Data Preparation: Load the OHLC data into a DataFrame using Python (Pandas).

In [2]:
```
import pandas as pd
import numpy as np
```

In [3]:
```
# Load data
file_path = r"C:\Users\HP\Downloads\bn_intra.csv"
ohlc_data = pd.read_csv(file_path)
```

In [4]:
```
ohlc_data
```

Out[4]:

	datetime	O	H	L	C
0	1/11/2023 9:15	42694.2	42774.65	42665.10	42708.6
1	1/11/2023 9:18	42710.4	42786.45	42710.40	42779.6
2	1/11/2023 9:21	42779.4	42815.15	42747.95	42779.2
3	1/11/2023 9:24	42780.1	42791.35	42729.35	42760.0
4	1/11/2023 9:27	42759.5	42767.90	42706.70	42738.4
...	...	...	...	...	...
1765	21-11-2023 15:15	43681.6	43695.30	43672.60	43692.4
1766	21-11-2023 15:18	43693.9	43697.85	43676.00	43681.1
1767	21-11-2023 15:21	43677.1	43705.20	43677.10	43699.5
1768	21-11-2023 15:24	43700.0	43710.05	43693.15	43704.1
1769	21-11-2023 15:27	43702.9	43710.00	43681.65	43694.9

1770 rows × 5 columns

In [5]:
```
ohlc_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1770 entries, 0 to 1769
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   datetime  1770 non-null   object 
 1   O         1770 non-null   float64
 2   H         1770 non-null   float64
 3   L         1770 non-null   float64
 4   C         1770 non-null   float64
dtypes: float64(4), object(1)
memory usage: 69.3+ KB
```

In [6]:
```
ohlc_data.describe()
```

Out[6]:

	O	H	L	C
count	1770.000000	1770.000000	1770.000000	1770.000000
mean	43611.126497	43625.357090	43596.144831	43611.129887
std	407.054460	406.016421	407.997071	406.816566
min	42613.600000	42651.350000	42589.650000	42616.300000
25%	43456.450000	43466.762500	43439.187500	43455.850000
50%	43657.150000	43671.625000	43644.300000	43656.500000
75%	43768.225000	43782.100000	43754.775000	43768.325000
max	44400.900000	44420.550000	44386.500000	44401.600000

In [7]:
```
# Ensure the timestamp is parsed correctly
ohlc_data['timestamp'] = pd.to_datetime(ohlc_data['datetime'])
```

In [8]:
```
# Sort data by timestamp (if not already sorted)
ohlc_data.sort_values('timestamp', inplace=True)
```

In [9]:
```
ohlc_data
```

Out[9]:

	datetime	O	H	L	C	timestamp
0	1/11/2023 9:15	42694.2	42774.65	42665.10	42708.6	2023-01-11 09:15:00
1	1/11/2023 9:18	42710.4	42786.45	42710.40	42779.6	2023-01-11 09:18:00
2	1/11/2023 9:21	42779.4	42815.15	42747.95	42779.2	2023-01-11 09:21:00
3	1/11/2023 9:24	42780.1	42791.35	42729.35	42760.0	2023-01-11 09:24:00
4	1/11/2023 9:27	42759.5	42767.90	42706.70	42738.4	2023-01-11 09:27:00
...	...	...	...	...	...	...
1015	12/11/2023 19:00	44001.0	44008.95	43988.55	44008.4	2023-12-11 19:00:00
1016	12/11/2023 19:03	44009.9	44013.30	43964.50	43964.5	2023-12-11 19:03:00
1017	12/11/2023 19:06	43969.6	43996.20	43969.05	43988.9	2023-12-11 19:06:00
1018	12/11/2023 19:09	43990.3	43992.20	43971.45	43986.6	2023-12-11 19:09:00
1019	12/11/2023 19:12	43985.4	43992.70	43971.25	43978.7	2023-12-11 19:12:00

1770 rows × 6 columns

Movement Calculation:

Compute the percentage movement between close values for each row.

Movement (%)= ((Close current−Close fixed)/Close fixed)×100

In [19]:
```
max_timeframe = pd.Timedelta(minutes=90) #given in the pdf
max_timeframe
```

Out[19]:
```
Timedelta('0 days 01:30:00')
```

In [11]:
```
def max_mov(n):
    start_time = n

    # Check for maximum movements within 90 minutes
    window_data = ohlc_data[(ohlc_data['timestamp'] >= start_time) &
                            (ohlc_data['timestamp'] <= start_time + max_timeframe)]
    window_data = window_data.reset_index()

    # Ensure there are rows in the filtered data
    if window_data.empty:
        return pd.DataFrame()  # Return an empty DataFrame if no data matches

    # Calculate percentage movements
    window_data['movement_perc'] = ((window_data['C'] - window_data['C'].iloc[0]) / wi

    # Find the row with the maximum movement
    max_row = window_data.loc[window_data['movement_perc'].abs().idxmax()]

    return pd.Series([window_data['timestamp'].iloc[0], max_row['timestamp'],window_da
                      index=['starttime', 'endtime','closing at start', 'closing at enc
```

In [12]:
```
mv = ohlc_data.apply(lambda row: max_mov(row['timestamp']),axis=1)
```

In [13]:
```
mv['time_difference'] = (mv['endtime'] - mv['starttime']).dt.total_seconds() / 60
```

