ISA 444: Business Forecasting

06: Centered Moving Average and Seasonal Decomposition

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

- ✅ Generate and interpret simple line charts
- ✅ Create seasonal plots and subplots
- ✅ Describe a lag and create lag scatter plots
- ✅ Plot and interpret the autocorrelation function (ACF) for a time-series

Learning Objectives for Today's 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

Motivation and Context

Recall: Time Series Components

- **Trend**: Long-term increase or decrease in the data
- **Seasonal**: Regular pattern of up and down fluctuations influenced by calendar or time of day (e.g., quarterly, monthly, day of the week, hourly)
- **Cyclical**: Repeating up and down movements that are **not of a fixed period** (e.g., business cycles, economic cycles), typically over multiple years

Time Series Decomposition

- **Decomposition** is the process of breaking down a time series into its constituent components: trend-cycle, seasonal, and residual/remainder.
- **Mathematically**, we can write a time series y_t as a function of these components:

$$y_t = f(T_t,\ S_t,\ R_t),$$

where:

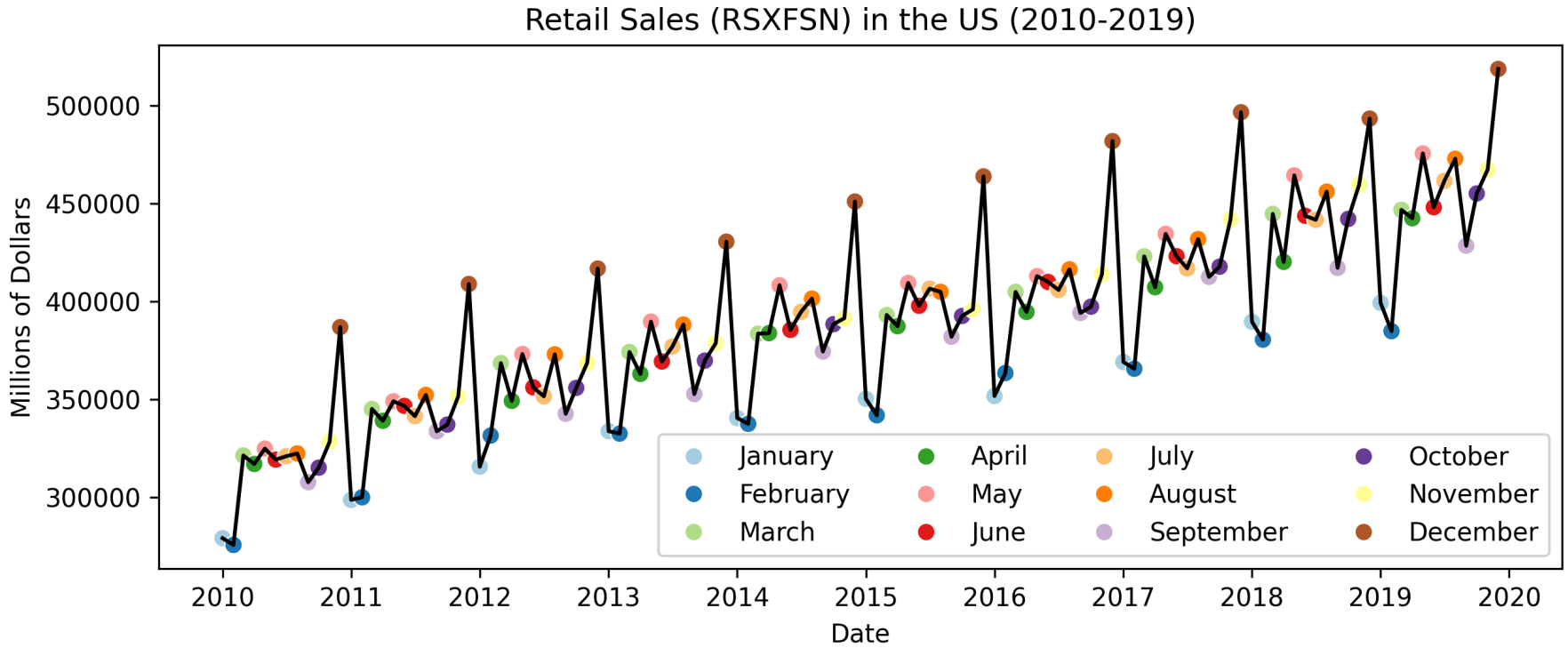
  - T_t: Trend-cycle component
  - S_t: Seasonal component
  - R_t: Remainder component
  - $f()$: Some function that combines the components

- The decomposition can be **additive** or **multiplicative**.

Seasonality: Additive Models

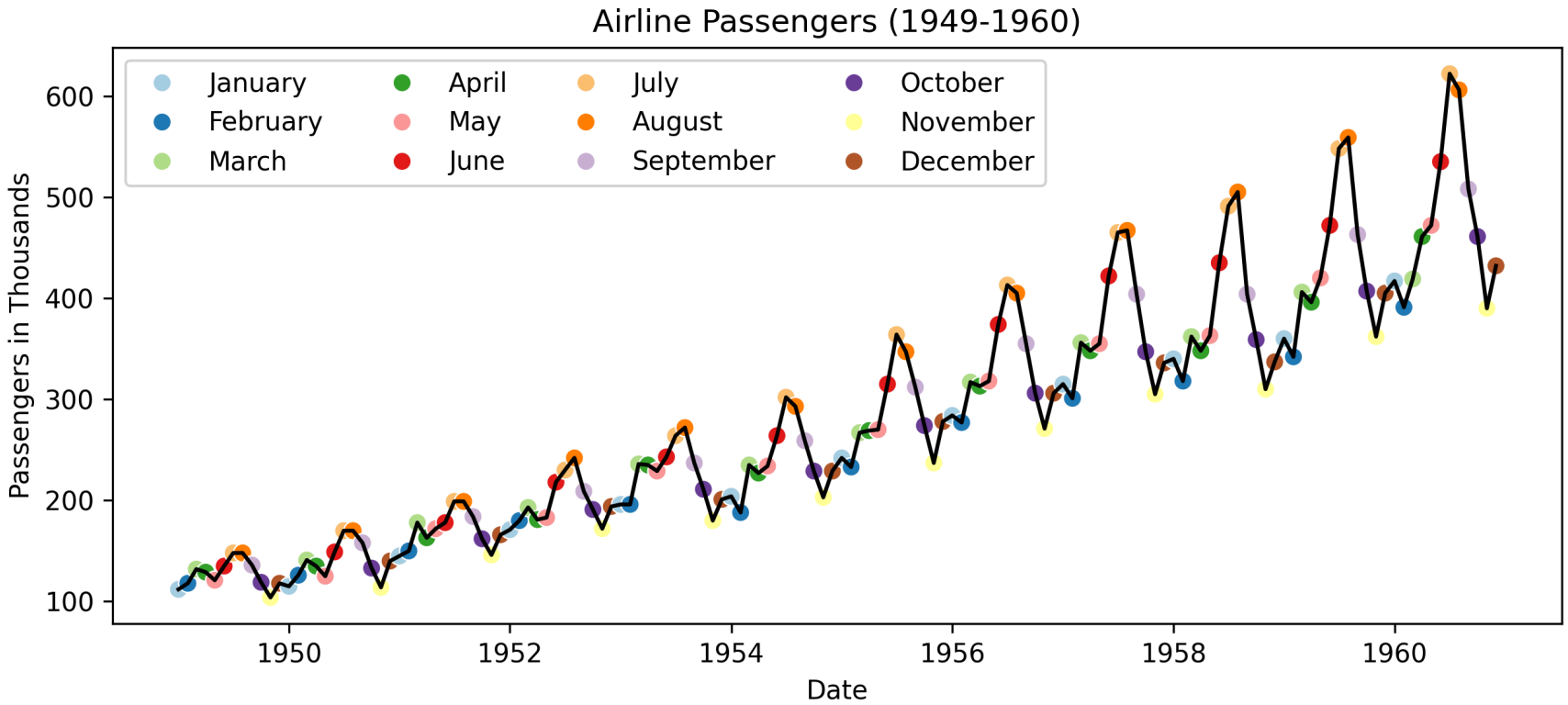
Definition: An additive model is appropriate when the **trend is approximately linear**, and the **seasonal components stays constant over time.**

It can be written as: $y_t = T_t + S_t + R_t$.



Seasonality: Multiplicative Models

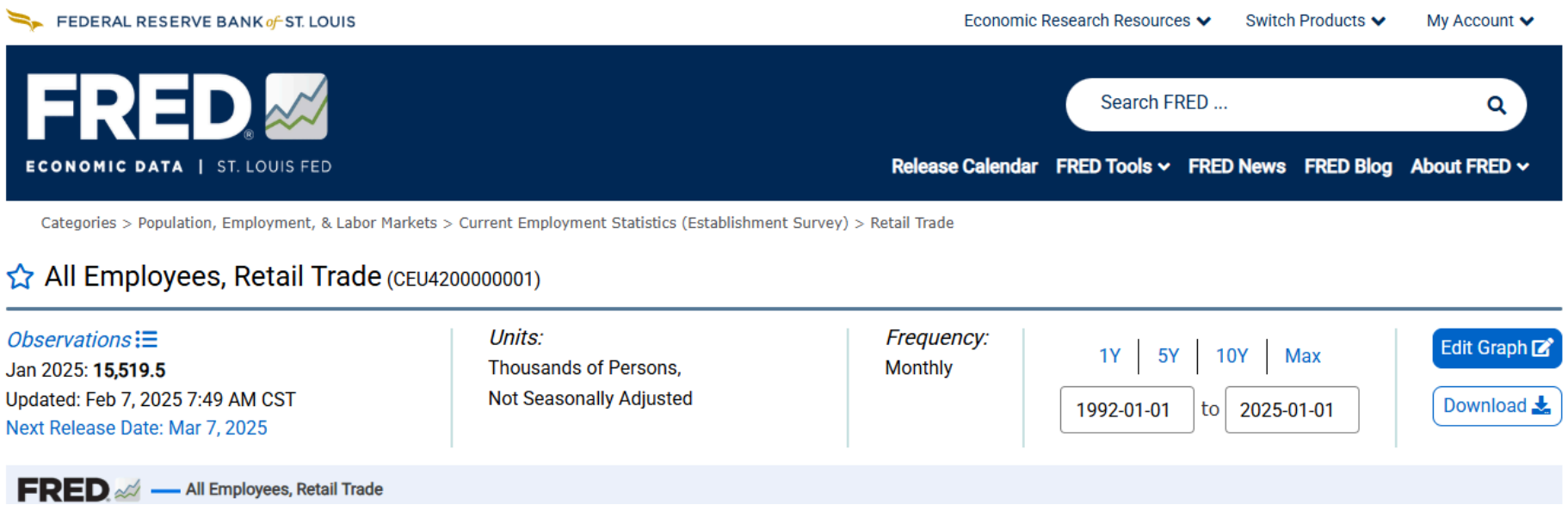
Definition: A multiplicative model is appropriate when the **trend is nonlinear (e.g., exponential)**, and/or the **seasonal component changes proportionally with the level of the time series.**
It can be written as: $y_t = T_t \times S_t \times R_t$.



Motivating Example: US Retail Trade Employment

Data | Plot | Decomp | Decomp | Trend-Cyc | DeTrend | SeasAdj | Notes

- The data contains the **total employment in U.S. retail trade** sector from 1992 to 2025.
- The data is **monthly** and **not seasonally adjusted**.



Centered Moving Averages

Relation of Moving Averages to Classical Decomposition

- The **classical method of time series decomposition** originated in the 1920s and was widely used until the 1950s.
- It still **forms the basis of many time series decomposition methods**, so it is important to understand how it works.
- The **first step in a classical decomposition is to use a moving average method to estimate the trend-cycle**, so we begin by discussing moving averages.

Source: Hyndman, R.J., & Athanasopoulos, G. (2021) Forecasting: principles and practice, 3rd edition, OTexts: Melbourne, Australia. https://otexts.com/fpp3/. Accessed on February 12, 2025. Chapter 3.3

Moving Average (MA) Smoothing

- **Moving averages** are used to **smooth out short-term fluctuations** and highlight longer-term trends or cycles.
- In classical decomposition, the **trend-cycle component is estimated using a centered moving average**.
- A **centered moving average** is calculated by taking the average of values on either side of the data point in question.
- Mathematically,a centered moving average of order m can be written as:

$$\hat{T}_t = \frac{1}{m} \sum_{j=-k}^{k} y_{t+j},$$

where $m = 2k + 1$ is the number of values in the moving average.

Activity: Centered Moving Average

03:00

Let's consider the following series: $y_t = \{4, 7, 7, 10, 13, 13, 16\}$

Calculate a **centered moving average of order 3**, and **input it in the CMA3 column below.**

- Note each **cell** in the table is **editable**, but please **only edit the CMA3 column**.

