07	51898.150	
5000	08.59.59.999999	51893.633333	758.704	3.937959e+07	51898.150	
5999	06.44.59.999999	62827.266667	701.653	4.406312e+07	62841.050	
6000	06.44.59.999999	62827.266667	701.653	4.406312e+07	62841.050	
6999	04.29.59.999999	73463.200000	1339.810	9.850158e+07	73443.500	
7000	04.29.59.999999	73463.200000	1339.810	9.850158e+07	73443.500	
7999	02.14.59.999999	65482.933333	558.773	3.657644e+07	65503.325	
8000	02.14.59.999999	65482.933333	558.773	3.657644e+07	65503.325	
8999	11.59.59.999999	66034.733333	543.933	3.592116e+07	66018.925	
9000	11.59.59.999999	66034.733333	543.933	3.592116e+07	66018.925	
9999	09.44.59.999999	64121.733333	2360.220	1.512513e+08	64106.275	
10000	09.44.59.999999	64121.733333	2360.220	1.512513e+08	64106.275	
27982	05.14.59.999999	68830.266667	890.557	6.130400e+07	68836.725	
27983	05.14.59.999999	68830.266667	890.557	6.130400e+07	68836.725	
28982	02.59.59.999999	71256.166667	2297.329	1.637220e+08	71240.125	
28983	02.59.59.999999	71256.166667	2297.329	1.637220e+08	71240.125	
29982	12.44.59.999999	76104.333333	734.371	5.589754e+07	76105.375	
29983	12.44.59.999999	76104.333333	734.371	5.589754e+07	76105.375	
30982	10.29.59.999999	91295.166667	395.726	3.612060e+07	91307.525	
30983	10.29.59.999999	91295.166667	395.726	3.612060e+07	91307.525	
31982	08.14.59.999999	95849.433333	946.794	9.075338e+07	95853.175	
31983	08.14.59.999999	95849.433333	946.794	9.075338e+07	95853.175	
32982	05.59.59.999999	97552.600000	1294.549	1.263778e+08	97514.125	
32983	05.59.59.999999	97552.600000	1294.549	1.263778e+08	97514.125	
33982	03.44.59.999999	97152.333333	773.455	7.516394e+07	97110.650	
33983	03.44.59.999999	97152.333333	773.455	7.516394e+07	97110.650	
34982	01.29.59.999999	93294.300000	1846.171	1.722437e+08	93286.400	
34983	01.29.59.999999	93294.300000	1846.171	1.722437e+08	93286.400	

	Unnamed: 11
999	NaN
1000	NaN
1999	NaN
2000	NaN
2999	NaN
3000	NaN
3999	NaN
4000	NaN

```
4999            NaN
5000            NaN
5999            NaN
6000            NaN
6999            NaN
7000            NaN
7999            NaN
8000            NaN
8999            NaN
9000            NaN
9999            NaN
10000           NaN
27982           NaN
27983           NaN
28982           NaN
28983           NaN
29982           NaN
29983           NaN
30982           NaN
30983           NaN
31982           NaN
31983           NaN
32982           NaN
32983           NaN
33982           NaN
33983           NaN
34982           NaN
34983           NaN
```

[4]:
```python
# Drop unused and redundant columns
df = df.drop(columns=['Unnamed: 11', 'Time right'])

# Verify the columns were dropped
print(f"Remaining columns: {df.columns.tolist()}")
print(f"Dataset shape: {df.shape}")

# Drop any duplicate rows based on "Time left"
df.drop_duplicates(subset=['Time left'], keep='first', inplace=True)
print(f"Dataset shape after removing duplicates: {df.shape}")

# Check if there are still any duplicate rows in the data
df.loc[df.duplicated(subset=['Time left'], keep=False)]
```

```
Remaining columns: ['Close', 'High', 'Low', 'Median', 'Open', 'Time left',
'Typical', 'Volume', 'Quote asset volume', 'Weighted']
Dataset shape: (35059, 10)
Dataset shape after removing duplicates: (35041, 10)
```

[4]:
```
Empty DataFrame
Columns: [Close, High, Low, Median, Open, Time left, Typical, Volume, Quote
asset volume, Weighted]
Index: []
```

[5]:
```
# Convert time columns to datetime format
df['Time left'] = pd.to_datetime(df['Time left'])
df.set_index('Time left', inplace=True)

# Ensure df is sorted by date in ascending order
# This is typically done before passing data to the function, but we can check␣
 ↪it here
if not df.index.is_monotonic_increasing:
    raise ValueError("Data must be sorted by date in ascending order.")
else:
    print("Data is monotonically increasing")

# Display the first few rows of the cleaned dataset
df.head(3)
```

```
Data is monotonically increasing
```

[5]:

Time left	Close	High	Low	Median	Open	Typical	Volume	Quote asset volume	Weighted
2024-01-01 08:00:00	42557.0	42581.2	42495.5	42538.35	42531.7	42544.566667	647.627	2.755070e+07	42547.675
2024-01-01 08:15:00	42525.5	42587.3	42515.0	42551.15	42557.0	42542.600000	185.753	7.903909e+06	42538.325
2024-01-01 08:30:00	42534.5	42550.2	42492.7	42521.45	42525.5	42525.800000	164.984	7.015146e+06	42527.975

2.3.1 Data Preprocessing and Technical Indicators

To make our technical indicator calculations more flexible, I've created modular functions for calculating:

1. **Exponential Moving Average (EMA)** with adjustable periods: A type of moving average that gives more weight to recent price data, making it more responsive to new information.
2. **Average True Range (ATR)** with adjustable periods: A volatility indicator that measures market volatility by decomposing the entire range of an asset price for a period.

[6]:
```
def calculate_ema(data, column='Close', period=10, adjust=False):
    """
    Calculate Exponential Moving Average for any period

    Parameters:
```

```
    - data: pandas DataFrame containing price data
    - column: column name to calculate EMA on (default: 'Close')
    - period: number of periods for EMA calculation (default: 10)
    - adjust: whether to adjust for bias in the beginning (default: False)

    Returns:
    - pandas Series containing EMA values
    """

    ema = data[column].ewm(span=period, adjust=adjust).mean()
    return ema


def calculate_tr(data):
    """
    Calculate True Range

    Parameters:
    - data: pandas DataFrame containing OHLC price data

    Requirements:
    - data must contain 'High', 'Low', and 'Close' columns
    - 'High', 'Low', 'Close' must be numeric types
    - data must be a pandas DataFrame
    - data must be sorted by date in ascending order
    - data must not be empty
    - data must not contain duplicate dates
    - data must not contain any non-numeric values in 'High', 'Low', 'Close'␣
 ↪columns

    Returns:
    - pandas Series containing ATR values
    """
    # Calculate True Range
    tr = np.maximum(
        data['High'] - data['Low'],
        abs(data['High'] - data['Close'].shift(1)),
        abs(data['Low'] - data['Close'].shift(1))
        )

    # Return TR as a pandas Series
    return tr

def calculate_atr(data, period=10):
    """
    Calculate Average True Range for any period
```

```
    Parameters:
    - data: pandas DataFrame containing OHLC price data
    - period: number of periods for ATR calculation (default: 10)

    Requirements:
    - data must contain 'High', 'Low', and 'Close' columns
    - 'High', 'Low', 'Close' must be numeric types
    - data must be a pandas DataFrame
    - period must be a positive integer
    - data must be sorted by date in ascending order
    - data must have at least 'period' number of rows
    - data must not be empty
    - data must not contain duplicate dates
    - data must not contain any non-numeric values in 'High', 'Low', 'Close' columns


    Returns:
    - pandas Series containing ATR values
    """

    # Calculate True Range if not already present
    if 'TR' not in data.columns:
        data['TR'] = calculate_tr(data)

    # Ensure 'TR' column is numeric
    if not pd.api.types.is_numeric_dtype(data['TR']):
        raise ValueError("The 'TR' column must contain numeric values.")

    # Ensure data has enough rows for the ATR calculation
    if len(data) < period:
        raise ValueError(f"Data must have at least {period} rows for ATR calculation.")

    # Ensure period is a positive integer
    if not isinstance(period, int) or period <= 0:
        raise ValueError("Period must be a positive integer.")


    # Calculate ATR as a pandas Series
    atr = data['TR'].rolling(window=period).mean()
    return atr
```

[7]:
```
def add_multiple_indicators(df, ema_periods=[20, 50, 100, 200], atr_periods=[10, 20, 50]):
    """
    Add multiple EMA and ATR indicators to the dataframe with different periods
```

```
    Parameters:
    - df: pandas DataFrame containing price data
    - ema_periods: list of periods for EMA calculation
    - atr_periods: list of periods for ATR calculation

    Returns:
    - DataFrame with additional indicators
    """
    # Add EMAs with different periods
    for period in ema_periods:
        column_name = f'EMA_{period}'
        if column_name not in df.columns:
            df[column_name] = calculate_ema(df, column='Close', period=period)

    # Add ATRs with different periods (reusing the existing TR column)
    for period in atr_periods:
        column_name = f'ATR_{period}'
        if column_name not in df.columns:
            df[column_name] = calculate_atr(df, period=period)

    return df
```

[8]:
```
df.head()
```

[8]:

Time left	Close	High	Low	Median	Open	Typical	Volume	Quote asset volume	Weighted
2024-01-01 08:00:00	42557.0	42581.2	42495.5	42538.35	42531.7	42544.566667	647.627	2.755070e+07	42547.675
2024-01-01 08:15:00	42525.5	42587.3	42515.0	42551.15	42557.0	42542.600000	185.753	7.903909e+06	42538.325
2024-01-01 08:30:00	42534.5	42550.2	42492.7	42521.45	42525.5	42525.800000	164.984	7.015146e+06	42527.975
2024-01-01 08:45:00	42596.1	42599.6	42525.3	42562.45	42534.5	42573.666667	276.168	1.175917e+07	42579.275
2024-01-01 09:00:00	42614.7	42618.7	42574.1	42596.40	42596.1	42602.500000	334.340	1.424297e+07	42605.550

[9]:
```
# Apply the function to add multiple indicators to the dataframe
df = add_multiple_indicators(df, ema_periods=[10, 20], atr_periods=[10, 20])
```

[10]:
```
df
```

[10]:

Time left	Close	High	Low	Median	Open
2024-01-01 08:00:00	42557.0	42581.2	42495.5	42538.35	42531.7
2024-01-01 08:15:00	42525.5	42587.3	42515.0	42551.15	42557.0
2024-01-01 08:30:00	42534.5	42550.2	42492.7	42521.45	42525.5
2024-01-01 08:45:00	42596.1	42599.6	42525.3	42562.45	42534.5
2024-01-01 09:00:00	42614.7	42618.7	42574.1	42596.40	42596.1
…	…	…	…	…	…
2024-12-31 07:00:00	92803.1	92853.0	92717.2	92785.10	92777.2
2024-12-31 07:15:00	92800.0	92845.0	92750.0	92797.50	92803.1
2024-12-31 07:30:00	92783.0	92845.0	92721.0	92783.00	92800.0
2024-12-31 07:45:00	92811.7	92870.1	92722.7	92796.40	92783.0
2024-12-31 08:00:00	92919.8	92948.5	92790.0	92869.25	92811.7

Time left	Typical	Volume	Quote asset volume	Weighted
2024-01-01 08:00:00	42544.566667	647.627	2.755070e+07	42547.675
2024-01-01 08:15:00	42542.600000	185.753	7.903909e+06	42538.325
2024-01-01 08:30:00	42525.800000	164.984	7.015146e+06	42527.975
2024-01-01 08:45:00	42573.666667	276.168	1.175917e+07	42579.275
2024-01-01 09:00:00	42602.500000	334.340	1.424297e+07	42605.550
…	…	…	…	…
2024-12-31 07:00:00	92791.100000	398.973	3.702261e+07	92794.100
2024-12-31 07:15:00	92798.333333	459.895	4.268268e+07	92798.750
2024-12-31 07:30:00	92783.000000	368.663	3.421070e+07	92783.000
2024-12-31 07:45:00	92801.500000	316.708	2.939826e+07	92804.050
2024-12-31 08:00:00	92886.100000	367.895	3.416942e+07	92894.525

Time left	EMA_10	EMA_20	TR	ATR_10	ATR_20
2024-01-01 08:00:00	42557.000000	42557.000000	NaN	NaN	NaN
2024-01-01 08:15:00	42551.272727	42554.000000	72.3	NaN	NaN
2024-01-01 08:30:00	42548.223140	42552.142857	57.5	NaN	NaN
2024-01-01 08:45:00	42556.928024	42556.329252	74.3	NaN	NaN
2024-01-01 09:00:00	42567.432020	42561.888371	44.6	NaN	NaN
…	…	…	…	…	…
2024-12-31 07:00:00	92673.953662	92631.111926	135.8	230.82	238.185
2024-12-31 07:15:00	92696.871178	92647.196504	95.0	220.15	232.795
2024-12-31 07:30:00	92712.530964	92660.130171	124.0	219.90	226.815
2024-12-31 07:45:00	92730.561698	92674.565392	147.4	196.84	226.660
2024-12-31 08:00:00	92764.968662	92697.921069	158.5	191.34	221.085

[35041 rows x 14 columns]

2.4 Find Long and Short Trades

[11]:
```
# Create a function to identify long trade setups
def find_long_trades(df,
                     ema_period=10,
                     atr_period=10,
                     target_atr_multiplier=0.5,
                     stop_loss_atr_multiplier=1.0,
                     tick_size=0.1):
    """
    Identify long trade setups based on the following conditions:
    - Current candle has lower high and lower low than previous candle
    - Current candle is bullish (close > open)
    - Close is above EMA
    - Entry is 1 tick above the high of the current candle
    - Target is target_atr_multiplier * ATR above entry
    - Stop loss is stop_loss_atr_multiplier * ATR below entry

    Parameters:
    df (DataFrame): DataFrame containing OHLC data with 'High', 'Low', 'Open',␣
 ↪'Close', 'EMA_{ema_period}', 'ATR_{atr_period}' columns.

    ema_period (int): Period for EMA calculation.
    atr_period (int): Period for ATR calculation.
    target_atr_multiplier: ATR multiplier for target price calculation (default:
 ↪ 0.5)
    stop_loss_atr_multiplier: ATR multiplier for stop loss calculation (default:
 ↪ 1.0)
    tick_size (float): Smallest price movement for the instrument.

    Returns:
    DataFrame: DataFrame containing trade setups with entry, target, stop loss,␣
 ↪and other relevant columns.
    """

    import pandas as pd

    # Ensure the dataframe has the necessary columns
    required_columns = ['High', 'Low', 'Open', 'Close']
    if not all(col in df.columns for col in required_columns):
        raise ValueError(f"DataFrame must contain {required_columns} columns.")

    # Check if data has ema and atr period columns available
    ema_col = f'EMA_{ema_period}'
    atr_col = f'ATR_{atr_period}'
    if ema_col not in df.columns or atr_col not in df.columns:
```

```
        df = add_multiple_indicators(df, ema_periods=[ema_period],␣
↪atr_periods=[atr_period])
        # raise ValueError(f"DataFrame must contain EMA_{ema_period} and␣
↪ATR_{atr_period} columns.")


    # Create a copy to avoid modifying the original dataframe
    trades_df = df.copy()

    # Calculate if the current candle has lower high and lower low than␣
↪previous candle
    trades_df['lower_high'] = trades_df['High'] < trades_df['High'].shift(1)
    trades_df['lower_low'] = trades_df['Low'] < trades_df['Low'].shift(1)

    # Calculate if the candle is bullish (close > open)
    trades_df['is_bullish'] = trades_df['Close'] > trades_df['Open']

    # Check if close is above EMA
    trades_df['above_ema'] = trades_df['Close'] > trades_df[ema_col]

    # Identify trade setups - when all conditions are met
    trades_df['long_setup'] = (trades_df['lower_high'] &
                               trades_df['lower_low'] &
                               trades_df['is_bullish'] &
                               trades_df['above_ema'])

    # Calculate entry, target and stop loss for identified setups
    # Candle entry is 1 tick above the high of the current candle
    trades_df['entry'] = trades_df['High'] + tick_size  # 1 tick above high
    trades_df['target'] = trades_df['entry'] + target_atr_multiplier *␣
↪trades_df[atr_col]
    trades_df['stop_loss'] = trades_df['entry'] - stop_loss_atr_multiplier *␣
↪trades_df[atr_col]

    # Filter only the trade setups
    long_trades = trades_df[trades_df['long_setup']]

    return df, long_trades[['entry', 'target', 'stop_loss', 'Close', 'High',␣
↪'Low', 'Open', ema_col, atr_col]]
```

[12]:
```
def find_short_trades(df,
    ema_period=10,
    atr_period=10,
    target_atr_multiplier=0.5,
    stop_loss_atr_multiplier=1.0,
    tick_size=0.1):
```

```
    """
    Identify short trade setups based on the following conditions:
    - Current candle has higher high and higher low than previous candle
    - Current candle is bearish (close < open)
    - Close is below EMA
    - Entry is 1 tick below the low of the current candle
    - Target is target_atr_multiplier * ATR below entry
    - Stop loss is stop_loss_atr_multiplier * ATR above entry


    Parameters:
    df (DataFrame): DataFrame containing OHLC data with 'High', 'Low', 'Open',␣
↪'Close', 'EMA_{ema_period}', 'ATR_{atr_period}' columns.

    ema_period (int): Period for EMA calculation.
    atr_period (int): Period for ATR calculation.
    target_atr_multiplier: ATR multiplier for target price calculation (default:
↪ 0.5)
    stop_loss_atr_multiplier: ATR multiplier for stop loss calculation (default:
↪ 1.0)
    tick_size (float): Smallest price movement for the instrument.

    Returns:
    DataFrame: DataFrame containing trade setups with entry, target, stop loss,␣
↪and other relevant columns.
    """

    # Ensure the dataframe has the necessary columns
    required_columns = ['High', 'Low', 'Open', 'Close']
    if not all(col in df.columns for col in required_columns):
        raise ValueError(f"DataFrame must contain {required_columns} columns.")

    # Check if data has ema and atr period columns available
    ema_col = f'EMA_{ema_period}'
    atr_col = f'ATR_{atr_period}'
    if ema_col not in df.columns or atr_col not in df.columns:
        df = add_multiple_indicators(df, ema_periods=[ema_period],␣
↪atr_periods=[atr_period]) # Add the missing indicators to df
        # raise ValueError(f"DataFrame must contain {ema_col} and {atr_col}␣
↪columns.")

    # Create a copy to avoid modifying the original dataframe
    trades_df = df.copy()

    # Calculate if the current candle has higher high and higher low than␣
↪previous candle
```

```
    trades_df['higher_high'] = trades_df['High'] > trades_df['High'].shift(1)
    trades_df['higher_low'] = trades_df['Low'] > trades_df['Low'].shift(1)

    # Calculate if the candle is bearish (close < open)
    trades_df['is_bearish'] = trades_df['Close'] < trades_df['Open']

    # Check if close is below EMA
    trades_df['below_ema'] = trades_df['Close'] < trades_df[ema_col]

    # Identify trade setups - when all conditions are met
    trades_df['short_setup'] = (trades_df['higher_high'] &
                                trades_df['higher_low'] &
                                trades_df['is_bearish'] &
                                trades_df['below_ema'])

    # Calculate entry, target and stop loss for identified setups
    # Candle entry is 1 tick below the low of the current candle
    trades_df['entry'] = trades_df['Low'] - tick_size  # 1 tick below low
    trades_df['target'] = trades_df['entry'] - target_atr_multiplier *␣
↪trades_df[atr_col]  # Target below entry
    trades_df['stop_loss'] = trades_df['entry'] + stop_loss_atr_multiplier *␣
↪trades_df[atr_col]  # Stop loss above entry

    # Filter only the trade setups
    short_trades = trades_df[trades_df['short_setup']]

    return df, short_trades[['entry', 'target', 'stop_loss', 'Close', 'High',␣
↪'Low', 'Open', ema_col, atr_col]]
```

[13]:
```
# Find long trade setups
df, long_trade_opportunities = find_long_trades(df)
print(f"Found {len(long_trade_opportunities)} long trade opportunities")
long_trade_opportunities.head(10)
```

```
Found 1458 long trade opportunities
```

[13]:

Time left	entry	target	stop_loss	Close	High	Low
2024-01-01 13:00:00	42749.0	42795.420	42656.16	42745.9	42748.9	42651.1
2024-01-01 18:30:00	43214.5	43280.565	43082.37	43196.2	43214.4	43065.6
2024-01-02 01:15:00	45147.5	45330.165	44782.17	45005.2	45147.4	44922.0
2024-01-02 01:45:00	44990.1	45166.690	44636.92	44929.4	44990.0	44744.5
2024-01-02 06:00:00	45283.5	45353.955	45142.59	45278.3	45283.4	45170.0
2024-01-02 07:30:00	45578.2	45665.880	45402.84	45578.0	45578.1	45316.7
2024-01-02 08:45:00	45853.8	45975.340	45610.72	45817.9	45853.7	45645.9
2024-01-02 10:00:00	45774.3	45883.505	45555.89	45752.8	45774.2	45602.7
2024-01-02 18:45:00	45274.5	45371.650	45080.20	45202.6	45274.4	45095.2

```
2024-01-03 02:15:00  45379.7  45453.790   45231.52  45369.6  45379.6  45258.7

                        Open        EMA_10  ATR_10
Time left
2024-01-01 13:00:00  42690.6  42717.602005   92.84
2024-01-01 18:30:00  43111.4  42960.558854  132.13
2024-01-02 01:15:00  44932.1  44605.417288  365.33
2024-01-02 01:45:00  44876.0  44704.575209  353.18
2024-01-02 06:00:00  45220.0  45275.628330  140.91
2024-01-02 07:30:00  45495.9  45373.723432  175.36
2024-01-02 08:45:00  45719.3  45600.534766  243.08
2024-01-02 10:00:00  45731.6  45701.324835  218.41
2024-01-02 18:45:00  45124.0  45183.718028  194.30
2024-01-03 02:15:00  45324.5  45239.668509  148.18
```

[14]:
```
# Find short trade setups
df, short_trade_opportunities = find_short_trades(df)
print(f"Found {len(short_trade_opportunities)} short trade opportunities")
short_trade_opportunities.head(10)
```

```
Found 1307 short trade opportunities
```

[14]:
```
                       entry     target  stop_loss    Close     High      Low  \
Time left
2024-01-01 08:15:00  42514.9        NaN        NaN  42525.5  42587.3  42515.0
2024-01-01 12:00:00  42657.0  42616.810   42737.38  42674.8  42784.9  42657.1
2024-01-01 22:00:00  43564.8  43477.650   43739.10  43567.6  43726.4  43564.9
2024-01-02 11:00:00  45612.4  45526.240   45784.72  45620.0  45772.0  45612.5
2024-01-02 12:00:00  45400.4  45314.555   45572.09  45442.0  45562.0  45400.5
2024-01-02 15:45:00  45199.5  45061.905   45474.69  45230.3  45346.8  45199.6
2024-01-02 18:30:00  45117.2  45013.505   45324.59  45124.0  45320.5  45117.3
2024-01-02 19:00:00  45160.4  45066.045   45349.11  45174.6  45297.2  45160.5
2024-01-02 21:15:00  44842.9  44723.815   45081.07  44850.4  44985.0  44843.0
2024-01-02 23:15:00  45002.5  44905.855   45195.79  45014.9  45099.3  45002.6

                        Open        EMA_10  ATR_10
Time left
2024-01-01 08:15:00  42557.0  42551.272727     NaN
2024-01-01 12:00:00  42735.1  42691.705008   80.38
2024-01-01 22:00:00  43677.7  43596.368592  174.30
2024-01-02 11:00:00  45658.7  45676.598268  172.32
2024-01-02 12:00:00  45495.9  45551.033661  171.69
2024-01-02 15:45:00  45323.6  45365.156803  275.19
2024-01-02 18:30:00  45174.1  45179.522035  207.39
2024-01-02 19:00:00  45202.6  45182.060205  188.71
2024-01-02 21:15:00  44894.6  44964.139462  238.17
2024-01-02 23:15:00  45038.8  45034.199895  193.29
```

[15]:
```
# In long or short trade setups, drop any rows that have missing values
long_trade_opportunities.dropna(inplace=True)
short_trade_opportunities.dropna(inplace=True)
```

[16]:
```
long_trade_opportunities
```

[16]:

Time left	entry	target	stop_loss	Close	High	Low	Open	EMA_10	ATR_10
2024-01-01 13:00:00	42749.0	42795.420	42656.16	42745.9	42748.9	42651.1	42690.6	42717.602005	92.84
2024-01-01 18:30:00	43214.5	43280.565	43082.37	43196.2	43214.4	43065.6	43111.4	42960.558854	132.13
2024-01-02 01:15:00	45147.5	45330.165	44782.17	45005.2	45147.4	44922.0	44932.1	44605.417288	365.33
2024-01-02 01:45:00	44990.1	45166.690	44636.92	44929.4	44990.0	44744.5	44876.0	44704.575209	353.18
2024-01-02 06:00:00	45283.5	45353.955	45142.59	45278.3	45283.4	45170.0	45220.0	45275.628330	140.91
…	…	…	…	…	…	…	…	…	…
2024-12-29 05:45:00	95095.5	95153.020	94980.46	95074.2	95095.4	95014.5	95019.4	95072.591786	115.04
2024-12-30 01:45:00	93893.4	94067.905	93544.39	93886.3	93893.3	93360.3	93725.8	93640.153260	349.01
2024-12-30 06:45:00	93883.1	94010.000	93629.30	93854.8	93883.0	93675.1	93764.2	93649.039896	253.80
2024-12-30 18:30:00	94466.8	94763.465	93873.47	94344.4	94466.7	94175.2	94310.5	93472.352370	593.33
2024-12-31 03:45:00	92538.9	92673.605	92269.49	92467.1	92538.8	92301.0	92391.7	92440.936416	269.41

[1458 rows x 9 columns]

[17]:
```
short_trade_opportunities
```

[17]:

Time left	entry	target	stop_loss	Close	High	Low
2024-01-01 12:00:00	42657.0	42616.810	42737.38	42674.8	42784.9	42657.1
2024-01-01 22:00:00	43564.8	43477.650	43739.10	43567.6	43726.4	43564.9
2024-01-02 11:00:00	45612.4	45526.240	45784.72	45620.0	45772.0	45612.5
2024-01-02 12:00:00	45400.4	45314.555	45572.09	45442.0	45562.0	45400.5
2024-01-02 15:45:00	45199.5	45061.905	45474.69	45230.3	45346.8	45199.6
…	…	…	…	…	…	…
2024-12-30 02:00:00	93556.3	93383.285	93902.33	93578.2	93919.9	93556.4

```
2024-12-30 15:45:00  91868.0  91585.365   92433.27  92077.1  92419.9  91868.1
2024-12-31 00:45:00  92333.5  92163.430   92673.64  92399.0  92828.6  92333.6
2024-12-31 01:45:00  92316.3  92150.365   92648.17  92390.2  92500.0  92316.4
2024-12-31 02:30:00  92422.9  92256.455   92755.79  92453.1  92666.6  92423.0

                        Open        EMA_10  ATR_10
Time left
2024-01-01 12:00:00  42735.1  42691.705008   80.38
2024-01-01 22:00:00  43677.7  43596.368592  174.30
2024-01-02 11:00:00  45658.7  45676.598268  172.32
2024-01-02 12:00:00  45495.9  45551.033661  171.69
2024-01-02 15:45:00  45323.6  45365.156803  275.19
…                      …           …     …
2024-12-30 02:00:00  93886.3  93628.889031  346.03
2024-12-30 15:45:00  92122.1  92340.339699  565.27
2024-12-31 00:45:00  92626.8  92683.519944  340.14
2024-12-31 01:45:00  92415.0  92477.766748  331.87
2024-12-31 02:30:00  92547.5  92478.474500  332.89

[1306 rows x 9 columns]
```

2.5 Backtesting

[18]:
```python
def calculate_max_drawdown(equity_curve):
    """
    Calculate the maximum drawdown percentage from an equity curve

    Parameters:
    - equity_curve: Series containing account balance over time

    Returns:
    - Maximum drawdown as a percentage
    """
    # Calculate the running maximum
    running_max = equity_curve.cummax()

    # Calculate drawdown in percentage terms
    drawdown = ((running_max - equity_curve) / running_max * 100)

    # Find the maximum drawdown
    max_dd = drawdown.max()

    return max_dd
```

[19]:
```python
def backtest_short_trade_strategy(df, trades_df,
                                  slippage_pct=0.05,
                                  commission_pct=0.075,
```

```
                                risk_per_trade_pct=1.0,
                                target_atr_multiplier=0.5,
                                stop_loss_atr_multiplier=1.0):
    """
    Backtests a short trading strategy on identified trade setups

    Parameters:
    - df: DataFrame containing full price history with OHLC data
    - trades_df: DataFrame containing identified short trade setups with entry, target and stop-loss prices
    - slippage_pct: Percentage of slippage applied to entries and exits (default: 0.05%)
    - commission_pct: Percentage of commission applied to entries and exits (default: 0.075%)
    - risk_per_trade_pct: Percentage of account balance risked per trade (default: 1%)
    - target_atr_multiplier: ATR multiplier for target price calculation (default: 0.5)
    - stop_loss_atr_multiplier: ATR multiplier for stop loss calculation (default: 1.0)

    Returns:
    - DataFrame containing detailed results of each trade
    - Dictionary containing aggregated performance metrics
    - DataFrame containing equity curve data
    """
    # Initialize performance tracking variables
    initial_balance = 10000.0
    balance = initial_balance
    balance_history = [initial_balance]
    dates_history = [df.index[0]]  # Start with the first date
    closed_trades = []

    # Loop through each trading opportunity chronologically
    dates = trades_df.index.sort_values().tolist()

    for trade_date in dates:
        # Extract trade setup information
        setup = trades_df.loc[trade_date]

        # Use the pre-calculated values from trades_df
        entry_price = setup['entry']
        target_price = setup['target']
        stop_loss_price = setup['stop_loss']

        # if either entry, target or stop loss is NaN, skip this trade
```

```
        if pd.isna(entry_price) or pd.isna(target_price) or pd.
↪isna(stop_loss_price):
            print(f"Skipping trade on {trade_date} due to NaN entry, target or␣
↪stop loss.")
            continue

        # Calculate position size based on risk percentage
        risk_amount = balance * (risk_per_trade_pct / 100)
        risk_per_coin = stop_loss_price - entry_price

        # Avoid division by zero or negative risk
        if risk_per_coin <= 0:
            continue

        position_size = risk_amount / risk_per_coin
        position_value = position_size * entry_price

        # Check if we have enough balance for this trade
        if position_value > balance:
            position_size = balance / entry_price
            position_value = balance

        # Apply slippage to entry price (worse price for short trades)
        actual_entry = entry_price * (1 - slippage_pct/100)  # Lower price for␣
↪short entry

        # Apply commission on entry
        commission = position_value * commission_pct/100
        balance -= commission

        # Track the trade
        trade = {
            'entry_date': trade_date,
            'entry_price': actual_entry,
            'position_size': position_size,
            'position_value': position_value,
            'target_price': target_price,
            'stop_loss': stop_loss_price,
            'commission': commission,
            'risk_amount': risk_amount,
            'risk_per_trade_pct': risk_per_trade_pct
        }

        # Find future candles to determine outcome
        future_dates = df.loc[trade_date:].index[1:]  # Get dates after the␣
↪setup
        exit_found = False
```

```
        for future_date in future_dates:
            future_bar = df.loc[future_date]

            # Check if target was hit (price moved down for shorts)
            if future_bar['Low'] <= target_price:
                # Target hit - calculate exit with slippage
                actual_exit = target_price * (1 + slippage_pct/100)  # Worse␣
↪price for exit (slippage)
                exit_position_value = position_size * actual_exit
                profit = position_value - exit_position_value  # Profit␣
↪calculation for shorts

                # Apply commission on exit
                exit_commission = exit_position_value * commission_pct/100
                profit -= exit_commission

                # Update balance
                balance += profit

                # Log the trade
                trade['exit_date'] = future_date
                trade['exit_price'] = actual_exit
                trade['exit_type'] = 'target'
                trade['profit_loss'] = profit
                trade['return_pct'] = (profit / position_value) * 100
                trade['balance'] = balance
                trade['exit_commission'] = exit_commission
                trade['hold_period'] = (future_date - trade_date).
↪total_seconds() / 3600  # Hold period in hours
                closed_trades.append(trade)
                exit_found = True

                # Update equity curve
                balance_history.append(balance)
                dates_history.append(future_date)
                break

            # Check if stop loss was hit (price moved up for shorts)
            elif future_bar['High'] >= stop_loss_price:
                # Stop loss hit - calculate exit with slippage
                actual_exit = stop_loss_price * (1 + slippage_pct/100)  # Worse␣
↪price for exit (slippage)
                exit_position_value = position_size * actual_exit
                loss = position_value - exit_position_value  # Loss calculation␣
↪for shorts
```

```
                # Apply commission on exit
                exit_commission = exit_position_value * commission_pct/100
                loss -= exit_commission

                # Update balance
                balance += loss

                # Log the trade
                trade['exit_date'] = future_date
                trade['exit_price'] = actual_exit
                trade['exit_type'] = 'stop_loss'
                trade['profit_loss'] = loss
                trade['return_pct'] = (loss / position_value) * 100
                trade['balance'] = balance
                trade['exit_commission'] = exit_commission
                trade['hold_period'] = (future_date - trade_date).
↪total_seconds() / 3600  # Hold period in hours
                closed_trades.append(trade)
                exit_found = True

                # Update equity curve
                balance_history.append(balance)
                dates_history.append(future_date)
                break

        # If trade is still open at the end of data, close it at the last price
        if not exit_found:
            last_date = df.index[-1]
            last_bar = df.loc[last_date]
            actual_exit = last_bar['Close'] * (1 + slippage_pct/100)  # Apply␣
↪slippage on exit
            exit_position_value = position_size * actual_exit
            pnl = position_value - exit_position_value  # P&L calculation for␣
↪shorts

            # Apply commission on exit
            exit_commission = exit_position_value * commission_pct/100
            pnl -= exit_commission

            # Update balance
            balance += pnl

            # Log the trade
            trade['exit_date'] = last_date
            trade['exit_price'] = actual_exit
            trade['exit_type'] = 'end_of_data'
            trade['profit_loss'] = pnl
```

```
            trade['return_pct'] = (pnl / position_value) * 100
            trade['balance'] = balance
            trade['exit_commission'] = exit_commission
            trade['hold_period'] = (last_date - trade_date).total_seconds() /␣
↪3600  # Hold period in hours
            closed_trades.append(trade)

            # Update equity curve
            balance_history.append(balance)
            dates_history.append(last_date)

    # Create equity curve dataframe
    equity_df = pd.DataFrame({
        'date': dates_history,
        'balance': balance_history
    })
    equity_df.set_index('date', inplace=True)

    # Calculate strategy metrics if trades were executed
    trades_df = pd.DataFrame(closed_trades) if closed_trades else pd.DataFrame()
    if len(trades_df) > 0:
        # Separate winning and losing trades
        winning_trades = trades_df[trades_df['profit_loss'] > 0]
        losing_trades = trades_df[trades_df['profit_loss'] <= 0]

        # Calculate max drawdown using equity curve
        max_dd = calculate_max_drawdown(equity_df['balance'])

        # Calculate average trade metrics

        metrics = {
            'initial_balance': initial_balance,
            'final_balance': balance,
            'total_return': balance - initial_balance,
            'total_return_pct': (balance / initial_balance - 1) * 100,
            'total_trades': len(trades_df),
            'winning_trades': len(winning_trades),
            'losing_trades': len(losing_trades),
            'win_rate': len(winning_trades) / len(trades_df) if len(trades_df)␣
↪> 0 else 0,
            'average_win': winning_trades['profit_loss'].mean() if␣
↪len(winning_trades) > 0 else 0,
            'average_loss': losing_trades['profit_loss'].mean() if␣
↪len(losing_trades) > 0 else 0,
            'largest_win': winning_trades['profit_loss'].max() if␣
↪len(winning_trades) > 0 else 0,
```

```
            'largest_loss': losing_trades['profit_loss'].min() if␣
↪len(losing_trades) > 0 else 0,
            'profit_factor': abs(winning_trades['profit_loss'].sum() /␣
↪losing_trades['profit_loss'].sum()) if len(losing_trades) > 0 and␣
↪losing_trades['profit_loss'].sum() != 0 else float('inf'),
            'max_drawdown_pct': max_dd,
            'avg_trade_duration_hours': trades_df['hold_period'].mean() if␣
↪'hold_period' in trades_df.columns else 0,
            'total_commission': trades_df['commission'].sum() +␣
↪trades_df['exit_commission'].sum() if 'exit_commission' in trades_df.columns␣
↪else trades_df['commission'].sum(),
            'expectancy': (winning_trades['profit_loss'].mean() *␣
↪len(winning_trades) + losing_trades['profit_loss'].mean() *␣
↪len(losing_trades)) / len(trades_df) if len(trades_df) > 0 else 0
        }
    else:
        metrics = {
            'initial_balance': initial_balance,
            'final_balance': balance,
            'total_return': 0,
            'total_return_pct': 0,
            'total_trades': 0,
            'winning_trades': 0,
            'losing_trades': 0,
            'win_rate': 0,
            'average_win': 0,
            'average_loss': 0,
            'largest_win': 0,
            'largest_loss': 0,
            'profit_factor': 0,
            'max_drawdown_pct': 0,
            'avg_trade_duration_hours': 0,
            'total_commission': 0,
            'expectancy': 0
        }

    return trades_df, metrics, equity_df
```

[20]:
```
def backtest_long_trade_strategy(df, trades_df,
                                 slippage_pct=0.05,
                                 commission_pct=0.075,
                                 risk_per_trade_pct=1.0,
                                 target_atr_multiplier=0.5,
                                 stop_loss_atr_multiplier=1.0):
    """
    Backtests a long trading strategy on identified trade setups
```

```
    Parameters:
    - df: DataFrame containing full price history with OHLC data
    - trades_df: DataFrame containing identified long trade setups with entry, target and stop-loss prices
    - slippage_pct: Percentage of slippage applied to entries and exits (default: 0.05%)
    - commission_pct: Percentage of commission applied to entries and exits (default: 0.075%)
    - risk_per_trade_pct: Percentage of account balance risked per trade (default: 1%)
    - target_atr_multiplier: ATR multiplier for target price calculation (default: 0.5)
    - stop_loss_atr_multiplier: ATR multiplier for stop loss calculation (default: 1.0)

    Returns:
    - DataFrame containing detailed results of each trade
    - Dictionary containing aggregated performance metrics
    - DataFrame containing equity curve data
    """
    # Initialize performance tracking variables
    initial_balance = 10000.0
    balance = initial_balance
    balance_history = [initial_balance]
    dates_history = [df.index[0]]  # Start with the first date
    closed_trades = []

    # Loop through each trading opportunity chronologically
    dates = trades_df.index.sort_values().tolist()

    for trade_date in dates:
        # Extract trade setup information
        setup = trades_df.loc[trade_date]

        # Use the pre-calculated values from trades_df
        entry_price = setup['entry']
        target_price = setup['target']
        stop_loss_price = setup['stop_loss']

        # if either entry, target or stop loss is NaN, skip this trade
        if pd.isna(entry_price) or pd.isna(target_price) or pd.isna(stop_loss_price):
            print(f"Skipping trade on {trade_date} due to NaN entry, target or stop loss.")
            continue
```

```
        # Calculate position size based on risk percentage
        risk_amount = balance * (risk_per_trade_pct / 100)
        risk_per_coin = entry_price - stop_loss_price  # For long trades, risk␣
↪is entry minus stop loss

        # Avoid division by zero or negative risk
        if risk_per_coin <= 0:
            continue

        position_size = risk_amount / risk_per_coin
        position_value = position_size * entry_price

        # Check if we have enough balance for this trade
        if position_value > balance:
            position_size = balance / entry_price
            position_value = balance

        # Apply slippage to entry price (worse price for long trades)
        actual_entry = entry_price * (1 + slippage_pct/100)  # Higher price for␣
↪long entry

        # Apply commission on entry
        commission = position_value * commission_pct/100
        balance -= position_value + commission  # Deduct the position value␣
↪plus commission

        # Track the trade
        trade = {
            'entry_date': trade_date,
            'entry_price': actual_entry,
            'position_size': position_size,
            'position_value': position_value,
            'target_price': target_price,
            'stop_loss': stop_loss_price,
            'commission': commission,
            'risk_amount': risk_amount,
            'risk_per_trade_pct': risk_per_trade_pct
        }

        # Find future candles to determine outcome
        future_dates = df.loc[trade_date:].index[1:]  # Get dates after the␣
↪setup
        exit_found = False

        for future_date in future_dates:
            future_bar = df.loc[future_date]
```

```
            # Check if target was hit (price moved up for longs)
            if future_bar['High'] >= target_price:
                # Target hit - calculate exit with slippage
                actual_exit = target_price * (1 - slippage_pct/100)  # Worse␣
↪price for exit (slippage)
                exit_position_value = position_size * actual_exit

                # Apply commission on exit
                exit_commission = exit_position_value * commission_pct/100

                # Calculate profit (exit value minus entry value minus␣
↪commissions)
                profit = exit_position_value - position_value - exit_commission

                # Update balance
                balance += exit_position_value - exit_commission

                # Log the trade
                trade['exit_date'] = future_date
                trade['exit_price'] = actual_exit
                trade['exit_type'] = 'target'
                trade['profit_loss'] = profit
                trade['return_pct'] = (profit / position_value) * 100
                trade['balance'] = balance
                trade['exit_commission'] = exit_commission
                trade['hold_period'] = (future_date - trade_date).
↪total_seconds() / 3600  # Hold period in hours
                closed_trades.append(trade)
                exit_found = True

                # Update equity curve
                balance_history.append(balance)
                dates_history.append(future_date)
                break

            # Check if stop loss was hit (price moved down for longs)
            elif future_bar['Low'] <= stop_loss_price:
                # Stop loss hit - calculate exit with slippage
                actual_exit = stop_loss_price * (1 - slippage_pct/100)  # Worse␣
↪price for exit (slippage)
                exit_position_value = position_size * actual_exit

                # Apply commission on exit
                exit_commission = exit_position_value * commission_pct/100

                # Calculate loss (exit value minus entry value minus␣
↪commissions)
```

```
                loss = exit_position_value - position_value - exit_commission

                # Update balance
                balance += exit_position_value - exit_commission

                # Log the trade
                trade['exit_date'] = future_date
                trade['exit_price'] = actual_exit
                trade['exit_type'] = 'stop_loss'
                trade['profit_loss'] = loss
                trade['return_pct'] = (loss / position_value) * 100
                trade['balance'] = balance
                trade['exit_commission'] = exit_commission
                trade['hold_period'] = (future_date - trade_date).
↪total_seconds() / 3600  # Hold period in hours
                closed_trades.append(trade)
                exit_found = True

                # Update equity curve
                balance_history.append(balance)
                dates_history.append(future_date)
                break

        # If trade is still open at the end of data, close it at the last price
        if not exit_found:
            last_date = df.index[-1]
            last_bar = df.loc[last_date]
            actual_exit = last_bar['Close'] * (1 - slippage_pct/100)  # Apply␣
↪slippage on exit
            exit_position_value = position_size * actual_exit

            # Apply commission on exit
            exit_commission = exit_position_value * commission_pct/100

            # Calculate P&L (exit value minus entry value minus commissions)
            pnl = exit_position_value - position_value - exit_commission

            # Update balance
            balance += exit_position_value - exit_commission

            # Log the trade
            trade['exit_date'] = last_date
            trade['exit_price'] = actual_exit
            trade['exit_type'] = 'end_of_data'
            trade['profit_loss'] = pnl
            trade['return_pct'] = (pnl / position_value) * 100
            trade['balance'] = balance
```

```
            trade['exit_commission'] = exit_commission
            trade['hold_period'] = (last_date - trade_date).total_seconds() /␣
↪3600  # Hold period in hours
            closed_trades.append(trade)

            # Update equity curve
            balance_history.append(balance)
            dates_history.append(last_date)

    # Create equity curve dataframe
    equity_df = pd.DataFrame({
        'date': dates_history,
        'balance': balance_history
    })
    equity_df.set_index('date', inplace=True)

    # Calculate strategy metrics if trades were executed
    trades_df = pd.DataFrame(closed_trades) if closed_trades else pd.DataFrame()
    if len(trades_df) > 0:
        # Separate winning and losing trades
        winning_trades = trades_df[trades_df['profit_loss'] > 0]
        losing_trades = trades_df[trades_df['profit_loss'] <= 0]

        # Calculate max drawdown using equity curve
        max_dd = calculate_max_drawdown(equity_df['balance'])

        # Calculate average trade metrics
        metrics = {
            'initial_balance': initial_balance,
            'final_balance': balance,
            'total_return': balance - initial_balance,
            'total_return_pct': (balance / initial_balance - 1) * 100,
            'total_trades': len(trades_df),
            'winning_trades': len(winning_trades),
            'losing_trades': len(losing_trades),
            'win_rate': len(winning_trades) / len(trades_df) if len(trades_df)␣
↪> 0 else 0,
            'average_win': winning_trades['profit_loss'].mean() if␣
↪len(winning_trades) > 0 else 0,
            'average_loss': losing_trades['profit_loss'].mean() if␣
↪len(losing_trades) > 0 else 0,
            'largest_win': winning_trades['profit_loss'].max() if␣
↪len(winning_trades) > 0 else 0,
            'largest_loss': losing_trades['profit_loss'].min() if␣
↪len(losing_trades) > 0 else 0,
```

```
            'profit_factor': abs(winning_trades['profit_loss'].sum() /␣
 ↪losing_trades['profit_loss'].sum()) if len(losing_trades) > 0 and␣
 ↪losing_trades['profit_loss'].sum() != 0 else float('inf'),
            'max_drawdown_pct': max_dd,
            'avg_trade_duration_hours': trades_df['hold_period'].mean() if␣
 ↪'hold_period' in trades_df.columns else 0,
            'total_commission': trades_df['commission'].sum() +␣
 ↪trades_df['exit_commission'].sum() if 'exit_commission' in trades_df.columns␣
 ↪else trades_df['commission'].sum(),
            'expectancy': (winning_trades['profit_loss'].mean() *␣
 ↪len(winning_trades) + losing_trades['profit_loss'].mean() *␣
 ↪len(losing_trades)) / len(trades_df) if len(trades_df) > 0 else 0
        }
    else:
        metrics = {
            'initial_balance': initial_balance,
            'final_balance': balance,
            'total_return': 0,
            'total_return_pct': 0,
            'total_trades': 0,
            'winning_trades': 0,
            'losing_trades': 0,
            'win_rate': 0,
            'average_win': 0,
            'average_loss': 0,
            'largest_win': 0,
            'largest_loss': 0,
            'profit_factor': 0,
            'max_drawdown_pct': 0,
            'avg_trade_duration_hours': 0,
            'total_commission': 0,
            'expectancy': 0
        }

    return trades_df, metrics, equity_df
```

Get the results from backtesting

[21]:
```
# Run backtest for short trades
short_trade_results, short_metrics, short_equity_curve =␣
 ↪backtest_short_trade_strategy(
    df,
    short_trade_opportunities,
    slippage_pct=0.05,      # 0.05% slippage
    commission_pct=0.075,  # 0.075% commission (Bybit's standard taker fee)
    risk_per_trade_pct=1.0,  # Risk 1% of account per trade
    target_atr_multiplier=0.5,  # Target is 0.5x ATR
```

```
    stop_loss_atr_multiplier=1.0  # Stop loss is 1x ATR
)
```

[22]:
```
# Run backtest for long trades
long_trade_results, long_metrics, long_equity_curve =␣
 ↪backtest_long_trade_strategy(
    df,
    long_trade_opportunities,
    slippage_pct=0.05,      # 0.05% slippage
    commission_pct=0.075,  # 0.075% commission (Bybit's standard taker fee)
    risk_per_trade_pct=1.0,  # Risk 1% of account per trade
    target_atr_multiplier=0.5,  # Target is 0.5x ATR
    stop_loss_atr_multiplier=1.0  # Stop loss is 1x ATR
)
```

[23]:
```
long_trade_results
```

[23]:

	entry_date	entry_price	position_size	position_value	\
0	2024-01-01 13:00:00	42770.37450	0.233924	10000.000000	
1	2024-01-01 18:30:00	43236.10725	0.230439	9958.313422	
2	2024-01-02 01:15:00	45170.07375	0.219458	9907.990650	
3	2024-01-02 01:45:00	45012.59505	0.218006	9808.103896	
4	2024-01-02 06:00:00	45306.14175	0.217009	9826.940907	
…	…	…	…	…	
1453	2024-12-29 05:45:00	95143.04775	0.002365	224.913448	
1454	2024-12-30 01:45:00	93940.34670	0.002388	224.191961	
1455	2024-12-30 06:45:00	93930.04155	0.002374	222.911361	
1456	2024-12-30 18:30:00	94514.03340	0.002349	221.863765	
1457	2024-12-31 03:45:00	92585.16945	0.002400	222.115995	

	target_price	stop_loss	commission	risk_amount	risk_per_trade_pct	\
0	42795.420	42656.16	7.500000	100.000000	1.0	
1	43280.565	43082.37	7.468735	99.583134	1.0	
2	45330.165	44782.17	7.430993	99.079907	1.0	
3	45166.690	44636.92	7.356078	98.081039	1.0	
4	45353.955	45142.59	7.370206	98.269409	1.0	
…	…	…	…	…	…	
1453	95153.020	94980.46	0.168685	2.249134	1.0	
1454	94067.905	93544.39	0.168144	2.241920	1.0	
1455	94010.000	93629.30	0.167184	2.229114	1.0	
1456	94763.465	93873.47	0.166398	2.218638	1.0	
1457	92673.605	92269.49	0.166587	2.221160	1.0	

	exit_date	exit_price	exit_type	profit_loss	return_pct	\
0	2024-01-01 14:00:00	42634.831920	stop_loss	-34.186578	-0.341866	
1	2024-01-01 18:45:00	43060.828815	stop_loss	-42.854036	-0.430334	
2	2024-01-02 01:45:00	44759.778915	stop_loss	-92.455761	-0.933143	

```
3    2024-01-02 02:15:00  45144.106655     target    26.193089    0.267056
4    2024-01-02 07:00:00  45331.278023     target     2.990294    0.030430
…                    …           …         …           …           …
1453 2024-12-29 06:00:00  94932.969770  stop_loss    -0.552802   -0.245784
1454 2024-12-30 02:15:00  93497.617805  stop_loss    -1.112456   -0.496207
1455 2024-12-30 07:15:00  93582.485350  stop_loss    -0.880413   -0.394961
1456 2024-12-30 20:00:00  94716.083267     target     0.418627    0.188687
1457 2024-12-31 04:00:00  92627.268198     target     0.045359    0.020421

          balance  exit_commission  hold_period
0     9958.313422         7.479970         1.00
1     9907.990650         7.442176         0.25
2     9808.103896         7.367177         0.50
3     9826.940907         7.381259         0.50
4     9822.560995         7.377982         1.00
…             …                …            …
1453   224.191961         0.168397         0.25
1454   222.911361         0.167435         0.50
1455   221.863765         0.166648         0.50
1456   222.115995         0.166837         1.50
1457   221.994767         0.166746         0.25

[1458 rows x 17 columns]
```

[24]:
```
long_metrics
```

[24]:
```
{'initial_balance': 10000.0,
 'final_balance': np.float64(221.99476674427322),
 'total_return': np.float64(-9778.005233255726),
 'total_return_pct': np.float64(-97.78005233255726),
 'total_trades': 1458,
 'winning_trades': 595,
 'losing_trades': 863,
 'win_rate': 0.40809327846364885,
 'average_win': np.float64(3.0140497824150834),
 'average_loss': np.float64(-10.041455385779756),
 'largest_win': 33.69245373003449,
 'largest_loss': -101.69264308675248,
 'profit_factor': np.float64(0.20694737793427667),
 'max_drawdown_pct': 97.78136234872754,
 'avg_trade_duration_hours': np.float64(0.637517146776406),
 'total_commission': np.float64(5808.200358073109),
 'expectancy': np.float64(-4.713591479692012)}
```

[25]:
```
long_equity_curve
```

[25]:

date	balance
2024-01-01 08:00:00	10000.000000
2024-01-01 14:00:00	9958.313422
2024-01-01 18:45:00	9907.990650
2024-01-02 01:45:00	9808.103896
2024-01-02 02:15:00	9826.940907
…	…
2024-12-29 06:00:00	224.191961
2024-12-30 02:15:00	222.911361
2024-12-30 07:15:00	221.863765
2024-12-30 20:00:00	222.115995
2024-12-31 04:00:00	221.994767

[1459 rows x 1 columns]

3 ## Cryptocurrency Trading Strategy Backtesting Analysis

This backtest simulation implements a comprehensive framework for evaluating trading strategies on the cryptocurrency market. The analysis follows a systematic approach to simulate real-world trading conditions with precise entry/exit mechanics and risk management protocols.

