analysis

January 6, 2025

[1]:
```python
import pandas as pd
from pathlib import Path
import sys
parent_dir = Path().resolve().parent
sys.path.append(str(parent_dir))
from utils.general import check_missing_timestamps
import numpy as np
import ccxt
import matplotlib.pyplot as plt


def get_exchange_timeframes(exchange_id):
    """
    Get the list of available timeframes for a given exchange.

    Args:
        exchange_id (str): The exchange ID (e.g., "binance", "bitget").

    Returns:
        list: A list of available timeframes for the exchange.
    """
    try:
        # Initialize the exchange
        exchange = getattr(ccxt, exchange_id)({'enableRateLimit': True})

        # Check if the exchange supports timeframes
        if hasattr(exchange, 'timeframes') and exchange.timeframes:
            return list(exchange.timeframes.keys())
        else:
            return []
    except Exception as e:
        print(f"Error fetching timeframes for {exchange_id}: {e}")
        return []

# Example Usage
exchange_id = "bitget"  # Replace with your desired exchange ID
timeframes = get_exchange_timeframes(exchange_id)
```

```
print(f"Available timeframes for {exchange_id}: {timeframes}")
```

```
Available timeframes for bitget: ['1m', '3m', '5m', '15m', '30m', '1h', '2h',
'4h', '6h', '12h', '1d', '3d', '1w', '1M']
```

[2]:
```
PATH = "/home/ubuntu/project/finance/cex-market-analysis/src/data/bitget/future"
TIMEFRAME = "1m"
```

[3]:
```
df = pd.read_csv("/home/ubuntu/project/finance/cex-market-analysis/src/data/
 ↪bitget/future/BTC_USDT:USDT_1m.csv", header=None)
df = pd.DataFrame(df.values, columns=['date', 'open', 'high', 'low', 'close',␣
 ↪'volume'])
df['date'] = pd.to_datetime(df['date'], unit='ms')
missing = check_missing_timestamps(df, freq='1min')
df.set_index('date', inplace=True)

df = df.resample('1d').agg({
    'open': 'first',     # First price in the 1-hour window (Open)
    'high': 'max',       # Maximum price in the 1-hour window (High)
    'low': 'min',        # Minimum price in the 1-hour window (Low)
    'close': 'last',     # Last price in the 1-hour window (Close)
    'volume': 'sum'      # Total volume in the 1-hour window
})

import plotly.graph_objects as go
# Create a candlestick chart using Plotly
fig = go.Figure(data=[go.Candlestick(
    x=df.index,
    open=df['open'],
    high=df['high'],
    low=df['low'],
    close=df['close'],
    increasing_line_color='green',  # Green for price increase
    decreasing_line_color='red',    # Red for price decrease
)])
# Customize layout
fig.update_layout(
    title='',
    xaxis_title='Date',
    yaxis_title='Price (USDT)',
    template='plotly_dark',  # Set a dark theme for the plot
    xaxis_rangeslider_visible=False  # Optionally hide the range slider
)
# Show the plot
```

[4]:
```
## Get Top 100 Future Volume Market
import pandas as pd
```

```
df_volume = pd.read_csv("/home/ubuntu/project/finance/cex-market-analysis/
 ↪symbols/top_100_bitget.csv",index_col=0)

duration_days= []

for i in range(len(df_volume)):

    symbol = df_volume["symbol"][i]
    filename = symbol.replace("/","_")


    filename = f"{PATH}/{filename}_{TIMEFRAME}.csv"
    try:
        df = pd.read_csv(filename)
        df = pd.DataFrame(df.values, columns=['date', 'open', 'high', 'low',␣
 ↪'close', 'volume'])
        df['date'] = pd.to_datetime(df['date'], unit='ms')
        # missing = check_missing_timestamps(df, freq='1min')
        df.set_index('date', inplace=True)

        duration_days.append(df.index[-1] - df.index[0])

    except:
        duration_days.append(np.nan)

df_volume['duration'] = duration_days
df_volume['duration_days'] = pd.to_timedelta(df_volume['duration']).dt.
 ↪total_seconds() / (24 * 3600)
```