t	y_t	cma3
1	4	
2	7	
3	7	
4	10	
5	13	
6	13	
7	16	

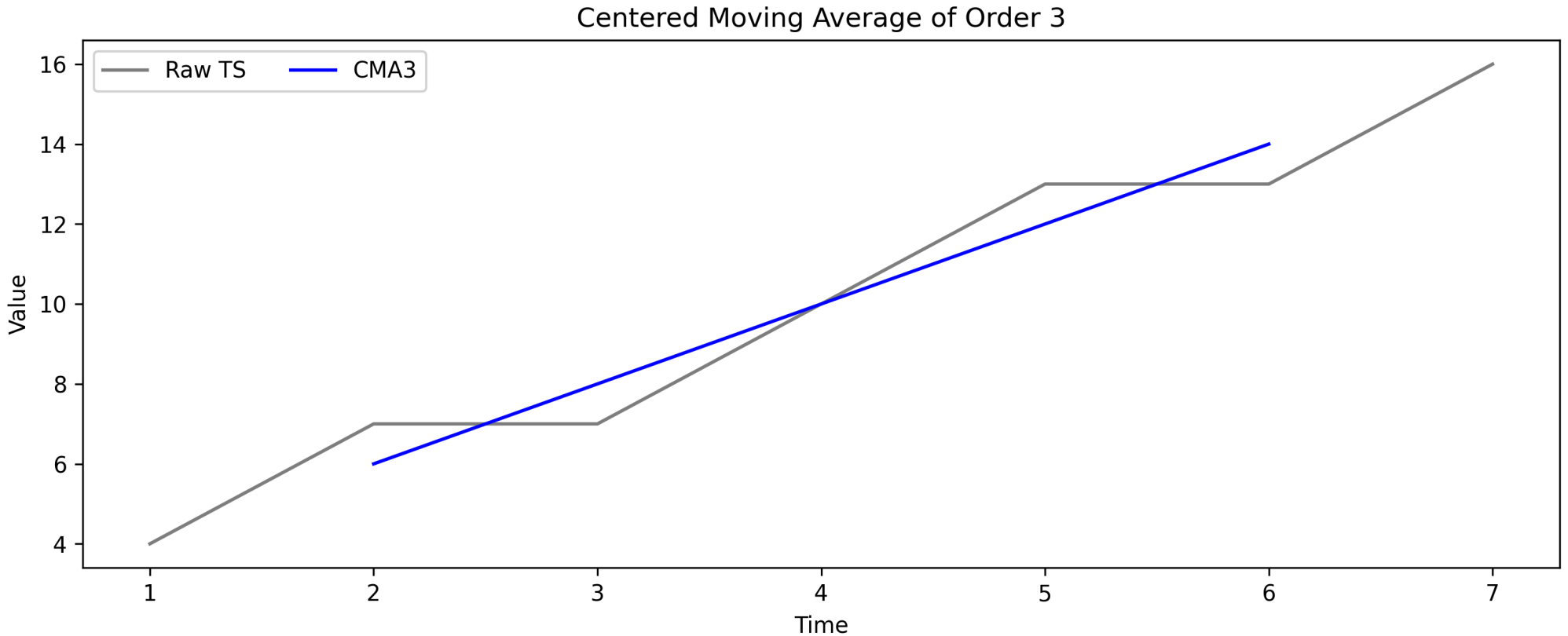
Centered Moving Averages in Python

```
y_t = [4, 7, 7, 10, 13, 13, 16]

df = (
  pd.DataFrame({"t": range(1, len(y_t) + 1), "y_t": y_t})
  .assign(
    cma3 = lambda x: x['y_t'].rolling(window = 3, center = True).mean(),
    cma4r = lambda x: x['y_t'].rolling(window = 4, center = True).mean(),
    cma4l = lambda x: x['cma4r'].shift(-1)
    )
  )
df.head(7)
```

```
##    t  y_t  cma3  cma4r  cma4l
## 0  1    4   NaN    NaN    NaN
## 1  2    7   6.0    NaN   7.00
## 2  3    7   8.0   7.00   9.25
## 3  4   10  10.0   9.25  10.75
## 4  5   13  12.0  10.75  13.00
## 5  6   13  14.0  13.00    NaN
## 6  7   16   NaN    NaN    NaN
```

Which of the two CMA4 cols is correct? Edit me

Centered Moving Averages as Trend-Cycle Estimators



Moving Average of Moving Averages

When m is even, we often resort to using a **moving average of moving averages** to center the moving average. Let us revisit our previous example:

```
df = (
  pd.DataFrame({"t": range(1, len(y_t) + 1), "y_t": y_t})
  .assign(
    cma3 = lambda x: x['y_t'].rolling(window = 3, center = True).mean(),
    cma4r = lambda x: x['y_t'].rolling(window = 4, center = True).mean(),
    cma4l = lambda x: x['cma4r'].shift(-1),
    cma4_2 = lambda x: x['cma4l'].rolling(window = 2, center = True).mean(),
    cma4_2_alt = lambda x: (x['cma4r'] + x['cma4l'])/2
    )
  )
df.head(7)
```

```
##    t  y_t  cma3  cma4r  cma4l  cma4_2  cma4_2_alt
## 0  1    4   NaN    NaN    NaN     NaN         NaN
## 1  2    7   6.0    NaN   7.00     NaN         NaN
## 2  3    7   8.0   7.00   9.25   8.125       8.125
## 3  4   10  10.0   9.25  10.75  10.000      10.000
## 4  5   13  12.0  10.75  13.00  11.875      11.875
## 5  6   13  14.0  13.00    NaN     NaN         NaN
## 6  7   16   NaN    NaN    NaN     NaN         NaN
```

Moving Average of Moving Averages (Cont.)

When a 2-MA follows a moving average of an even order (such as 4), it is called a “centered moving average of order 4”.

This is because the results are now symmetric.

To see that this is the case, we can write the 2×4-MA (named as `cma4_2` or `cma4_2_alt` in our Python code) as follows:

$$\begin{aligned}\hat{T}_t &= \frac{1}{2}\left[\frac{1}{4}(y_{t-2} + y_{t-1} + y_t + y_{t+1}) + \frac{1}{4}(y_{t-1} + y_t + y_{t+1} + y_{t+2})\right] \\ &= \frac{1}{8}y_{t-2} + \frac{1}{4}y_{t-1} + \frac{1}{4}y_t + \frac{1}{4}y_{t+1} + \frac{1}{8}y_{t+2}.\end{aligned}$$

It is now a **weighted average of observations that is symmetric**.

Centered Moving Averages for Trend-Cycle Estimation

📊 **Key Idea:**
Centered moving averages (CMAs) help estimate the **trend-cycle** by smoothing seasonal fluctuations.

Example: 2 × 4 - MA

$$\hat{T}_t = \frac{1}{8}y_{t-2} + \frac{1}{4}y_{t-1} + \frac{1}{4}y_t + \frac{1}{4}y_{t+1} + \frac{1}{8}y_{t+2}$$

✅ **Balances** data across quarters
✅ **Averages out seasonal /quarterly variation**

Choosing the Right Moving Average

Seasonal Period (m)	Recommended MA
Even (e.g., 12 months, 4 quarters)	**2 × m - MA**
Odd (e.g., 7 days)	**m-MA**

✓ **2 × 12 - MA** for monthly seasonality in annual data
✓ **2 × 4 - MA** for quarterly seasonality in annual data
✓ **7-MA** for weekly seasonality in daily data

⚠ **Beware!**
Using the wrong MA order **retains seasonal noise** instead of extracting the trend.

Activity: Trend-Cycle Estimation Using CMAs

05:00

Data | Plot | Printout

- Download the CEU4200000001, capturing the **total employment in U.S. retail trade** sector from 1992 to 2025.
- The data is **monthly** and **not seasonally adjusted**.
- Use an **appropriate moving average** to estimate the trend-cycle component.
- Recreate the plot of raw series and the trend-cycle component from the next tab.

Classical Decomposition

Classical Decomposition

Additive decomposition: $y_t = T_t + S_t + R_t = \hat{T}_t + \hat{S}_t + \hat{R}_t$

Multiplicative decomposition: $y_t = T_t \times S_t \times R_t = \hat{T}_t \times \hat{S}_t \times \hat{R}_t$

- **Estimate** $\hat{T}$ using $(2 \times m)$-MA if m is even. Otherwise, estimate $\hat{T}$ using m-MA
- **Compute de-trended series** using: (a) $y_t - \hat{T}_t$ for additive decomposition, and (b) $y_t/\hat{T}_t$ for multiplicative decomposition.
- Once de-trended, **we are left with** the S_t and R_t components. Specifically:
  - **Additive model:** $y_t - \hat{T}_t = (\hat{T}_t + \hat{S}_t + \hat{R}_t) - \hat{T}_t = \hat{S}_t + \hat{R}_t$
  - **Multiplicative model:** $\frac{y_t}{\hat{T}_t} = \frac{\hat{T}_t \times \hat{S}_t \times \hat{R}_t}{\hat{T}_t} = \hat{S}_t \times \hat{R}_t$

Source: Hyndman, R.J., & Athanasopoulos, G. (2021) Forecasting: principles and practice, 3rd edition, OTexts: Melbourne, Australia. https://otexts.com/fpp3/. Accessed on February 12, 2025. Chapter 3.4 Slides

Estimating Seasonal Component

- The **seasonal component** can be estimated by **averaging** the **detrended series** for that season across all years.
- For example, if we have monthly data, we can estimate the seasonal index for January by averaging all the January values in the detrended series. Do this for each month to get the seasonal index.
- **Adjust** the seasonal indices to ensure that that:
  - for additive: $S^{(1)} + S^{(2)} + \ldots + S^{(12)} = 0$
  - for multiplicative: $S^{(1)} + S^{(2)} + \ldots + S^{(12)} = m$
- The **seasonal component** is then obtained by **repeating** the **seasonal indices** for each year in the data.

Estimating the Remainder Component

Additive decomposition: $\hat{R}_t = y_t - \hat{T}_t - \hat{S}_t$

Multiplicative decomposition: $\hat{R}_t = y_t / (\hat{T}_t \hat{S}_t)$

Classical Decomposition in Python (Ref)

statsmodels.tsa.seasonal.seasonal_decompose

```
statsmodels.tsa.seasonal.seasonal_decompose(
    x,
    model='additive',
    filt=None,
    period=None,
    two_sided=True,
    extrapolate_trend=0
)                                                                    [source]
```

Seasonal decomposition using moving averages.

Parameters

x : array_like

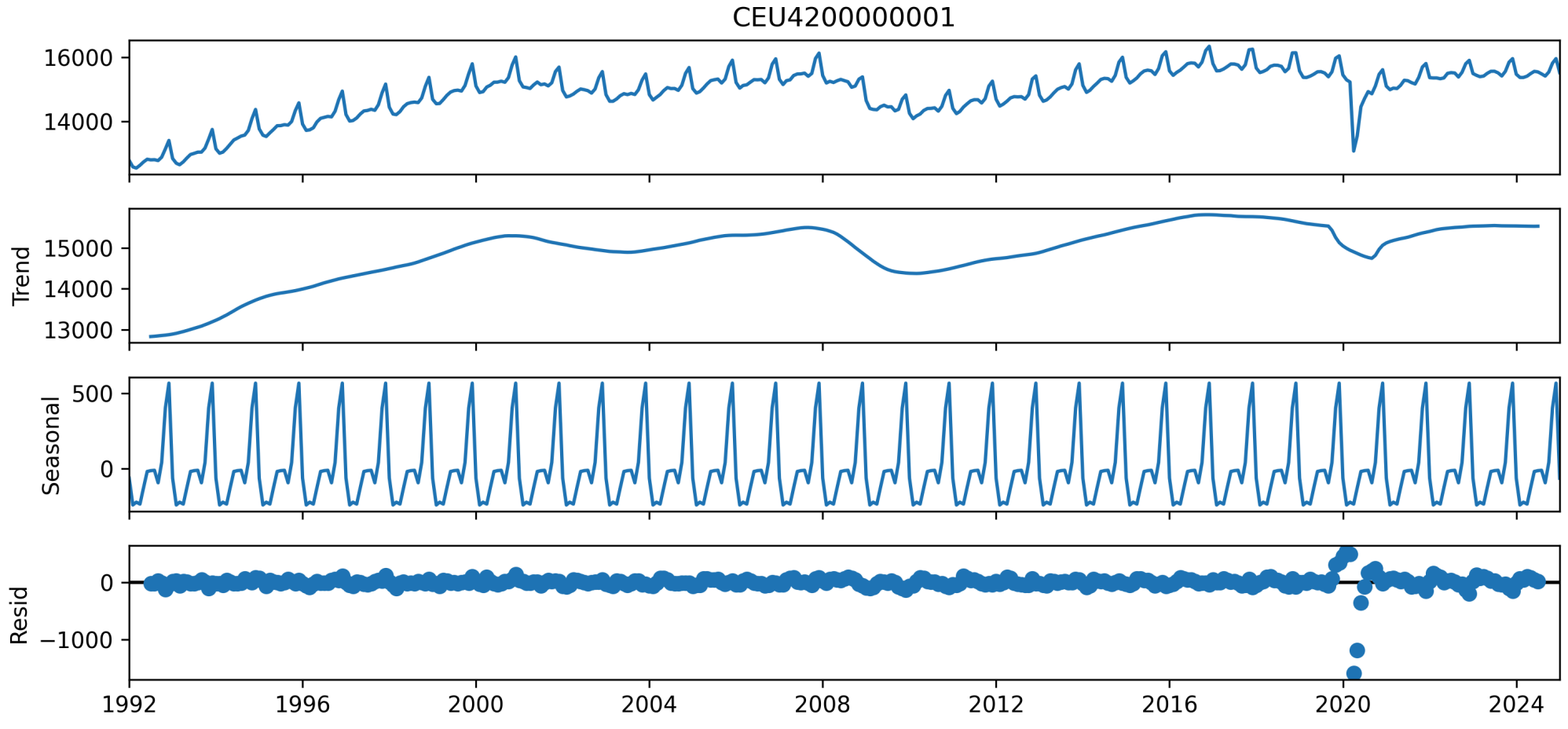
Time series. If 2d, individual series are in columns. x must contain 2 complete cycles.

model : {“additive”, “multiplicative”} optional

Classical Decomposition in Python (Code-Add)

```
from statsmodels.tsa.seasonal import seasonal_decompose

retail_empl = (
  pd.read_csv("https://fred.stlouisfed.org/graph/fredgraph.csv?bgcolor=%23ebf3fb&chart_type=line&drp=0
  .rename(columns = {"observation_date": "date", "CEU4200000001": "CEU4200000001"})
  .assign(
    date = lambda x: pd.to_datetime(x["date"]),
      )
  )

# Set the date as index and set the frequency of the data
retail_empl = retail_empl.set_index("date")
retail_empl = retail_empl.asfreq("MS")

# Decompose the time series
res = seasonal_decompose( retail_empl["CEU4200000001"], model = "additive")

# Extract the trend, seasonal, and residual components
res_df = pd.DataFrame({
  "observed": res.observed, "trend": res.trend, "seasonal": res.seasonal, "residual": res.resid
  })

# Plot the decomposed time series
fig = res.plot()
fig.set_size_inches(10, 4.75)
fig.tight_layout()
```

Classical Decomposition in Python (Output-Add)

