Feature Engineering for Time Series Data

Minh-Duc Bui

Year 2023

Outline

- Feature Engineering in General
- Feature Engineering for Time Series Data

Outline

- Feature Engineering in General
- Feature Engineering for Time Series Data

Feature Engineering in General

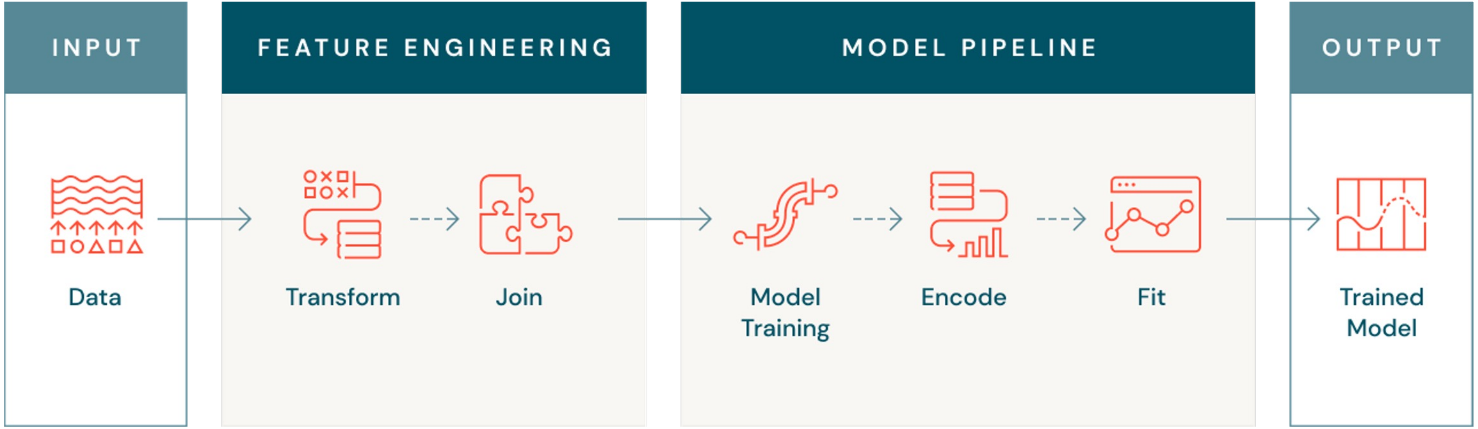
Feature engineering is the process of extracting features (characteristics, properties, attributes) from raw data.

- Missing values
- Data Normalization/Standardization
- Encoding Categorical Data
- Dimensionality Reduction
- Temporal Feature Engineering
- Domain-Specific Features

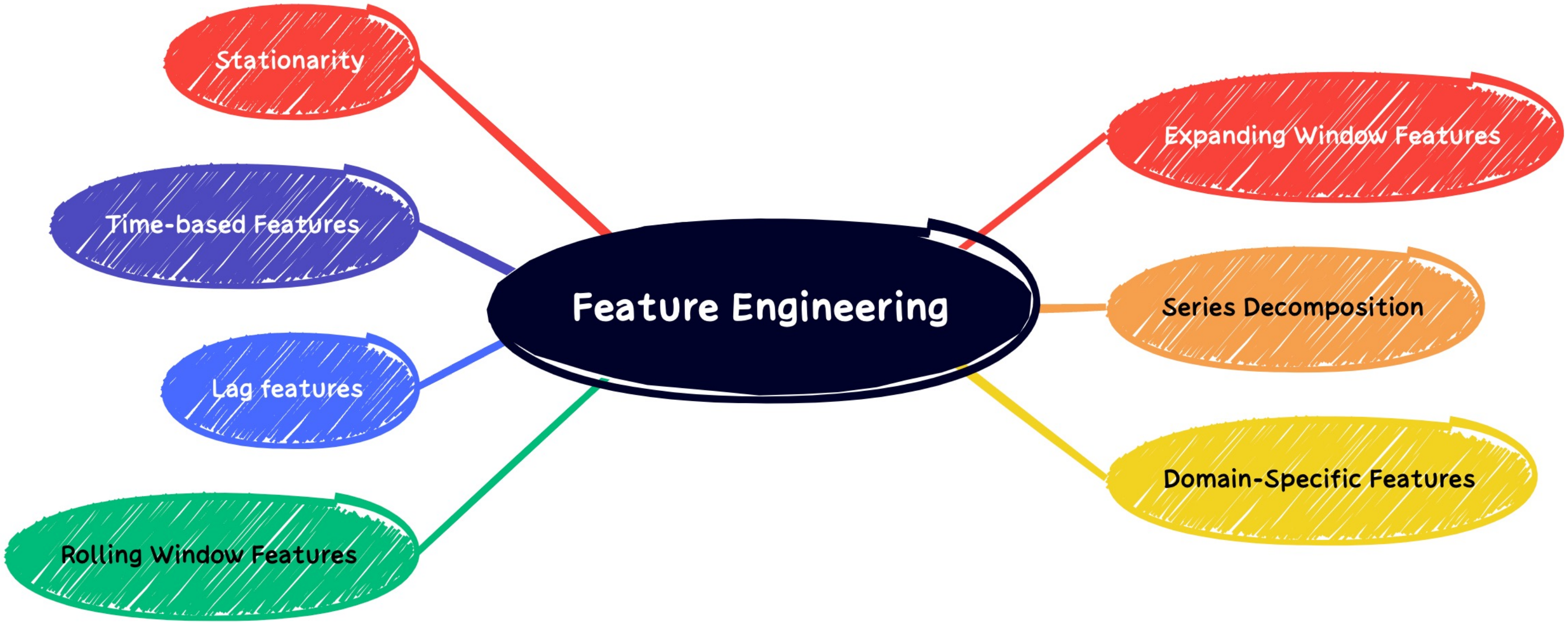
Feature Engineering in General

- Having the right features tends to **give the biggest performance boost compared to clever algorithmic techniques** such as hyper-parameter tuning.

- State-of-the-art model architectures **can still perform poorly if they don't use a good set of features**.



Feature Engineering for Time Series



Outline

- Feature Engineering in General
- Feature Engineering for Time Series Data

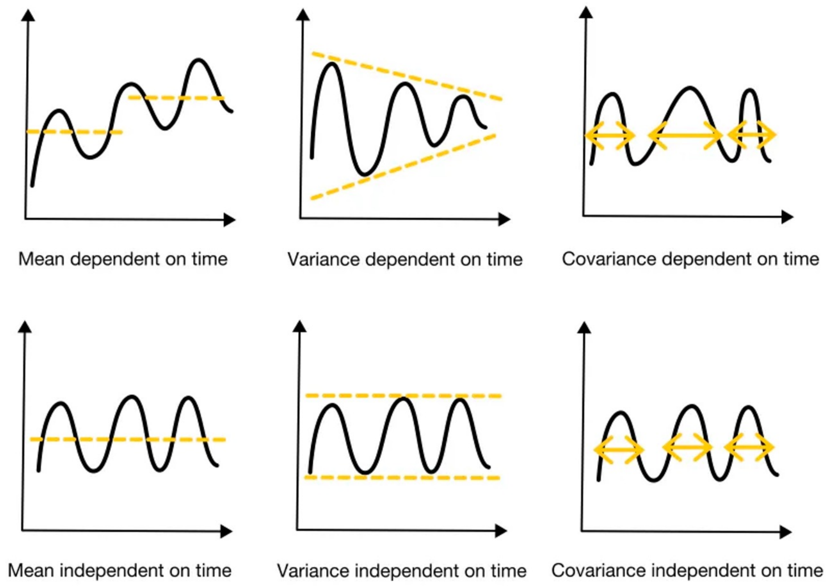
Stationarity

❖ What is Stationarity?

Stationarity describes the concept that how a time series is changing will remain the same in the future.

