ISA 444: Business Forecasting

07: Visualizing Many Time-Series

Fadel M. Megahed, PhD

Professor
Farmer School of Business
Miami University

@FadelMegahed
fmegahed
fmegahed@miamioh.edu
? Automated Scheduler for Office Hours

Spring 2025

Quick Refresher of Last Class

- ✅ Describe and compute centered moving averages
- ✅ Estimate trend-cycle via moving averages
- ✅ Perform classical decomposition (trend-cycle, seasonal, residual/remainder)
- ✅ Understand STL / MSTL as alternatives to classical decomposition

Learning Objectives for Today's Class

- Explain the differences between wide vs. long format
- Use seaborn to plot multiple time-series
- Convert a data set to Nixtla's long format (`unique_id`, `ds`, `y`)
- Use UtilsForecast to visualize multiple series

Wide Vs. Long Format

Wide Format

Date	Temperature (°F)	Muggy Days	Rainy/Snowy Days
2024-01-01	31	0.0	7
2024-02-01	34	0.0	6
2024-03-01	44	0.0	9
2024-04-01	54	0.1	10
2024-05-01	64	3.4	12
2024-06-01	72	12.3	11
2024-07-01	76	18.2	11
2024-08-01	74	15.5	9
2024-09-01	67	7.0	7
2024-10-01	55	0.9	7
2024-11-01	45	0.0	7
2024-12-01	35	0.0	8

Characteristics of Wide TS Data

- Each row represents a single observation
- Each column represents a different time series
- Easy to read and understand (but not appropriate for analysis if you are using the nixtlaverse group of Python packages)