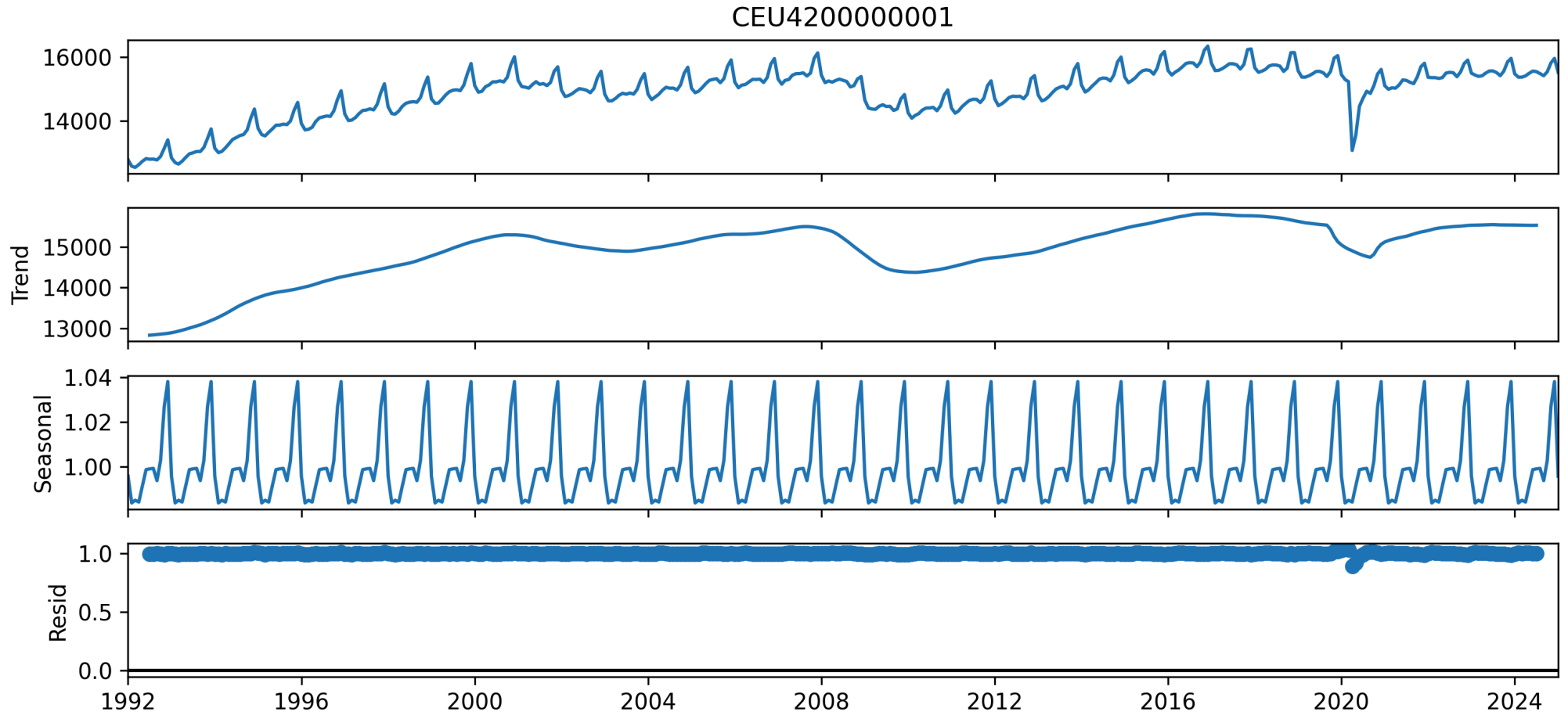


Comment: You can have more control over the plot by plotting the data frame `res_df` directly. See this example from StackOverflow, while noting that their `decomposition` is equivalent to our `res` object. You can also convert the data into a long format using `pd.melt`, then `.reset_index()` to convert date into a column, and plot it using `sns.relplot`.

Classical Decomposition in Python (Code-Mult)

```
from statsmodels.tsa.seasonal import seasonal_decompose

retail_empl = (
  pd.read_csv("https://fred.stlouisfed.org/graph/fredgraph.csv?bgcolor=%23ebf3fb&chart_type=line&drp=0&
  .rename(columns = {"observation_date": "date", "CEU4200000001": "CEU4200000001"})
  .assign(
    date = lambda x: pd.to_datetime(x["date"]),
      )
  )

# Set the date as index and set the frequency of the data
retail_empl = retail_empl.set_index("date")
retail_empl = retail_empl.asfreq("MS")

# Decompose the time series
res = seasonal_decompose( retail_empl["CEU4200000001"], model = "multiplicative")

# Extract the trend, seasonal, and residual components
res_df = pd.DataFrame({
  "observed": res.observed, "trend": res.trend, "seasonal": res.seasonal, "residual": res.resid
  })

# Plot the decomposed time series
fig = res.plot()
fig.set_size_inches(10, 4.75)
fig.tight_layout()
```

Classical Decomposition in Python (Output-Mult)



Comment: You can have more control over the plot by plotting the data frame `res_df` directly. See this example from StackOverflow, while noting that their `decomposition` is equivalent to our `res` object. You can also convert the data into a long format using `pd.melt`, then `.reset_index()` to convert date into a column, and plot it using `sns.relplot`.

Comments on the Classical Decomposition

- Estimate of trend is **unavailable** for first few and last few observations.
- The **seasonal component repeats** from year to year, which may not be realistic (especially for long time-series).
- Since the trend-cycle is estimated using an average-based method, it is **not robust to outliers**.
- **Newer methods** are designed to overcome these problems.

STL Decomposition

Seasonal-Trend Decomposition using LOESS (STL)

- **STL** is a very flexible method for decomposing time series.
- It can handle **any type of seasonality**, and allows for **non-constant** seasonal patterns.
- It is robust to **outliers**.
- However, it:
  - **cannot** handle trading day or calendar adjustments, and
  - it is only **additive** (take logs to get multiplicative decomposition).

STL Decomposition in Python (Ref)