In **mathematical terms**, a time series is stationary when its **statistical properties are independent of time**:

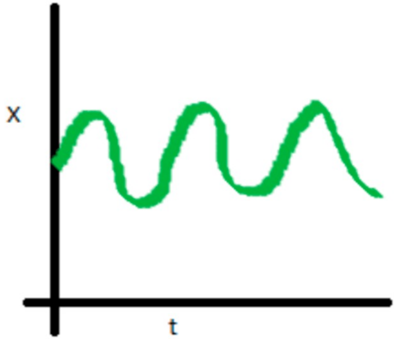
- constant mean,
- constant variance, and
- covariance is independent of time.



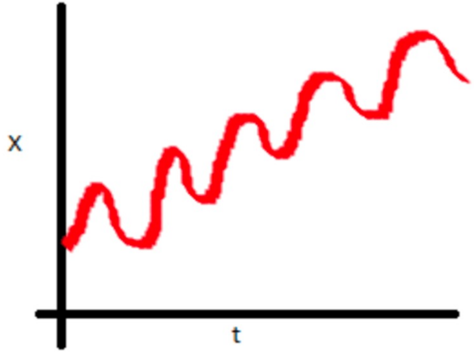
Stationarity

❖ **Why it is important?**

Some time series forecasting models (e.g., autoregressive models) require a stationary time series because they are easier to model due to their constant statistical properties.

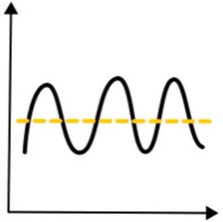


Stationary series

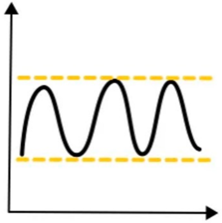


Non-Stationary series

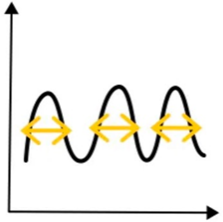
Stationarity



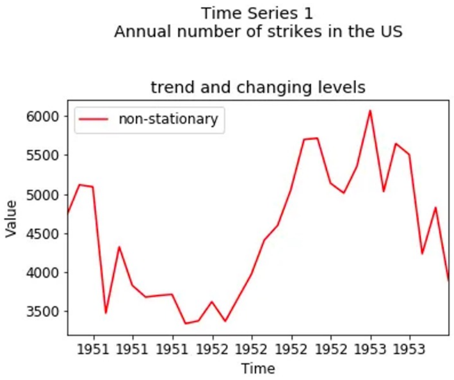
Mean independent on time

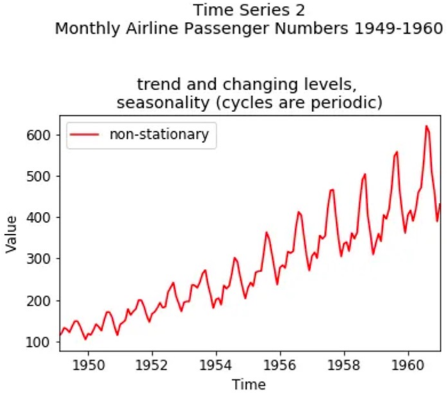


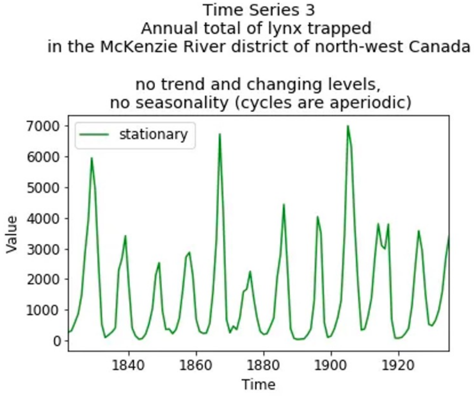
Variance independent on time

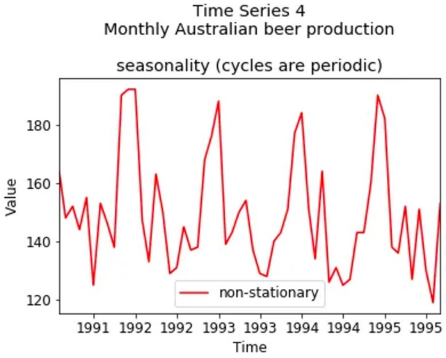


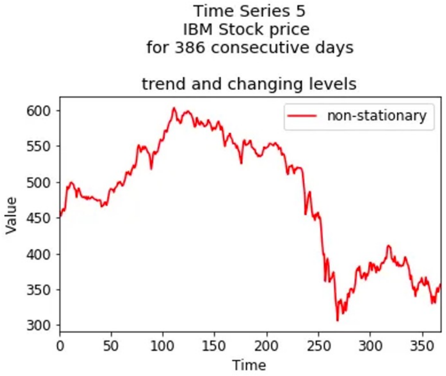
Covariance independent on time

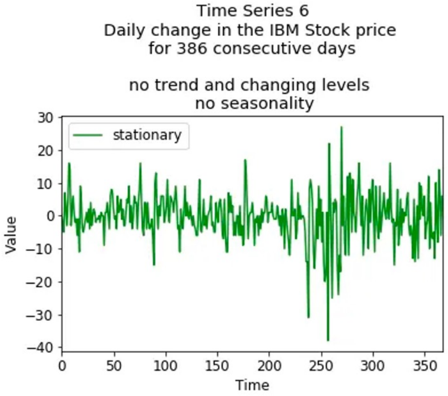










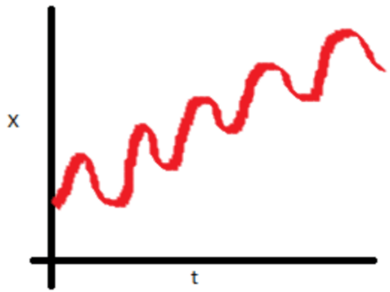


Stationarity

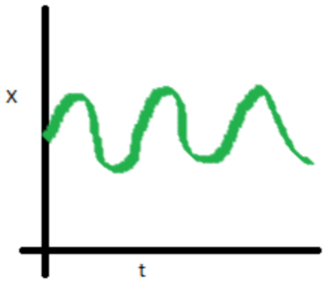
❖ Non-stationarity => Stationarity

You can apply different transformations to a non-stationary time series to try to make it stationary:

- Differencing
- Detrending by model fitting
- Log transformation



Non-Stationary series



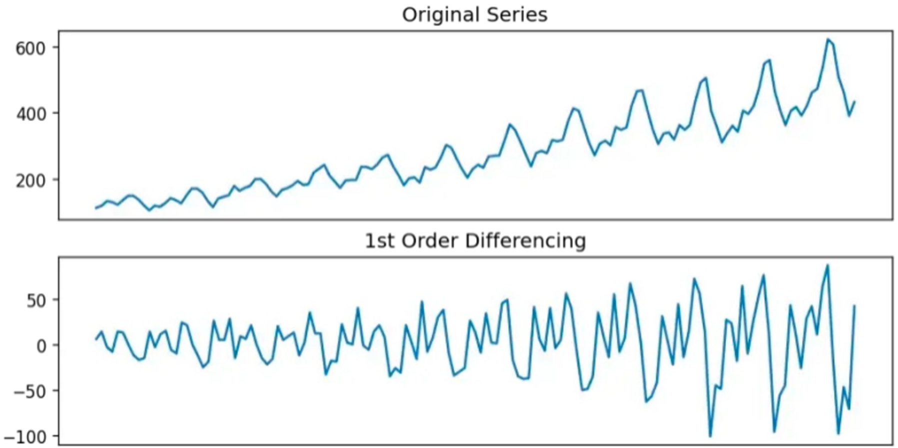
Stationary series

Stationarity

❖ First-order differencing

The differenced series is the change between consecutive observations in the original series:

$$y'_t = y_t - y_{t-1}.$$




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

df = pd.read_csv('ch2_co2_levels.csv')