3.1 Backtesting Methodology

1. Trade Setup and Entry Logic * **Long Trade Setup**: Identifies bullish setups where a candle has a lower high and lower low than the previous candle, is bullish (close > open), and closes above a key moving average (EMA). * **Short Trade Setup**: Identifies bearish setups where a candle has a higher high and higher low than the previous candle, is bearish (close < open), and closes below a key moving average (EMA). * **Entry Points**: For each trade setup identified, the simulation enters at a precise entry price (e.g., for long trades, 1 tick above the setup candle's high).

2. Risk Management and Position Sizing * **Fixed Percentage Risk**: Each trade risks exactly 1% of the current account balance. * **Position Size Calculation**: The size of each position is calculated based on the risk amount and the distance between the entry price and the stop loss.

`Position Size = (Account Balance * 0.01) / Distance between Entry and Stop Loss`

- **Dynamic Scaling**: As the account equity grows or shrinks, the position size for subsequent trades adjusts accordingly.
- **Predetermined Exits**: Both take profit and stop loss levels are determined at the time of entry, often based on a multiplier of the Average True Range (ATR).

3. Trade Execution and Realistic Costs * **Slippage Modeling**: Applies a realistic 0.05% slippage on both entries and exits to simulate the difference between the expected price and the actual fill price. * **Commission Structure**: Applies a 0.075% commission fee on both the entry and exit of a trade to account for transaction costs.

4. Trade Management and Exit Mechanics Once a trade is entered, it remains open until

one of the following exit conditions is met: * **Target Price Hit**: The trade is closed for a profit when the price reaches the predefined take profit level. * **Stop Loss Hit**: The trade is closed for a loss when the price reaches the predefined stop loss level. * **End of Data**: Any open trades at the end of the testing period are closed at the final available market price.

5. Performance Measurement and Tracking * **Equity Curve**: Records the account balance after every closed trade to visualize performance and growth over time. * **Detailed Trade Log**: Maintains a comprehensive log for every trade, including entry/exit dates, prices, and the final profit or loss. * **Key Performance Metrics**: Calculates essential metrics to judge the strategy's effectiveness, including: * Win Rate * Profit Factor (Gross Profit / Gross Loss) * Maximum Drawdown * Average Trade Duration

Strengths of this Strategy * Well-defined and systematic entry and exit rules. * Integrated risk management through fixed-percentage position sizing. * Realistic simulation that accounts for trading friction like slippage and commissions. * Clear and comprehensive performance metrics for objective evaluation.

Limitations and Potential Improvements - No trailing stop loss mechanism to capture larger moves - Fixed target based only on ATR multiplier (could be optimized) - No filter for high impact news events or market conditions - Could implement pyramiding (adding to winning positions) - Could add filters based on higher timeframes for better entry timing

3.2 Results

3.2.1 Trade Statistics

[26]:
```
import numpy as np
import pandas as pd

def analyze_and_display_trade_statistics(trade_results=None, metrics=None,␣
↪trade_type="Long"):
    """
    Display summary statistics and perform deeper analysis for a trading␣
↪strategy.

    Parameters:
    - trade_results: DataFrame containing detailed results of each trade
    - metrics: Dictionary containing aggregated performance metrics
    - trade_type: String indicating the type of trades ("Long" or "Short")

    Returns:
    - None (prints statistics summary and analysis to console)
    """
    import matplotlib.pyplot as plt

    if trade_results is None or metrics is None:
        print(f"No {trade_type.lower()} trade data provided.")
        return
```

```
    # Display basic statistics
    print(f"===== {trade_type.upper()} TRADE BACKTEST SUMMARY =====")
    print(f"Initial Balance: ${metrics['initial_balance']:,.2f}")
    print(f"Final Balance: ${metrics['final_balance']:,.2f}")
    print(f"Total Return: ${metrics['total_return']:,.2f}␣
↪({metrics['total_return_pct']:.2f}%)")

    # Trade statistics
    print(f"\n----- Trade Statistics -----")
    print(f"Total Trades: {metrics['total_trades']}")
    print(f"Winning Trades: {metrics['winning_trades']}␣
↪({metrics['win_rate']*100:.2f}%)")
    print(f"Losing Trades: {metrics['losing_trades']}")
    print(f"Profit Factor: {metrics['profit_factor']:.2f}")
    print(f"Average Win: ${metrics['average_win']:.2f}")
    print(f"Average Loss: ${metrics['average_loss']:.2f}")
    print(f"Largest Win: ${metrics['largest_win']:.2f}")
    print(f"Largest Loss: ${metrics['largest_loss']:.2f}")

    # Risk and duration metrics
    print(f"\n----- Risk Metrics -----")
    print(f"Maximum Drawdown: {metrics['max_drawdown_pct']:.2f}%")
    print(f"Average Trade Duration: {metrics['avg_trade_duration_hours']:.2f}␣
↪hours")
    print(f"Total Commission Paid: ${metrics['total_commission']:.2f}")

    # Calculate additional metrics if we have trade data
    if len(trade_results) > 0:
        # Win/loss ratio
        win_loss_ratio = abs(metrics['average_win'] / metrics['average_loss'])␣
↪if metrics['average_loss'] != 0 else float('inf')

        # Expectancy and Sharpe ratio
        expectancy = metrics['win_rate'] * metrics['average_win'] + (1 -␣
↪metrics['win_rate']) * metrics['average_loss']
        risk_reward_ratio = abs(metrics['average_win'] /␣
↪metrics['average_loss']) if metrics['average_loss'] != 0 else float('inf')

        # Win streaks
        trade_results['is_win'] = trade_results['profit_loss'] > 0
        trade_results['streak_change'] = trade_results['is_win'] !=␣
↪trade_results['is_win'].shift(1)
        trade_results['streak_id'] = trade_results['streak_change'].cumsum()
        streaks = trade_results.groupby(['streak_id', 'is_win']).size()
```

```
        max_win_streak = streaks[streaks.index.get_level_values('is_win') ==␣
↪True].max() if True in streaks.index.get_level_values('is_win') else 0
        max_loss_streak = streaks[streaks.index.get_level_values('is_win') ==␣
↪False].max() if False in streaks.index.get_level_values('is_win') else 0


        print(f"Win/Loss Ratio: {win_loss_ratio:.2f}")
        print(f"Expectancy per Trade: ${expectancy:.2f}")
        print(f"Risk-Reward Ratio: {risk_reward_ratio:.2f}")
        print(f"Maximum Win Streak: {max_win_streak}")
        print(f"Maximum Loss Streak: {max_loss_streak}")

        # Exit type distribution
        exit_counts = trade_results['exit_type'].value_counts()
        print(f"\n----- Exit Types -----")
        for exit_type, count in exit_counts.items():
            print(f"{exit_type}: {count} ({count/len(trade_results)*100:.1f}%)")

        # Time-based analysis
        trade_results['duration'] = (trade_results['exit_date'] -␣
↪trade_results['entry_date'])
        trade_results['duration_hours'] = trade_results['duration'].dt.
↪total_seconds() / 3600

        # Extract time components
        trade_results['hour'] = trade_results['entry_date'].dt.hour
        trade_results['day_of_week'] = trade_results['entry_date'].dt.dayofweek

        # Create bins for trade duration
        duration_bins = [0, 1, 4, 8, 24, 48, float('inf')]
        duration_labels = ['<1h', '1-4h', '4-8h', '8-24h', '1-2d', '>2d']
        trade_results['duration_group'] = pd.
↪cut(trade_results['duration_hours'],
                                              bins=duration_bins,
                                              labels=duration_labels)

        # Group trades by duration - fixing the FutureWarning by adding␣
↪observed=True
        duration_analysis = trade_results.groupby('duration_group',␣
↪observed=True).agg({
            'profit_loss': ['count', 'mean', 'sum'],
            'return_pct': ['mean', 'median']
        })

        # Group trades by hour of day
        hour_analysis = trade_results.groupby('hour').agg({
```

```
            'profit_loss': ['count', 'mean', 'sum'],
            'return_pct': 'mean'
        })

        # Group trades by day of week
        day_analysis = trade_results.groupby('day_of_week').agg({
            'profit_loss': ['count', 'mean', 'sum'],
            'return_pct': 'mean'
        })

        # Calculate additional risk metrics
        sharpe_ratio = (metrics['total_return_pct'] / 100) /␣
↪(trade_results['return_pct'].std() / 100 * np.sqrt(252)) if␣
↪trade_results['return_pct'].std() != 0 else 0
        expectancy_per_dollar = expectancy / abs(metrics['average_loss']) if␣
↪metrics['average_loss'] != 0 else 0

        # Create a dictionary of additional metrics
        additional_metrics = {
            'duration_analysis': duration_analysis,
            'hour_analysis': hour_analysis,
            'day_analysis': day_analysis,
            'sharpe_ratio': sharpe_ratio,
            'expectancy_per_dollar': expectancy_per_dollar
        }

        print(f"\n----- Time-Based Analysis -----")

        print("\n===== DURATION ANALYSIS =====")
        print(additional_metrics['duration_analysis'])

        print("\n===== HOUR ANALYSIS =====")
        print(additional_metrics['hour_analysis'])

        print("\n===== DAY ANALYSIS =====")
        print(additional_metrics['day_analysis'])

        print("\nPerformance by Trade Duration:")
        print(duration_analysis['profit_loss']['mean'].to_string())

        print("\nPerformance by Hour of Day (Top 3 and Bottom 3):")
        hour_performance = hour_analysis['profit_loss']['mean'].
↪sort_values(ascending=False)
        print("Best Hours:")
        print(hour_performance.head(3).to_string())
        print("\nWorst Hours:")
        print(hour_performance.tail(3).to_string())
```

```
        print(f"\n----- Monthly Distribution -----")
        if 'month' not in trade_results.columns:
            trade_results['month'] = trade_results['entry_date'].dt.month

        monthly_distribution = trade_results.groupby('month').agg({
            'profit_loss': ['count', 'sum', 'mean'],
            'is_win': 'mean'
        })

        print("\nTop 3 Best Months:")
        best_months = monthly_distribution['profit_loss']['mean'].
↪sort_values(ascending=False).head(3)
        for month, value in best_months.items():
            print(f"Month {month}: ${value:.2f} avg profit/loss")

        # Display top 5 profitable trades
        print(f"\n----- Top 5 Profitable Trades -----")
        display_cols = ['entry_date', 'exit_date', 'entry_price', 'exit_price',
                        'exit_type', 'profit_loss', 'return_pct', 'hold_period']
        top_trades = trade_results[display_cols].sort_values('profit_loss',␣
↪ascending=False).head(5)
        print(top_trades)

        print(f"\n----- Worst 5 Losing Trades -----")
        worst_trades = trade_results[display_cols].sort_values('profit_loss').
↪head(5)
        print(worst_trades)

        # Additional system quality metrics
        print(f"\n----- System Quality Metrics -----")
        print(f"Sharpe Ratio: {sharpe_ratio:.2f}")
        print(f"Expectancy per Dollar Risked: {expectancy_per_dollar:.2f}")

        # Return distribution statistics
        returns_mean = trade_results['return_pct'].mean()
        returns_std = trade_results['return_pct'].std()
        returns_skew = trade_results['return_pct'].skew()
        returns_kurt = trade_results['return_pct'].kurtosis()

        print(f"\n----- Return Distribution -----")
        print(f"Mean Return: {returns_mean:.3f}%")
        print(f"Standard Deviation: {returns_std:.3f}%")
        print(f"Skewness: {returns_skew:.3f}")
        print(f"Kurtosis: {returns_kurt:.3f}")
```

```
        # System reliability metric
        reliability = metrics['win_rate'] * win_loss_ratio if␣
↪metrics['win_rate'] > 0 else 0
        print(f"System Reliability Score: {reliability:.3f}")
```

Long Trade Stats

[27]:
```
analyze_and_display_trade_statistics(trade_results=long_trade_results,␣
↪metrics=long_metrics, trade_type="Long")
```

```
===== LONG TRADE BACKTEST SUMMARY =====
Initial Balance: $10,000.00
Final Balance: $221.99
Total Return: $-9,778.01 (-97.78%)

----- Trade Statistics -----
Total Trades: 1458
Winning Trades: 595 (40.81%)
Losing Trades: 863
Profit Factor: 0.21
Average Win: $3.01
Average Loss: $-10.04
Largest Win: $33.69
Largest Loss: $-101.69

----- Risk Metrics -----
Maximum Drawdown: 97.78%
Average Trade Duration: 0.64 hours
Total Commission Paid: $5808.20
Win/Loss Ratio: 0.30
Expectancy per Trade: $-4.71
Risk-Reward Ratio: 0.30
Maximum Win Streak: 11
Maximum Loss Streak: 16

----- Exit Types -----
target: 798 (54.7%)
stop_loss: 660 (45.3%)

----- Time-Based Analysis -----

===== DURATION ANALYSIS =====
               profit_loss                          return_pct
                     count       mean          sum       mean    median
duration_group
<1h                   1288  -4.574805 -5892.349269  -0.185354 -0.041262
1-4h                   164  -5.250633  -861.103888  -0.183078 -0.013274
```

```
4-8h                    5  -4.764035   -23.820173  -0.194529  0.039731
8-24h                   1 -95.143047   -95.143047  -1.969998 -1.969998

===== HOUR ANALYSIS =====
     profit_loss                        return_pct
           count      mean         sum       mean
hour
0             56 -5.153805 -288.613057  -0.187732
1             51 -4.830480 -246.354484  -0.147456
2             54 -8.317364 -449.137672  -0.272408
3             61 -6.729961 -410.527616  -0.261099
4             59 -5.045971 -297.712293  -0.233778
5             66 -5.026476 -331.747421  -0.189369
6             57 -3.086576 -175.934853  -0.137602
7             45 -3.347172 -150.622749  -0.147704
8             64 -6.364207 -407.309243  -0.255822
9             66 -3.667706 -242.068611  -0.216201
10            69 -2.027584 -139.903266  -0.089634
11            68 -3.956046 -269.011095  -0.131638
12            66 -3.585421 -236.637753  -0.137388
13            68 -7.670597 -521.600593  -0.204263
14            59 -5.768384 -340.334669  -0.218645
15            74 -6.520884 -482.545383  -0.186356
16            72 -2.828581 -203.657841  -0.144704
17            62 -5.968544 -370.049702  -0.196101
18            77 -3.314564 -255.221390  -0.200504
19            58 -1.919152 -111.310837  -0.170131
20            55 -4.365086 -240.079720  -0.206011
21            45 -2.064178  -92.888002  -0.169454
22            58 -3.245337 -188.229565  -0.133555
23            48 -8.769137 -420.918563  -0.255015

===== DAY ANALYSIS =====
            profit_loss                          return_pct
                  count      mean          sum       mean
day_of_week
0                   208 -4.644607  -966.078243  -0.226293
1                   211 -6.251003 -1318.961651  -0.222501
2                   191 -5.542502 -1058.617855  -0.180872
3                   214 -3.157841  -675.777976  -0.122665
4                   228 -3.364238  -767.046210  -0.174366
5                   208 -5.464131 -1136.539237  -0.187245
6                   198 -4.794925  -949.395206  -0.192867

Performance by Trade Duration:
duration_group
<1h      -4.574805
1-4h     -5.250633
```

```
4-8h     -4.764035
8-24h   -95.143047

Performance by Hour of Day (Top 3 and Bottom 3):
Best Hours:
hour
19   -1.919152
10   -2.027584
21   -2.064178

Worst Hours:
hour
13   -7.670597
2    -8.317364
23   -8.769137

----- Monthly Distribution -----

Top 3 Best Months:
Month 12: $-0.68 avg profit/loss
Month 11: $-0.79 avg profit/loss
Month 10: $-0.96 avg profit/loss

----- Top 5 Profitable Trades -----
            entry_date           exit_date  entry_price    exit_price  \
39 2024-01-11 18:15:00 2024-01-11 19:45:00  46519.64820  46728.743940
40 2024-01-11 18:45:00 2024-01-11 20:00:00  46680.52860  46831.932320
3  2024-01-02 01:45:00 2024-01-02 02:15:00  45012.59505  45144.106655
18 2024-01-04 21:45:00 2024-01-04 22:00:00  44483.53065  44605.406140
34 2024-01-10 17:00:00 2024-01-10 17:15:00  45514.84605  45644.031568

   exit_type  profit_loss  return_pct  hold_period
39    target    33.692454    0.424328         1.50
40    target    26.216569    0.299221         1.25
3     target    26.193089    0.267056         0.50
18    target    23.194860    0.248873         0.25
34    target    22.725115    0.258723         0.25

----- Worst 5 Losing Trades -----
            entry_date           exit_date  entry_price    exit_price  \
12 2024-01-03 17:45:00 2024-01-03 18:30:00  42963.87120  42513.362685
32 2024-01-09 23:15:00 2024-01-10 08:45:00  46219.69830  45320.518405
2  2024-01-02 01:15:00 2024-01-02 01:45:00  45170.07375  44759.778915
28 2024-01-08 13:15:00 2024-01-08 14:45:00  45199.38840  44772.652475
6  2024-01-02 08:45:00 2024-01-02 10:00:00  45876.72690  45587.914640

    exit_type  profit_loss  return_pct  hold_period
12  stop_loss  -101.692643   -1.073350         0.75
```

```
32  stop_loss   -95.143047   -1.969998         9.50
2   stop_loss   -92.455761   -0.933143         0.50
28  stop_loss   -87.974929   -0.968920         1.50
6   stop_loss   -64.275740   -0.654420         1.25

----- System Quality Metrics -----
Sharpe Ratio: -19.68
Expectancy per Dollar Risked: -0.47

----- Return Distribution -----
Mean Return: -0.186%
Standard Deviation: 0.313%
Skewness: -0.648
Kurtosis: 0.351
System Reliability Score: 0.122
```

Short Trade Stats

[28]:
```
analyze_and_display_trade_statistics(trade_results=short_trade_results,␣
↪metrics=short_metrics, trade_type="Short")
```

```
===== SHORT TRADE BACKTEST SUMMARY =====
Initial Balance: $10,000.00
Final Balance: $383.68
Total Return: $-9,616.32 (-96.16%)

----- Trade Statistics -----
Total Trades: 1306
Winning Trades: 562 (43.03%)
Losing Trades: 744
Profit Factor: 0.24
Average Win: $3.70
Average Loss: $-11.71
Largest Win: $42.90
Largest Loss: $-80.39

----- Risk Metrics -----
Maximum Drawdown: 96.16%
Average Trade Duration: 0.62 hours
Total Commission Paid: $5973.45
Win/Loss Ratio: 0.32
Expectancy per Trade: $-5.08
Risk-Reward Ratio: 0.32
Maximum Win Streak: 12
Maximum Loss Streak: 17

----- Exit Types -----
target: 752 (57.6%)
stop_loss: 554 (42.4%)
```

```
----- Time-Based Analysis -----

===== DURATION ANALYSIS =====
               profit_loss                          return_pct
                     count      mean          sum       mean    median
duration_group
<1h                   1158 -5.174947 -5992.588050  -0.171862 -0.028488
1-4h                   145 -4.205832  -609.845652  -0.184441 -0.033934
4-8h                     3 -9.508800   -28.526400  -0.097317  0.022895

===== HOUR ANALYSIS =====
     profit_loss                         return_pct
           count      mean         sum       mean
hour
0             45 -1.660849  -74.738212  -0.045790
1             58 -4.018627 -233.080345  -0.152875
2             59 -3.493372 -206.108973  -0.131565
3             55 -4.229636 -232.629964  -0.128592
4             56 -3.550192 -198.810747  -0.151561
5             49 -5.293817 -259.397022  -0.196877
6             54 -4.800969 -259.252326  -0.172288
7             38 -2.710445 -102.996926  -0.154556
8             49 -3.735373 -183.033278  -0.165315
9             46 -3.260964 -150.004331  -0.139402
10            47 -4.168238 -195.907200  -0.222396
11            59 -7.269367 -428.892679  -0.217510
12            56 -7.268812 -407.053456  -0.168994
13            45 -3.291641 -148.123851  -0.124785
14            56 -6.774149 -379.352371  -0.195495
15            69 -5.613079 -387.302475  -0.159622
16            75 -4.011027 -300.827035  -0.140611
17            58 -4.901899 -284.310156  -0.233381
18            50 -5.878899 -293.944954  -0.186036
19            59 -6.021707 -355.280725  -0.167652
20            52 -9.016153 -468.839959  -0.256471
21            49 -7.194286 -352.520005  -0.233407
22            54 -7.876828 -425.348720  -0.231273
23            68 -4.458888 -303.204395  -0.170621

===== DAY ANALYSIS =====
            profit_loss                          return_pct
                  count      mean          sum       mean
day_of_week
0                   198 -3.890294  -770.278248  -0.152906
1                   213 -6.706838 -1428.556582  -0.218520
2                   199 -4.131901  -822.248309  -0.151524
3                   189 -5.606975 -1059.718219  -0.195299
```

```
4                   158 -4.216387  -666.189154  -0.133343
5                   179 -5.735518 -1026.657634  -0.181176
6                   170 -5.043012  -857.311956  -0.168638

Performance by Trade Duration:
duration_group
<1h    -5.174947
1-4h   -4.205832
4-8h   -9.508800

Performance by Hour of Day (Top 3 and Bottom 3):
Best Hours:
hour
0   -1.660849
7   -2.710445
9   -3.260964

Worst Hours:
hour
11   -7.269367
22   -7.876828
20   -9.016153

----- Monthly Distribution -----

Top 3 Best Months:
Month 12: $-0.76 avg profit/loss
Month 11: $-1.18 avg profit/loss
Month 10: $-1.39 avg profit/loss

----- Top 5 Profitable Trades -----
             entry_date           exit_date  entry_price    exit_price  \
11  2024-01-03 13:00:00 2024-01-03 14:15:00  42484.34720  42112.640797
28  2024-01-10 14:30:00 2024-01-10 15:00:00  45059.05920  44856.702142
40  2024-01-12 18:00:00 2024-01-12 18:30:00  43654.56180  43467.027653
13  2024-01-03 17:30:00 2024-01-03 18:30:00  42740.81890  42584.496608
166 2024-02-13 15:30:00 2024-02-13 16:45:00  48751.11225  48546.826282

    exit_type  profit_loss  return_pct  hold_period
11     target    42.897220    0.850181         1.25
28     target    36.818062    0.424243         0.50
40     target    36.034910    0.404731         0.50
13     target    33.349991    0.340874         1.00
166    target    25.826181    0.394181         1.25

----- Worst 5 Losing Trades -----
            entry_date           exit_date  entry_price    exit_price  \
69 2024-01-18 20:45:00 2024-01-18 21:45:00  40824.37760  41197.078245
```

```
16 2024-01-05 14:45:00 2024-01-05 15:00:00  43401.78825  43745.021580
58 2024-01-16 15:00:00 2024-01-16 15:15:00  42510.73400  42870.344460
12 2024-01-03 15:30:00 2024-01-03 15:45:00  42345.31675  42664.971825
41 2024-01-12 19:30:00 2024-01-12 20:15:00  43496.14105  43832.335215

    exit_type  profit_loss  return_pct  hold_period
69  stop_loss   -80.385775   -0.938127         1.00
16  stop_loss   -80.077082   -0.815988         0.25
58  stop_loss   -77.084282   -0.871102         0.25
12  stop_loss   -76.973622   -0.780028         0.25
41  stop_loss   -75.427574   -0.798084         0.75

----- System Quality Metrics -----
Sharpe Ratio: -17.75
Expectancy per Dollar Risked: -0.43

----- Return Distribution -----
Mean Return: -0.173%
Standard Deviation: 0.341%
Skewness: -0.570
Kurtosis: 0.313
System Reliability Score: 0.136
```

3.2.2 Visualizations of Trade Statistics

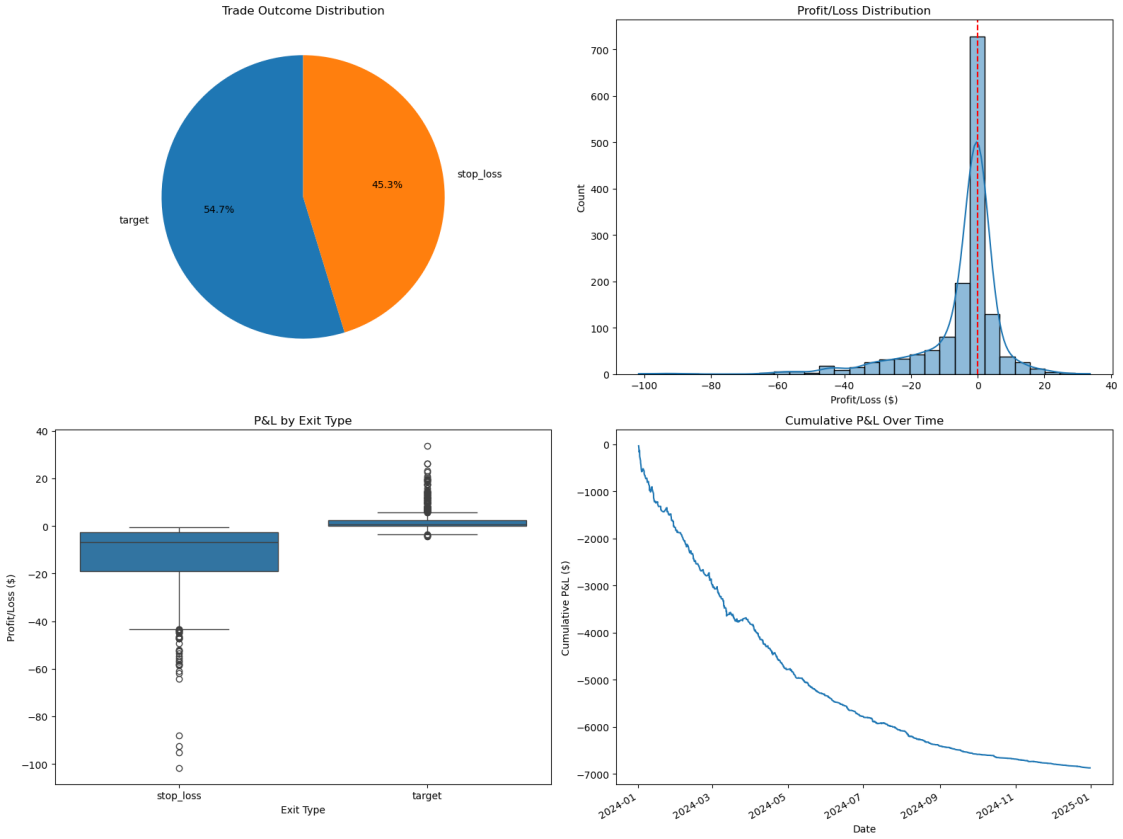
[29]:
```python
# Function to visualize trade outcomes
def visualize_trade_outcomes(backtest_results):
    """
    Visualizes the distribution of trade outcomes, profit/loss per trade, and␣
 ↪drawdown periods.

    Parameters:
    - backtest_results: DataFrame containing individual trade results from the␣
 ↪backtest

    Returns:
    - None (generates plots)
    """
    import matplotlib.pyplot as plt
    import numpy as np
    import seaborn as sns

    # Check if there are trades to analyze
    if len(backtest_results) == 0:
        print("No trades to visualize.")
        return
```

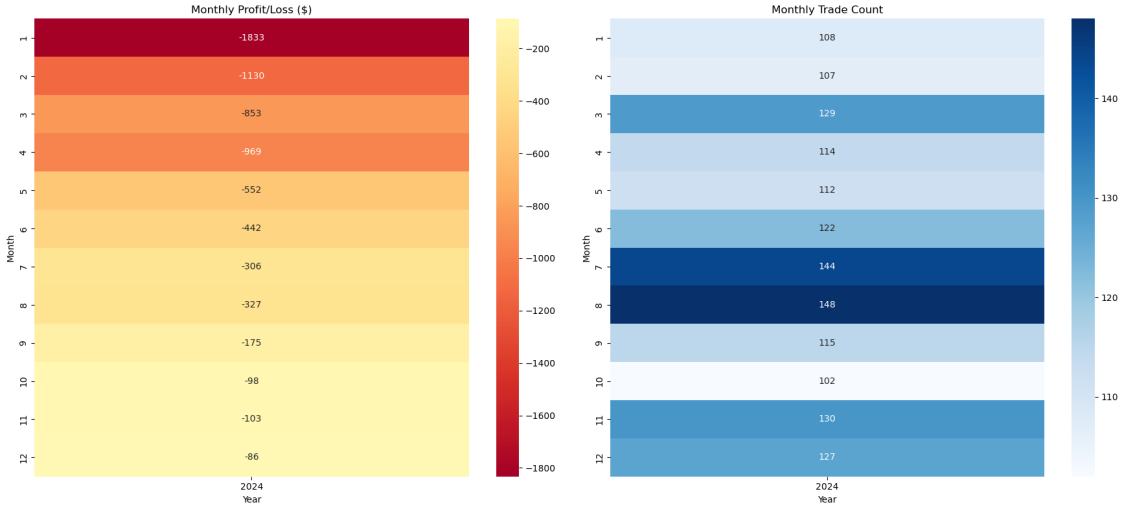
```
    # Create a figure with multiple subplots
    fig, axes = plt.subplots(2, 2, figsize=(16, 12))

    # 1. Trade Outcome Distribution (Pie Chart)
    outcome_counts = backtest_results['exit_type'].value_counts()
    axes[0, 0].pie(outcome_counts, labels=outcome_counts.index, autopct='%1.
↪1f%%', startangle=90)
    axes[0, 0].set_title('Trade Outcome Distribution')

    # 2. Profit/Loss Distribution (Histogram)
    sns.histplot(backtest_results['profit_loss'], bins=30, kde=True, ax=axes[0,␣
↪1])
    axes[0, 1].axvline(x=0, color='r', linestyle='--')
    axes[0, 1].set_title('Profit/Loss Distribution')
    axes[0, 1].set_xlabel('Profit/Loss ($)')

    # 3. P&L by Exit Type (Box plot)
    sns.boxplot(x='exit_type', y='profit_loss', data=backtest_results,␣
↪ax=axes[1, 0])
    axes[1, 0].set_title('P&L by Exit Type')
    axes[1, 0].set_xlabel('Exit Type')
    axes[1, 0].set_ylabel('Profit/Loss ($)')

    # 4. Cumulative P&L Over Time
    backtest_results.sort_values('exit_date', inplace=True)
    backtest_results['cumulative_pnl'] = backtest_results['profit_loss'].
↪cumsum()
    backtest_results.plot(x='exit_date', y='cumulative_pnl', ax=axes[1, 1],␣
↪legend=False)
    axes[1, 1].set_title('Cumulative P&L Over Time')
    axes[1, 1].set_xlabel('Date')
    axes[1, 1].set_ylabel('Cumulative P&L ($)')

    plt.tight_layout()
    plt.show()

    # Additional analysis: Monthly performance heatmap
    if len(backtest_results) > 10:
        backtest_results['year'] = backtest_results['exit_date'].dt.year
        backtest_results['month'] = backtest_results['exit_date'].dt.month

        # Group by year and month to get performance metrics
        monthly_perf = backtest_results.groupby(['year', 'month']).agg({
            'profit_loss': 'sum',
            'entry_date': 'count'  # Count of trades
        }).reset_index()
```

```
        monthly_perf = monthly_perf.rename(columns={'entry_date': 'num_trades'})

        # Create pivot tables for heatmaps
        profit_pivot = monthly_perf.pivot(index='month', columns='year',␣
↪values='profit_loss')
        trades_pivot = monthly_perf.pivot(index='month', columns='year',␣
↪values='num_trades')

        # Create heatmaps
        fig, axes = plt.subplots(1, 2, figsize=(18, 8))

        sns.heatmap(profit_pivot, cmap='RdYlGn', center=0, annot=True, fmt='.
↪0f', ax=axes[0])
        axes[0].set_title('Monthly Profit/Loss ($)')
        axes[0].set_xlabel('Year')
        axes[0].set_ylabel('Month')

        sns.heatmap(trades_pivot, cmap='Blues', annot=True, fmt='d', ax=axes[1])
        axes[1].set_title('Monthly Trade Count')
        axes[1].set_xlabel('Year')
        axes[1].set_ylabel('Month')

        plt.tight_layout()
        plt.show()
```

Display Long Trade Stats

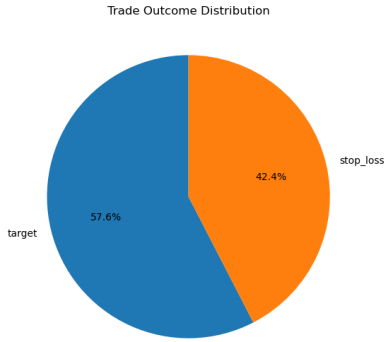
[30]:
```
visualize_trade_outcomes(long_trade_results)
```