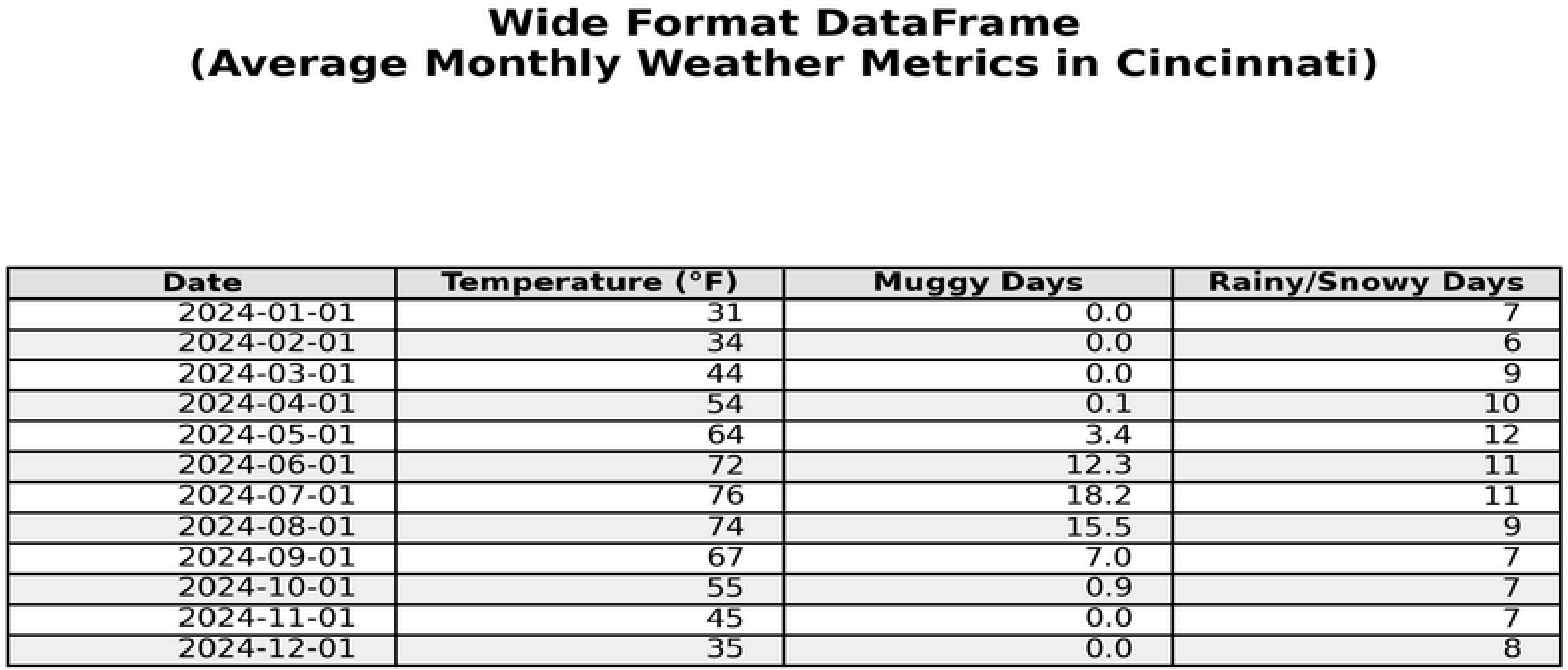
Long Format

Date	Variable	Value
2024-01-01	Muggy Days	0.0
2024-01-01	Rainy/Snowy Days	7.0
2024-01-01	Temperature (°F)	31.0
2024-02-01	Muggy Days	0.0
2024-02-01	Rainy/Snowy Days	6.0
2024-02-01	Temperature (°F)	34.0
2024-03-01	Muggy Days	0.0
2024-03-01	Rainy/Snowy Days	9.0
2024-03-01	Temperature (°F)	44.0
2024-04-01	Muggy Days	0.1
2024-04-01	Rainy/Snowy Days	10.0
2024-04-01	Temperature (°F)	54.0

Characteristics of Long TS Data

- Observations are now split into multiple rows
- Variable ids/labels are stored in a single column, and their corresponding values are stored in another column
- Easy to analyze and visualize (especially with the nixtlaverse group of Python packages)

A Visual Comparison



Wide Format DataFrame
(Average Monthly Weather Metrics in Cincinnati)

Date	Temperature (°F)	Muggy Days	Rainy/Snowy Days
2024-01-01	31	0.0	7
2024-02-01	34	0.0	6
2024-03-01	44	0.0	9
2024-04-01	54	0.1	10
2024-05-01	64	3.4	12
2024-06-01	72	12.3	11
2024-07-01	76	18.2	11
2024-08-01	74	15.5	9
2024-09-01	67	7.0	7
2024-10-01	55	0.9	7
2024-11-01	45	0.0	7
2024-12-01	35	0.0	8

Python pandas function:
long_data = wide_data.reset_index().melt(id_vars=['Date'], var_name='Metric', value_name='Value')

Class Activity: From Wide to Long Format

05:00

Description Starter Code Notes

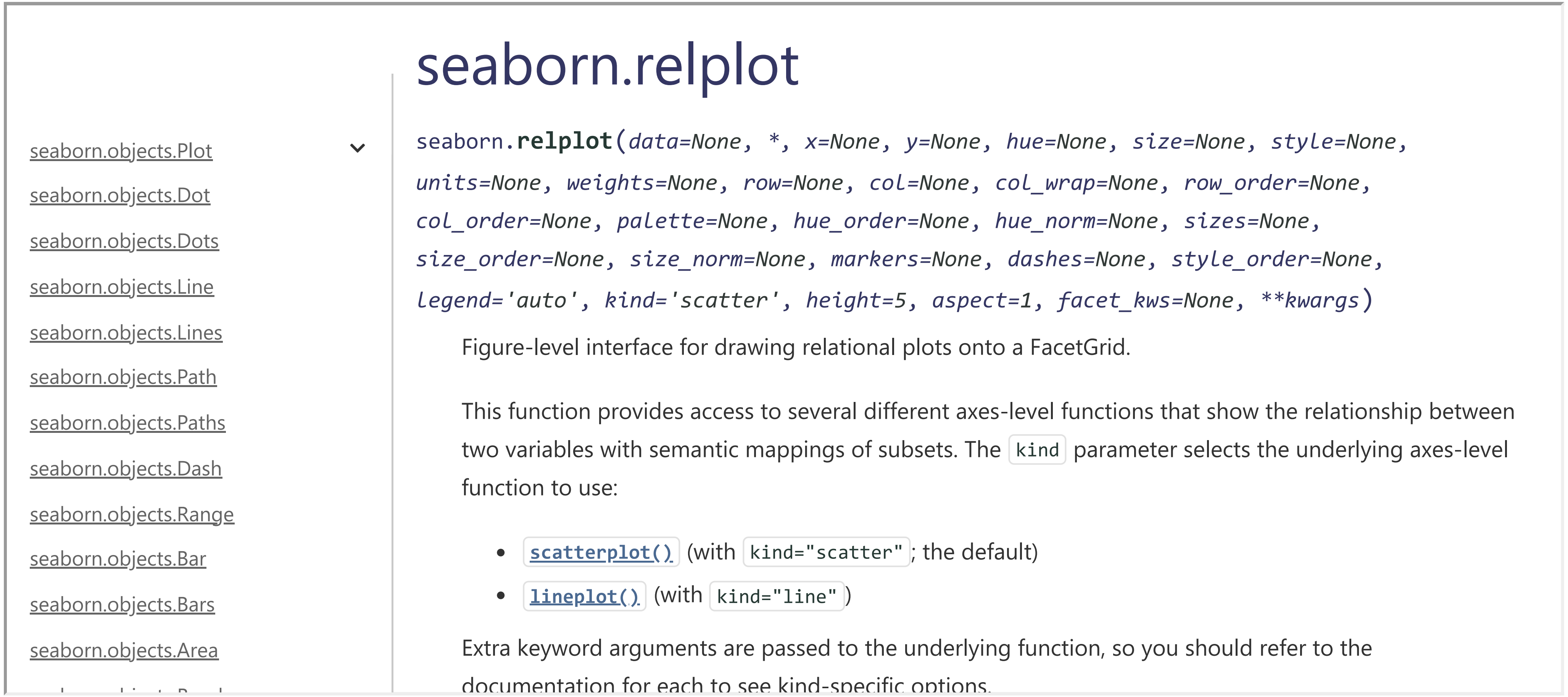
In this activity, you will extract the adjusted closing prices of five stocks (AAPL, MSFT, GOOGL, AMZN, TSLA) from Yahoo Finance. Once you extract the data, you should **report** the following:

- **Summary statistics** of the **adjusted closing prices for each stock**.
- **Convert the data** from wide **to long format**, and **report the shape** of the long format data.
- **Save** the long format data to a CSV file.

Hint: Read and convert the column names prior to converting the data to long format.

Visualizing Multiple Time-Series Using Seaborn

Recall: Seaborn's `relplot` Function



- seaborn.objects.Plot
- seaborn.objects.Dot
- seaborn.objects.Dots
- seaborn.objects.Line
- seaborn.objects.Lines
- seaborn.objects.Path
- seaborn.objects.Paths
- seaborn.objects.Dash
- seaborn.objects.Range
- seaborn.objects.Bar
- seaborn.objects.Bars
- seaborn.objects.Area

seaborn.relplot

```
seaborn.relplot(data=None, *, x=None, y=None, hue=None, size=None, style=None,
units=None, weights=None, row=None, col=None, col_wrap=None, row_order=None,
col_order=None, palette=None, hue_order=None, hue_norm=None, sizes=None,
size_order=None, size_norm=None, markers=None, dashes=None, style_order=None,
legend='auto', kind='scatter', height=5, aspect=1, facet_kws=None, **kwargs)
```

Figure-level interface for drawing relational plots onto a FacetGrid.

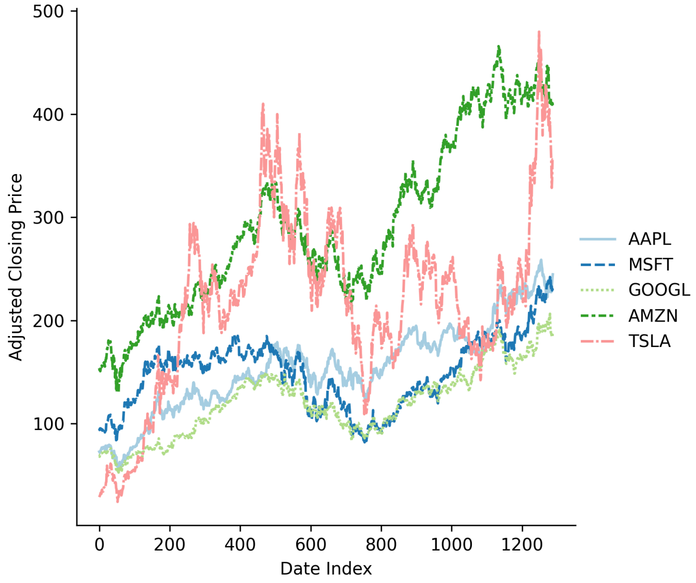
This function provides access to several different axes-level functions that show the relationship between two variables with semantic mappings of subsets. The `kind` parameter selects the underlying axes-level function to use:

- `scatterplot()` (with `kind="scatter"`; the default)
- `lineplot()` (with `kind="line"`)

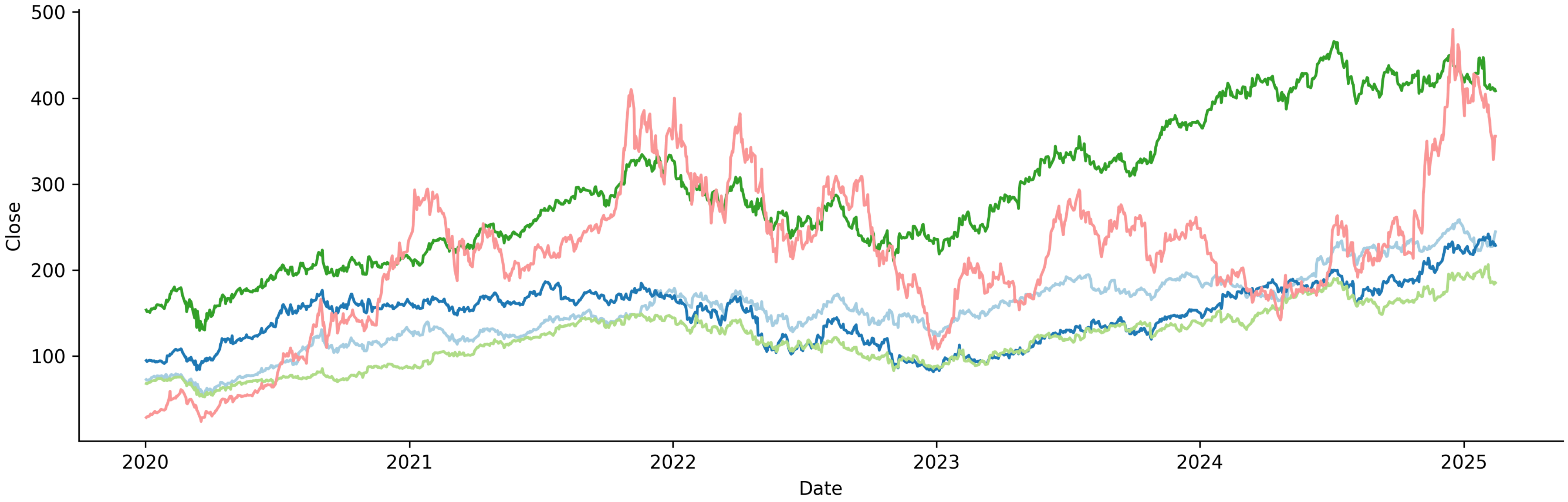
Extra keyword arguments are passed to the underlying function, so you should refer to the documentation for each to see kind-specific options.

