Time-series,
Spring, 2026

Python & Time Series Fundamentals

Faculty of DS & AI
Spring semester, 2026

Trong-Nghia Nguyen

Business AI Lab

Content

- NumPy Review for Time Series
- Pandas for Time Series
- Technical Indicators and EDA

Content

- NumPy Review for Time Series
- Pandas for Time Series
- Technical Indicators and EDA

NumPy Review for Time Series

NumPy arrays

Exercise:

1. Create a NumPy array with 50 evenly spaced values from 0 to 10 (inclusive). What is the shape and dtype?

2. Given the array `arr = np.array([10, 20, 30, 40, 50, 60, 70, 80, 90, 100])`:
   Extract the first 5 elements
   Extract elements from index 3 to 7 (inclusive)
   Extract every other element starting from index 0

3. Create two arrays: `a = np.array([1, 2, 3])` and `b = np.array([10, 20, 30])`.
   Add them elementwise
   Multiply them elementwise
   What happens if you try `a + 5`? (Broadcasting)

4. Create a time series array with 100 daily values starting from January 1, 2020. Use `np.arange()` to create day numbers (0 to 99), then create values as `100 + 2day + np.random.normal(0, 5, 100)`. What is the mean and standard deviation?

NumPy Review for Time Series

NumPy Operations for Time Series

General representation of time series data:

$$\mathbf{X} = \{x_t \in \mathbb{R}^d \mid t = 1, 2, \ldots, T\} \quad (1)$$

- **X**: the complete time-series dataset
- **t**: time index
- **T**: total number of observed time steps
- $\mathbf{x_t}$: feature vector at time step *t*
- **d**: dimensionality of the feature space
- $\mathbb{R}^{\mathbf{d}}$: *d*-dimensional real-valued space

Interpretation:

The time series is modeled as a sequence of *d*-dimensional observations indexed by time.

NumPy Review for Time Series

NumPy Operations for Time Series

Time-series decomposition:

$$x_t = \tau_t + s_t + c_t + \epsilon_t \quad (2)$$

- **x_t**: observed value at time step *t*
- **τ_t (trend)**: long-term progression of the series
- **s_t (seasonality)**: repeating patterns with a fixed period
- **c_t (cyclic component)**: non-periodic, long-term oscillations
- **ε_t (noise)**: random fluctuations or unexplained residuals

Interpretation:

Each observation is decomposed into interpretable structural components plus random noise.

NumPy Review for Time Series

NumPy Operations for Time Series

Forecasting formulation:

$$\widehat{x}_{t+1} = f(x_t, x_{t-1}, \dots, x_{t-k}) \quad (3)$$

- $\hat{x}_{t+1}$: predicted value at the next time step
- **f(·)**: forecasting function (e.g., ARIMA, LSTM, Transformer)
- **k**: look-back window size
- x_{t-i}: historical observations used for prediction

Interpretation:

Future values are estimated based on a finite window of past observations.

NumPy Review for Time Series

NumPy Operations for Time Series

Multivariate time series:

$$x_t = \begin{bmatrix} v_t^{(1)} \\ v_t^{(2)} \\ \vdots \\ v_t^{(m)} \end{bmatrix}, \quad X \in \mathbb{R}^{T \times m} \tag{4}$$

- x_t: observation vector at time step *t*
- $v_t^{(i)}$: the *i*-th variable at time step *t*
- **m**: number of variables (features or channels)
- **T**: number of time steps
- $X \in \mathbb{R}^{T \times m}$: time–feature matrix representation of the dataset

Interpretation:

The time series is represented as a matrix, where rows correspond to time steps and columns correspond to variables.

NumPy Review for Time Series

NumPy Operations for Time Series

Scrolling (Sliding) Window Idea:

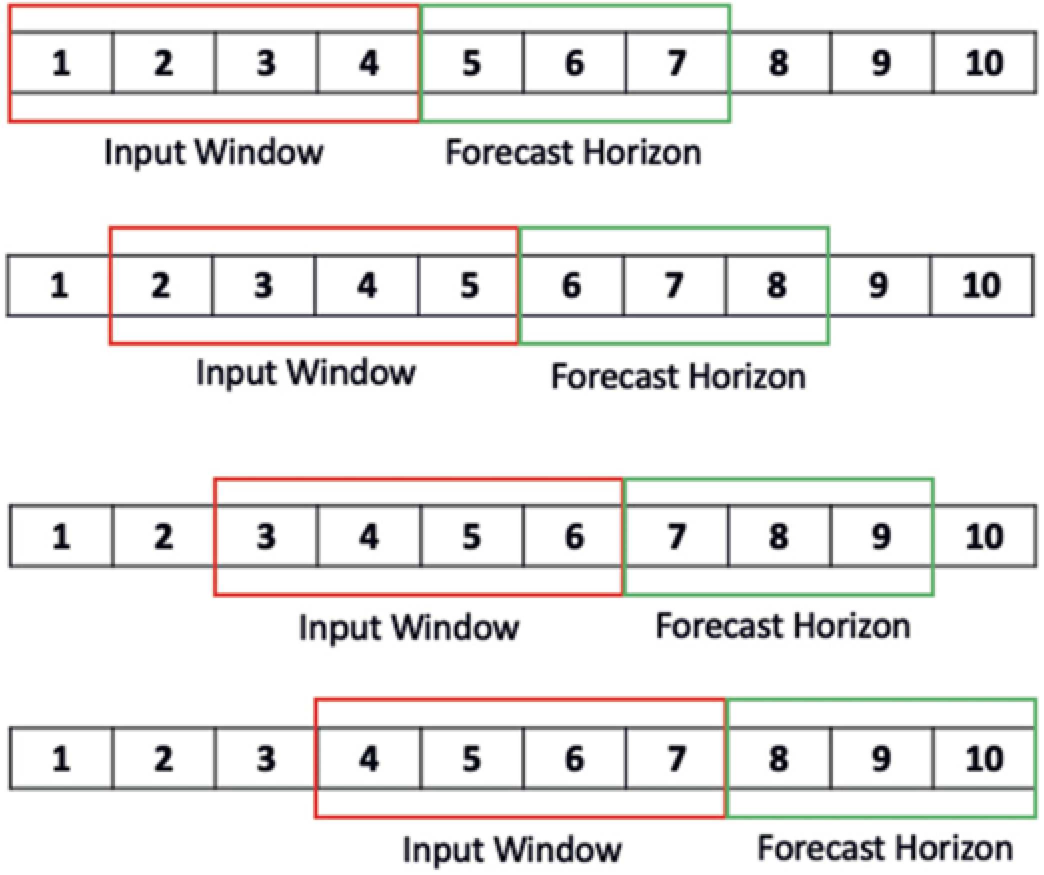
Choose a window size *k* (e.g., *k = 10*)

Slide the window one step at a time along the series

At each position *t*, compute $\bar{x}_t$ using the formula (5)

This produces a new, shorter series of smoothed values

```
# Rolling window calculation (manual)
window_size = 10
Kernel = np.ones(window_size)/window_size,
rolling_mean = np.convolve(values, kernel, mode =.
Valid)
```



NumPy Review for Time Series

NumPy Operations for Time Series

Simple Moving Average (SMA) over a window of size *k:*

$$\overline{x}_t = \frac{1}{k}\sum_{i=0}^{k-1} x_{t-i} \qquad (5)$$

k = window size

Larger *k* → smoother curve

This averages the most recent *k* points $x_t, x_{t-1}, \ldots, x_{t-k+1}$ to smooth shortterm fluctuations.

Content

- NumPy Review for Time Series
- Pandas for Time Series
- Technical Indicators and EDA

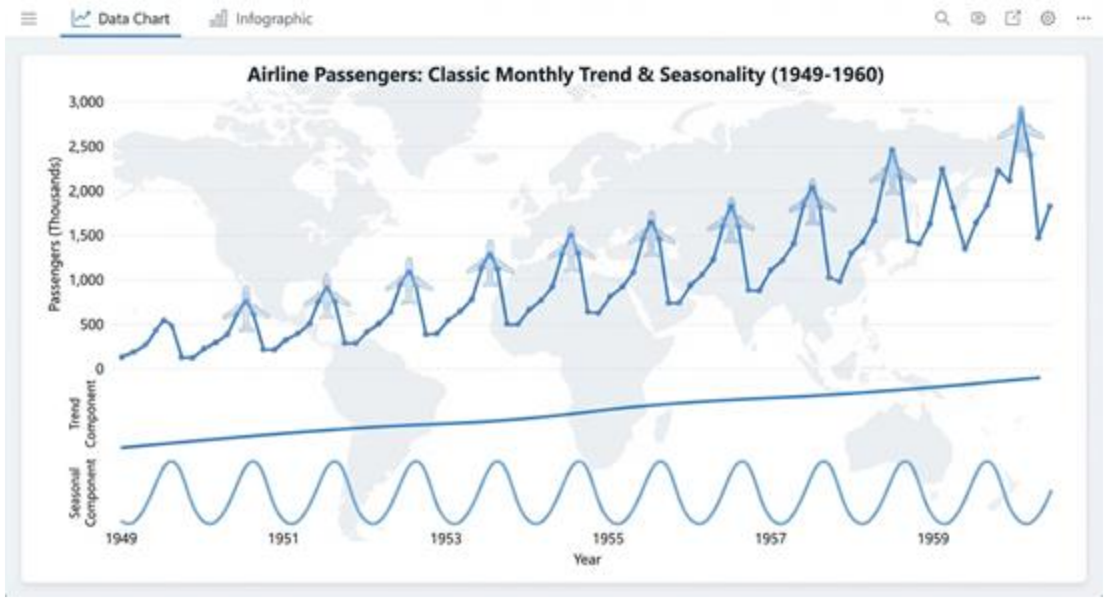
Pandas for Time Series

Pandas for Time Series

Dataset Description:

- Monthly airline passenger numbers (1949–1960)
- Univariate time series
- Clear trend and seasonality
- Widely used benchmark dataset in Time-series analysis

```python
from statsmodels.datasets import get_rdataset
import pandas as pd

# Load AirPassengers dataset
data = get_rdataset('AirPassengers', 'datasets')
df = data.data

# Convert to Pandas Series with datetime index
ts = pd.Series(df['value'].values,
               index=pd.date_range('1949-01', periods=len(df), freq='M'))
print(ts.head())
print(ts.info())
```



AirPassengers dataset

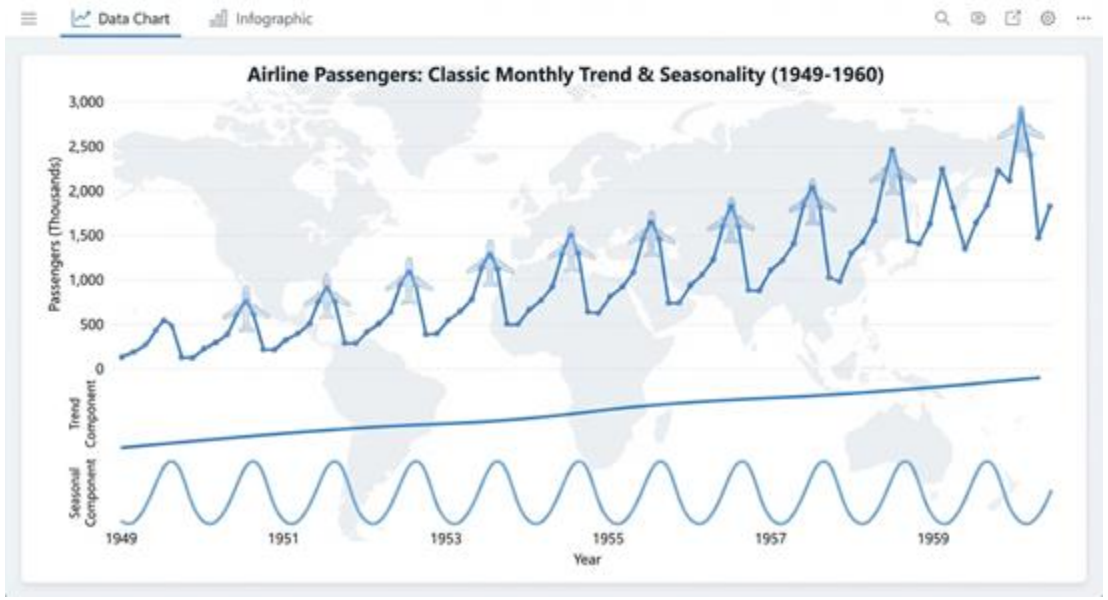
Pandas for Time Series

Pandas for Time Series

Time Indexing and Slicing:

- Datebased indexing
- Subsetting by year or date range
- Shifting the time series
- Differencing for trend removal

```python
# Datebased indexing
ts_1950 = ts['1950']  # All data from 1950
ts_1950_1955 = ts['1950':'1955']  # Date range

# Shifting
ts_lag1 = ts.shift(1)  # Shift by 1 period

# Differencing (removes trend)
ts_diff = ts.diff()  # First difference
```



AirPassengers dataset

Pandas for Time Series

Pandas for Time Series

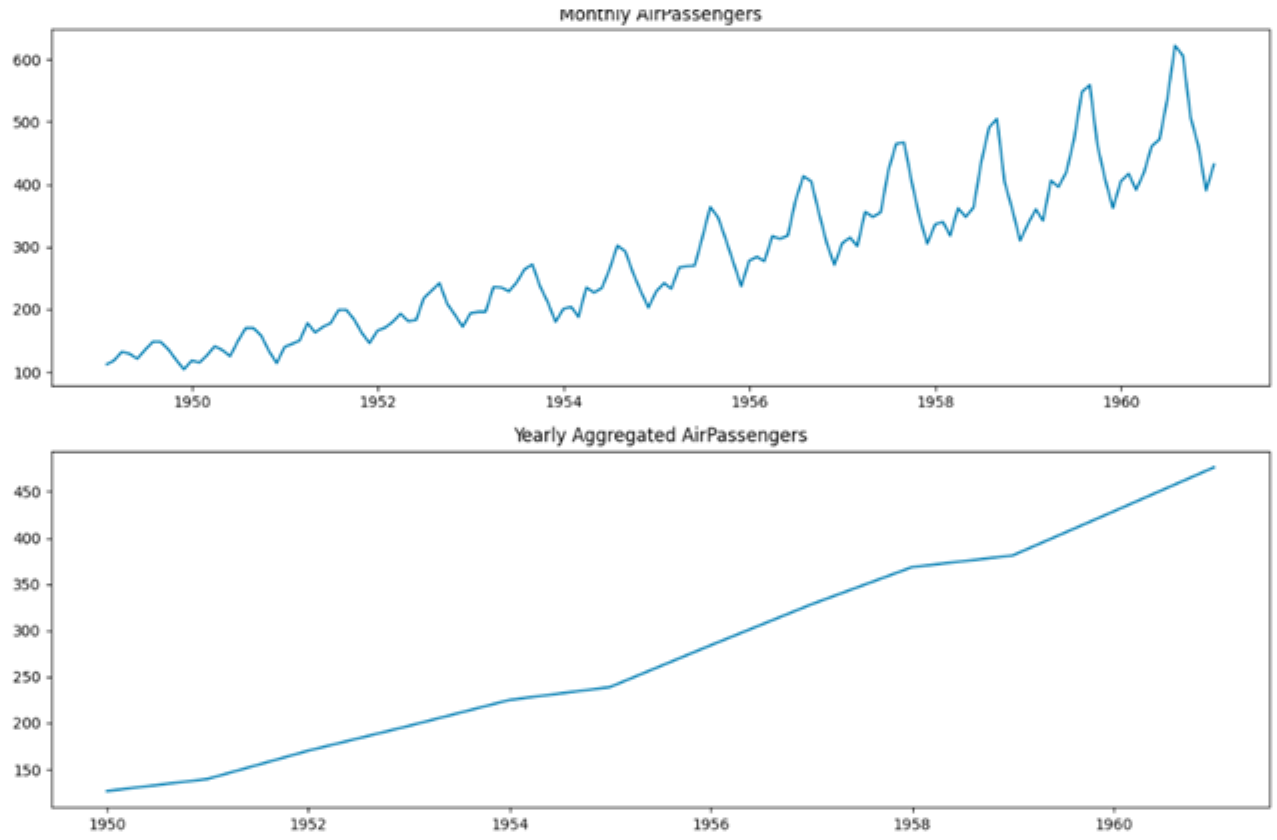
- Resampling:
  - Change frequency of observations
  - Aggregate data for analysis
  - Align multiple time series
  - Types:
    - Downsampling: Reduce frequency (e.g., daily → monthly)
    - Upsampling: Increase frequency (e.g., monthly → daily)
    - Common Frequencies:
      'D' (daily), 'W' (weekly), 'M' (monthly), 'Q' (quarterly), 'Y' (yearly)
    - Aggregation Functions:
      `mean()`, `sum()`, `last()`, `first()`, `max()`, `min()`

Pandas for Time Series

Pandas for Time Series

- Resampling AirPassengers
  - Monthly to yearly aggregation
  - Effect of resampling on trend and variance
  - Visual comparison of different frequencies

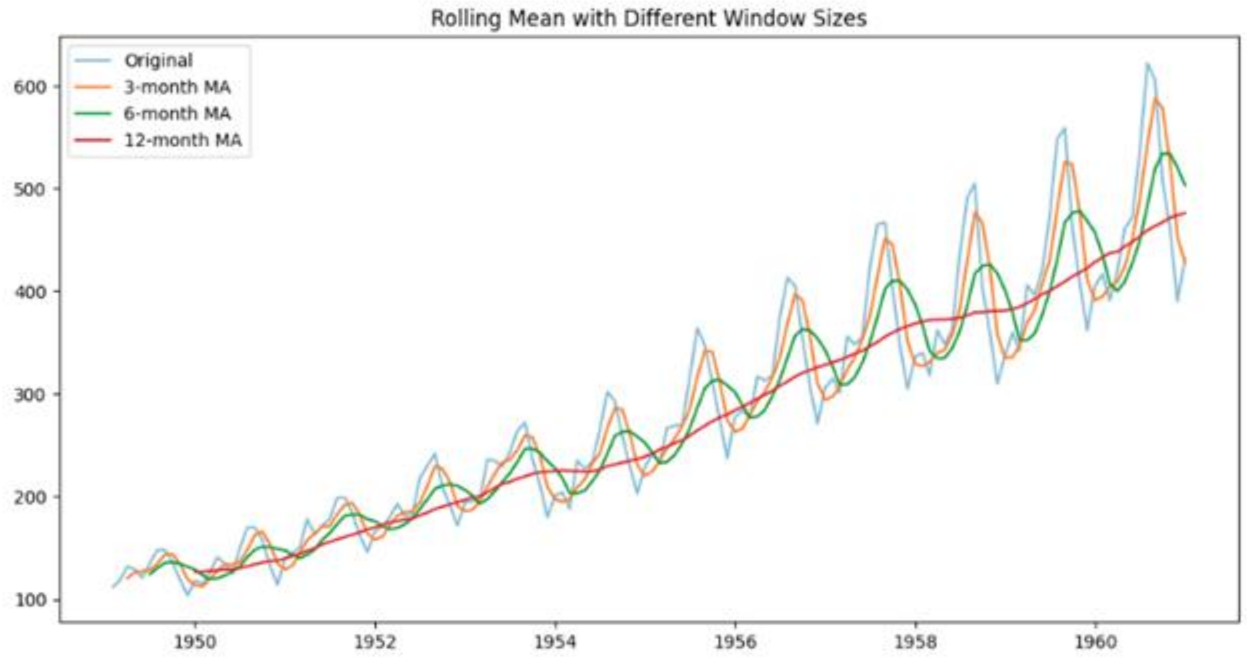
```
# Monthly to yearly aggregation
yearly = ts.resample('Y').mean()  # Yearly mean
yearly_sum = ts.resample('Y').sum()  # Yearly sum

# Visual comparison
import matplotlib.pyplot as plt
fig, axes = plt.subplots(2, 1, figsize=(12, 8))
axes[0].plot(ts.index, ts.values, label='Monthly')
axes[0].set_title('Monthly AirPassengers')
axes[1].plot(yearly.index, yearly.values, label='Yearly Mean')
axes[1].set_title('Yearly Aggregated AirPassengers')
plt.tight_layout()
plt.show()
```



Pandas for Time Series

Pandas for Time Series

- Rolling Statistics with Pandas
  - Rolling mean
  - Rolling standard deviation
  - Effect of window size (3, 6, 12)
  - Connection to NumPy sliding window

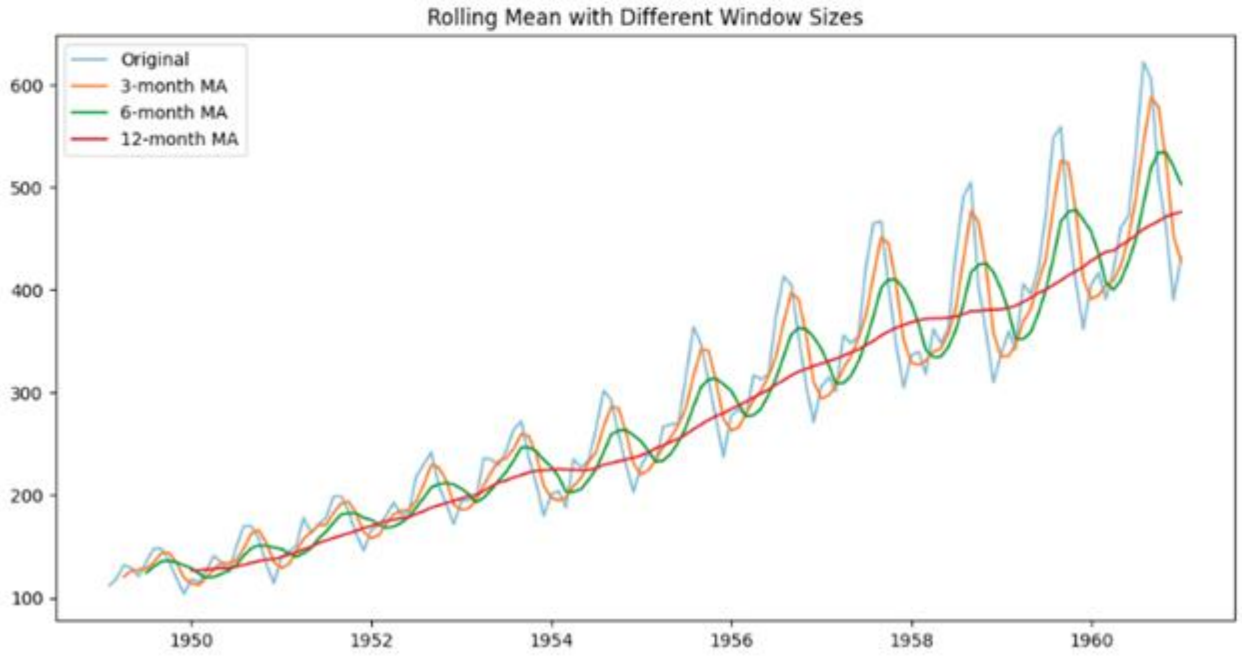


Pandas for Time Series

Pandas for Time Series

- Rolling Statistics with Pandas

```
# Rolling statistics
ts_rolling_mean_12 = ts.rolling(window=12).mean() # 12month rollir
ts_rolling_std_12 = ts.rolling(window=12).std()  # 12month rolling

# Different window sizes
ts_rolling_3 = ts.rolling(window=3).mean()
ts_rolling_6 = ts.rolling(window=6).mean()
ts_rolling_12 = ts.rolling(window=12).mean()

# Visualization
plt.figure(figsize=(12, 6))
plt.plot(ts.index, ts.values, label='Original', alpha=0.5)
plt.plot(ts_rolling_3.index, ts_rolling_3.values, label='3month MA')
plt.plot(ts_rolling_6.index, ts_rolling_6.values, label='6month MA')
plt.plot(ts_rolling_12.index, ts_rolling_12.values, label='12month MA')
plt.legend()
plt.title('Rolling Mean with Different Window Sizes')
plt.show()
```