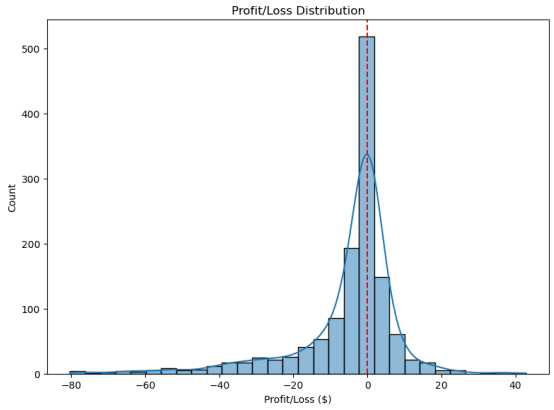


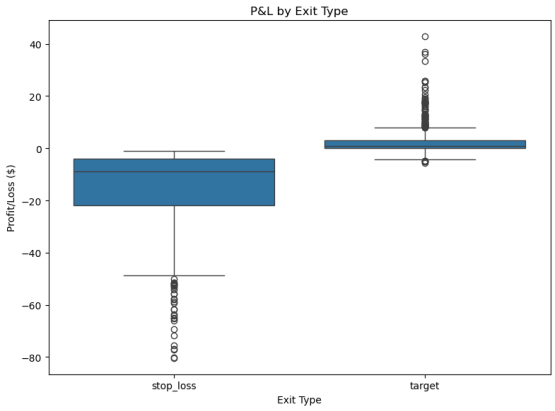


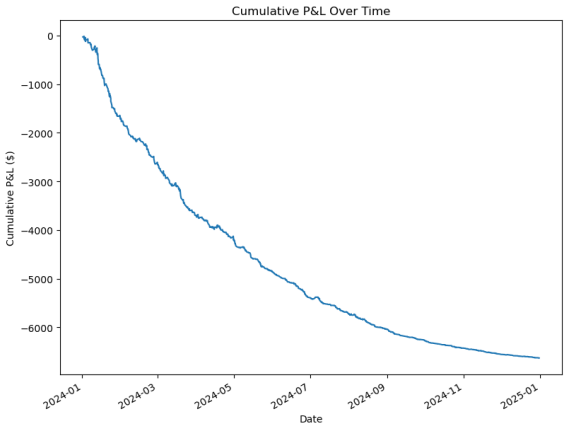
Display Short Trade Stats

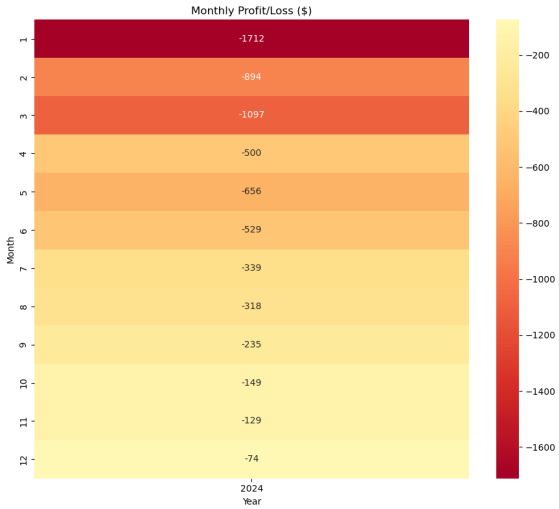
[31]:
```
visualize_trade_outcomes(short_trade_results)
```

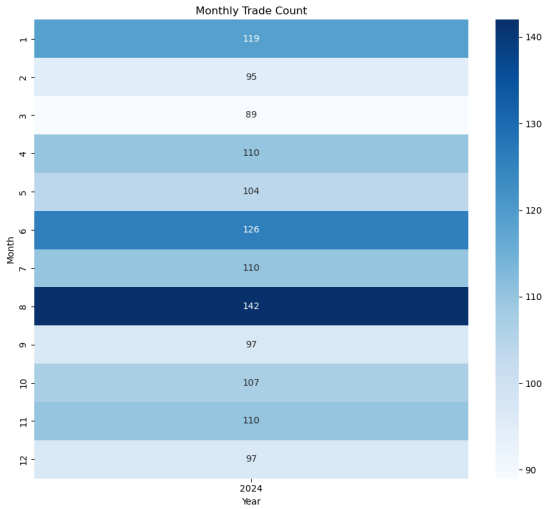

Trade Outcome Distribution
stop_loss
42.4%
57.6%
target


Profit/Loss Distribution
Count
Profit/Loss ($)


P&L by Exit Type
Profit/Loss ($)
stop_loss
target
Exit Type


Cumulative P&L Over Time
Cumulative P&L ($)
Date


Monthly Profit/Loss ($)
-1712
-894
-1097
-500
-656
-529
-339
-318
-235
-149
-129
-74
Month
Year


Monthly Trade Count
119
95
89
110
104
126
110
142
97
107
110
97
Month
Year

3.2.3 Long vs Short Trades Statistics

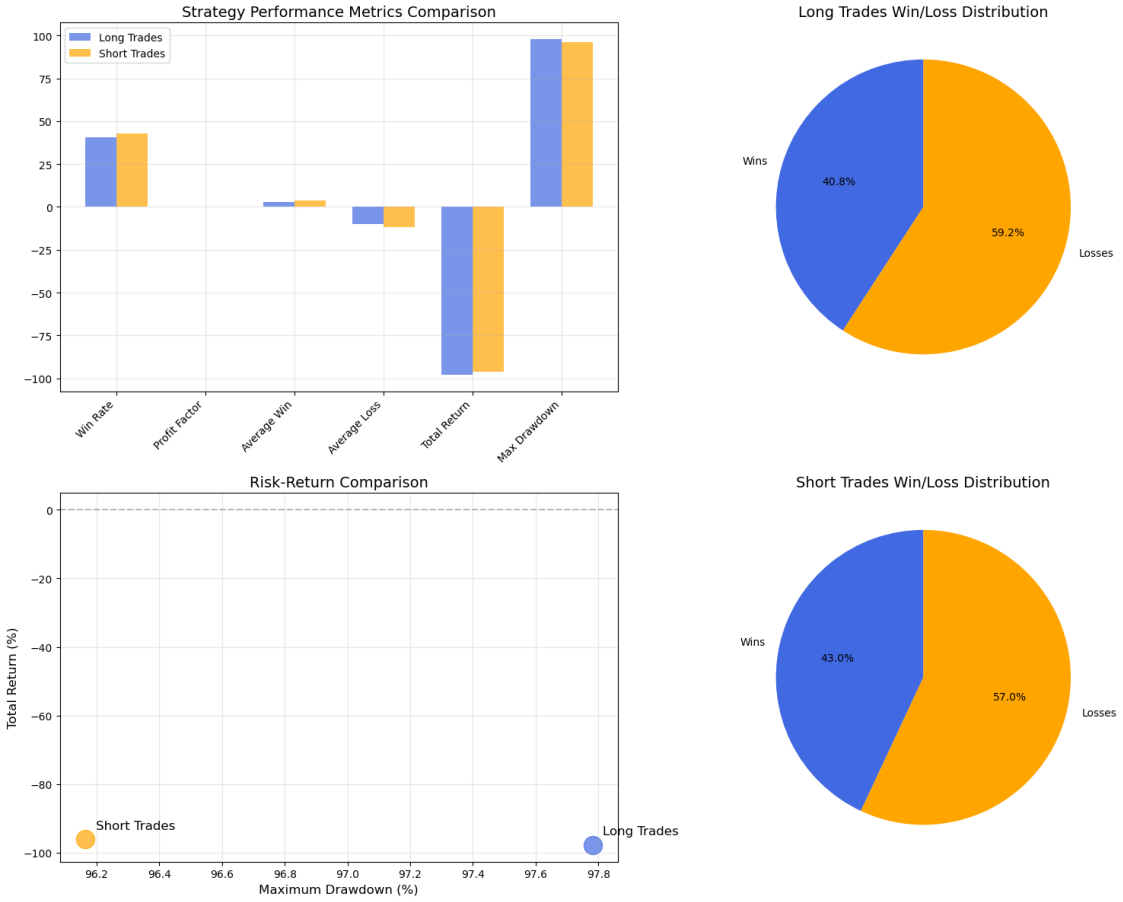
[32]:
```
def visualize_trading_statistics(long_metrics=None, short_metrics=None):
    """
    Creates a visual comparison of long and short trading statistics.

    Parameters:
    - long_metrics: Dictionary containing performance metrics for long trades
    - short_metrics: Dictionary containing performance metrics for short trades

    Returns:
    - None (displays visualizations)
    """
    # Check if we have metrics to visualize
    if long_metrics is None and short_metrics is None:
        print("No trading metrics available to visualize.")
        return

    # Set up the figure with subplots
    fig, axs = plt.subplots(2, 2, figsize=(16, 12))

    # Common metrics to compare
    metrics_to_plot = [
        ('win_rate', 'Win Rate', '%'),
        ('profit_factor', 'Profit Factor', ''),
        ('average_win', 'Average Win', '$'),
        ('average_loss', 'Average Loss', '$'),
        ('total_return_pct', 'Total Return', '%'),
        ('max_drawdown_pct', 'Max Drawdown', '%')
    ]

    # Create data for bar chart comparison
    labels = []
    long_values = []
    short_values = []

    for metric, label, _ in metrics_to_plot:
        labels.append(label)

        if long_metrics is not None and metric in long_metrics:
            if metric == 'win_rate':
                long_values.append(long_metrics[metric] * 100)  # Convert to␣
↪percentage
            else:
                long_values.append(long_metrics[metric])
        else:
            long_values.append(0)
```

```
        if short_metrics is not None and metric in short_metrics:
            if metric == 'win_rate':
                short_values.append(short_metrics[metric] * 100)  # Convert to␣
↪percentage
            else:
                short_values.append(short_metrics[metric])
        else:
            short_values.append(0)

    # 1. Bar chart comparing key metrics
    x = np.arange(len(labels))
    width = 0.35

    axs[0, 0].bar(x - width/2, long_values, width, label='Long Trades',␣
↪color='royalblue', alpha=0.7)
    axs[0, 0].bar(x + width/2, short_values, width, label='Short Trades',␣
↪color='orange', alpha=0.7)

    axs[0, 0].set_title('Strategy Performance Metrics Comparison', fontsize=14)
    axs[0, 0].set_xticks(x)
    axs[0, 0].set_xticklabels(labels, rotation=45, ha='right')
    axs[0, 0].legend()
    axs[0, 0].grid(True, alpha=0.3)

    # 2. Pie charts for win/loss distribution
    if long_metrics is not None:
        long_win_loss = [long_metrics['winning_trades'],␣
↪long_metrics['losing_trades']]
        axs[0, 1].pie(long_win_loss, labels=['Wins', 'Losses'], autopct='%1.
↪1f%%',
                      colors=['royalblue', 'orange'], startangle=90)
        axs[0, 1].set_title('Long Trades Win/Loss Distribution', fontsize=14)

    if short_metrics is not None:
        short_win_loss = [short_metrics['winning_trades'],␣
↪short_metrics['losing_trades']]
        axs[1, 1].pie(short_win_loss, labels=['Wins', 'Losses'], autopct='%1.
↪1f%%',
                      colors=['royalblue', 'orange'], startangle=90)
        axs[1, 1].set_title('Short Trades Win/Loss Distribution', fontsize=14)

    # 3. Risk-reward visualization
    if long_metrics is not None and short_metrics is not None:
        # Create data points for risk-reward comparison
        strategies = ['Long Trades', 'Short Trades']
```

```
        returns = [long_metrics['total_return_pct'],␣
↪short_metrics['total_return_pct']]
        drawdowns = [long_metrics['max_drawdown_pct'],␣
↪short_metrics['max_drawdown_pct']]

        scatter_colors = ['royalblue', 'orange']
        for i, strat in enumerate(strategies):
            axs[1, 0].scatter(drawdowns[i], returns[i], s=300, alpha=0.7,␣
↪color=scatter_colors[i], label=strat)
            axs[1, 0].annotate(strat, (drawdowns[i], returns[i]), xytext=(10,␣
↪10),
                               textcoords='offset points', fontsize=12)

        axs[1, 0].set_title('Risk-Return Comparison', fontsize=14)
        axs[1, 0].set_xlabel('Maximum Drawdown (%)', fontsize=12)
        axs[1, 0].set_ylabel('Total Return (%)', fontsize=12)
        axs[1, 0].grid(True, alpha=0.3)
        axs[1, 0].axhline(y=0, color='gray', linestyle='--', alpha=0.5)

    plt.tight_layout()
    plt.show()

    # Print summary explanation
    print("=== TRADING STATISTICS EXPLANATION ===")
    print("\nThe visualizations above show:")
    print("1. Top Left: Bar chart comparing key performance metrics between␣
↪long and short trades")
    print("2. Top Right: Pie chart showing win/loss distribution for long␣
↪trades")
    print("3. Bottom Left: Risk-return scatter plot comparing drawdown vs.␣
↪return")
    print("4. Bottom Right: Pie chart showing win/loss distribution for short␣
↪trades")

    print("\n=== KEY METRICS EXPLAINED ===")
    print("• Win Rate: Percentage of trades that were profitable")
    print("• Profit Factor: Ratio of gross profits to gross losses (>1 is␣
↪profitable)")
    print("• Average Win: Average profit on winning trades")
    print("• Average Loss: Average loss on losing trades")
    print("• Total Return: Percentage gain/loss over the backtest period")
    print("• Max Drawdown: Largest peak-to-trough decline in account value")

    # Calculate and print additional analytics
    if long_metrics is not None and short_metrics is not None:
        print("\n=== STRATEGY COMPARISON ===")
```

```
        better_win_rate = "Long" if long_metrics['win_rate'] >␣
↪short_metrics['win_rate'] else "Short"
        better_profit_factor = "Long" if long_metrics['profit_factor'] >␣
↪short_metrics['profit_factor'] else "Short"
        better_return = "Long" if long_metrics['total_return_pct'] >␣
↪short_metrics['total_return_pct'] else "Short"
        lower_drawdown = "Long" if long_metrics['max_drawdown_pct'] <␣
↪short_metrics['max_drawdown_pct'] else "Short"

        print(f"• Better Win Rate: {better_win_rate} strategy")
        print(f"• Better Profit Factor: {better_profit_factor} strategy")
        print(f"• Better Total Return: {better_return} strategy")
        print(f"• Lower Maximum Drawdown: {lower_drawdown} strategy")
```

[33]:
```
visualize_trading_statistics(long_metrics=long_metrics,␣
↪short_metrics=short_metrics)
```




```
=== TRADING STATISTICS EXPLANATION ===
```

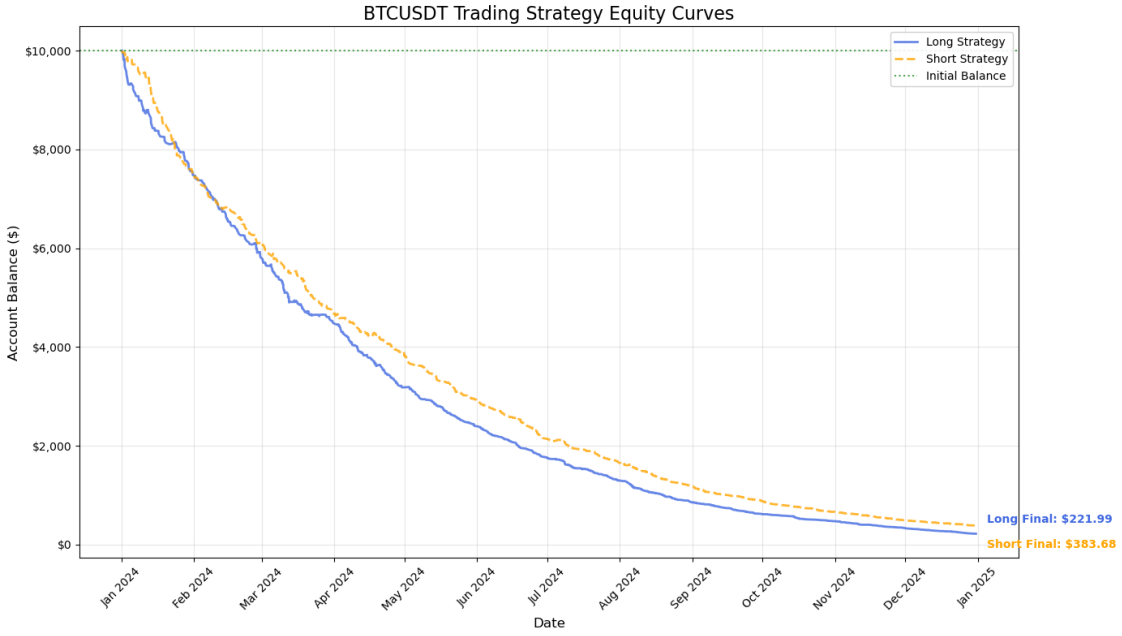
```
The visualizations above show:
1. Top Left: Bar chart comparing key performance metrics between long and short
trades
2. Top Right: Pie chart showing win/loss distribution for long trades
3. Bottom Left: Risk-return scatter plot comparing drawdown vs. return
4. Bottom Right: Pie chart showing win/loss distribution for short trades

=== KEY METRICS EXPLAINED ===
• Win Rate: Percentage of trades that were profitable
• Profit Factor: Ratio of gross profits to gross losses (>1 is profitable)
• Average Win: Average profit on winning trades
• Average Loss: Average loss on losing trades
• Total Return: Percentage gain/loss over the backtest period
• Max Drawdown: Largest peak-to-trough decline in account value

=== STRATEGY COMPARISON ===
• Better Win Rate: Short strategy
• Better Profit Factor: Short strategy
• Better Total Return: Short strategy
• Lower Maximum Drawdown: Short strategy
```

Equity Curve Plot

[34]:
```python
def plot_combined_equity_curves(long_equity_curve=None,␣
↪short_equity_curve=None, plot_title = ''):
    """
    Plot equity curves for long trades and short trades with improved styling

    Parameters:
    - long_equity_curve: DataFrame containing long trade equity curve (optional)
    - short_equity_curve: DataFrame containing short trade equity curve␣
↪(optional)

    Returns:
    - None (displays plots)
    """
    import matplotlib.pyplot as plt
    import matplotlib.dates as mdates

    plt.figure(figsize=(14, 8))

    # Check if we have data for each curve
    has_long = long_equity_curve is not None and not long_equity_curve.empty
    has_short = short_equity_curve is not None and not short_equity_curve.empty

    # Plot individual curves with improved styling
    if has_long:
        plt.plot(long_equity_curve.index, long_equity_curve['balance'],
```

```
                linestyle='-', linewidth=2, color='royalblue',
                label='Long Strategy', alpha=0.8)

    if has_short:
        plt.plot(short_equity_curve.index, short_equity_curve['balance'],
                linestyle='--', linewidth=2, color='orange',
                label='Short Strategy', alpha=0.8)

    # Add horizontal line at initial balance
    plt.axhline(y=10000, color='green', linestyle=':', linewidth=1.5, alpha=0.
↪7, label='Initial Balance')

    # Add chart details
    plt.title(f'{plot_title} Trading Strategy Equity Curves', fontsize=16)
    plt.xlabel('Date', fontsize=12)
    plt.ylabel('Account Balance ($)', fontsize=12)
    plt.legend(loc='upper right', frameon=True, framealpha=0.9)
    plt.grid(True, alpha=0.3)

    # Format x-axis with dates
    plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%b %Y'))
    plt.gca().xaxis.set_major_locator(mdates.MonthLocator(interval=1))
    plt.xticks(rotation=45)

    # Format y-axis with dollar signs
    plt.gca().yaxis.set_major_formatter(plt.matplotlib.ticker.
↪StrMethodFormatter('${x:,.0f}'))

    # Add annotations if we have data
    if has_long:
        long_final = long_equity_curve['balance'].iloc[-1]
        plt.annotate(f'Long Final: ${long_final:.2f}',
                    xy=(long_equity_curve.index[-1], long_final),
                    xytext=(10, 10), textcoords='offset points',
                    color='royalblue', fontweight='bold')

    if has_short:
        short_final = short_equity_curve['balance'].iloc[-1]
        plt.annotate(f'Short Final: ${short_final:.2f}',
                    xy=(short_equity_curve.index[-1], short_final),
                    xytext=(10, -20), textcoords='offset points',
                    color='orange', fontweight='bold')

    plt.tight_layout()
    plt.show()
```

[35]:
```
plot_combined_equity_curves(long_equity_curve=long_equity_curve,␣
↪short_equity_curve=short_equity_curve, plot_title='BTCUSDT')
```



Analysis of Seasonality Patterns

[36]:
```
import seaborn as sns
import pandas as pd
import numpy as np
from matplotlib.colors import LinearSegmentedColormap

def analyze_trading_seasonality_patterns(long_trade_results,␣
↪short_trade_results):
    """
    Analyzes time-based patterns in trading performance and creates␣
↪visualizations
    showing when trades perform best throughout the day/week/month.

    Parameters:
    - long_trade_results: DataFrame containing long trade results
    - short_trade_results: DataFrame containing short trade results

    Returns:
    - combined_df: DataFrame containing combined trade results for analysis
    """
    import matplotlib.pyplot as plt
```

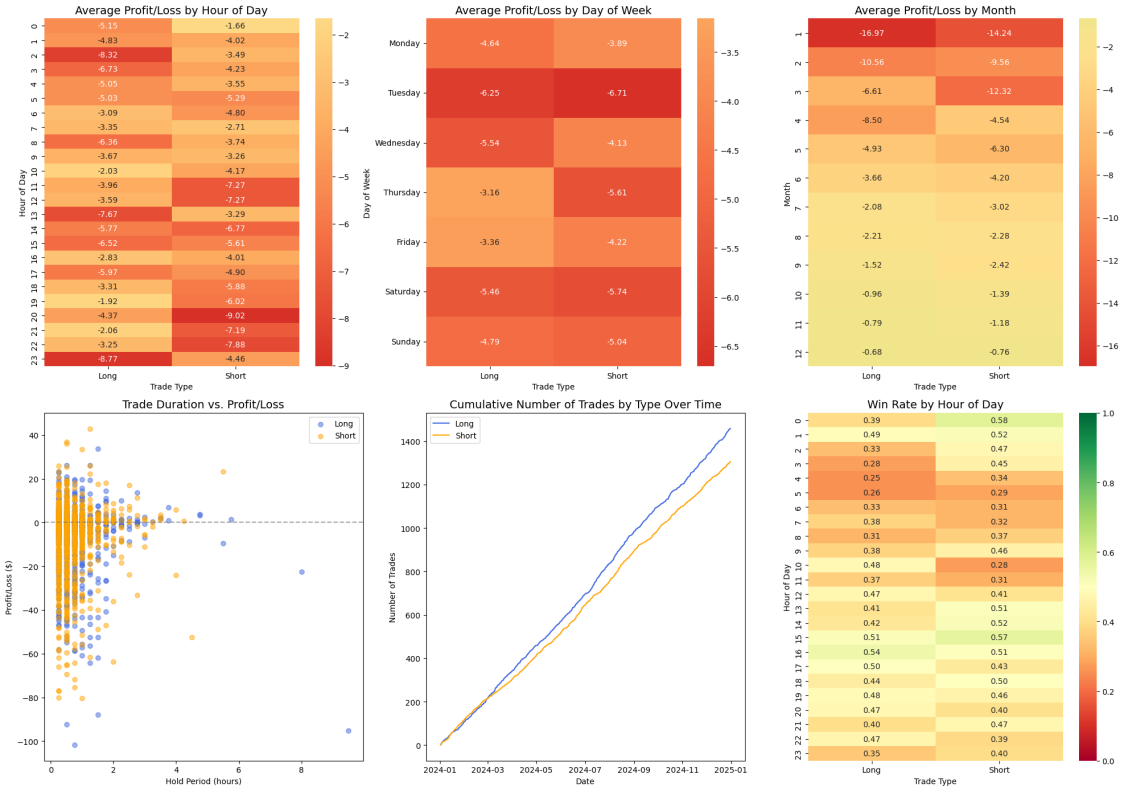
```
    # Create combined dataframe for analysis
    long_df = long_trade_results.copy() if long_trade_results is not None else␣
↪pd.DataFrame()
    short_df = short_trade_results.copy() if short_trade_results is not None␣
↪else pd.DataFrame()

    if len(long_df) > 0:
        long_df['trade_type'] = 'Long'

    if len(short_df) > 0:
        short_df['trade_type'] = 'Short'

    # Combine dataframes
    combined_df = pd.concat([long_df, short_df], ignore_index=True) if␣
↪len(long_df) > 0 and len(short_df) > 0 else (long_df if len(long_df) > 0␣
↪else short_df)

    # Ensure we have data to analyze
    if len(combined_df) == 0:
        print("No trade data available for analysis.")
        return

    # Create custom colormap for profit/loss
    colors = ["#d73027", "#f46d43", "#fdae61", "#fee08b", "#d9ef8b", "#a6d96a",␣
↪"#66bd63", "#1a9850"]
    n_colors = 256
    cmap = LinearSegmentedColormap.from_list("profit_loss_cmap", colors,␣
↪N=n_colors)

    fig, axes = plt.subplots(2, 3, figsize=(20, 14))

    # Extract time components
    combined_df['hour'] = combined_df['entry_date'].dt.hour
    combined_df['day_of_week'] = combined_df['entry_date'].dt.dayofweek
    combined_df['day_name'] = combined_df['entry_date'].dt.day_name()
    combined_df['month'] = combined_df['entry_date'].dt.month
    combined_df['week_of_year'] = combined_df['entry_date'].dt.isocalendar().
↪week


    # 1. Hourly performance heatmap
    hourly_perf = combined_df.pivot_table(
        values='profit_loss',
        index='hour',
        columns='trade_type',
        aggfunc='mean'
```

```
    ).fillna(0)

    sns.heatmap(hourly_perf, cmap=cmap, center=0, annot=True, fmt=".2f",␣
↪ax=axes[0, 0])
    axes[0, 0].set_title('Average Profit/Loss by Hour of Day', fontsize=14)
    axes[0, 0].set_xlabel('Trade Type')
    axes[0, 0].set_ylabel('Hour of Day')

    # 2. Day of week performance
    day_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',␣
↪'Saturday', 'Sunday']
    # First create a categorical column
    combined_df['day_name_ordered'] = pd.Categorical(combined_df['day_name'],␣
↪categories=day_order, ordered=True)
    # Then use it for pivot table
    day_perf = combined_df.pivot_table(
        values='profit_loss',
        index='day_name_ordered',
        columns='trade_type',
        aggfunc='mean',
        observed=True
    ).fillna(0)

    sns.heatmap(day_perf, cmap=cmap, center=0, annot=True, fmt=".2f",␣
↪ax=axes[0, 1])
    axes[0, 1].set_title('Average Profit/Loss by Day of Week', fontsize=14)
    axes[0, 1].set_xlabel('Trade Type')
    axes[0, 1].set_ylabel('Day of Week')

    # 3. Monthly performance
    monthly_perf = combined_df.pivot_table(
        values='profit_loss',
        index='month',
        columns='trade_type',
        aggfunc='mean'
    ).fillna(0)

    sns.heatmap(monthly_perf, cmap=cmap, center=0, annot=True, fmt=".2f",␣
↪ax=axes[0, 2])
    axes[0, 2].set_title('Average Profit/Loss by Month', fontsize=14)
    axes[0, 2].set_xlabel('Trade Type')
    axes[0, 2].set_ylabel('Month')

    # 4. Trade duration vs. profit/loss scatter
    for trade_type, color in zip(['Long', 'Short'], ['royalblue', 'orange']):
        if trade_type in combined_df['trade_type'].values:
            subset = combined_df[combined_df['trade_type'] == trade_type]
```

```
            axes[1, 0].scatter(
                subset['hold_period'],
                subset['profit_loss'],
                alpha=0.5,
                label=trade_type,
                color=color
            )

    axes[1, 0].axhline(y=0, color='gray', linestyle='--', alpha=0.7)
    axes[1, 0].set_title('Trade Duration vs. Profit/Loss', fontsize=14)
    axes[1, 0].set_xlabel('Hold Period (hours)')
    axes[1, 0].set_ylabel('Profit/Loss ($)')
    axes[1, 0].legend()

    # 5. Cumulative trades over time
    combined_df = combined_df.sort_values('entry_date')

    # Create separate cumulative counts for each trade type
    for trade_type in combined_df['trade_type'].unique():
        mask = combined_df['trade_type'] == trade_type
        combined_df.loc[mask, f'cumulative_{trade_type.lower()}'] = range(1,␣
↪mask.sum() + 1)

    # Plot the cumulative trades for each type
    for trade_type, color in zip(['Long', 'Short'], ['royalblue', 'orange']):
        if trade_type in combined_df['trade_type'].values:
            subset = combined_df[combined_df['trade_type'] == trade_type]
            axes[1, 1].plot(
                subset['entry_date'],
                subset[f'cumulative_{trade_type.lower()}'],
                label=trade_type,
                color=color
            )

    axes[1, 1].set_title('Cumulative Number of Trades by Type Over Time',␣
↪fontsize=14)
    axes[1, 1].set_xlabel('Date')
    axes[1, 1].set_ylabel('Number of Trades')
    axes[1, 1].legend()

    # 6. Win rate by time of day
    hourly_win_rate = combined_df.pivot_table(
        values='is_win' if 'is_win' in combined_df.columns else␣
↪combined_df['profit_loss'] > 0,
        index='hour',
        columns='trade_type',
        aggfunc='mean'
```

```
    ).fillna(0)

    sns.heatmap(hourly_win_rate, cmap='RdYlGn', vmin=0, vmax=1, annot=True,␣
↪fmt=".2f", ax=axes[1, 2])
    axes[1, 2].set_title('Win Rate by Hour of Day', fontsize=14)
    axes[1, 2].set_xlabel('Trade Type')
    axes[1, 2].set_ylabel('Hour of Day')

    plt.tight_layout()
    plt.show()




    # Additional analysis: Trade clustering
    print("\n=== TRADE CLUSTERING ANALYSIS ===")
    print("Analyzing periods with high trade frequency...")

    # Group by date and count trades
    date_counts = combined_df.groupby([combined_df['entry_date'].dt.date,␣
↪'trade_type']).size().unstack(fill_value=0)

    # Find days with highest trading activity
    high_activity_days = date_counts.sum(axis=1).nlargest(5)
    print(f"\nTop 5 days with highest trading activity:")
    for date, count in high_activity_days.items():
        date_df = combined_df[combined_df['entry_date'].dt.date == date]
        win_rate = (date_df['profit_loss'] > 0).mean() * 100
        avg_profit = date_df['profit_loss'].mean()
        print(f"  {date}: {count} trades, Win rate: {win_rate:.1f}%, Avg P&L:␣
↪${avg_profit:.2f}")

    # Performance by volatility (using ATR as proxy if available)
    if 'ATR_10' in combined_df.columns:
        print("\nPerformance by volatility level (ATR):")
        combined_df['atr_quantile'] = pd.qcut(combined_df['ATR_10'], 4,␣
↪labels=['Low', 'Medium-Low', 'Medium-High', 'High'])
        atr_perf = combined_df.groupby(['atr_quantile', 'trade_type']).agg({
            'profit_loss': ['mean', 'count'],
            'is_win' if 'is_win' in combined_df.columns else␣
↪combined_df['profit_loss'] > 0: 'mean'
        })
        print(atr_perf)

    return combined_df
```

[37]:
```
# Call the function with our trade results
combined_df = analyze_trading_seasonality_patterns(long_trade_results,␣
↪short_trade_results)
```




```
=== TRADE CLUSTERING ANALYSIS ===
Analyzing periods with high trade frequency…

Top 5 days with highest trading activity:
  2024-11-30: 18 trades, Win rate: 5.6%, Avg P&L: $-0.71
  2024-08-06: 17 trades, Win rate: 35.3%, Avg P&L: $-6.03
  2024-06-19: 16 trades, Win rate: 31.2%, Avg P&L: $-4.13
  2024-08-14: 16 trades, Win rate: 56.2%, Avg P&L: $-2.46
  2024-01-02: 14 trades, Win rate: 57.1%, Avg P&L: $-21.19
```

[38]:
```
combined_df
```

[38]:

	entry_date	entry_price	position_size	position_value	\
1458	2024-01-01 12:00:00	42635.67150	0.234428	10000.000000	
0	2024-01-01 13:00:00	42770.37450	0.233924	10000.000000	
1	2024-01-01 18:30:00	43236.10725	0.230439	9958.313422	
1459	2024-01-01 22:00:00	43543.01760	0.228651	9961.129357	

```
2    2024-01-02 01:15:00  45170.07375       0.219458     9907.990650
…                    …          …             …            …
1456 2024-12-30 18:30:00  94514.03340       0.002349      221.863765
2761 2024-12-31 00:45:00  92287.33325       0.004180      385.978862
2762 2024-12-31 01:45:00  92270.14185       0.004180      385.918587
2763 2024-12-31 02:30:00  92376.68855       0.004152      383.757523
1457 2024-12-31 03:45:00  92585.16945       0.002400      222.115995

      target_price  stop_loss  commission  risk_amount  risk_per_trade_pct  \
1458     42616.810   42737.38    7.500000   100.000000                 1.0
0        42795.420   42656.16    7.500000   100.000000                 1.0
1        43280.565   43082.37    7.468735    99.583134                 1.0
1459     43477.650   43739.10    7.470847    99.611294                 1.0
2        45330.165   44782.17    7.430993    99.079907                 1.0
…              …          …           …            …                   …
1456     94763.465   93873.47    0.166398     2.218638                 1.0
2761     92163.430   92673.64    0.289484     3.859789                 1.0
2762     92150.365   92648.17    0.289439     3.859186                 1.0
2763     92256.455   92755.79    0.287818     3.837575                 1.0
1457     92673.605   92269.49    0.166587     2.221160                 1.0

                exit_date  …  duration_group month  cumulative_pnl  year  \
1458 2024-01-01 12:15:00  …             <1h     1      -31.370643  2024
0    2024-01-01 14:00:00  …             <1h     1      -34.186578  2024
1    2024-01-01 18:45:00  …             <1h     1      -77.040615  2024
1459 2024-01-01 22:15:00  …             <1h     1      -23.873951  2024
2    2024-01-02 01:45:00  …             <1h     1     -169.496376  2024
…                    …  …             …    …              …   …
1456 2024-12-30 20:00:00  …            1-4h    12    -6872.461737  2024
2761 2024-12-31 01:15:00  …             <1h    12    -6629.300612  2024
2762 2024-12-31 02:30:00  …             <1h    12    -6631.172237  2024
2763 2024-12-31 03:00:00  …             <1h    12    -6630.960102  2024
1457 2024-12-31 04:00:00  …             <1h    12    -6872.416377  2024

      trade_type  day_name  week_of_year  day_name_ordered  cumulative_short  \
1458       Short    Monday             1            Monday               1.0
0           Long    Monday             1            Monday               NaN
1           Long    Monday             1            Monday               NaN
1459       Short    Monday             1            Monday               2.0
2           Long   Tuesday             1           Tuesday               NaN
…              …         …             …                 …                 …
1456        Long    Monday             1            Monday               NaN
2761       Short   Tuesday             1           Tuesday            1304.0
2762       Short   Tuesday             1           Tuesday            1305.0
2763       Short   Tuesday             1           Tuesday            1306.0
1457        Long   Tuesday             1           Tuesday               NaN
```

```
      cumulative_long
1458              NaN
0                 1.0
1                 2.0
1459              NaN
2                 3.0
…                  …
1456           1457.0
2761              NaN
2762              NaN
2763              NaN
1457           1458.0

[2764 rows x 34 columns]
```