# Convert to datetime
df['datestamp'] =
pd.to_datetime(df['datestamp'])

# Set datestamp column as index
df = df.set_index('datestamp')



df_diff = df.diff()
```

Stationarity

❖ Second-order differencing

Occasionally the differenced data will not appear to be stationary and it may be necessary to difference the data a second time to obtain a stationary series:

$$\begin{aligned} y''_t &= y'_t - y'_{t-1} \\ &= (y_t - y_{t-1}) - (y_{t-1} - y_{t-2}) \\ &= y_t - 2y_{t-1} + y_{t-2}. \end{aligned}$$

$$y'_t = y_t - y_{t-1}.$$

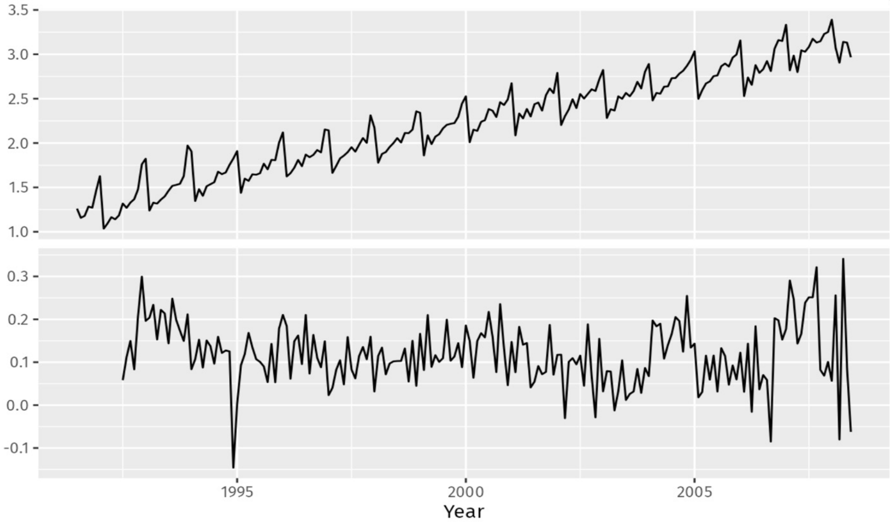

```python
import pandas as pd
import matplotlib.pyplot as plt


df = pd.read_csv('ch2_co2_levels.csv')

# Convert to datetime
df['datestamp'] =
pd.to_datetime(df['datestamp'])

# Set datestamp column as index
df = df.set_index('datestamp')



df_diff = df.diff().diff()
```

Stationarity

❖ Seasonal differencing

A seasonal difference is the difference between an observation and the previous observation from the same season.

$$y'_t = y_t - y_{t-m},$$

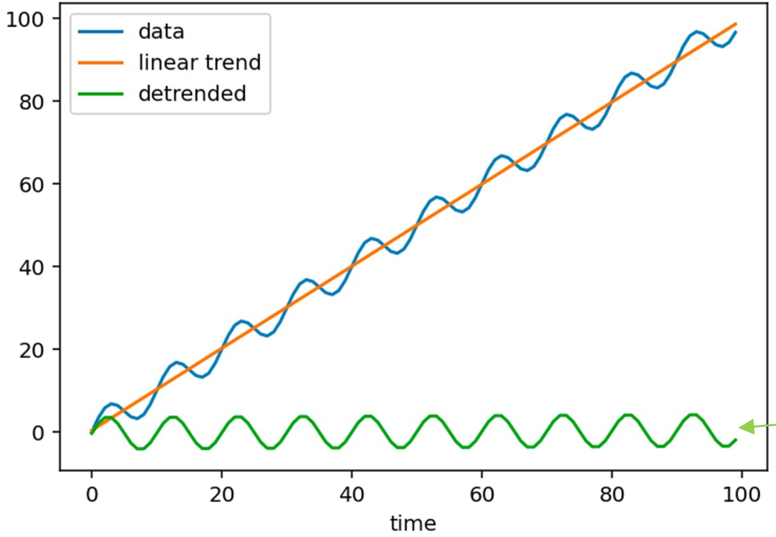