Content

- NumPy Review for Time Series
- Pandas for Time Series
- Technical Indicators and EDA

Technical Indicators and EDA

From SMA to EMA

- Simple Moving Average (SMA): Equal weights for all observations in window.

$$\mathrm{SMA}_n = \frac{1}{n} \sum_{i=1}^{n} P_i \qquad (6)$$

- Exponential Moving Average (EMA): More weight to recent observations

$$\mathrm{EMA}_t = \alpha P_t + (1 - \alpha)\, \mathrm{EMA}_{t-1} \qquad (7)$$

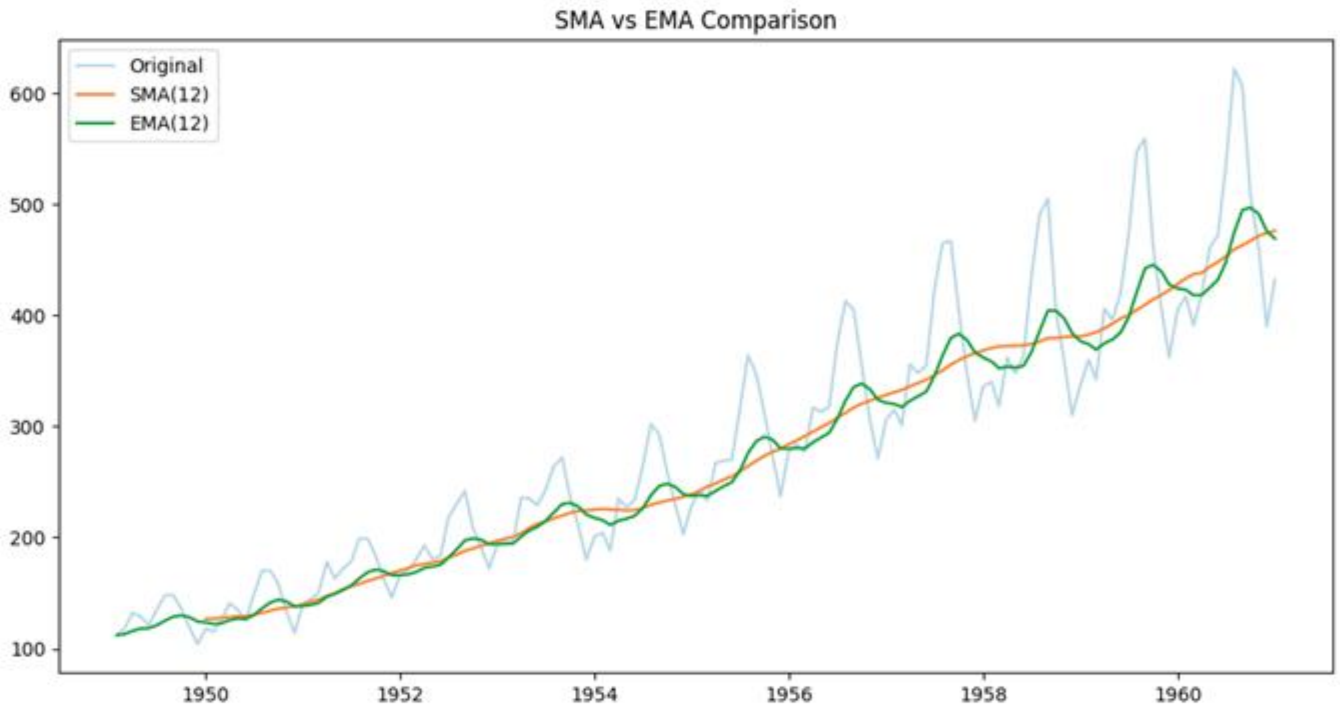
Where $\alpha = 2 / (n + 1)$

EMA is more responsive to recent changes
SMA is smoother but less reactive
Smoothing any time series data
Trend detection in various domains

Technical Indicators and EDA

From SMA to EMA

```
# SMA
sma_12 = ts.rolling(window=12).mean()

# EMA
ema_12 = ts.ewm(span=12, adjust=False).mean()

# Comparison plot
plt.figure(figsize=(12, 6))
plt.plot(ts.index, ts.values, label='Original',
alpha=0.3)
plt.plot(sma_12.index, sma_12.values, label='SMA(12)')
plt.plot(ema_12.index, ema_12.values, label='EMA(12)')
plt.legend()
plt.title('SMA vs EMA Comparison')
plt.show()
```