[5]:
```
df_volume
```

[5]:
```
                 symbol    volume_24h        price          duration  \
0         BTC/USDT:USDT  6.354086e+09  98235.30000 371 days 00:23:00
1         ETH/USDT:USDT  3.395894e+09   3636.78000 371 days 00:23:00
2         XRP/USDT:USDT  1.325817e+09      2.37160 371 days 00:23:00
3        HIVE/USDT:USDT  3.086087e+08      0.56790  13 days 17:29:00
4        MOCA/USDT:USDT  2.346044e+08      0.31892 176 days 22:08:00
..                  …           …           …                …
95        SXP/USDT:USDT  5.675550e+06      0.35760              NaT
96        FIL/USDT:USDT  5.537617e+06      5.64000              NaT
97       TROY/USDT:USDT  5.467954e+06      0.00433              NaT
98      DRIFT/USDT:USDT  5.420605e+06      1.37660              NaT
99  NEIROETH/USDT:USDT  5.356493e+06      0.04870              NaT

    duration_days
0      371.015972
```

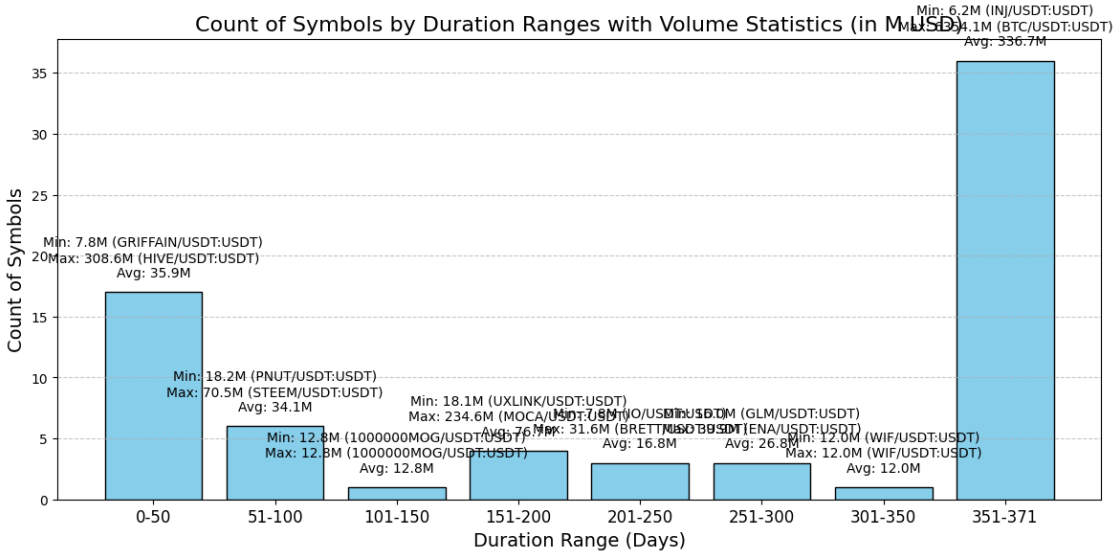
```
1      371.015972
2      371.015972
3       13.728472
4      176.922222
..            …
95            NaN
96            NaN
97            NaN
98            NaN
99            NaN

[100 rows x 5 columns]
```

[6]:
```
# Define bins for ranges
bins = [0, 50, 100, 150, 200, 250, 300, 350, 372]
labels = ['0-50', '51-100', '101-150', '151-200', '201-250', '251-300',␣
 ↪'301-350', '351-371']

# Categorize durations into bins
df_volume['duration_range'] = pd.cut(df_volume['duration_days'], bins=bins,␣
 ↪labels=labels, right=True)

# Calculate statistics for each range
range_stats = df_volume.groupby('duration_range')['volume_24h'].agg(['min',␣
 ↪'max', 'mean', 'count'])

# Add symbols with max and min volumes
range_symbols = df_volume.groupby('duration_range').apply(
    lambda group: pd.Series({
        'max_symbol': group.loc[group['volume_24h'].idxmax(), 'symbol'] if not␣
 ↪group.empty else None,
        'min_symbol': group.loc[group['volume_24h'].idxmin(), 'symbol'] if not␣
 ↪group.empty else None
    })
).reset_index()

# Merge the stats and symbols
range_stats = range_stats.merge(range_symbols, on='duration_range')

# Convert volume to M USD
range_stats['min'] /= 1e6
range_stats['max'] /= 1e6
range_stats['mean'] /= 1e6

# Plotting
plt.figure(figsize=(12, 6))
```

```
# Bar plot for count of symbols
plt.bar(range_stats['duration_range'], range_stats['count'], color='skyblue',␣
 ↪edgecolor='black')

# Formatting
plt.title('Count of Symbols by Duration Ranges with Volume Statistics (in M␣
 ↪USD)', fontsize=16)
plt.xlabel('Duration Range (Days)', fontsize=14)
plt.ylabel('Count of Symbols', fontsize=14)
plt.xticks(fontsize=12)
plt.grid(axis='y', linestyle='--', alpha=0.7)

# Annotate bars with volume statistics and symbols
for i, (count, min_vol, max_vol, mean_vol, max_symbol, min_symbol) in␣
 ↪enumerate(zip(
    range_stats['count'], range_stats['min'], range_stats['max'],␣
 ↪range_stats['mean'],
    range_stats['max_symbol'], range_stats['min_symbol']
)):
    annotation = (
        f"Min: {min_vol:.1f}M ({min_symbol})\n"
        f"Max: {max_vol:.1f}M ({max_symbol})\n"
        f"Avg: {mean_vol:.1f}M"
    )
    plt.text(i, count + 1, annotation, ha='center', va='bottom', fontsize=10,␣
 ↪color='black')

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

```
/tmp/ipykernel_1758103/2273443294.py:9: FutureWarning:

The default of observed=False is deprecated and will be changed to True in a
future version of pandas. Pass observed=False to retain current behavior or
observed=True to adopt the future default and silence this warning.

/tmp/ipykernel_1758103/2273443294.py:12: FutureWarning:

The default of observed=False is deprecated and will be changed to True in a
future version of pandas. Pass observed=False to retain current behavior or
observed=True to adopt the future default and silence this warning.

/tmp/ipykernel_1758103/2273443294.py:12: DeprecationWarning:

DataFrameGroupBy.apply operated on the grouping columns. This behavior is
deprecated, and in a future version of pandas the grouping columns will be
excluded from the operation. Either pass `include_groups=False` to exclude the
```

groupings or explicitly select the grouping columns after groupby to silence this warning.



0.1 Candle Wick Analysis

- Count the number of Candle that upper wick > 0 and upper wick $= 0$
- Make the ratio of upper wick higher than zero and equal to zero.
- Perform the same analysis for each timeframe
- Perform the same analysis during different market period: American / Europe / Asian (week end not included)

[7]:
```
df = pd.read_csv("/home/ubuntu/project/finance/cex-market-analysis/src/data/
 ↪bitget/future/BTC_USDT:USDT_1m.csv", header=None)
df = pd.DataFrame(df.values, columns=['date', 'open', 'high', 'low', 'close',␣
 ↪'volume'])
df['date'] = pd.to_datetime(df['date'], unit='ms')
missing = check_missing_timestamps(df, freq='1min')
df.set_index('date', inplace=True)

df_resample = df.resample('1h').agg({
    'open': 'first',      # First price in the 1-hour window (Open)
    'high': 'max',        # Maximum price in the 1-hour window (High)
    'low': 'min',         # Minimum price in the 1-hour window (Low)
    'close': 'last',      # Last price in the 1-hour window (Close)
    'volume': 'sum'       # Total volume in the 1-hour window
})
```

```
df_resample.tail()
```

[7]:

date	open	high	low	close	volume
2025-01-05 19:00:00	97872.0	98085.2	97804.9	98045.5	3075.543
2025-01-05 20:00:00	98045.5	98675.1	97989.4	98250.0	5729.170
2025-01-05 21:00:00	98250.0	98500.0	98186.0	98460.9	1624.727
2025-01-05 22:00:00	98460.9	98745.7	98310.8	98661.5	3238.308
2025-01-05 23:00:00	98661.5	98824.3	98482.5	98516.3	1542.044

[8]:
```
df_resample['upper_wick'] = df_resample['high'] - df_resample[['open','close']].max(axis=1)

# Count the days where the wick is greater than zero
upper_wick_count = df_resample[df_resample['upper_wick'] > 0].shape[0]
neutral_wick_count = df_resample[df_resample['upper_wick'] == 0].shape[0]

total_candles = len(df_resample)
upper_wick_percentage = upper_wick_count * 100 / total_candles
neutral_wick_percentage = neutral_wick_count * 100 / total_candles

print(f"Total Number of candles: {total_candles}")
print(f"Number of upper wick > 0 : {upper_wick_count} -- {upper_wick_percentage:.2f}%")
print(f"Number of neutral wick == 0 : {neutral_wick_count} --{neutral_wick_percentage:.2f}%")
```

```
Total Number of candles: 8906
Number of upper wick > 0 : 8738 -- 98.11%
Number of neutral wick == 0 : 168 -- 1.89%
```

0.1.1 Analysis for all timeframe

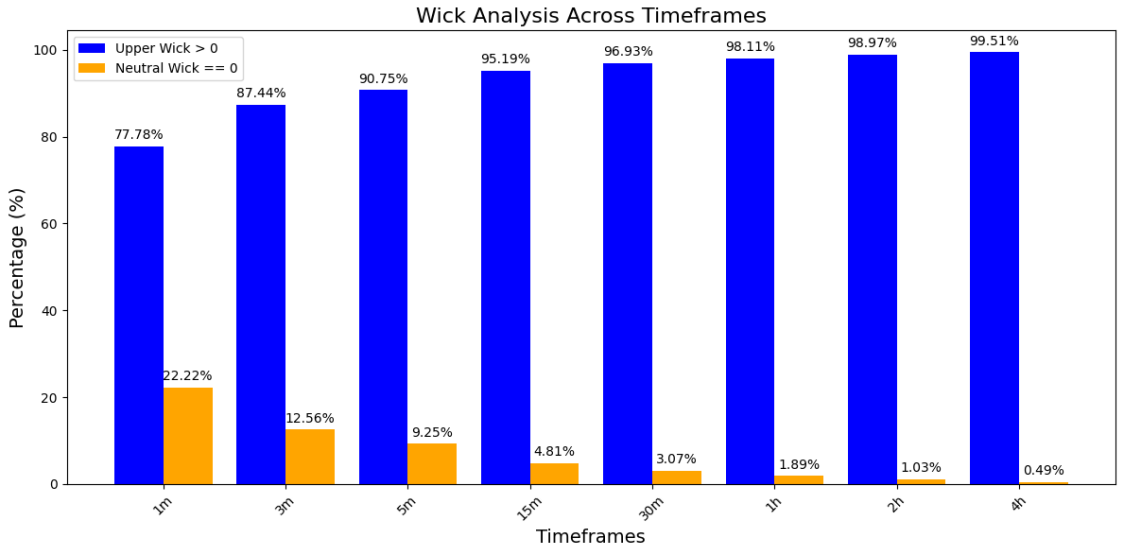
[9]:
```
# Mapping of timeframes to pandas resampling strings
timeframe_resampling_map = {
    '1m': '1min',   # 1 minute
    '3m': '3min',   # 3 minutes
    '5m': '5min',   # 5 minutes
    '15m': '15min', # 15 minutes
    '30m': '30min', # 30 minutes
    '1h': '1h',   # 1 hour
    '2h': '2h',   # 2 hours
    '4h': '4h',   # 4 hours
}
```

[10]:
```
results = {}
```

```
for timeframe, resample_str in timeframe_resampling_map.items():
    # Resample the DataFrame
    df_resampled = df.resample(resample_str).agg({
        'open': 'first',
        'high': 'max',
        'low': 'min',
        'close': 'last',
        'volume': 'sum'
    })

    # Calculate the upper wick
    df_resampled['upper_wick'] = df_resampled['high'] - df_resampled[['open',␣
 ↪'close']].max(axis=1)

    # Perform analysis
    total_candles = len(df_resampled)
    upper_wick_count = df_resampled[df_resampled['upper_wick'] > 0].shape[0]
    neutral_wick_count = df_resampled[df_resampled['upper_wick'] == 0].shape[0]

    upper_wick_percentage = upper_wick_count * 100 / total_candles if␣
 ↪total_candles > 0 else 0
    neutral_wick_percentage = neutral_wick_count * 100 / total_candles if␣
 ↪total_candles > 0 else 0

    # Store results
    results[timeframe] = {
        'total_candles': total_candles,
        'upper_wick_count': upper_wick_count,
        'neutral_wick_count': neutral_wick_count,
        'upper_wick_percentage': round(upper_wick_percentage, 2),
        'neutral_wick_percentage': round(neutral_wick_percentage, 2)
    }
```

[11]:
```
import matplotlib.pyplot as plt

# Prepare data for plotting
timeframes = list(results.keys())
upper_wick_percentages = [stats['upper_wick_percentage'] for stats in results.
 ↪values()]
neutral_wick_percentages = [stats['neutral_wick_percentage'] for stats in␣
 ↪results.values()]

# Plotting
fig, ax = plt.subplots(figsize=(12, 6))

# Bar width
bar_width = 0.4
```

```
# Plot upper wick percentages
ax.bar([i - bar_width / 2 for i in range(len(timeframes))],␣
 ↪upper_wick_percentages,
       width=bar_width, label='Upper Wick > 0', color='blue')

# Plot neutral wick percentages
ax.bar([i + bar_width / 2 for i in range(len(timeframes))],␣
 ↪neutral_wick_percentages,
       width=bar_width, label='Neutral Wick == 0', color='orange')

# Formatting
ax.set_title('Wick Analysis Across Timeframes', fontsize=16)
ax.set_xlabel('Timeframes', fontsize=14)
ax.set_ylabel('Percentage (%)', fontsize=14)
ax.set_xticks(range(len(timeframes)))
ax.set_xticklabels(timeframes, rotation=45)
ax.legend()

# Add percentage labels on bars
for i, (upper, neutral) in enumerate(zip(upper_wick_percentages,␣
 ↪neutral_wick_percentages)):
    ax.text(i - bar_width / 2, upper + 1, f'{upper:.2f}%', ha='center',␣
 ↪va='bottom', fontsize=10)
    ax.text(i + bar_width / 2, neutral + 1, f'{neutral:.2f}%', ha='center',␣
 ↪va='bottom', fontsize=10)

# Show plot
plt.tight_layout()
plt.show()
```



[12]:
```
# Function to calculate wick statistics
def calculate_wick_statistics(df, timeframe_resampling_map):
    """
    Calculate wick statistics for a DataFrame across multiple timeframes.

    Parameters:
        df (pd.DataFrame): Input DataFrame with 'open', 'high', 'low', 'close',␣
↪'volume'.
        timeframe_resampling_map (dict): Mapping of timeframes to pandas␣
↪resampling strings.

    Returns:
        dict: Wick statistics for each timeframe.
    """
    results = {}

    for timeframe, resample_str in timeframe_resampling_map.items():
        # Resample the DataFrame
        df_resampled = df.resample(resample_str).agg({
            'open': 'first',
            'high': 'max',
            'low': 'min',
            'close': 'last',
            'volume': 'sum'
        }).dropna()

        # Calculate the upper wick
        df_resampled['upper_wick'] = df_resampled['high'] -␣
↪df_resampled[['open', 'close']].max(axis=1)

        # Perform analysis
        total_candles = len(df_resampled)
        upper_wick_count = df_resampled[df_resampled['upper_wick'] > 0].shape[0]
        neutral_wick_count = df_resampled[df_resampled['upper_wick'] == 0].
↪shape[0]

        upper_wick_percentage = upper_wick_count * 100 / total_candles if␣
↪total_candles > 0 else 0
        neutral_wick_percentage = neutral_wick_count * 100 / total_candles if␣
↪total_candles > 0 else 0

        # Store results
        results[timeframe] = {
            'total_candles': total_candles,
```

```
            'upper_wick_count': upper_wick_count,
            'neutral_wick_count': neutral_wick_count,
            'upper_wick_percentage': round(upper_wick_percentage, 2),
            'neutral_wick_percentage': round(neutral_wick_percentage, 2)
        }

    return results

# Function to plot wick statistics
def plot_wick_statistics(results, crypto_name):
    """
    Plot wick statistics across timeframes.

    Parameters:
        results (dict): Wick statistics for each timeframe.
        crypto_name (str): Name of the cryptocurrency.
    """
    # Prepare data for plotting
    timeframes = list(results.keys())
    upper_wick_percentages = [stats['upper_wick_percentage'] for stats in␣
↪results.values()]
    neutral_wick_percentages = [stats['neutral_wick_percentage'] for stats in␣
↪results.values()]

    # Plotting
    fig, ax = plt.subplots(figsize=(12, 6))

    # Bar width
    bar_width = 0.4

    # Plot upper wick percentages
    ax.bar([i - bar_width / 2 for i in range(len(timeframes))],␣
↪upper_wick_percentages,
           width=bar_width, label='Upper Wick > 0', color='blue')

    # Plot neutral wick percentages
    ax.bar([i + bar_width / 2 for i in range(len(timeframes))],␣
↪neutral_wick_percentages,
           width=bar_width, label='Neutral Wick == 0', color='orange')

    # Formatting
    ax.set_title(f'Wick Analysis Across Timeframes for {crypto_name}',␣
↪fontsize=16)
    ax.set_xlabel('Timeframes', fontsize=14)
    ax.set_ylabel('Percentage (%)', fontsize=14)
    ax.set_xticks(range(len(timeframes)))
    ax.set_xticklabels(timeframes, rotation=45)
```

```
    ax.legend()

    # Add percentage labels on bars
    for i, (upper, neutral) in enumerate(zip(upper_wick_percentages,␣
 ↪neutral_wick_percentages)):
        ax.text(i - bar_width / 2, upper + 1, f'{upper:.2f}%', ha='center',␣
 ↪va='bottom', fontsize=10)
        ax.text(i + bar_width / 2, neutral + 1, f'{neutral:.2f}%', ha='center',␣
 ↪va='bottom', fontsize=10)

    # Show plot
    plt.tight_layout()
    plt.show()
```

0.2 Perform wick statistics on top 100 crypto

[13]:
```
import pandas as pd
import natsort
import glob
import os

PATH_SAVE = "/home/ubuntu/project/finance/cex-market-analysis/src/data/bitget/
 ↪future/"
files_path = natsort.natsorted(glob.glob(os.path.join(PATH_SAVE, "*.csv"),␣
 ↪recursive=False))

# Initialize an empty list to store results
all_results = []

for file_path in files_path:
    # Extract the symbol name from the filename
    path, filename = os.path.split(file_path)

    print(file_path)
    symbol = filename.replace(".csv", "")  # Remove the .csv extension

    # Load the data
    df = pd.read_csv(file_path, header=None)
    df = pd.DataFrame(df.values, columns=['date', 'open', 'high', 'low',␣
 ↪'close', 'volume'])
    df['date'] = pd.to_datetime(df['date'], unit='ms')
    df.set_index('date', inplace=True)

    # Calculate wick statistics
    results = calculate_wick_statistics(df, timeframe_resampling_map)
    # plot_wick_statistics(results, filename)
```

```
    # Flatten results and append to all_results
    for timeframe, stats in results.items():
        all_results.append({
            'symbol': symbol,
            'timeframe': timeframe,
            'total_candles': stats['total_candles'],
            'upper_wick_count': stats['upper_wick_count'],
            'neutral_wick_count': stats['neutral_wick_count'],
            'upper_wick_percentage': stats['upper_wick_percentage'],
            'neutral_wick_percentage': stats['neutral_wick_percentage']
        })

# Convert all_results to a DataFrame
df_results = pd.DataFrame(all_results)

# Save to a CSV file
output_path = "/home/ubuntu/project/finance/cex-market-analysis/src/
 ↪results_summary.csv"
df_results.to_csv(output_path, index=False)

print(f"Results saved to {output_path}")
```

```
/home/ubuntu/project/finance/cex-market-
analysis/src/data/bitget/future/1000000MOG_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-
analysis/src/data/bitget/future/AAVE_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-
analysis/src/data/bitget/future/ACT_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-
analysis/src/data/bitget/future/ADA_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-
analysis/src/data/bitget/future/AGLD_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-
analysis/src/data/bitget/future/AI16Z_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-
analysis/src/data/bitget/future/AIXBT_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-
analysis/src/data/bitget/future/ALGO_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-
analysis/src/data/bitget/future/ARB_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-
analysis/src/data/bitget/future/AVAX_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-
analysis/src/data/bitget/future/BCH_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-
analysis/src/data/bitget/future/BGB_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-
analysis/src/data/bitget/future/BIO_USDT:USDT_1m.csv
```

/home/ubuntu/project/finance/cex-market-analysis/src/data/bitget/future/BRETT_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-analysis/src/data/bitget/future/BTC_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-analysis/src/data/bitget/future/CHILLGUY_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-analysis/src/data/bitget/future/DF_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-analysis/src/data/bitget/future/DOGE_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-analysis/src/data/bitget/future/DOT_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-analysis/src/data/bitget/future/DegenReborn_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-analysis/src/data/bitget/future/ENA_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-analysis/src/data/bitget/future/ENS_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-analysis/src/data/bitget/future/ETC_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-analysis/src/data/bitget/future/ETH_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-analysis/src/data/bitget/future/FARTCOIN_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-analysis/src/data/bitget/future/FET_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-analysis/src/data/bitget/future/FUEL_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-analysis/src/data/bitget/future/GALA_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-analysis/src/data/bitget/future/GIGA_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-analysis/src/data/bitget/future/GLM_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-analysis/src/data/bitget/future/GRIFFAIN_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-analysis/src/data/bitget/future/HBAR_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-analysis/src/data/bitget/future/HIVE_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-analysis/src/data/bitget/future/HYPE_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-analysis/src/data/bitget/future/INJ_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-analysis/src/data/bitget/future/IOTA_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-analysis/src/data/bitget/future/IO_USDT:USDT_1m.csv

/home/ubuntu/project/finance/cex-market-analysis/src/data/bitget/future/JASMY_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-analysis/src/data/bitget/future/KMNO_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-analysis/src/data/bitget/future/LINK_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-analysis/src/data/bitget/future/LTC_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-analysis/src/data/bitget/future/MOCA_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-analysis/src/data/bitget/future/MOVE_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-analysis/src/data/bitget/future/NEAR_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-analysis/src/data/bitget/future/NOT_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-analysis/src/data/bitget/future/ONDO_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-analysis/src/data/bitget/future/PENGU_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-analysis/src/data/bitget/future/PEPE_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-analysis/src/data/bitget/future/PHA_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-analysis/src/data/bitget/future/PNUT_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-analysis/src/data/bitget/future/PRCL_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-analysis/src/data/bitget/future/SAND_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-analysis/src/data/bitget/future/SEI_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-analysis/src/data/bitget/future/SHIB_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-analysis/src/data/bitget/future/SOL_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-analysis/src/data/bitget/future/STEEM_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-analysis/src/data/bitget/future/STG_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-analysis/src/data/bitget/future/STX_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-analysis/src/data/bitget/future/SUI_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-analysis/src/data/bitget/future/SUSHI_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-analysis/src/data/bitget/future/TIA_USDT:USDT_1m.csv

```
/home/ubuntu/project/finance/cex-market-
analysis/src/data/bitget/future/TRX_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-
analysis/src/data/bitget/future/UNI_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-
analysis/src/data/bitget/future/USUAL_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-
analysis/src/data/bitget/future/UXLINK_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-
analysis/src/data/bitget/future/VELO_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-
analysis/src/data/bitget/future/VET_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-
analysis/src/data/bitget/future/VIRTUAL_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-
analysis/src/data/bitget/future/WIF_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-
analysis/src/data/bitget/future/WLD_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-
analysis/src/data/bitget/future/XLM_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-
analysis/src/data/bitget/future/XRP_USDT:USDT_1m.csv
/home/ubuntu/project/finance/cex-market-
analysis/src/data/bitget/future/ZEREBRO_USDT:USDT_1m.csv
Results saved to /home/ubuntu/project/finance/cex-market-
analysis/src/results_summary.csv
```

[14]:
```
# Filter for rows with timeframe == "1h"
filtered_df = df_results[df_results['timeframe'] == "1h"]

# Find the row with the maximum upper_wick_percentage
max_upper_wick_row = filtered_df.loc[filtered_df['upper_wick_percentage'].
  ↪idxmax()]

# Extract the symbol name
max_symbol = max_upper_wick_row['symbol']

print(f"Symbol with max upper_wick_percentage for 1h timeframe: {max_symbol}")
df_results[df_results["symbol"]==max_symbol]
```

```
Symbol with max upper_wick_percentage for 1h timeframe: BTC_USDT:USDT_1m
```

[14]:

	symbol	timeframe	total_candles	upper_wick_count	\
112	BTC_USDT:USDT_1m	1m	534265	415541	
113	BTC_USDT:USDT_1m	3m	178089	155725	
114	BTC_USDT:USDT_1m	5m	106854	96975	
115	BTC_USDT:USDT_1m	15m	35619	33905	
116	BTC_USDT:USDT_1m	30m	17810	17264	

```
117  BTC_USDT:USDT_1m        1h            8906              8738
118  BTC_USDT:USDT_1m        2h            4453              4407
119  BTC_USDT:USDT_1m        4h            2227              2216

     neutral_wick_count  upper_wick_percentage  neutral_wick_percentage
112              118724                  77.78                   22.22
113               22364                  87.44                   12.56
114                9879                  90.75                    9.25
115                1714                  95.19                    4.81
116                 546                  96.93                    3.07
117                 168                  98.11                    1.89
118                  46                  98.97                    1.03
119                  11                  99.51                    0.49
```

[15]:
```
## Same analysis only during the week
```

[16]:
```
import pandas as pd
import plotly.graph_objects as go

# Read and process data
df = pd.read_csv("/home/ubuntu/project/finance/cex-market-analysis/src/data/
 ↪bitget/future/BTC_USDT:USDT_1m.csv", header=None)
df = pd.DataFrame(df.values, columns=['date', 'open', 'high', 'low', 'close',␣
 ↪'volume'])
df['date'] = pd.to_datetime(df['date'], unit='ms')

# Extract the weekday (0=Monday, 1=Tuesday, ..., 4=Friday)
df['weekday'] = df['date'].dt.weekday

# Filter for weekdays (Monday to Friday)
df = df.loc[df['weekday'] < 5]

# Set the date as the index
df.set_index('date', inplace=True)

# Resample to daily data
df = df.resample('1d').agg({
    'open': 'first',      # First price in the 1-day window (Open)
    'high': 'max',        # Maximum price in the 1-day window (High)
    'low': 'min',         # Minimum price in the 1-day window (Low)
    'close': 'last',      # Last price in the 1-day window (Close)
    'volume': 'sum'       # Total volume in the 1-day window
}).dropna()  # Drop rows with NaN values (e.g., days with no data)