```python
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv('ch2_co2_levels.csv')

# Convert to datetime
df['datestamp'] =
pd.to_datetime(df['datestamp'])

# Set datestamp column as index
df = df.set_index('datestamp')



df_diff = df.diff(periods=12)
```

Stationarity

❖ Detrending by model fitting

Fit a trend model and then subtracting the trend component from the original series.



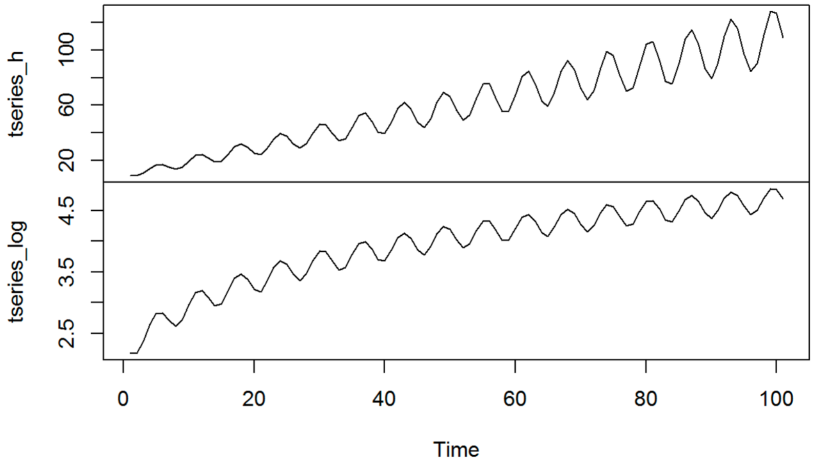

```
from sklearn.linear_model import LinearRegression

# Create a numerical time index (e.g., 0, 1, 2, ...)
df['time_index'] = np.arange(len(df))

# Prepare the data for the linear model
X = df['time_index'].values.reshape(-1, 1)
y = df['co2'].values

# Create and fit the linear model
model = LinearRegression()
model.fit(X, y)

# Predict CO2 levels using the model
predicted_co2 = model.predict(X)

# Detrend the data by subtracting the predicted
values from the original CO2 levels
detrended_co2 = df['co2'] - predicted_co2
```

Stationarity

❖ Log transformation

Apply the logarithm function to each data point in a dataset.

=> Stabilizing Variance, Reducing Skewness




```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv('ch2_co2_levels.csv')

# Convert to datetime
df['datestamp'] =
pd.to_datetime(df['datestamp'])

# Set datestamp column as index
df = df.set_index('datestamp')



df_log_transforme = np.log(df['co2'])
```

Time-based Features

- Create new features from date and time information.
- Extract the day of the week, hour of the day, or month of the year.

	co2	month	day
datestamp			
1958-03-29	316.1	3	29
1958-04-05	317.3	4	5
1958-04-12	317.6	4	12
1958-04-19	317.5	4	19
1958-04-26	316.4	4	26
1958-05-03	316.9	5	3
1958-05-10	317.5	5	10
1958-05-17	317.5	5	17
1958-05-24	317.9	5	24
1958-05-31	315.8	5	31

```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv('ch2_co2_levels.csv')

# Convert to datetime
df['datestamp'] =
pd.to_datetime(df['datestamp'])

# Set datestamp column as index
df = df.set_index('datestamp')

df['month'] = df.index.month
df['day']   = df.index.day
```

Lag Features

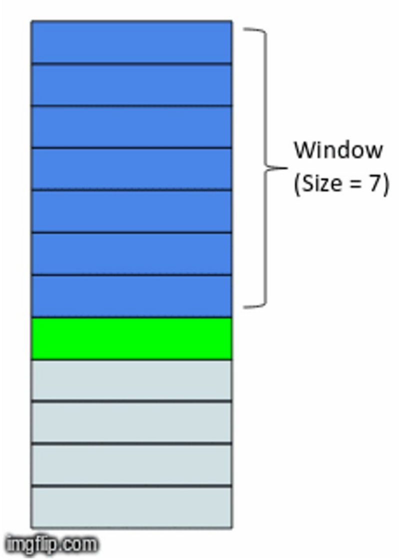
- Shift the values of a variable backward in time by a certain number of time periods.
- Lagged features can capture temporal dependencies and trends in the data.

datestamp	co2	lag_1	lag_2	lag_3
1958-03-29	316.1	NaN	NaN	NaN
1958-04-05	317.3	316.1	NaN	NaN
1958-04-12	317.6	317.3	316.1	NaN
1958-04-19	317.5	317.6	317.3	316.1
1958-04-26	316.4	317.5	317.6	317.3
1958-05-03	316.9	316.4	317.5	317.6
1958-05-10	317.5	316.9	316.4	317.5
1958-05-17	317.5	317.5	316.9	316.4
1958-05-24	317.9	317.5	317.5	316.9
1958-05-31	315.8	317.9	317.5	317.5

```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv('ch2_co2_levels.csv’)

# Convert to datetime
df['datestamp’] =
pd.to_datetime(df['datestamp'])

# Set datestamp column as index
df = df.set_index('datestamp’)

df['lag_1'] = df['co2'].shift(1)
df['lag_2'] = df['co2'].shift(2)
df['lag_3'] = df['co2'].shift(3)
```

Rolling Window Features

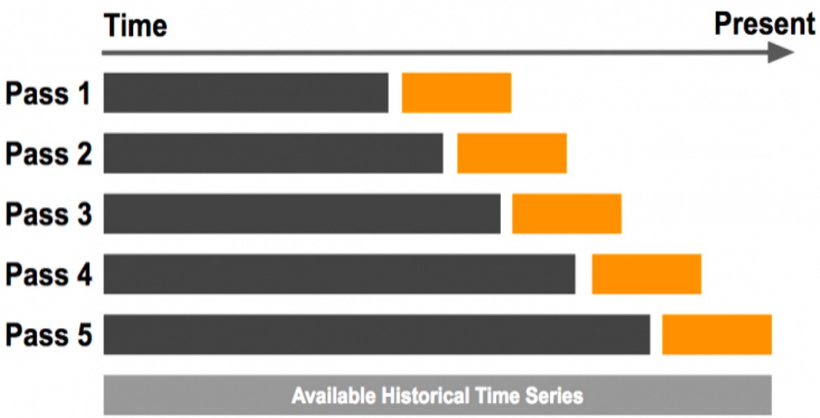
- Calculating summary statistics, such as the **mean** or **standard deviation**, over a sliding window of previous values.
  - highlighting long-term trends or cycles
  - smoothing out short-term fluctuations
  - removing outliers

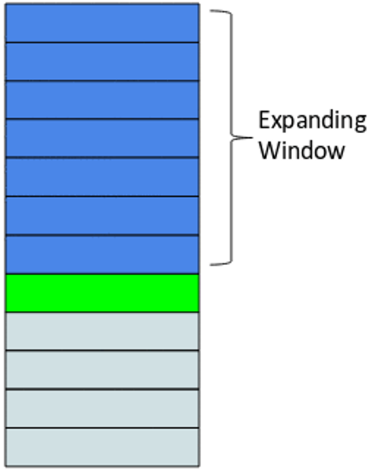