statsmodels.tsa.seasonal.STL

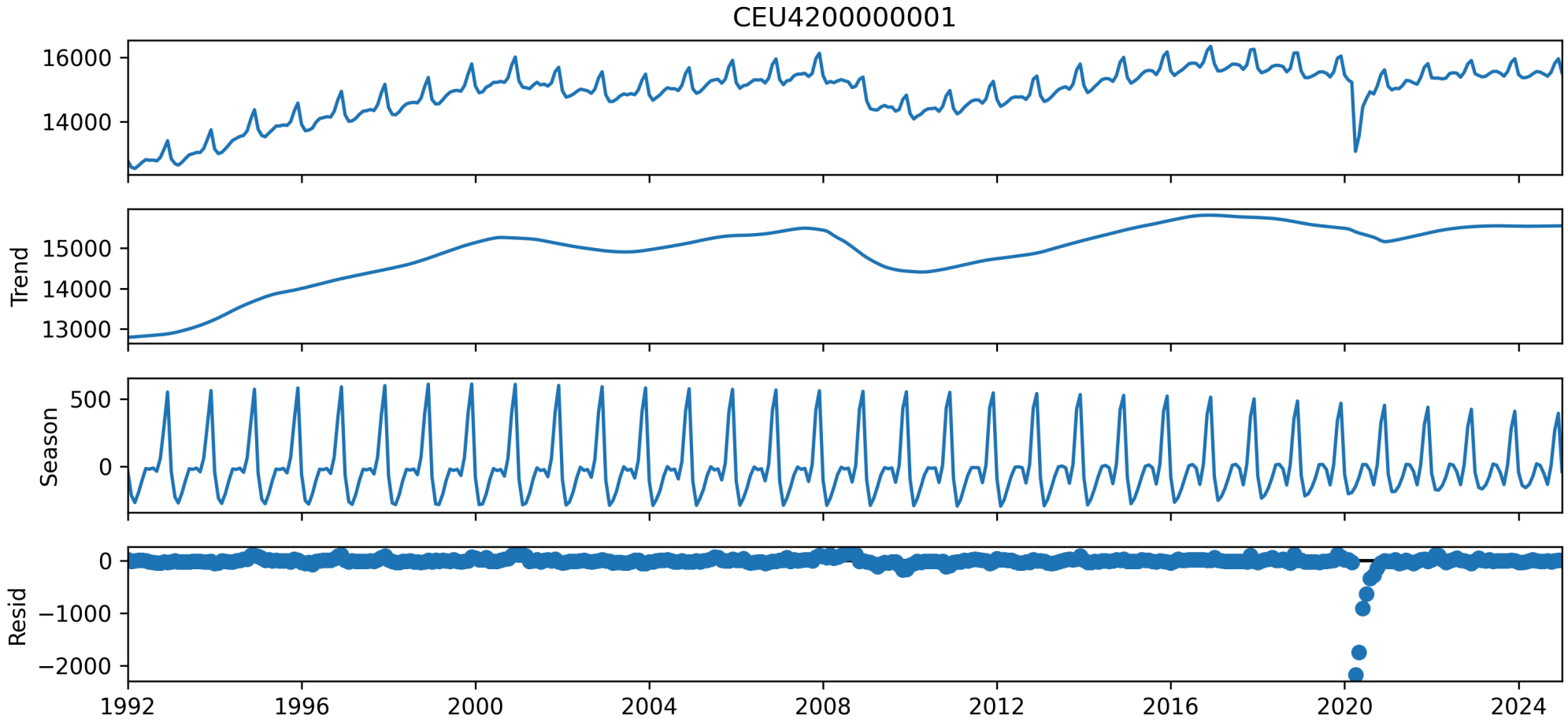
```
class statsmodels.tsa.seasonal.STL(
    endog,
    period=None,
    seasonal=7,
    trend=None,
    low_pass=None,
    seasonal_deg=1,
    trend_deg=1,
    low_pass_deg=1,
    robust=False,
    seasonal_jump=1,
    trend_jump=1,
    low_pass_jump=1
)
```

Season-Trend decomposition using LOESS.

STL Decomposition in Python (Code)

```
from statsmodels.tsa.seasonal import STL

retail_empl = (
  pd.read_csv("https://fred.stlouisfed.org/graph/fredgraph.csv?bgcolor=%23ebf3fb&chart_type=line&drp=0
  .rename(columns = {"observation_date": "date", "CEU4200000001": "CEU4200000001"})
  .assign(
    date = lambda x: pd.to_datetime(x["date"]),
      )
  )

retail_empl = retail_empl.set_index("date")
retail_empl = retail_empl.asfreq("MS")

# Decompose the time series
stl = STL(retail_empl["CEU4200000001"], seasonal = 13, robust = True)
res = stl.fit()

# Extract the trend, seasonal, and residual components
res_df = pd.DataFrame({
  "observed": res.observed, "trend": res.trend, "seasonal": res.seasonal, "residual": res.resid
  })

# Plot the decomposed time series
fig = res.plot()
fig.set_size_inches(10, 4.75)
fig.tight_layout()
```

STL Decomposition in Python (Out)



STL Decomposition vs Classical Decomposition

02:00

In two minutes, examine the differences between the output figures from the classical decomposition (additive) and the STL decomposition.

- Edit me
- ...
- ...
- ...
- ...
- ...

Recap

Summary of Main Points

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

- 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

 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.

🎯 Assignment 🎯

- In preparation for the assignment, you are encouraged to thoroughly read the following:
  - MSTL API Reference from the statsmodels package.
  - Multi-Seasonal Time Series Decomposition using MSTL in Python, a Towards Data Science Blog, which demonstrates how the MSTL method works and can be implemented on a similar dataset.
- Skim through the MSTL Research Paper.
- Once you read the references and went over our class notes, you are now ready to examine and complete Assignment 04.