Seaborn's `relplot` Function with Wide Data

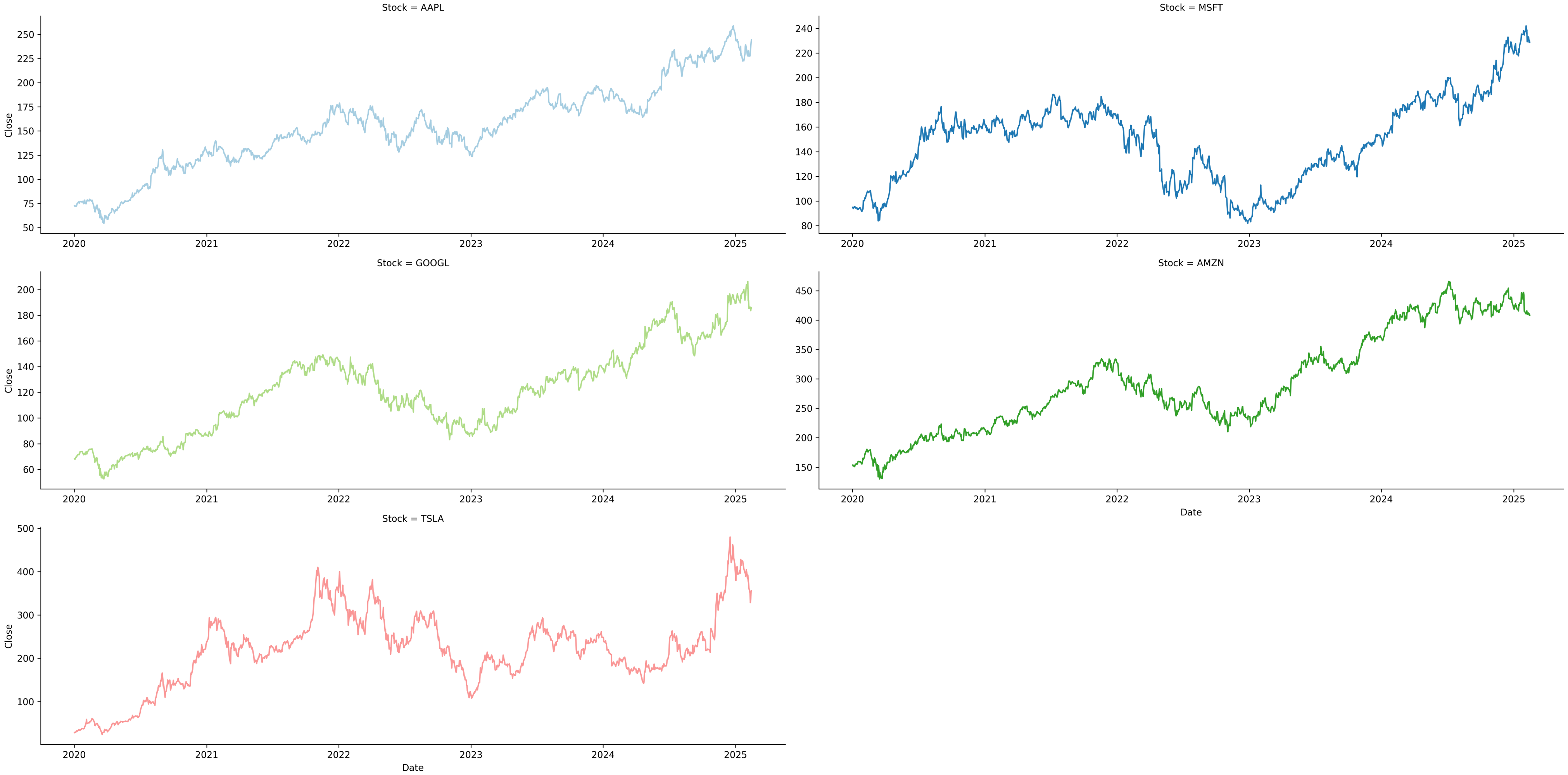
```python
import datetime as dt
import yfinance as yf
import pandas as pd
import seaborn as sns

# Download the stock data for the following companies
stock_data = (
  yf.download(
    ['AAPL', 'MSFT', 'GOOGL', 'AMZN', 'TSLA'],
    start='2020-01-01',
    end= (dt.datetime.now().date() - dt.timedelta(days=1))
  )
  # Keep only the adjusted closing price
  [['Adj Close']].reset_index()
)

# Overwrite the multi-index column names w/ single level
stock_data.columns = (
  ['Date', 'AAPL', 'MSFT', 'GOOGL', 'AMZN', 'TSLA'] )

# Plot the closing prices
sns.relplot(
  data=stock_data, kind='line', palette ='Paired'
)
```



Seaborn's `relplot` Function with Long Data



Seaborn's `relplot` Function with Long Data (Facets)



Activity: Reflect on the Previous Seaborn Plots

03:00

- Whether you are plotting multiple lines in a single plot or using facets, the `relplot` function is quite versatile. However, this approach is only suitable for a few (in my opinion ≤ 9 time series). **So what options, do we have if we have more than 9 time series?**
- I think there are **three alternative charting approaches** (I am not talking about specific libraries). **Can you guess what they are?**
- In the next three minutes, edit the bullet points below to reflect the three alternative charting approaches.
  - ...
  - ...
  - ...

Advanced Visualizations with Seaborn

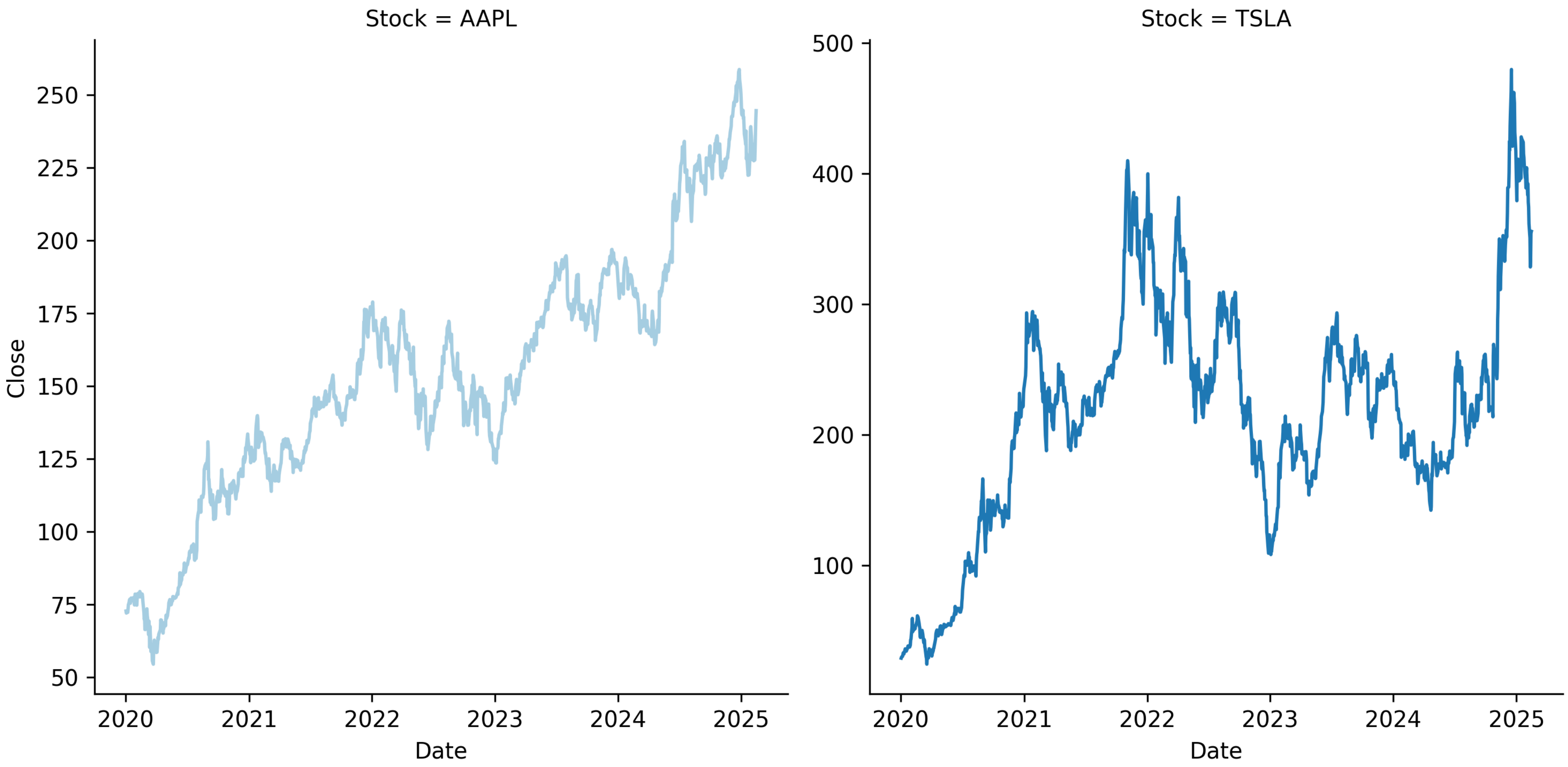
Approach 1: Sample

- **Sample** a subset of the time series to plot. This approach is useful when you have a large number of time series and you want to visualize a **representative sample**.
- **Key Point:** The sample should be **representative** of the entire data set.

Approach 1: Sample (Code Example)

```python
import pandas as pd
import seaborn as sns
import random

random.seed(2025) # for reproducibility

# Start with the long format data and sample two stocks
sampled_stocks = stock_data['Stock'].unique().tolist()
sampled_stocks = random.sample(sampled_stocks, 2)

# Plot the adjusted closing prices
fig = sns.relplot(
  data=stock_data.query('Stock in @sampled_stocks'),
  x='Date', y='Close', kind='line', legend=False, hue='Stock', col='Stock',
  palette='Paired', col_wrap=2,  facet_kws={'sharey': False, 'sharex': False}
)
plt.tight_layout() # improves title visibility
```

Appraoch 1: Sample (Result)



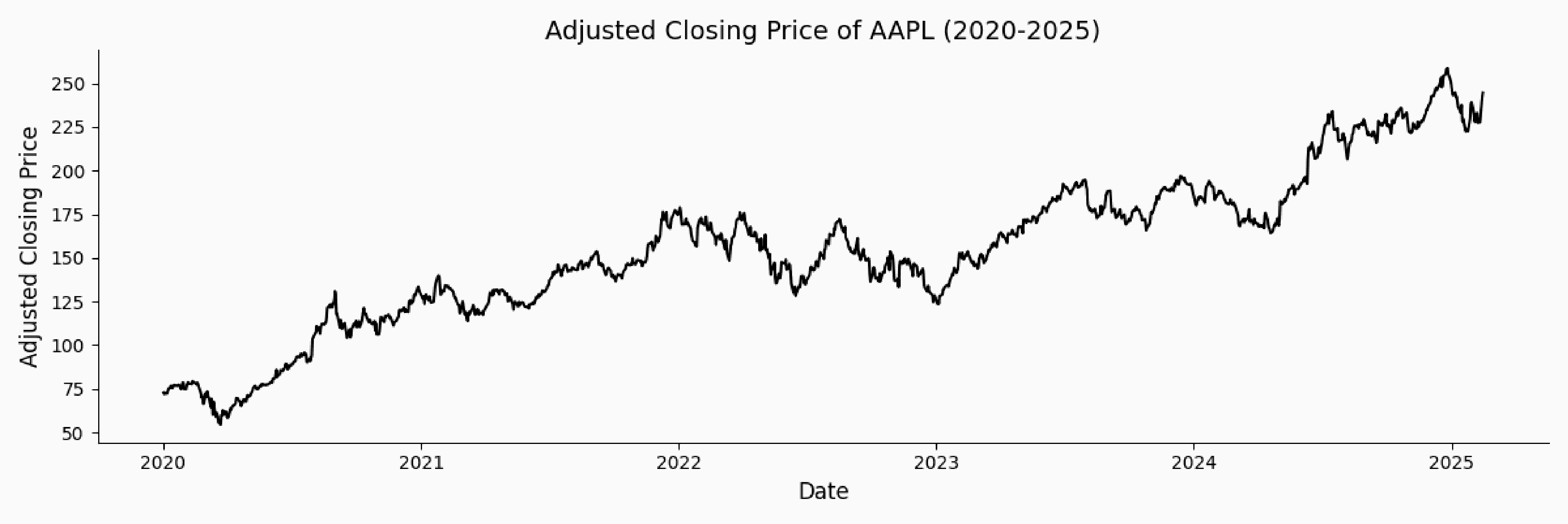
Approach 2: Animated Plots

- **Animate** the time series data. This approach is useful when you have a large number of time series and you want to visualize all of them.
- **Key Point:** Animated plots can be **interactive** and **engaging**.

Approach 2: Animated Plots (Code Example)

```python
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import imageio.v2 as imageio

# Start with long format data
stocks = stock_data['Stock'].unique().tolist()

image_paths = []

for stock in stocks:
    sns.relplot(
      data=stock_data.query('Stock == @stock'), kind ='line',
      x='Date', y='Close', color ='black',
      height = 4, aspect = 3
    )
    plt.title(f"Adjusted Closing Price of {stock} (2020-2025)", fontsize=14)
    plt.xlabel("Date", fontsize=12)
    plt.ylabel("Adjusted Closing Price", fontsize=12)
    plt.tight_layout()
    plt.savefig(f'../../figures/{stock}_animated_plot.png')
    image_paths.append(f'../../figures/{stock}_animated_plot.png')

# Create a GIF from the images
images = [imageio.imread(path) for path in image_paths]
imageio.mimsave('../../figures/animated_stock_lineplot.gif', images, fps=0.25)
```

Approach 2: Animated Plots (Result)