Technical Indicators and EDA

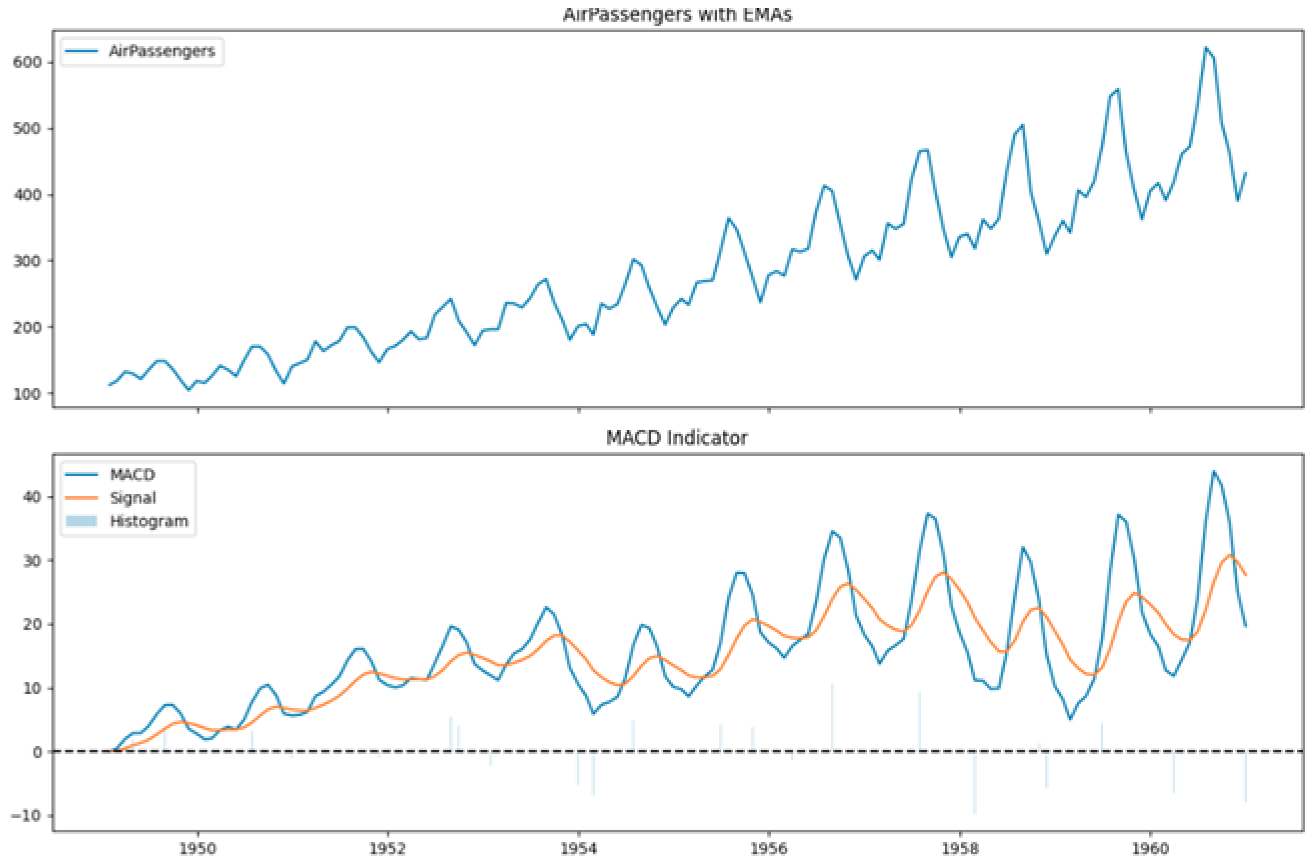
Moving Average Convergence Divergence (MACD) Indicator

- Components:
  - Fast EMA (typically 12 periods)
  - Slow EMA (typically 26 periods)
  - MACD line = Fast EMA Slow EMA
  - Signal line = EMA(9) of MACD line
- Interpretation:
  - Momentum indicator
  - MACD > Signal: Bullish momentum
  - MACD < Signal: Bearish momentum

Technical Indicators and EDA

MACD Indicator

```python
# Calculate MACD
ema_fast = ts.ewm(span=12, adjust=False).mean()
ema_slow = ts.ewm(span=26, adjust=False).mean()
macd_line = ema_fast - ema_slow
signal_line = macd_line.ewm(span=9, adjust=False).mean()
histogram = macd_line - signal_line

# Visualization
fig, axes = plt.subplots(2, 1, figsize=(12, 8), sharex=True)
axes[0].plot(ts.index, ts.values, label='AirPassengers')
axes[0].set_title('AirPassengers with EMAs')
axes[0].legend()

axes[1].plot(macd_line.index, macd_line.values, label='MACD')
axes[1].plot(signal_line.index, signal_line.values, label='Signal')
axes[1].bar(histogram.index, histogram.values, alpha=0.3, label='Histogram')
axes[1].axhline(y=0, color='black', linestyle='--')
axes[1].set_title('MACD Indicator')
axes[1].legend()
plt.tight_layout()
plt.show()
```



