```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        import pandas as pd
        import plotly.graph_objs as go
        import plotly.express as px
```

```
/Users/ommehta/anaconda3/lib/python3.9/site-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (detecte
d version 1.25.2
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

```
In [2]: df = pd.read_csv('spyData.csv')

        df.head()
```

Out[2]:

	Unnamed: 0	#RIC	Date-Time	Open	High	Low	Last	Volume	No. Trades	Dates
0	0	SPY	2010-01-04T08:00:00.000000000-05	112.12	112.18	111.44	112.14	1765282.0	428.0	2010-01-04
1	1	SPY	2010-01-04T08:15:00.000000000-05	112.14	112.16	112.09	112.15	682644.0	674.0	2010-01-04
2	2	SPY	2010-01-04T08:30:00.000000000-05	112.16	112.23	112.14	112.22	988952.0	873.0	2010-01-04
3	3	SPY	2010-01-04T08:45:00.000000000-05	112.22	112.33	112.22	112.32	378532.0	1167.0	2010-01-04
4	4	SPY	2010-01-04T09:00:00.000000000-05	112.33	112.46	112.33	112.39	554833.0	1641.0	2010-01-04

```
In [3]: prices = df['Last']
        log_prices = np.log(prices)
        log_prices[0:5]
```

```
Out[3]: 0    4.719748
        1    4.719837
        2    4.720461
        3    4.721352
        4    4.721975
        Name: Last, dtype: float64
```

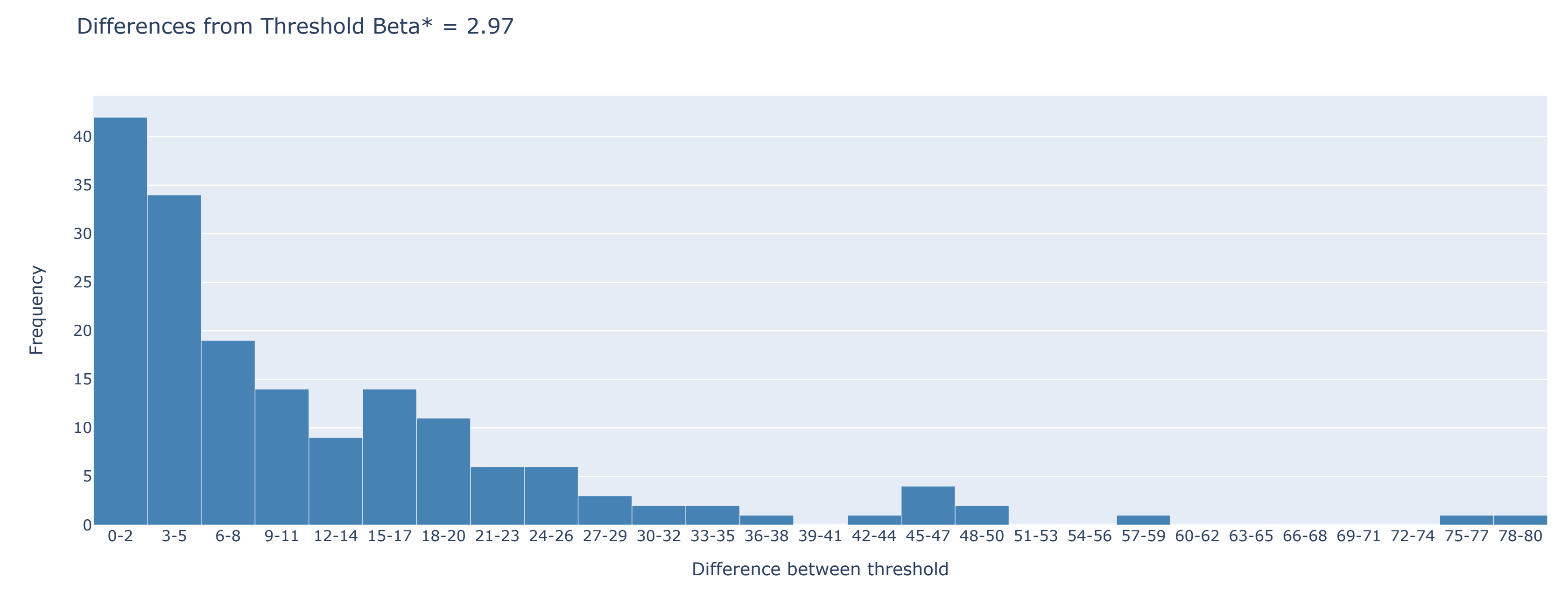
New Lee-Mykland Calibration (2/7/24)

```
In [4]: alpha_K = 0.5
K = 120

# Calibrating the Lee-Mykland Parameters

def calculate_sigma_hat_squared(log_prices, i, K):
    # Ensure we have enough data points to look back K periods from index i
    if i < K + 1:
        return np.nan  # return NaN if there isn't enough history
    # Calculate the sum of squared log returns over the window ending at index i
    sum_squared_log_returns = sum([
        np.abs(log_prices[j] - log_prices[j-1]) * np.abs(log_prices[j-1] - log_prices[j-2])
        for j in range(i-K+2, i)
    ])
    # Compute sigma_hat_squared
    sigma_hat_squared = (1 / (K - 2)) * sum_squared_log_returns
    return sigma_hat_squared

# Assuming the log_prices is a pandas Series, let's apply the function to each index
# We'll use a rolling apply to do this efficiently
sigma_hat_squared_series = pd.Series([calculate_sigma_hat_squared(log_prices, i, K)
                                      for i in range(len(log_prices))])

L_i_series = pd.Series([(log_prices[i]-log_prices[i-1])/np.sqrt(sigma_hat_squared_series[i])
                        for i in range(K-1, len(sigma_hat_squared_series))])
```

```
In [5]: alpha = 0.05

n = len(log_prices)
c = np.sqrt(2/np.pi)
C_n = (np.sqrt(2*np.log(n)))/c - (np.log(np.pi)+np.log(np.log(n)))/(2*c*np.sqrt(2*np.log(n)))
S_n = 1/(c*np.sqrt(2*np.log(n)))
df['Test Statistic'] = pd.Series([(np.abs(L_i_series[i]) - C_n)/S_n for i in range(len(L_i_series))])

Beta_star = -np.log(-np.log(1-alpha))
df['Jump'] = (df['Test Statistic'] > Beta_star).astype(int)
```

Distribution of jumps

```
In [8]: diffs = [i for i in df['Test Statistic'] if i > Beta_star] - Beta_star
# Define bucket size and range
bucket_size = 3
buckets = np.arange(0, max(diffs) + bucket_size, bucket_size)

# Group data into buckets
counts, bins = np.histogram(diffs, bins=buckets)

# Plot histogram
fig = go.Figure(data=[go.Bar(x=bins, y=counts, width=bucket_size, marker_color='steelblue')])
fig.update_layout(title=f'Differences from Threshold Beta* = {round(Beta_star, 2)}',
                  xaxis_title='Difference between threshold',
                  yaxis_title='Frequency',
                  xaxis=dict(tickvals=bins, ticktext=[f'{int(bins[i])}-{int(bins[i+1]-1)}' for i in range(len(bins)-1)]),
                  bargap=1)
fig.show()
```




```
In [54]: jumps_df = df.loc[df['Jump']==1]
jumps_df = jumps_df.drop(columns = ['#RIC', 'Open', 'High', 'Low', 'Unnamed: 0'])
jumps_df['Difference'] = df['Test Statistic'] - Beta_star
jumps_df = jumps_df.sort_values(by=['Difference'])
jumps_df['Date-Time'].tail()
```

```
Out[54]: 399      2010-01-15T17:45:00.000000000-05
15117    2011-07-05T08:45:00.000000000-04
7520     2010-10-01T08:00:00.000000000-04
7521     2010-10-01T08:15:00.000000000-04
3307     2010-05-03T14:45:00.000000000-04
Name: Date-Time, dtype: object
```

5 most severe jumps:

January 15, 2010, 5:45pm

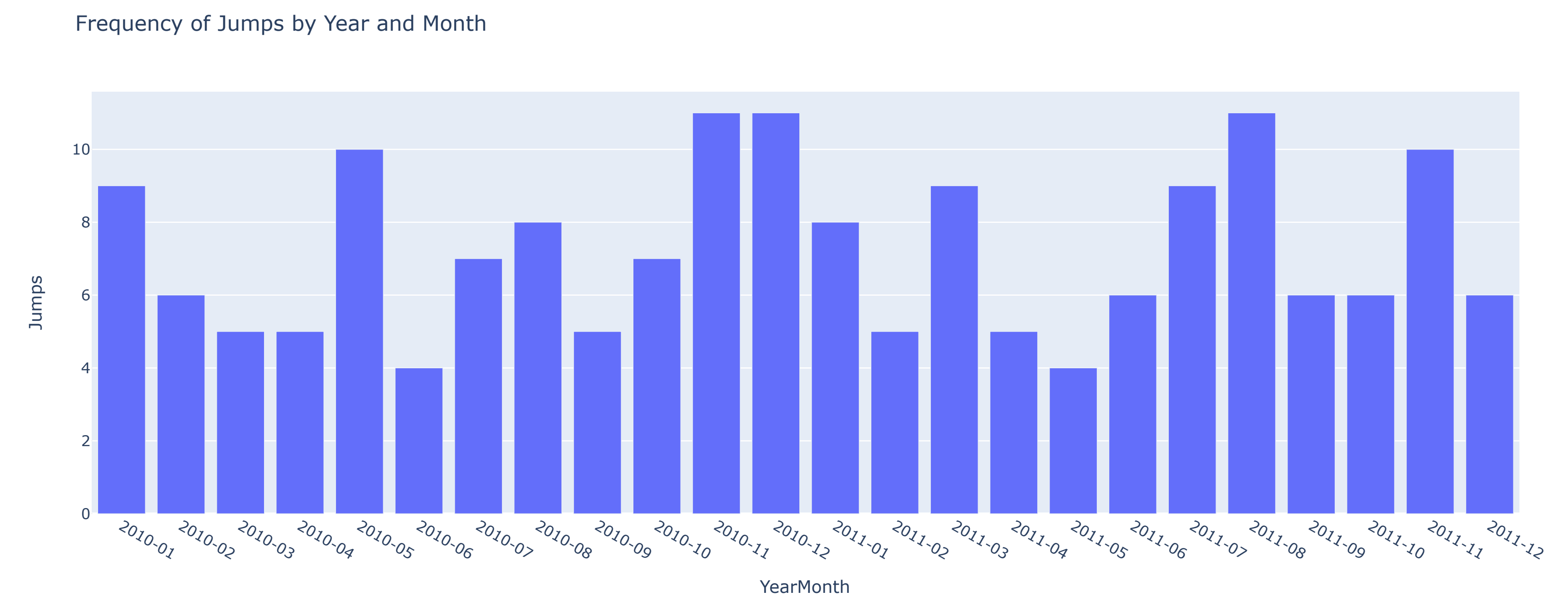
July 5, 2011, 8:45am

October 1, 2010, 8:00am

October 1, 2010, 8:15am

May 3, 2010, 2:45pm

```
In [9]: import plotly.express as px

# Assuming 'df' is your DataFrame and it contains the 'Date-Time' column in datetime format
# Convert 'Date-Time' to datetime format if it's not already
df['Date-Time'] = pd.to_datetime(df['Date-Time'])

# Create a 'YearMonth' column by extracting year and month as strings and concatenating them
df['YearMonth'] = df['Date-Time'].apply(lambda x: f"{x.year}-{str(x.month).zfill(2)}")

# Group by 'YearMonth' and sum 'Jumps'
monthly_jumps = df.groupby('YearMonth')['Jumps'].sum().reset_index()

# Plot the histogram
fig = px.bar(monthly_jumps, x='YearMonth', y='Jumps', title='Frequency of Jumps by Year and Month')
fig.update_xaxes(type='category')
fig.show()
```



There were 10-11 jumps in May 2010, Nov/Dec 2010, Aug 2011, Nov 2011.

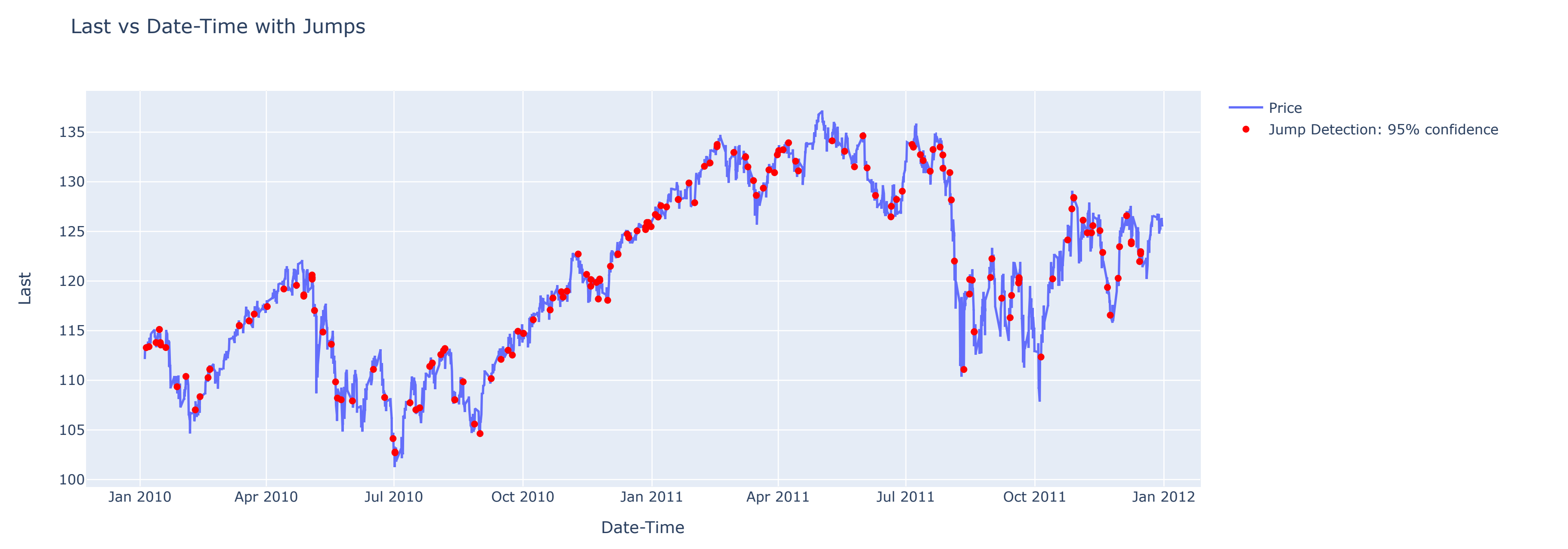
Possible Explanations:

May 2010: Flash Crash

Nov/Dec 2010 and Aug 2011: Concerns surrounding Euro sovereign debt and US debt ceiling

Nov 2011: First ever time S&P downgraded the US credit rating from AAA to AA+

```
In [10]: # Convert the 'Date-Time' column to datetime
#df['Date-Time'] = pd.to_datetime(df['Date-Time'])

# Create a scatter trace for the 'Last' values
trace0 = go.Scatter(
    x=df['Date-Time'],
    y=df['Last'],
    mode='lines',
    name='Price'
)

# Create a scatter trace for the jumps, where 'jump' equals 1
jump_points = df[df['Jumps'] == 1]
trace1 = go.Scatter(
    x=jump_points['Date-Time'],
    y=jump_points['Last'],
    mode='markers',
    name='Jump Detection: 95% confidence',
    marker=dict(color='red')
)

# Define the layout for the plot
layout = go.Layout(
    title='Last vs Date-Time with Jumps',
    xaxis=dict(title='Date-Time'),
    yaxis=dict(title='Last'),
    showlegend=True
)

# Create the figure with both traces
fig = go.Figure(data=[trace0, trace1], layout=layout)

# Display the figure
fig.show()
```



173 total jumps at the 5% significance level.

Old "Lee-Mykland" calibration (Dec 2023) - similar but not exact

```
In [ ]: df['log_return'] = np.log(df['Last'] / df['Last'].shift(1))

# Drop the NaN values created by the shift operation
df = df.dropna()

# Set the window size K for calculating the rolling standard deviation of log returns
# Assuming K is given in the paper or can be chosen appropriately
K = 60  # Example window size, the appropriate value should be chosen based on the research paper

# Calculate the rolling standard deviation of log returns
df['rolling_std'] = df['log_return'].rolling(window=K).std()

# Calculate the test statistic L(i) for each point in time
df['L'] = df['log_return'] / df['rolling_std']

# Choose a significance level α (e.g., a5%)
alpha = 0.05

# Calculate the threshold value for jump detection
# Assuming the formula for the threshold is given in the paper and using norm.ppf for the inverse cumulative distribution function
# Adjust the formula according to the one provided in the paper
threshold = -np.log(-np.log(1 - alpha))

# Identify jumps by comparing the absolute value of L(i) with the threshold
df['jump'] = (df['L'].abs() > threshold).astype(int)

# Count the number of jumps detected
num_jumps = df['jump'].sum()

num_jumps
```

```
In [ ]: jumps = np.where(df['jump']==1)[0].tolist() #indexes of jumps
```

```
In [ ]: df['Return'] = np.log1p(df['Last'].pct_change())
```

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

```
In [ ]: jump_returns = df.loc[df['jump'] == 1, 'Return'].tolist()
```

```
In [ ]: positive = 0
negative = 0
pos_jumps = []
neg_jumps = []

for jump in jump_returns:

    if jump > 0:
        positive += 1
        pos_jumps.append(jump)

    if jump < 0:
        negative += 1
        neg_jumps.append(jump)
```

```
In [ ]: np.mean(neg_jumps)
```