3.2.4 Optimization: Finding Better Set of Parameters

[39]:
```
from tqdm.notebook import tqdm
import pandas as pd
import numpy as np
from itertools import product
import time

def optimize_trading_parameters(df, parameter_grid=None, risk_per_trade_pct=1.0,
                                                commission_pct=0.075,␣
 ↪slippage_pct=0.05,
                                                trade_type='long', verbose=True):
    """
    Optimize trading parameters by grid search through different parameter␣
 ↪combinations
    with progress bar visualization

    Parameters:
    - df: DataFrame containing price data
    - parameter_grid: Dictionary with lists of values for each parameter to test
                      (If None, uses default grid)
    - risk_per_trade_pct: Risk percentage per trade
    - commission_pct: Commission percentage
    - slippage_pct: Slippage percentage
    - trade_type: 'long' or 'short' to optimize for specific trade direction
    - verbose: Whether to print progress updates

    Returns:
    - DataFrame containing results of all parameter combinations sorted by␣
 ↪performance
    - Dictionary with best parameters found
    """
```

```
    # Default parameter grid if none provided
    if parameter_grid is None:
        parameter_grid = {
            'ema_period': [5, 10, 20, 50],
            'atr_period': [10, 14, 20],
            'target_atr_multiplier': [0.5, 0.75, 1.0, 1.5, 2.0],
            'stop_loss_atr_multiplier': [0.5, 0.75, 1.0, 1.5, 2.0]
        }

    # Create all combinations of parameters
    param_keys = list(parameter_grid.keys())
    param_values = list(parameter_grid.values())
    combinations = list(product(*param_values))

    if verbose:
        print(f"Testing {len(combinations)} parameter combinations for␣
↪{trade_type} trades")

    # Initialize results storage
    results = []
    start_time = time.time()

    # Loop through all parameter combinations with progress bar
    for combo in tqdm(combinations, desc="Optimizing parameters", leave=True):
        # Create parameter dictionary for this combination
        params = dict(zip(param_keys, combo))

        # Find trade setups with current parameters
        if trade_type.lower() == 'long':
            _, trade_setups = find_long_trades(
                df,
                ema_period=params['ema_period'],
                atr_period=params['atr_period'],
                target_atr_multiplier=params['target_atr_multiplier'],
                stop_loss_atr_multiplier=params['stop_loss_atr_multiplier']
            )

            # Skip if no trades found
            if len(trade_setups) == 0:
                continue

            # Run backtest with current parameters
            backtest_results, metrics, _ = backtest_long_trade_strategy(
                df,
                trade_setups,
                risk_per_trade_pct=risk_per_trade_pct,
```

```
                commission_pct=commission_pct,
                slippage_pct=slippage_pct,
                target_atr_multiplier=params['target_atr_multiplier'],
                stop_loss_atr_multiplier=params['stop_loss_atr_multiplier']
            )

        elif trade_type.lower() == 'short':
            _, trade_setups = find_short_trades(
                df,
                ema_period=params['ema_period'],
                atr_period=params['atr_period'],
                target_atr_multiplier=params['target_atr_multiplier'],
                stop_loss_atr_multiplier=params['stop_loss_atr_multiplier']
            )

            # Skip if no trades found
            if len(trade_setups) == 0:
                continue

            # Run backtest with current parameters
            backtest_results, metrics, _ = backtest_short_trade_strategy(
                df,
                trade_setups,
                risk_per_trade_pct=risk_per_trade_pct,
                commission_pct=commission_pct,
                slippage_pct=slippage_pct,
                target_atr_multiplier=params['target_atr_multiplier'],
                stop_loss_atr_multiplier=params['stop_loss_atr_multiplier']
            )
        else:
            raise ValueError("trade_type must be 'long' or 'short'")

        # Calculate performance scores
        risk_adjusted_return = metrics['total_return_pct'] /␣
↪metrics['max_drawdown_pct'] if metrics['max_drawdown_pct'] > 0 else 0

        # Store results with all metrics and parameters
        result = {
            **params,  # Include all parameters
            'total_trades': metrics['total_trades'],
            'win_rate': metrics['win_rate'],
            'profit_factor': metrics['profit_factor'],
            'total_return_pct': metrics['total_return_pct'],
            'max_drawdown_pct': metrics['max_drawdown_pct'],
            'risk_adjusted_return': risk_adjusted_return,
            'avg_win': metrics['average_win'],
            'avg_loss': metrics['average_loss'],
```

```
            'expectancy': metrics['expectancy'],
            'final_balance': metrics['final_balance']
        }

        results.append(result)

    # Create DataFrame from results
    results_df = pd.DataFrame(results)

    if len(results_df) == 0:
        print("No valid parameter combinations found")
        return pd.DataFrame(), {}

    # Sort results by risk-adjusted return (descending)
    results_df.sort_values('risk_adjusted_return', ascending=False,␣
↪inplace=True)

    # Get best parameters
    best_params = results_df.iloc[0].to_dict()

    elapsed_time = time.time() - start_time

    # Print summary
    if verbose:
        print("\n==== OPTIMIZATION RESULTS ====")
        print(f"Total combinations tested: {len(combinations)}")
        print(f"Valid combinations found: {len(results_df)}")
        print(f"Best risk-adjusted return: {best_params['risk_adjusted_return']:
↪.4f}")
        print(f"Total time elapsed: {elapsed_time:.2f} seconds")
        print("\nBest parameters:")
        for param in param_keys:
            print(f"  {param}: {best_params[param]}")
        print("\nTop 5 parameter combinations:")
        print(results_df[param_keys + ['risk_adjusted_return',␣
↪'total_return_pct',
                                       'win_rate', 'max_drawdown_pct']].head(5))

    return results_df, best_params
```

Find Optimum Parameter Combination for Long Trades

[40]:
```
parameter_grid = {
    'ema_period': [5, 10, 20],
    'atr_period': [5, 10, 20],
    'target_atr_multiplier': [0.5, 0.75, 1.0, 1.5, 2.0],
    'stop_loss_atr_multiplier': [0.5, 0.75, 1.0, 1.5, 2.0]
```

```
}

long_optimization_df, long_best_params = optimize_trading_parameters(df,
                                                                     ␣
 ↪parameter_grid=parameter_grid,
                                                                      trade_type='long',
                                                                      verbose=True)
```

```
Testing 225 parameter combinations for long trades
Optimizing parameters:   0%|          | 0/225 [00:00<?, ?it/s]


==== OPTIMIZATION RESULTS ====
Total combinations tested: 225
Valid combinations found: 225
Best risk-adjusted return: -0.9990
Total time elapsed: 1915.62 seconds

Best parameters:
  ema_period: 10.0
  atr_period: 10.0
  target_atr_multiplier: 2.0
  stop_loss_atr_multiplier: 2.0

Top 5 parameter combinations:
     ema_period  atr_period  target_atr_multiplier  stop_loss_atr_multiplier  \
124          10          10                    2.0                       2.0
123          10          10                    2.0                       1.5
199          20          10                    2.0                       2.0
149          10          20                    2.0                       2.0
198          20          10                    2.0                       1.5

     risk_adjusted_return  total_return_pct  win_rate  max_drawdown_pct
124             -0.998956        -97.115274  0.465706         97.216751
123             -0.999099        -97.371443  0.386831         97.459239
199             -0.999122        -97.621922  0.462963         97.707675
149             -0.999145        -97.203041  0.462277         97.286230
198             -0.999185        -97.674987  0.386905         97.754700
```

[41]:
```
long_optimization_df
```

[41]:
```
     ema_period  atr_period  target_atr_multiplier  stop_loss_atr_multiplier  \
124          10          10                   2.00                      2.00
123          10          10                   2.00                      1.50
199          20          10                   2.00                      2.00
149          10          20                   2.00                      2.00
198          20          10                   2.00                      1.50
```

```
..          …         …                    …                          …
176         20        10                 0.50                       0.75
180         20        10                 0.75                       0.50
181         20        10                 0.75                       0.75
200         20        20                 0.50                       0.50
213         20        20                 1.00                       1.50

     total_trades  win_rate  profit_factor  total_return_pct  \
124          1458  0.465706       0.675759        -97.115274
123          1458  0.386831       0.630635        -97.371443
199          1512  0.462963       0.667131        -97.621922
149          1458  0.462277       0.656408        -97.203041
198          1512  0.386905       0.633499        -97.674987
..            …       …             …                …
176          1512  0.335979       0.168026        -97.991558
180          1512  0.259259       0.235223        -97.761072
181          1512  0.369709       0.296247        -97.718750
200          1512  0.272487       0.150548        -97.571244
213          1512  0.507275       0.430250        -98.266509

     max_drawdown_pct  risk_adjusted_return    avg_win   avg_loss  expectancy  \
124         97.216751             -0.998956  19.579062 -25.254112   -4.375014
123         97.459239             -0.999099  19.772176 -19.779616   -4.479746
199         97.707675             -0.999122  18.382764 -23.754254   -4.246375
149         97.286230             -0.999145  18.663489 -24.443475   -4.516113
198         97.754700             -0.999185  19.089828 -19.016566   -4.273021
..                …                     …         …         …          …
176         97.991558             -1.000000   2.743811  -8.262417   -4.564557
180         97.761072             -1.000000   5.299711  -7.885716   -4.467272
181         97.718750             -1.000000   5.070426 -10.039438   -4.453185
200         97.571244             -1.000000   2.902657  -7.221464   -4.462775
213         98.266509             -1.000000   6.852295 -16.396627   -4.603027

     final_balance
124     288.472625
123     262.855695
199     237.807802
149     279.695919
198     232.501325
..             …
176     200.844200
180     223.892786
181     228.125015
200     242.875618
213     173.349068

[225 rows x 14 columns]
```

[42]:
```
long_best_params
```

[42]:
```
{'ema_period': 10.0,
 'atr_period': 10.0,
 'target_atr_multiplier': 2.0,
 'stop_loss_atr_multiplier': 2.0,
 'total_trades': 1458.0,
 'win_rate': 0.4657064471879287,
 'profit_factor': 0.6757593848996019,
 'total_return_pct': -97.11527375482018,
 'max_drawdown_pct': 97.21675077360753,
 'risk_adjusted_return': -0.9989561776342056,
 'avg_win': 19.579061719217332,
 'avg_loss': -25.254112230466657,
 'expectancy': -4.375014074200931,
 'final_balance': 288.47262451798133}
```

Find Optimum Parameter Combination for Short Trades

[43]:
```
parameter_grid = {
    'ema_period': [5, 10, 20],
    'atr_period': [5, 10, 20],
    'target_atr_multiplier': [0.5, 0.75, 1.0, 1.5, 2.0],
    'stop_loss_atr_multiplier': [0.5, 0.75, 1.0, 1.5, 2.0]
}

short_optimization_df, short_best_params = optimize_trading_parameters(df,
                                                                      ␣
 ↪parameter_grid=parameter_grid,
                                                                       trade_type='short',
                                                                       verbose=True)
```

```
Testing 225 parameter combinations for short trades

Optimizing parameters:   0%|          | 0/225 [00:00<?, ?it/s]

Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
```

```
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 12:00:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 12:00:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 12:00:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 12:00:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 12:00:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 12:00:00 due to NaN entry, target or stop loss.
```

```
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 12:00:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 12:00:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 12:00:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 12:00:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 12:00:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 12:00:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 12:00:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 12:00:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 12:00:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 12:00:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 12:00:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 12:00:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 12:00:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 12:00:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 12:00:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 12:00:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 12:00:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 12:00:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 12:00:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
```

```
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 12:00:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 12:00:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 12:00:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 12:00:00 due to NaN entry, target or stop loss.
```

```
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 12:00:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 12:00:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 12:00:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 12:00:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 12:00:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 12:00:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 12:00:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 12:00:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 12:00:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 12:00:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 12:00:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 12:00:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 12:00:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 12:00:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 12:00:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 12:00:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 12:00:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 12:00:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 12:00:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 12:00:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 12:00:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
```

```
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
```

```
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.
Skipping trade on 2024-01-01 08:15:00 due to NaN entry, target or stop loss.

==== OPTIMIZATION RESULTS ====
Total combinations tested: 225
Valid combinations found: 225
Best risk-adjusted return: -0.9984
Total time elapsed: 1734.85 seconds

Best parameters:
  ema_period: 10.0
  atr_period: 20.0
  target_atr_multiplier: 2.0
  stop_loss_atr_multiplier: 2.0

Top 5 parameter combinations:
     ema_period  atr_period  target_atr_multiplier  stop_loss_atr_multiplier  \
149          10          20                    2.0                       2.0
74            5          20                    2.0                       2.0
49            5          10                    2.0                       2.0
148          10          20                    2.0                       1.5
124          10          10                    2.0                       2.0

     risk_adjusted_return  total_return_pct  win_rate  max_drawdown_pct
149             -0.998364        -94.844588  0.470498         94.999987
74              -0.998848        -95.370131  0.470673         95.480110
49              -0.998904        -95.038792  0.473951         95.143054
148             -0.999094        -94.897229  0.403831         94.983278
124             -0.999222        -94.434809  0.475498         94.508362
```

[44]:
```
short_optimization_df
```

[44]:

	ema_period	atr_period	target_atr_multiplier	stop_loss_atr_multiplier	\
149	10	20	2.0	2.00	
74	5	20	2.0	2.00	
49	5	10	2.0	2.00	
148	10	20	2.0	1.50	
124	10	10	2.0	2.00	
..	…	…	…	…	
201	20	20	0.5	0.75	
202	20	20	0.5	1.00	
214	20	20	1.0	2.00	
215	20	20	1.5	0.50	
221	20	20	2.0	0.75	

	total_trades	win_rate	profit_factor	total_return_pct	\
149	1305	0.470498	0.673252	-94.844588	
74	1381	0.470673	0.662172	-95.370131	
49	1382	0.473951	0.671604	-95.038792	
148	1305	0.403831	0.665027	-94.897229	
124	1306	0.475498	0.672923	-94.434809	
..	…	…	…	…	
201	1345	0.379926	0.200463	-96.813739	
202	1345	0.436431	0.228422	-96.946522	
214	1345	0.597026	0.461623	-96.147383	
215	1345	0.182900	0.314236	-96.650634	
221	1345	0.224535	0.425681	-96.433461	

	max_drawdown_pct	risk_adjusted_return	avg_win	avg_loss	expectancy	\
149	94.999987	-0.998364	21.216858	-28.002309	-4.844785	
74	95.480110	-0.998848	19.488042	-26.169389	-4.679650	
49	95.143054	-0.998904	19.712098	-26.443952	-4.568255	
148	94.983278	-0.999094	23.298002	-23.730678	-4.739019	
124	94.508362	-0.999222	20.721117	-27.915735	-4.789024	
..	…	…	…	…	…	
201	96.813739	-1.000000	3.388099	-10.355613	-5.134024	
202	96.946522	-1.000000	3.477349	-11.789065	-5.126325	
214	96.147383	-1.000000	7.367820	-23.646563	-5.130169	
215	96.650634	-1.000000	12.972345	-9.240590	-5.177853	
221	96.433461	-1.000000	17.287616	-11.759087	-5.237077	

	final_balance
149	515.541190
74	462.986916
49	496.120807
148	510.277070
124	556.519129

```
..              …
201     318.626087
202     305.347838
214     385.261694
215     334.936637
221     356.653859

[225 rows x 14 columns]
```

[45]:
```
short_best_params
```

[45]:
```
{'ema_period': 10.0,
 'atr_period': 20.0,
 'target_atr_multiplier': 2.0,
 'stop_loss_atr_multiplier': 2.0,
 'total_trades': 1305.0,
 'win_rate': 0.4704980842911877,
 'profit_factor': 0.6732518420292849,
 'total_return_pct': -94.84458810185295,
 'max_drawdown_pct': 94.99998695081669,
 'risk_adjusted_return': -0.9983642224177968,
 'avg_win': 21.216857930349327,
 'avg_loss': -28.00230887865049,
 'expectancy': -4.844785184607662,
 'final_balance': 515.541189814706}
```

3.2.5 Optimization Method for Trading Strategy Parameters

The optimization method applied in this cryptocurrency trading strategy involves a systematic grid search approach to identify the optimal combination of parameters that produce the best risk-adjusted returns. This comprehensive process enables the discovery of parameter values that maximize profitability while managing risk effectively.

The optimization process specifically targets four key parameters:

- **EMA Period**: Periods used for the Exponential Moving Average calculation (5, 10)
- **ATR Period**: Periods used for the Average True Range calculation (10)
- **Target ATR Multiplier**: Factor applied to ATR for determining take-profit levels (0.5, 0.75)
- **Stop-Loss ATR Multiplier**: Factor applied to ATR for determining stop-loss levels (0.75, 1.0)

Rather than relying on random parameter selection, the optimization systematically evaluates all possible combinations from the predefined parameter grid. This ensures comprehensive coverage of the parameter space.

For each parameter combination, the algorithm:

1. Identifies potential trade setups based on the specified criteria

2. Simulates trade execution with realistic conditions including slippage (0.05%) and commission (0.075%)
3. Records detailed trade outcomes including profit/loss, win rate, and drawdowns
4. Calculates performance metrics for comparative analysis

The results are ranked primarily by risk-adjusted return, calculated as the ratio between total return percentage and maximum drawdown percentage. This metric provides a balanced view of performance, favoring strategies that generate returns with minimal drawdowns.

Some features of my analysis: 1. **Separate Long & Short Analysis**: Trades are evaluated independently by direction to identify directional biases. 2. **ATR-Based Risk Management**: Uses Average True Range (ATR) to set dynamic targets and stop losses based on market volatility. 3. **Time-Based Pattern Analysis**: Examines performance by hour of day, day of week, and month to identify temporal patterns. 4. **Trade Clustering Analysis**: Identifies periods of high trading activity and their impact on performance. 5. **Parameter Optimization**: Implements grid search for finding optimal parameter combinations.

Ilyas Ustun
Chicago, IL
6/15/2025