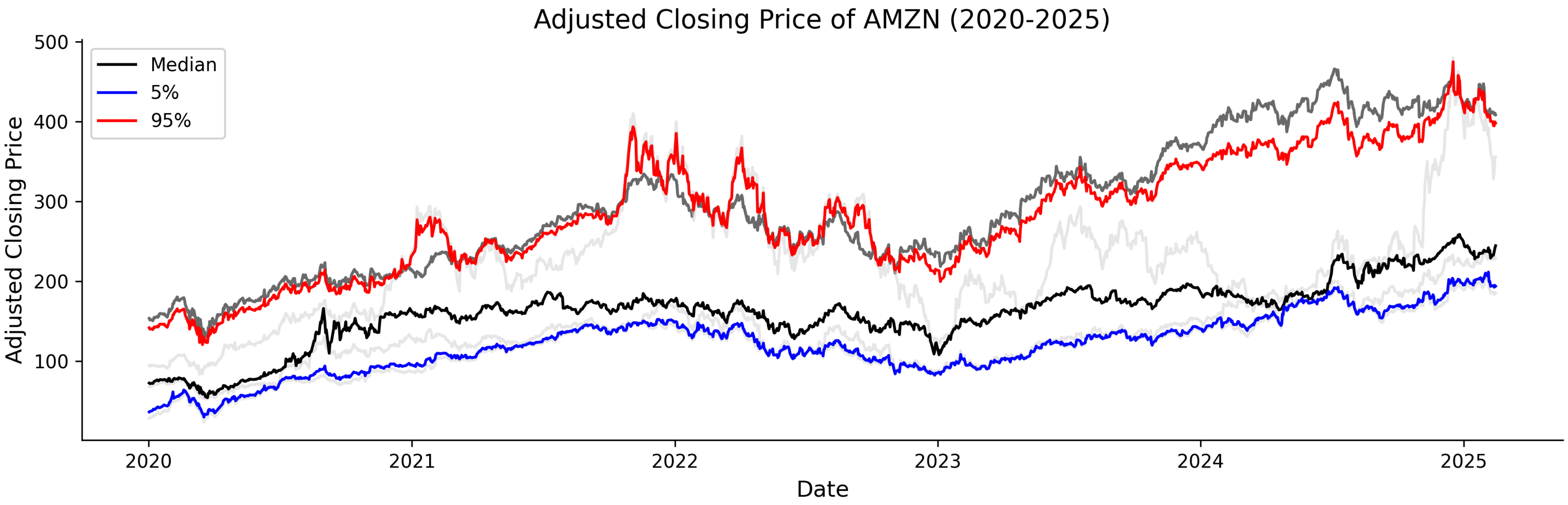
Approach 3: Spaghetti Plot

- **Plot all time series** on a single plot. This approach is useful when you have a large number of time series and you want to visualize all of them.
- **Key Points:**
  - Use **light gray** for the lines to **de-emphasize** individual time series.
  - Use **bold colors** for the lines of **specific time series (or summary statistics across all time series)** to **emphasize** them.

Approach 3: Spaghetti Plot (Code Example)

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# for each stock, plot the closing price over time as a light gray line
for stock, group in stock_data.groupby("Stock"):
  ax = sns.lineplot(data=group, x='Date', y='Close', color='lightgray', alpha=0.5)

# Calculate and overlay percentiles across all time series for each date
quantiles = stock_data.groupby('Date')['Close'].quantile([0.05, 0.5, 0.95]).unstack()

# Plot the median, 5th, and 95th percentiles
sns.lineplot(data=quantiles, x=quantiles.index, y=0.5, color='black', label='Median', ax = ax)
sns.lineplot(data=quantiles, x=quantiles.index, y=0.05, color='blue', label='5%', ax=ax)
sns.lineplot(data=quantiles, x=quantiles.index, y=0.95, color='red', label='95%', ax=ax)
```

Approach 3: Spaghetti Plot (Result)



Nixtla's Long Format

Class Activity: Convert Data to Nixtla's Format

08:00

- **Data:** This COVID-19 data set contains the daily cumulative number of confirmed cases for each county.
- **Objectives:**
  - Read the data set.
  - Filter the data to include only the 88 counties in Ohio, and dates from 2020-04-01 to 2022-12-31.
  - Convert the data to Nixtla's long format (`unique_id`, `ds`, `y`), where the:
    - `unique_id` column is used to identify each county.
    - `ds` column is used to represent the date.
    - `y` column is used to represent the cumulative number of confirmed cases.
  - Use the `plot_series` method from the UtilsForecast to visualize the data. See here to learn about how to import and here to see the arguments of the `plot_series` method.

Recap

Summary of Main Points

By now, you should be able to do the following:

- Explain the differences between wide vs. long format
- Use seaborn to plot multiple time-series
- Convert a data set to Nixtla's long format (`unique_id`, `ds`, `y`)
- Use UtilsForecast to visualize multiple series

📝 Review and Clarification 📝

- **Class Notes**: Take some time to revisit your class notes for key insights and concepts.
- **Zoom Recording**: The recording of today's class will be made available on Canvas approximately 3-4 hours after the session ends.
- **Questions**: Please don't hesitate to ask for clarification on any topics discussed in class. It's crucial not to let questions accumulate.