In [14]:
```
mv
```

Out[14]:

	starttime	endtime	closing at start	closing at end	moving_percentage	time_difference
0	2023-01-11 09:15:00	2023-01-11 09:36:00	42708.6	42782.1	0.172096	21.0
1	2023-01-11 09:18:00	2023-01-11 10:00:00	42779.6	42640.2	-0.325856	42.0
2	2023-01-11 09:21:00	2023-01-11 10:00:00	42779.2	42640.2	-0.324924	39.0
3	2023-01-11 09:24:00	2023-01-11 10:00:00	42760.0	42640.2	-0.280168	36.0
4	2023-01-11 09:27:00	2023-01-11 10:00:00	42738.4	42640.2	-0.229770	33.0
...	...	...	...	...	...	...
1015	2023-12-11 19:00:00	2023-12-11 19:03:00	44008.4	43964.5	-0.099754	3.0
1016	2023-12-11 19:03:00	2023-12-11 19:06:00	43964.5	43988.9	0.055499	3.0
1017	2023-12-11 19:06:00	2023-12-11 19:12:00	43988.9	43978.7	-0.023188	6.0
1018	2023-12-11 19:09:00	2023-12-11 19:12:00	43986.6	43978.7	-0.017960	3.0
1019	2023-12-11 19:12:00	2023-12-11 19:12:00	43978.7	43978.7	0.000000	0.0

1770 rows × 6 columns

In [15]:
```
# Initialize variables
used_timeframes = []
valid_entries = []
```

In [16]:
```
# Iterate over rows and filter data
for index, row in mv.iterrows():
    start_time = row['starttime']
    end_time = row['endtime']
    movement_perc = row['moving_percentage']

    # Check if movement is either greater than 0.3 or less than -0.3
    if not (movement_perc > 0.3 or movement_perc < -0.3):
        continue

    # Check for overlap with already used timeframes
    overlap = any((start_time < t[1]) and (end_time > t[0]) for t in used_timeframes)

    # If there's no overlap, add the current timeframe to valid entries and mark it as
    if not overlap:
        valid_entries.append({
            'starttime': start_time,
            'endtime': end_time,
            'closing at start': row['closing at start'],
            'closing at end': row['closing at end'],
```

```
            'time_difference': row['time_difference'],
            'moving_percentage': movement_perc

        })
        used_timeframes.append((start_time, end_time))
```

In [17]:
```
# Convert valid entries to a DataFrame
valid_df = pd.DataFrame(valid_entries)
```

In [23]:
```
valid_df['date'] = valid_df['starttime'].apply(lambda x :x.date())
```

In [25]:
```
result = valid_df[["date","starttime","endtime","time_difference","moving_percentage"]
```

In [26]:
```
result
```

Out[26]:

	date	starttime	endtime	time_difference	moving_percentage
0	2023-01-11	2023-01-11 09:18:00	2023-01-11 10:00:00	42.0	-0.325856
1	2023-01-11	2023-01-11 11:09:00	2023-01-11 12:27:00	78.0	0.316710
2	2023-01-11	2023-01-11 12:27:00	2023-01-11 13:15:00	48.0	-0.337926
3	2023-02-11	2023-02-11 09:15:00	2023-02-11 09:42:00	27.0	0.334648
4	2023-02-11	2023-02-11 09:42:00	2023-02-11 10:54:00	72.0	-0.460196
5	2023-02-11	2023-02-11 10:54:00	2023-02-11 12:06:00	72.0	-0.565423
6	2023-02-11	2023-02-11 13:27:00	2023-02-11 14:57:00	90.0	0.372063
7	2023-03-11	2023-03-11 09:48:00	2023-03-11 10:57:00	69.0	0.302387
8	2023-07-11	2023-07-11 10:27:00	2023-07-11 11:51:00	84.0	-0.306375
9	2023-07-11	2023-07-11 12:15:00	2023-07-11 13:45:00	90.0	0.311093
10	2023-07-11	2023-07-11 13:45:00	2023-07-11 15:15:00	90.0	0.662358
11	2023-08-11	2023-08-11 09:27:00	2023-08-11 10:12:00	45.0	-0.318336
12	2023-09-11	2023-09-11 09:15:00	2023-09-11 10:21:00	66.0	0.455593
13	2023-09-11	2023-09-11 10:27:00	2023-09-11 11:57:00	90.0	0.351465
14	2023-10-11	2023-10-11 09:15:00	2023-10-11 10:45:00	90.0	0.375102
15	2023-10-11	2023-10-11 11:30:00	2023-10-11 12:54:00	84.0	0.358015
16	2023-10-11	2023-10-11 13:54:00	2023-10-11 15:24:00	90.0	0.404235
17	2023-11-13	2023-11-13 09:15:00	2023-11-13 10:30:00	75.0	-0.306739
18	2023-11-13	2023-11-13 10:30:00	2023-11-13 11:42:00	72.0	0.396214
19	2023-11-15	2023-11-15 09:21:00	2023-11-15 10:51:00	90.0	-0.343958
20	2023-11-16	2023-11-16 09:45:00	2023-11-16 11:12:00	87.0	0.327275
21	2023-11-16	2023-11-16 13:36:00	2023-11-16 15:06:00	90.0	-0.356155
22	2023-11-17	2023-11-17 09:15:00	2023-11-17 10:18:00	63.0	-0.366463
23	2023-11-17	2023-11-17 10:54:00	2023-11-17 12:24:00	90.0	-0.333149
24	2023-11-17	2023-11-17 14:06:00	2023-11-17 15:24:00	78.0	-0.320277
25	2023-11-20	2023-11-20 09:15:00	2023-11-20 10:27:00	72.0	0.510636
26	2023-11-20	2023-11-20 10:27:00	2023-11-20 11:15:00	48.0	-0.436444
27	2023-11-20	2023-11-20 11:21:00	2023-11-20 12:51:00	90.0	0.302781
28	2023-11-21	2023-11-21 12:18:00	2023-11-21 13:09:00	51.0	-0.306848

In [ ]: