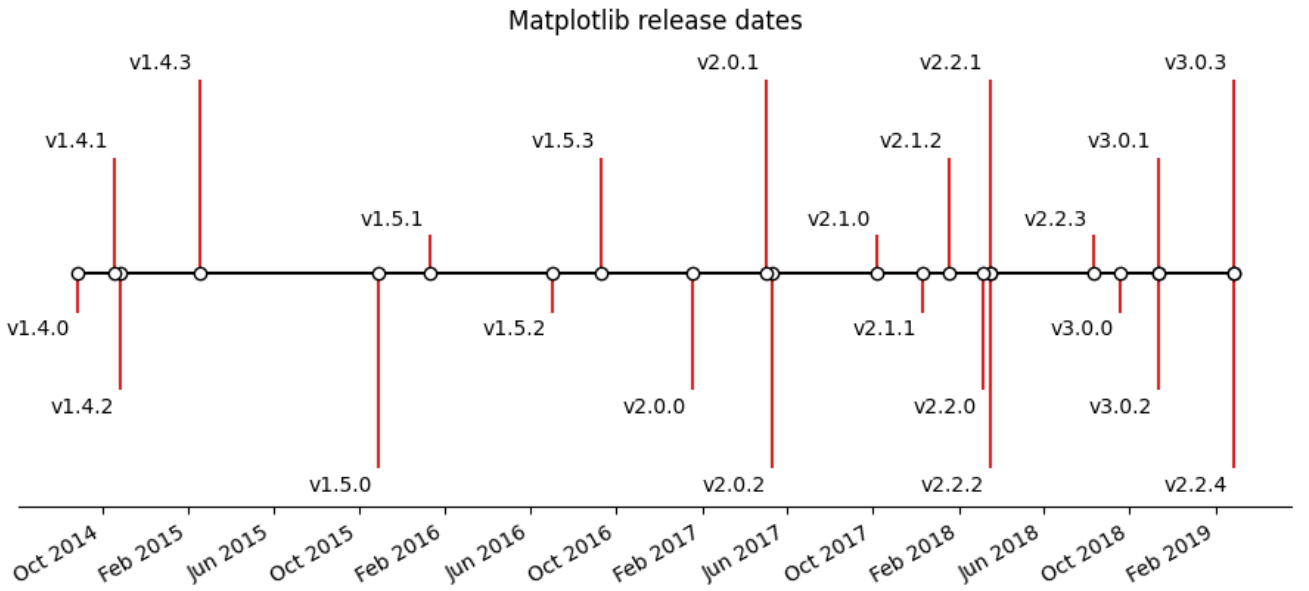
Chapter 8

Timeline Chart, candlestick chart, Cross spectral density (CSD)

Timeline Chart



- ใช้แสดงลำดับเหตุการณ์ตามเวลา ทำให้เห็นลำดับเหตุการณ์ที่เกิดขึ้นในช่วงเวลาต่างๆ หรือการวิเคราะห์และนำเสนอข้อมูลที่เกี่ยวข้องกับเวลา ช่วยให้ข้อมูลที่ซับซ้อนและละเอียดมากกลายเป็นข้อมูลที่เข้าใจง่ายและชัดเจน

Import packet

```
from datetime import datetime
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.dates as mdates
```

กำหนดข้อมูลใน list เก็บไว้ในตัวแปร


```
names = ['v2.2.4', 'v3.0.3', 'v3.0.2', 'v3.0.1', 'v3.0.0', 'v2.2.3',
         'v2.2.2', 'v2.2.1', 'v2.2.0', 'v2.1.2', 'v2.1.1', 'v2.1.0',
         'v2.0.2', 'v2.0.1', 'v2.0.0', 'v1.5.3', 'v1.5.2', 'v1.5.1',
         'v1.5.0', 'v1.4.3', 'v1.4.2', 'v1.4.1', 'v1.4.0']

dates = ['2019-02-26', '2019-02-26', '2018-11-10', '2018-11-10',
         '2018-09-18', '2018-08-10', '2018-03-17', '2018-03-16',
         '2018-03-06', '2018-01-18', '2017-12-10', '2017-10-07',
         '2017-05-10', '2017-05-02', '2017-01-17', '2016-09-09',
         '2016-07-03', '2016-01-10', '2015-10-29', '2015-02-16',
         '2014-10-26', '2014-10-18', '2014-08-26']
```

แปลงข้อมูล datetime และกำหนดข้อมูล levels

- `dates = [datetime.strptime(d, "%Y-%m-%d") for d in dates]`
- วนลูปแปลงข้อมูลในตัวแปร dates ที่ละตัวจากรูปแบบ string เป็น datetime จนครบทุกตัว และเก็บในตัวแปร dates

- `levels = np.tile([-5, 5, -3, 3, -1, 1],`
- `int(np.ceil(len(dates)/6)))[:len(dates)]`
- การกำหนดระดับความยาวของเส้นที่ลากจากเส้น timeline
- `[-5, 5, -3, 3, -1, 1]` คือกำหนดลำดับและความยาวเส้นที่ต้องการ ในที่นี้คือ 6 เส้น
- `int(np.ceil(len(dates)/6))` คือให้กำหนดความยาวเส้นทั้ง 6 ระดับ ตามจำนวนวันทั้งหมดใน dates
- `[:len(dates)]` คือกำหนดจำนวนวันทั้งหมด

กำหนดขนาดและชื่อของกราฟ

- `fig, ax = plt.subplots(figsize=(8.8, 4), layout="constrained")`
- `ax.set(title="Matplotlib release dates")`

- กำหนดขนาดของกราฟเป็นกว้าง 8.8 นิ้ว สูง 4 นิ้ว
- `layout="constrained"` เป็น perameter ที่จะช่วยจัดหน้าและปรับขนาดของแต่ละ subplot ให้มีขนาดที่เหมาะสม ไม่ซ้อนทับกันและไม่เกินขอบเขตของ figure size ที่กำหนดไว้ ช่วยลดความสับสนในการอ่านข้อมูลและทำให้กราฟสวยงาม
- กำหนดชื่อกราฟ Matplotlib release dates

สร้างเส้น timeline ของกราฟ

- `ax.vlines(dates, 0, levels, color="tab:red")`
- เป็นการลากเส้นจากเวลา(dates) แต่ละจุดเป็นเส้นตรงตั้งฉากเริ่มจาก 0 ของเส้น timeline ไปจนถึงระดับความยาวในตัวแปร levels ที่กำหนดไว้ และกำหนดสีเป็นสีแดง

- `ax.plot(dates, np.zeros_like(dates), "-o",`
- `color="k", markerfacecolor="w")`
- ใช้สร้างเส้น timeline และเครื่องหมายบนเส้นตามค่าเวลา(dates) แต่ละจุด
- `np.zeros_like(dates)` คือการกำหนดจุดเวลาทุกค่าใน dates จะมีค่าเท่ากับ 0 ทั้งหมดและลากเส้นกราฟเป็นเส้นตรงเชื่อมระหว่างจุดทุกจุด
- `"-o", color="k", markerfacecolor="w"` คือกำหนดเครื่องหมายเป็นจุด กำหนดสีจุดเป็นสีดำ และกำหนดตรงกลางจุดเป็นสีขาว

กำหนดตำแหน่งข้อความในกราฟ

```
for d, l, r in zip(dates, levels, names):
    ax.annotate(r, xy=(d, l),
                xytext=(-3, np.sign(l)*3), textcoords="offset points",
                horizontalalignment="right",
                verticalalignment="bottom" if l > 0 else "top")
```

```
for d, l, r in zip(dates, levels, names):
```

- วนลูปอ่านข้อมูล dates, levels, names แต่ละข้อมูลเก็บข้อมูลที่อ่านไว้ในตัวแปร d, l, r ตามลำดับ
- `ax.annotate(r, xy=(d, l)` คือแสดงข้อความในตัวแปร r (names) ในตำแหน่ง x = d(dates)และ y = l(levels)

กำหนดตำแหน่งข้อความในกราฟ

- `xytext=(-3, np.sign(l)*3)` ใช้กำหนดตำแหน่งข้อความที่จะแสดงในรูปแบบพิกเซล
- `-3` แทนตำแหน่งข้อความในแนวแกน x คือด้านซ้ายของจุดที่กำหนด
- `np.sign(l)*3` แทนตำแหน่งข้อความในแนวแกน y คือด้านบนหรือด้านล่างตามค่าระดับ levels ที่กำหนด

- `textcoords="offset points"` คือการระบุว่าตำแหน่งของข้อความที่แสดงจะใช้ระบบพิกเซล
- `horizontalalignment="right"` คือการจัดตำแหน่งจุดที่จะแสดงข้อความให้อยู่ทางขวาของข้อความในแนวแกน x
- `verticalalignment="bottom" if l > 0 else "top")`
- ใช้จัดตำแหน่งข้อความให้อยู่ด้านล่างของจุดที่กำหนดถ้าค่า l มากกว่า 0 ถ้าน้อยกว่า 0 ให้จัดตำแหน่งข้อความให้อยู่ด้านบน เพื่อปรับตำแหน่งข้อความให้อยู่ในทิศทางที่เหมาะสมตามค่า levels ที่กำหนด

กำหนดเครื่องหมายและข้อความบนแกน x

- `ax.xaxis.set_major_locator(mdates.MonthLocator(interval=4))`
- `ax.xaxis.set_major_formatter(mdates.DateFormatter("%b %Y"))`
- `plt.setp(ax.get_xticklabels(), rotation=30, ha="right")`

- `ax.xaxis.set_major_locator(mdates.MonthLocator(interval=4))`
- ใช้กำหนดตำแหน่งเครื่องหมายขีดบนแกน x ให้แสดงทุกๆ 4 เดือน

- `ax.xaxis.set_major_formatter(mdates.DateFormatter("%b %Y"))`
- ใช้กำหนดรูปแบบข้อความที่จะใช้แสดงบนแกน x %b คือชื่อเดือนแบบย่อ %y คือปี

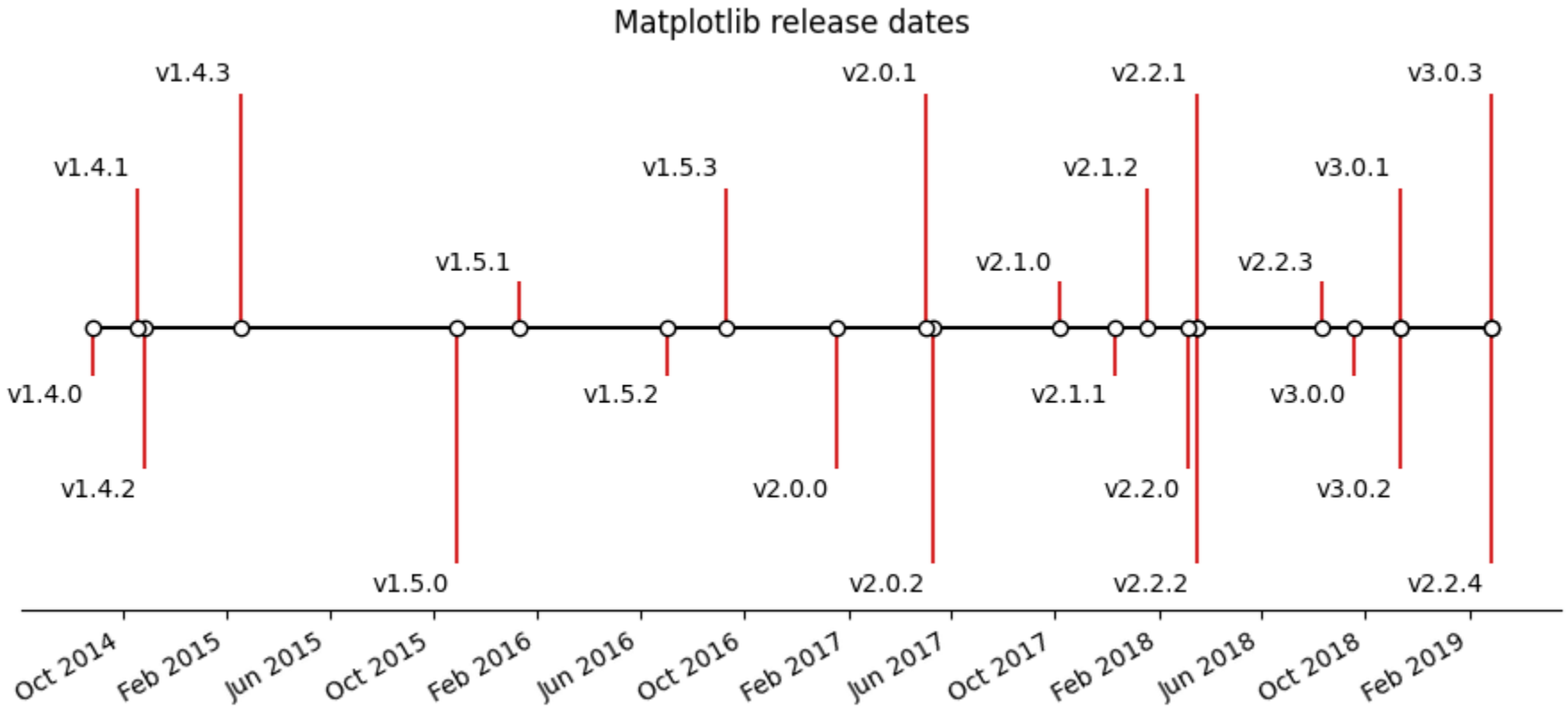
- `plt.setp(ax.get_xticklabels(), rotation=30, ha="right")`
- ใช้กำหนดให้ข้อความบนแกน x หมุนไปทางขวา 30 องศา เพื่อให้ข้อความไม่ทับซ้อนกัน

กำหนดส่วนประกอบต่างๆของกราฟ

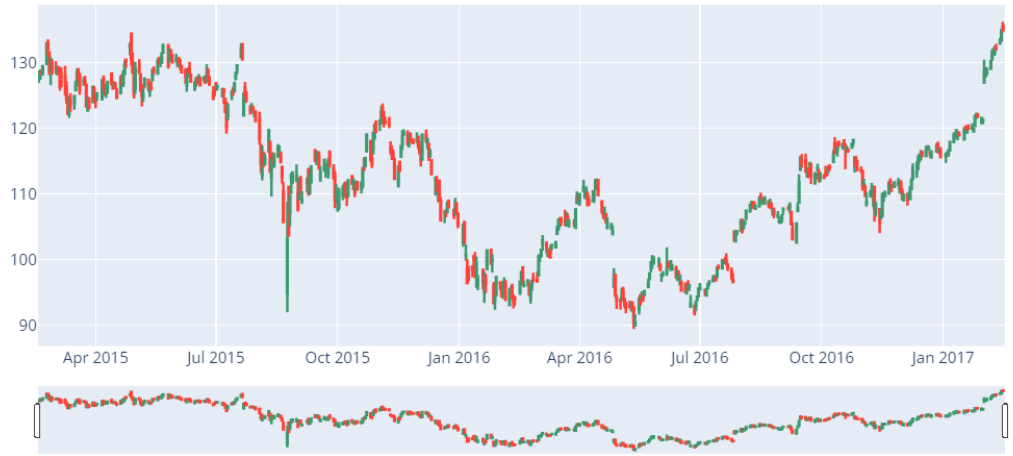
- `ax.yaxis.set_visible(False)`
- `ax.spines[["left", "top", "right"]].set_visible(False)`
- `ax.margins(y=0.1)`
- `plt.show()`

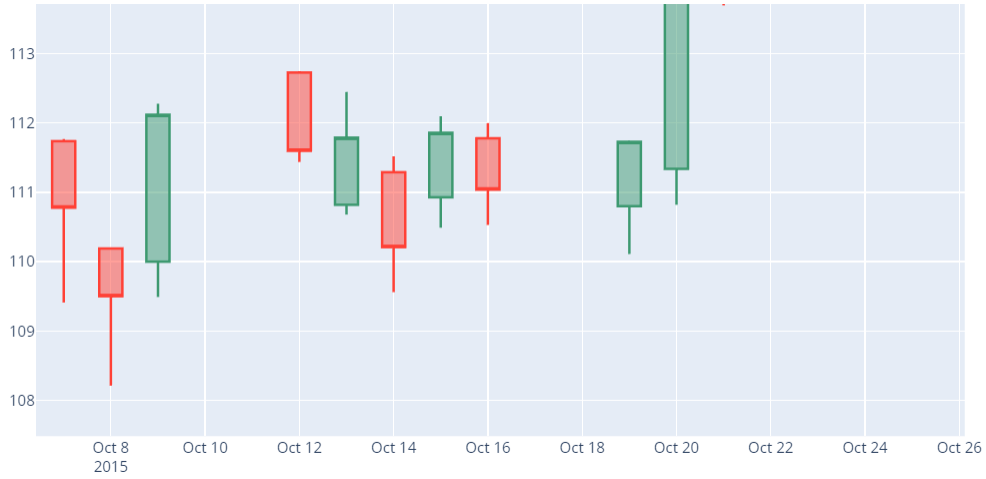
- `ax.yaxis.set_visible(False)` ปิดการแสดงแกน y บนกราฟ
- `ax.spines[["left", "top", "right"]].set_visible(False)`
- ปิดการแสดงเส้นขอบของกราฟด้านซ้าย ด้านบน ด้านขวา
- `ax.margins(y=0.1)`
- เพิ่มพื้นที่ช่องว่างด้านบนและด้านล่างของกราฟ 10% ของความสูงทั้งหมดของข้อมูลที่กราฟแสดงบนกราฟ เพื่อให้กราฟอยู่ตรงกลางและทำให้กราฟดูง่ายสวยงาม
- `plt.show()` แสดงกราฟที่สร้างขึ้นตามการกำหนดค่าต่างๆด้วย Matplotlib

ผลลัพธ์


Matplotlib release dates
v1.4.3
v2.0.1
v2.2.1
v3.0.3
v1.4.1
v1.5.3
v2.1.2
v3.0.1
v1.5.1
v2.1.0
v2.2.3
v1.4.0
v1.5.2
v2.1.1
v3.0.0
v1.4.2
v2.0.0
v2.2.0
v3.0.2
v1.5.0
v2.0.2
v2.2.2
v2.2.4
Oct 2014
Feb 2015
Jun 2015
Oct 2015
Feb 2016
Jun 2016
Oct 2016
Feb 2017
Jun 2017
Oct 2017
Feb 2018
Jun 2018
Oct 2018
Feb 2019

Candlestick Chart



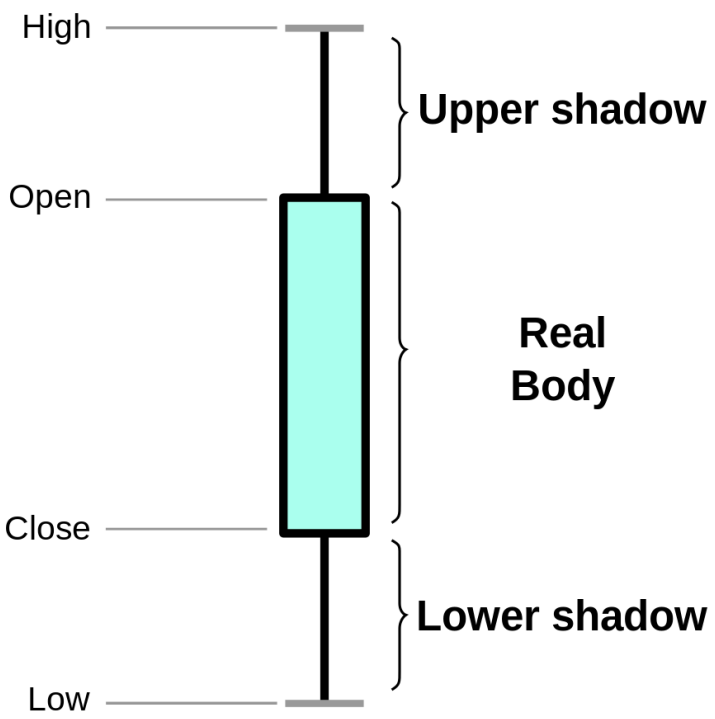


- Candlestick Chart หรือกราฟแท่งเทียน ใช้แสดงข้อมูลราคาของหลักทรัพย์หรือสินทรัพย์ ดูแนวโน้มของตลาด

ส่วนประกอบของ Candlestick Chart

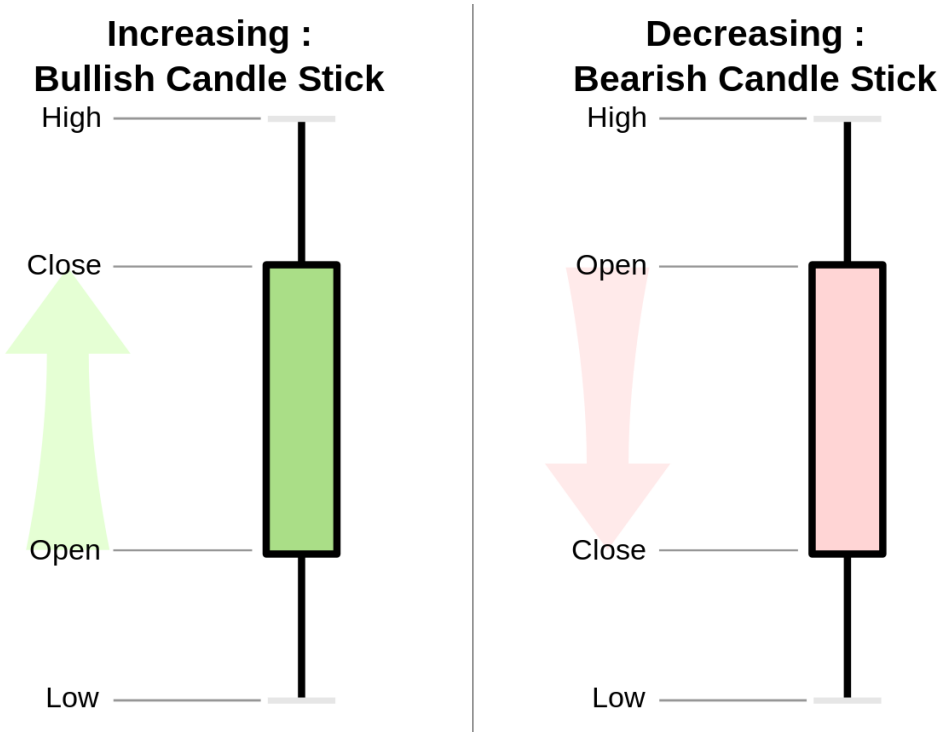
- ส่วนประกอบของ Candlestick Chart

1. จุดสูงสุด (High) คือ จุดที่สูงที่สุดของแท่งเทียนนั้น
2. จุดต่ำสุด (Low) คือ จุดที่ต่ำที่สุดของแท่งเทียนนั้น
3. ราคาเปิด (Open) คือ ช่วงที่เริ่มต้นของแท่งเทียน
4. ราคาปิด (Close) คือ ช่วงสุดท้ายก่อนจบแท่งเทียนนั้น
5. เนื้อเทียน (Body) คือ ช่วงระยะห่างของราคาเปิดและราคาปิด
6. ไส้เทียน (Shadow) คือ ช่วงที่เลยออกนอกราคาปิดและราคาเปิด เพื่อแสดงให้ทราบว่าราคาเคยไปถึงจุดนั้นแล้ว



วิธีการดูแนวโน้มเบื้องต้น

- วิธีการดูแนวโน้มเบื้องต้น แนวโน้มมี 2 แบบ
- Bullish แนวโน้มขาขึ้น
- Bearish แนวโน้มขาลง



Import packet

- `import plotly.graph_objects as go`
- `import pandas as pd`
- `from datetime import datetime`

Download data to memory

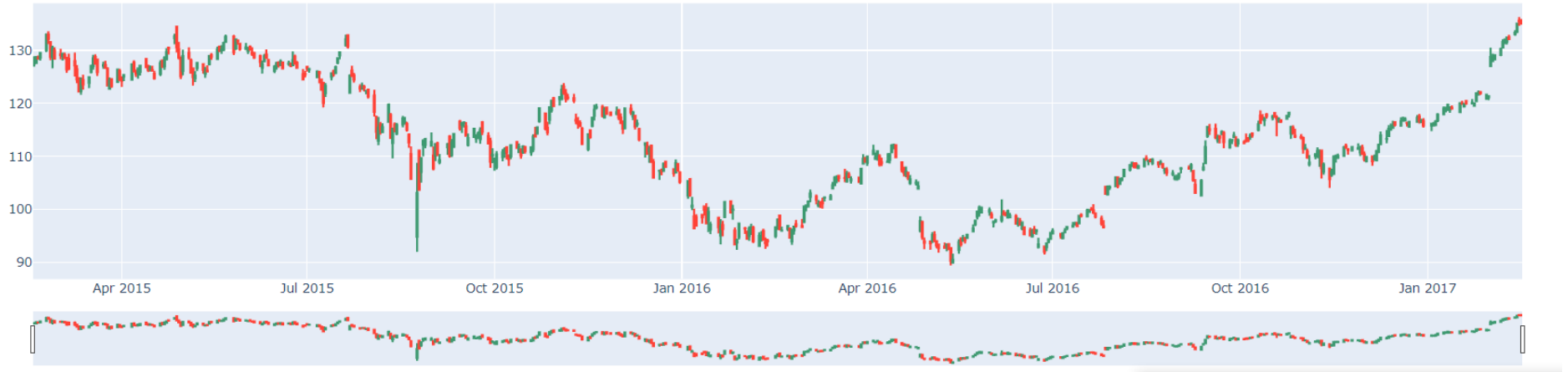
- `df = pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/master/finance-charts-apple.csv')`
- `df`

	Date	AAPL.Open	AAPL.High	AAPL.Low	AAPL.Close	AAPL.Volume	AAPL.Adjusted	dn	mavg	up	direction
0	2015-02-17	127.489998	128.880005	126.919998	127.830002	63152400	122.905254	106.741052	117.927667	129.114281	Increasing
1	2015-02-18	127.629997	128.779999	127.449997	128.720001	44891700	123.760965	107.842423	118.940333	130.038244	Increasing
2	2015-02-19	128.479996	129.029999	128.330002	128.449997	37362400	123.501363	108.894245	119.889167	130.884089	Decreasing
3	2015-02-20	128.619995	129.500000	128.050003	129.500000	48948400	124.510914	109.785449	120.763500	131.741551	Increasing
4	2015-02-23	130.020004	133.000000	129.660004	133.000000	70974100	127.876074	110.372516	121.720167	133.067817	Increasing
...	...	...	...	...	...	...	...	...	...	...	...
501	2017-02-10	132.460007	132.940002	132.050003	132.119995	20065500	132.119995	114.494004	124.498666	134.503328	Decreasing
502	2017-02-13	133.080002	133.820007	132.750000	133.289993	23035400	133.289993	114.820798	125.205166	135.589534	Increasing
503	2017-02-14	133.470001	135.089996	133.250000	135.020004	32815500	135.020004	115.175718	125.953499	136.731280	Increasing
504	2017-02-15	135.520004	136.270004	134.619995	135.509995	35501600	135.509995	115.545035	126.723499	137.901963	Decreasing
505	2017-02-16	135.669998	135.899994	134.839996	135.350006	22118000	135.350006	116.203299	127.504333	138.805366	Decreasing

506 rows × 11 columns

สร้าง Candlestick Chart

```
fig = go.Figure(data=[go.Candlestick(x=df['Date'],
                open=df['AAPL.Open'],
                high=df['AAPL.High'],
                low=df['AAPL.Low'],
                close=df['AAPL.Close'])])
fig.show()
```

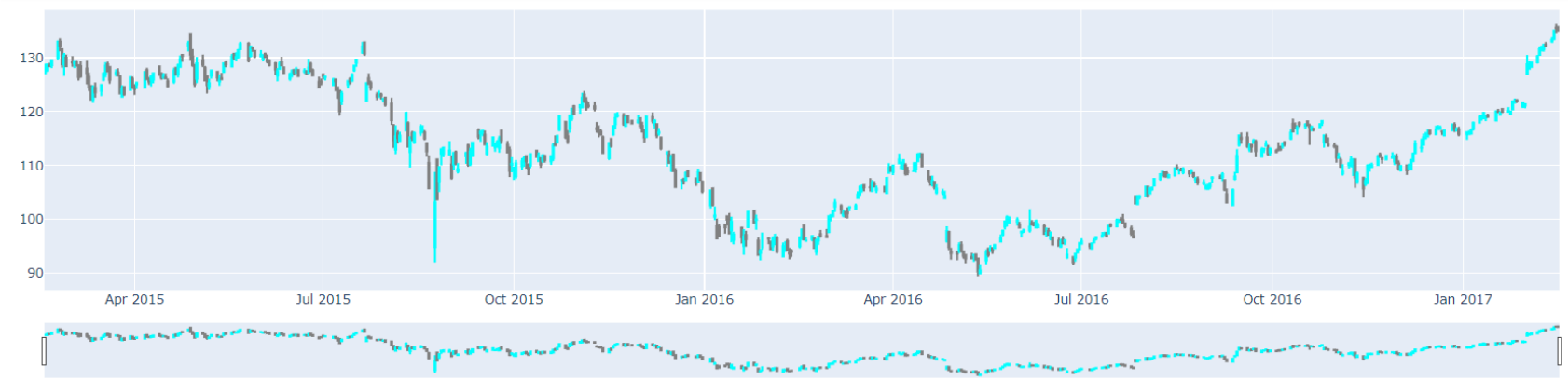


Candlestick without Rangeslider

```
fig = go.Figure(data=[go.Candlestick(x=df['Date'],
                open=df['AAPL.Open'], high=df['AAPL.High'],
                low=df['AAPL.Low'], close=df['AAPL.Close'])
                     ])
fig.update_layout(xaxis_rangeslider_visible=False)
fig.show()
```

Custom Candlestick Colors

```
fig = go.Figure(data=[go.Candlestick(
    x=df['Date'],
    open=df['AAPL.Open'], high=df['AAPL.High'],
    low=df['AAPL.Low'], close=df['AAPL.Close'],
    increasing_line_color= 'cyan', decreasing_line_color= 'gray'
)])
fig.show()
```

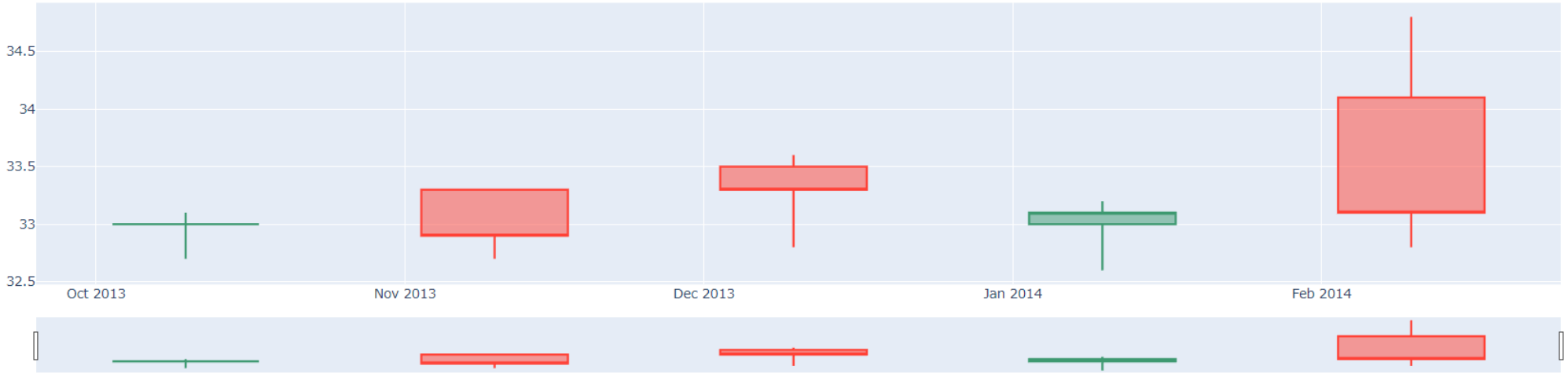


Simple Example with datetime Objects

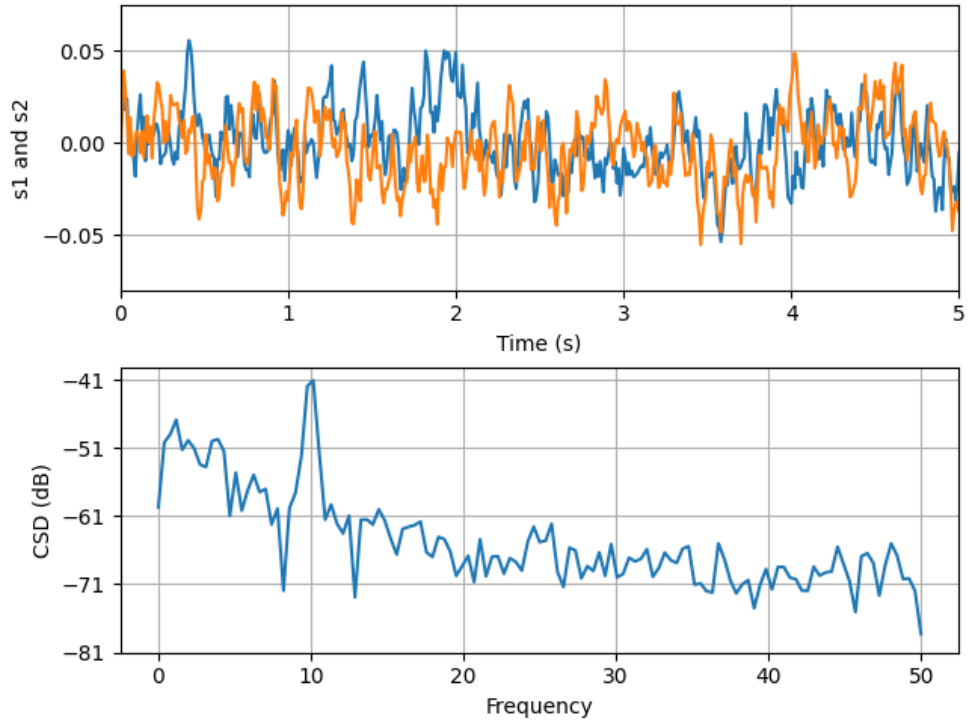

```
open_data = [33.0, 33.3, 33.5, 33.0, 34.1]
high_data = [33.1, 33.3, 33.6, 33.2, 34.8]
low_data = [32.7, 32.7, 32.8, 32.6, 32.8]
close_data = [33.0, 32.9, 33.3, 33.1, 33.1]
dates = [datetime(year=2013, month=10, day=10),
         datetime(year=2013, month=11, day=10),
         datetime(year=2013, month=12, day=10),
         datetime(year=2014, month=1, day=10),
         datetime(year=2014, month=2, day=10)]
```

Simple Example with datetime Objects

- `fig = go.Figure(data=[go.Candlestick(x=dates,`
- `open=open_data, high=high_data,`
- `low=low_data, close=close_data)])`
- `fig.show()`



Cross spectral density (CSD)



- ใช้ในการวิเคราะห์ความสัมพันธ์ระหว่างสองคลื่นสัญญาณที่มีความถี่ต่าง ๆ ในช่วงเวลาเดียวกัน ใช้ในการศึกษาความสัมพันธ์ระหว่างสองตัวแปรหรือสองคลื่นสัญญาณที่เป็นไปในทิศทางเดียวกันหรือทิศทางตรงกันข้าม CSD มีความสำคัญในด้านการวิเคราะห์การแสดงออกของข้อมูลที่มีความซับซ้อน โดยเฉพาะในงานที่เกี่ยวข้องกับการสังเคราะห์คลื่นสัญญาณ

วิธีอ่านกราฟ Cross spectral density (CSD)

- การอ่านกราฟ Cross Spectral Density (CSD)
- แกน x (Frequency): แสดงค่าความถี่ของสัญญาณที่ถูกวิเคราะห์ ความถี่ส่วนใหญ่จะแสดงหน่วยเป็น Hz
- แกน y (CSD): แสดงค่า Cross Spectral Density (CSD) หรือความสัมพันธ์ระหว่างสองสัญญาณที่ถูกวิเคราะห์ ค่า CSD สูงแสดงถึงความสัมพันธ์มากขึ้นระหว่างสองสัญญาณ ในขณะที่ค่า CSD ต่ำแสดงถึงความสัมพันธ์ที่น้อยลงระหว่างสองสัญญาณ
- การเปลี่ยนแปลงของกราฟ: การดูว่ากราฟ CSD มีลักษณะการเปลี่ยนแปลงอย่างไร ที่จุดไหนมีค่า CSD สูงขึ้นหรือต่ำลง วิเคราะห์ความสัมพันธ์ระหว่างสองสัญญาณในแต่ละความถี่
- ยกตัวอย่างการเปรียบเทียบคลื่นไฟฟ้าหัวใจกับคลื่นไฟฟ้าสมอง และการคำนวณ CSD นำไปสร้างกราฟ

Import

- `import numpy as np`
- `from scipy import signal`
- `import matplotlib.pyplot as plt`