Technical Indicators and EDA

Relative Strength Index (RSI) Indicator

- Relative Strength Index (RSI):
  - Momentum oscillator measuring speed and magnitude of changes
  - Formula: RSI = 100 (100 / (1 + RS))

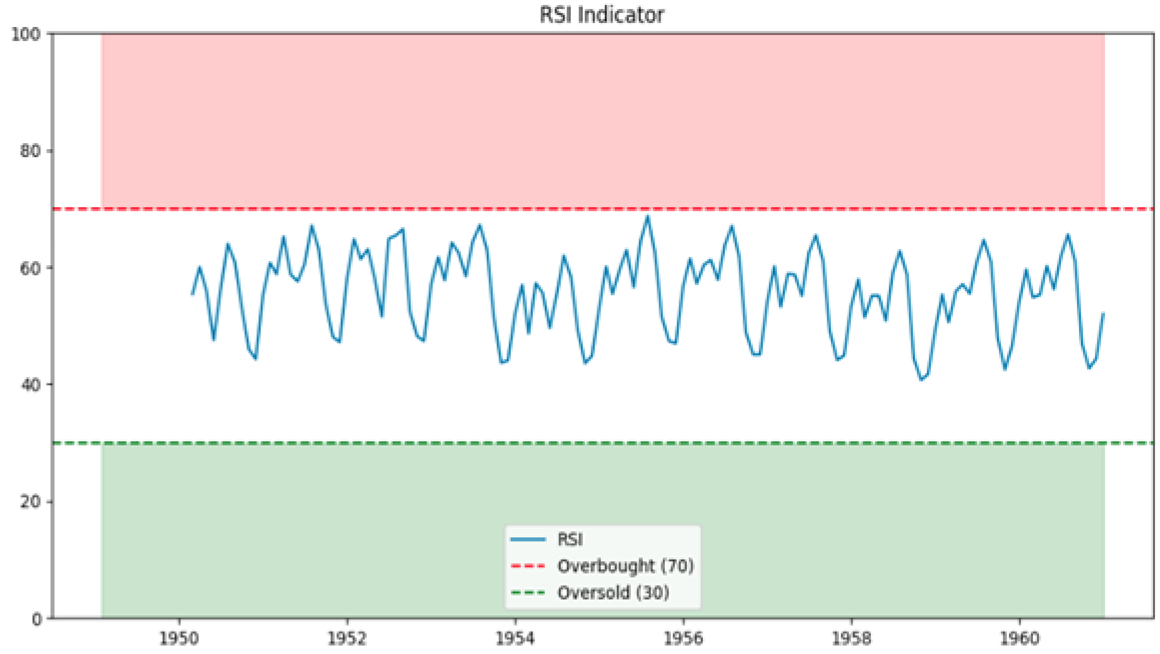
  Where RS = Average Gain / Average Loss (over n periods)

- Calculation:
  - Gains and losses over a window (typically 14 periods)
  - Separates positive and negative price changes
- Interpretation:
  - RSI > 70: Overbought (potential reversal)
  - RSI < 30: Oversold (potential reversal)
  - RSI = 50: Neutral
- NonFinancial Illustration Purpose:
  - Demonstrates momentum concepts applicable to any time series
  - Useful for identifying extreme values in any domain

Technical Indicators and EDA

RSI Indicator (Conceptual)

```
def calculate_rsi(data, window=14):
    delta = data.diff()
    gain = (delta.where(delta > 0, 0)).rolling(window=window).mean()
    loss = (-delta.where(delta < 0, 0)).rolling(window=window).mean()
    rs = gain / loss
    rsi = 100 - (100 / (1 + rs))
    return rsi

rsi = calculate_rsi(ts, window=14)

# Visualization
plt.figure(figsize=(12, 6))
plt.plot(rsi.index, rsi.values, label='RSI')
plt.axhline(y=70, color='r', linestyle='--', label='Overbought (70)')
plt.axhline(y=30, color='g', linestyle='--', label='Oversold (30)')
plt.fill_between(rsi.index, 70, 100, alpha=0.2, color='red')
plt.fill_between(rsi.index, 0, 30, alpha=0.2, color='green')
plt.ylim(0, 100)
plt.title('RSI Indicator')
plt.legend()
plt.show()
```



Technical Indicators and EDA

Exploratory Data Analysis (EDA)

- Line plot of the series
- Seasonal patterns identification
- Rolling mean and variance analysis
- Trend–seasonality decomposition

- Excercise: proceed EDA with AirPassengers dataset

Thank you!