# Create a candlestick chart using Plotly
fig = go.Figure(data=[go.Candlestick(
    x=df.index,
```

```
    open=df['open'],
    high=df['high'],
    low=df['low'],
    close=df['close'],
    increasing_line_color='green',  # Green for price increase
    decreasing_line_color='red',    # Red for price decrease
)])

# Customize layout
fig.update_layout(
    title='BTC/USDT Candlestick Chart (Weekdays)',
    xaxis_title='Date',
    yaxis_title='Price (USDT)',
    template='plotly_dark',  # Set a dark theme for the plot
    xaxis_rangeslider_visible=False  # Optionally hide the range slider
)

# Show the plot
fig.show()
```

[22]:
```
import pandas as pd
import natsort
import glob
import os

PATH_SAVE = "/home/ubuntu/project/finance/cex-market-analysis/src/data/bitget/future/"
files_path = natsort.natsorted(glob.glob(os.path.join(PATH_SAVE, "*.csv"), recursive=False))

# Initialize an empty list to store results
all_results = []

for file_path in files_path:
    # Extract the symbol name from the filename
    path, filename = os.path.split(file_path)
    symbol = filename.replace(".csv", "")  # Remove the .csv extension

    # Load the data
    df = pd.read_csv(file_path, header=None)
    df = pd.DataFrame(df.values, columns=['date', 'open', 'high', 'low', 'close', 'volume'])
    df['date'] = pd.to_datetime(df['date'], unit='ms')

    # Extract the weekday (0=Monday, 1=Tuesday, ..., 4=Friday)
    df['weekday'] = df['date'].dt.weekday
```

```
    # Filter for weekdays (Monday to Friday)
    df = df.loc[df['weekday'] < 5]
    df.set_index('date', inplace=True)

    # Calculate wick statistics
    results = calculate_wick_statistics(df, timeframe_resampling_map)
    # plot_wick_statistics(results, filename)
    # Flatten results and append to all_results
    for timeframe, stats in results.items():
        all_results.append({
            'symbol': symbol,
            'timeframe': timeframe,
            'total_candles': stats['total_candles'],
            'upper_wick_count': stats['upper_wick_count'],
            'neutral_wick_count': stats['neutral_wick_count'],
            'upper_wick_percentage': stats['upper_wick_percentage'],
            'neutral_wick_percentage': stats['neutral_wick_percentage']
        })
# Convert all_results to a DataFrame
df_results = pd.DataFrame(all_results)
# Save to a CSV file
output_path = "/home/ubuntu/project/finance/cex-market-analysis/src/
 ↪results_summary_weekday.csv"
df_results.to_csv(output_path, index=False)
print(f"Results saved to {output_path}")
```

```
Results saved to /home/ubuntu/project/finance/cex-market-
analysis/src/results_summary_weekday.csv
```

[23]:
```
# Filter for rows with timeframe == "1h"
filtered_df = df_results[df_results['timeframe'] == "1h"]

# Sort the DataFrame by 'upper_wick_percentage' in descending order
top_10_symbols_df = filtered_df.sort_values(by='upper_wick_percentage',␣
 ↪ascending=False).head(10)

# Extract the top 10 symbols
top_10_symbols = top_10_symbols_df[['symbol', 'upper_wick_percentage']]

print("Top 10 Symbols with highest upper_wick_percentage for 1h timeframe:")
print(top_10_symbols)
```

```
Top 10 Symbols with highest upper_wick_percentage for 1h timeframe:
                    symbol  upper_wick_percentage
229      GIGA_USDT:USDT_1m                  99.12
117       BTC_USDT:USDT_1m                  98.02
45      AI16Z_USDT:USDT_1m                  97.92
437       SOL_USDT:USDT_1m                  97.78
```

```
373     PENGU_USDT:USDT_1m                97.75
509     USUAL_USDT:USDT_1m                97.73
189       ETH_USDT:USDT_1m                97.61
389       PHA_USDT:USDT_1m                97.52
245  GRIFFAIN_USDT:USDT_1m                97.50
381      PEPE_USDT:USDT_1m                97.25
```

0.2.1 American Stock Market Open Hours (NYSE / NASDAQ)

0.2.2 Korean Stock Market Open Hours (KOSPI)

0.2.3 European Stock Market Open Hours (London Stock Exchange)