จำลองข้อมูลเวลาในรูปแบบความถี่

- `num_samples = 1000`
- `time = np.linspace(0, 1, num_samples)`

- ใช้สร้าง array ตัวแปร time ที่ประกอบด้วยค่าเวลาตั้งแต่ 0 ถึง 1 ที่มีขนาดทั้งหมด 1000 ตัวอย่างโดย `np.linspace()` จะแบ่งค่าเวลาตั้งแต่ 0 ถึง 1 ออกเป็นช่วงย่อยๆ ที่มีจำนวนเท่าๆ กัน ทั้งหมด 1000 ช่วง

จำลองข้อมูลคลื่นไฟฟ้าหัวใจ ECG

```
p_wave = np.sin(2 * np.pi * 2 * time)
qrs_complex = (
    + 0.2 * np.sin(2 * np.pi * 10 * time)
    + 0.3 * np.sin(2 * np.pi * 20 * time)
    + 0.1 * np.sin(2 * np.pi * 30 * time)
)
t_wave = np.sin(2 * np.pi * 1 * time)
ecg_signal = p_wave + qrs_complex + t_wave
```

- `np.sin()` คือฟังก์ชัน sine ที่ใช้สำหรับคำนวณค่า sine ของผลลัพธ์จากการคำนวณใน ()
- `np.pi` คือค่าของ Π (พาย) มีค่าประมาณเท่ากับ 3.14159 หรือ 22/7

จำลองข้อมูลคลื่นไฟฟ้าหัวใจ ECG

- `p_wave` จำลองคลื่นที่เกิดขึ้นเมื่อหัวใจขยับและส่งสัญญาณไปยังห้องหัวใจ (atria) มีความถี่ 2 Hz
- `qrs_complex` จำลองคลื่นที่เกิดขึ้นเมื่อสัญญาณไฟฟ้าเดินลงไปยังห้องหลัง (ventricles) จากการผสมคลื่น 3 ความถี่ 10 Hz, 20 Hz, 30 Hz ตามอัตราส่วนที่กำหนด
- `t_wave` จำลองคลื่นที่เกิดขึ้นเมื่อหัวใจเตรียมพร้อมที่จะเตรียมให้กลับไปสู่สถานะพัก (resting state) โดยเป็นคลื่น sine wave ที่มีความถี่ต่ำ 1 Hz
- `ecg_signal` รวมค่าคลื่นทั้ง 3 จำลองคลื่นสัญญาณทางไฟฟ้าที่เกิดขึ้นในหัวใจตลอดเวลาในรูปแบบของกราฟ ECG

จำลองข้อมูลคลื่นไฟฟ้าสมอง EEG ในสภาวะ BETA

- `brain_signal = np.cos(2 * np.pi * 20 * time)`
- `random_movement = np.random.normal(loc=0, scale=0.1, size=num_samples)`
- `brain_signal += random_movement`

- `np.cos()` คือฟังก์ชัน cosine ที่ใช้สำหรับคำนวณค่า cosine ของผลลัพธ์จากการคำนวณใน () คือคลื่นที่มีความถี่ 20 Hz
- `np.random.normal(loc=0, scale=0.1, size=num_samples)` คือการสร้างการกระจายแบบ normal distribution ที่มีค่าเฉลี่ยเท่ากับ 0 และส่วนเบี่ยงเบนเท่ากับ 0.1 ด้วยขนาดของข้อมูลที่กำหนด
- `brain_signal += random_movement` คือการบวกค่า random_movement ลงใน brain_signal เพื่อนำไปใช้ในการจำลองการทำงานของสมองซึ่ง random_movement ที่บวกเข้าไปเป็นค่าที่ถูกสุ่มมาจากการกระจาย normal distribution ทำให้คลื่นที่ออกมาดูเป็นธรรมชาติคล้ายของจริงมากขึ้น

คำนวณ CSD ระหว่างคลื่นไฟฟ้าหัวใจและสมอง

- `frequencies, csd = signal.csd(ecg_signal, brain_signal, fs=1.0, nperseg=100)`

- คำนวณความสัมพันธ์ของคลื่นระหว่างสัญญาณคลื่นทางไฟฟ้าของหัวใจ (ecg_signal) และสัญญาณทางไฟฟ้าของสมอง (brain_signal) โดยใช้ Cross Spectral Density (CSD)
- signal.csd() ฟังก์ชันที่ใช้ในการคำนวณ Cross Spectral Density (CSD) ระหว่างสองสัญญาณ
- ecg_signal, brain_signal: ตัวแปรสัญญาณคลื่นที่จะถูกนำมาคำนวณ CSD
- fs=1.0: ค่าอัตราสุ่มของสัญญาณ (sampling frequency) ตั้งค่าให้มีค่าเป็น 1.0 เนื่องจากเรากำหนดเวลาในหน่วยวินาที
- nperseg=100: number of points per segment ใช้ในการคำนวณ CSD ในแต่ละครั้ง กำหนดให้มีค่าเป็น 100 จุด
- ผลลัพธ์ที่ได้ frequencies เป็นค่าความถี่และ csd เป็นค่า Cross Spectral Density ระหว่างสองสัญญาณ

การ plot กราฟคลื่นไฟฟ้าหัวใจและสมอง และกราฟ CSD

- `fig, (ax1, ax2) = plt.subplots(2, 1, layout='constrained')`

- สร้างรูป (Figure) และ subplot 2 แถว 1 คอลัมน์ และเก็บตำแหน่งของแต่ละ subplot ไว้ในตัวแปร ax1 และ ax2 โดยใช้ layout เป็น 'constrained' เพื่อให้การจัดวาง subplot เป็นแบบที่สอดคล้องกับขนาดของรูป

สร้างกราฟคลื่นไฟฟ้าหัวใจและสมอง

- `ax1.plot(time, ecg_signal, label='Heart Signal (ECG)')`
- `ax1.plot(time, brain_signal, label='Brain Signal (EEG-BATA)’)`

- สร้างกราฟของสัญญาณคลื่นไฟฟ้าของหัวใจ (ECG) บนแกน x คือเวลา (time) และแกน y คือค่าของตัวแปร (ecg_signal) โดยใส่ label เพื่อให้เข้าใจง่ายว่าเป็นสัญญาณคลื่นไฟฟ้าของหัวใจ
- สร้างกราฟของสัญญาณคลื่นไฟฟ้าของสมอง (EEG-BATA) บนแกน x คือเวลา (time) และแกน y คือค่า ของตัวแปร (brain_signal) โดยใส่ label เพื่อให้เข้าใจง่ายว่าเป็นสัญญาณคลื่นไฟฟ้าของสมอง

กำหนดส่วนประกอบต่างๆ ของกราฟคลื่นไฟฟ้าหัวใจและสมอง

- `ax1.set_xlabel('Time')`
- `ax1.set_ylabel('Amplitude')`
- `ax1.set_title('Heart and Brain Signals')`
- `ax1.legend()`
- `ax1.grid(True)`

- กำหนดชื่อแกน X เป็น 'Time’
- กำหนดชื่อแกน y เป็น 'Amplitude’
- กำหนดชื่อกราฟเป็น 'Heart and Brain Signals’
- เพิ่มสัญลักษณ์ของคำอธิบายกราฟ (legend) แสดงความหมายของแต่ละสีของเส้นกราฟ
- เปิดการแสดงเส้นกริดบนกราฟเพื่อช่วยในการอ่านค่าในกราฟ

สร้างกราฟ CSD

- `ax2.semilogy(frequencies, np.abs(csd))`

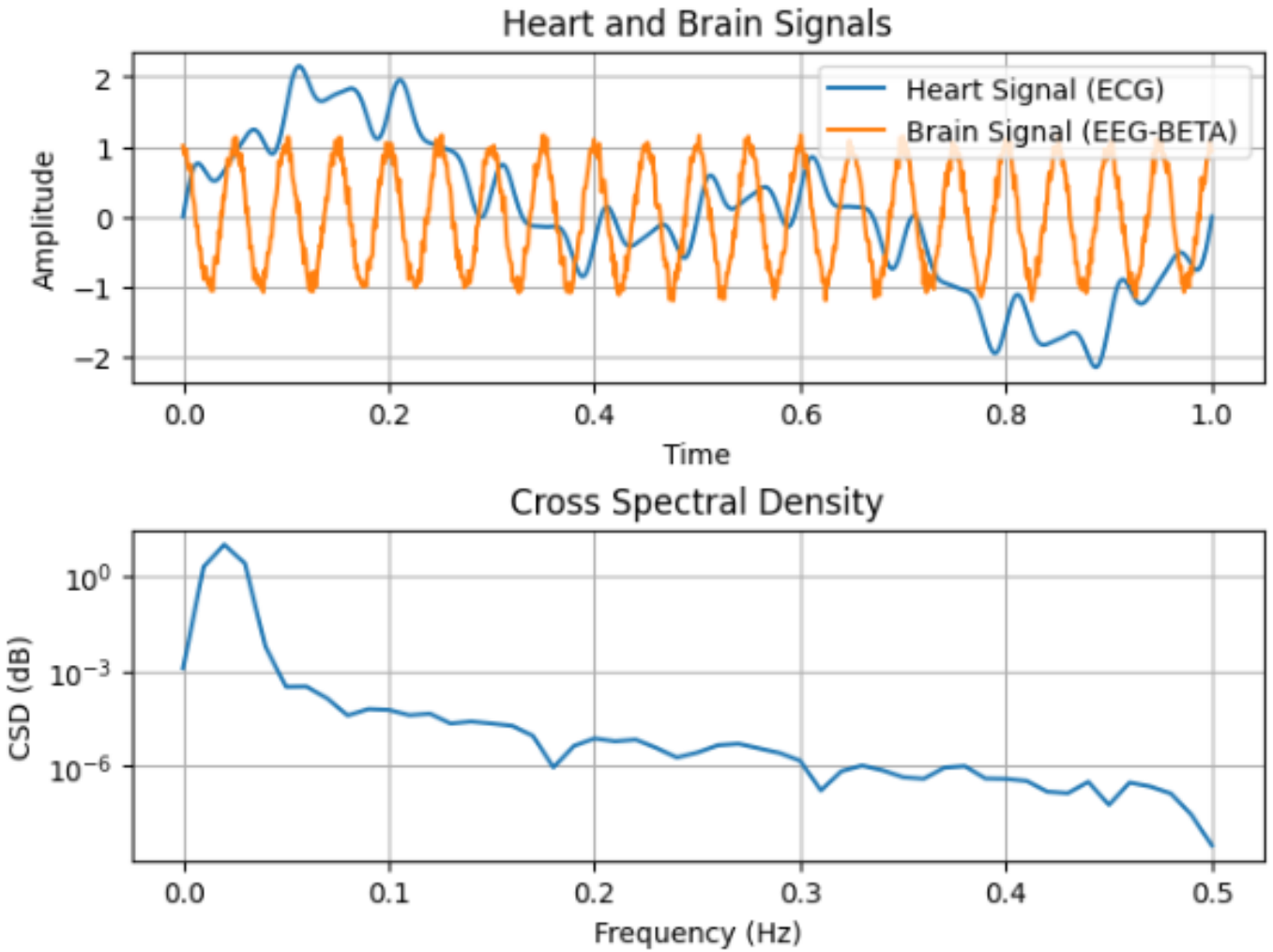
- สร้างกราฟของ Cross Spectral Density (CSD) โดยใช้ฟังก์ชัน semilogy เพื่อแสดงการเปลี่ยนแปลงของค่า CSD ในลักษณะ logarithmic scale บนแกน y ซึ่ง frequencies คือค่าความถี่และ csd คือค่า Cross Spectral Density ที่คำนวณมาก่อนหน้านี้

กำหนดส่วนประกอบต่างๆ ของกราฟ CSD

- `ax2.set_xlabel('Frequency (Hz)')`
- `ax2.set_ylabel('CSD (dB)')`
- `ax2.set_title('Cross Spectral Density')`
- `ax2.grid(True)`
- `plt.show()`

- กำหนดชื่อแกน x เป็น 'Frequency (Hz)’
- กำหนดชื่อแกน y เป็น 'CSD (dB)’
- กำหนดชื่อกราฟเป็น 'Cross Spectral Density’
- เปิดการแสดงเส้นกริดบนกราฟเพื่อช่วยในการอ่านค่าในกราฟ
- plt.show() แสดงผลกราฟบนหน้าจอ

ผลลัพธ์


Heart and Brain Signals
Heart Signal (ECG)
Brain Signal (EEG-BETA)
Amplitude
Time
Cross Spectral Density
CSD (dB)
Frequency (Hz)