```
df_mean = df.rolling(window=48).mean()
```

Expanding Window Features

- In zolling window technique, we consider only the most recent values and ignore the past values.
- The idea behind the expanding window feature is that it **takes all the past values into account**.



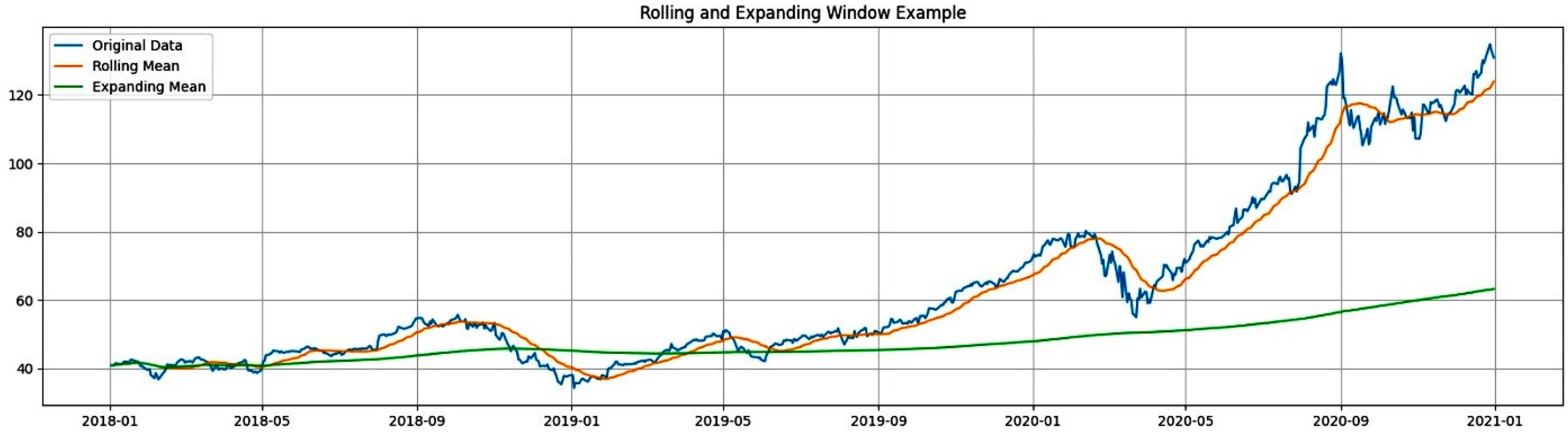



```
df_mean = df.expanding().mean()
```

Rolling & Expanding Window

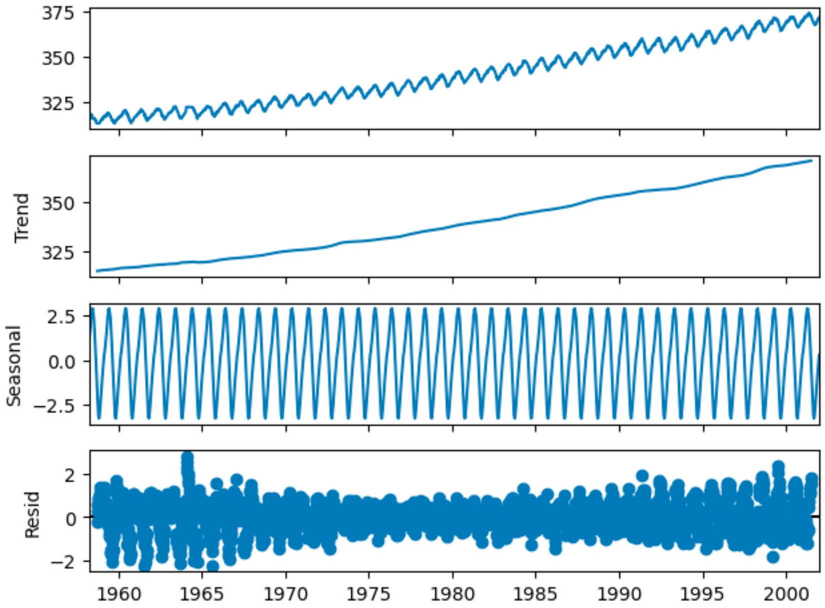
- Capture the local trends, fluctuations, and overall behavior.
  => Allow the model to learn from the temporal dynamics.

- **Rolling window**: useful when dealing with **noisy data** or **non-stationarity**.
- **Expanding window**: provide insights into the **cumulative effects** or **long-term trends**.



Series Decomposition

- Time series are a combination of (mainly) three components: *trend, seasonality, and residuals/remainder*
- Decomposition provides a useful abstract model:
  - thinking about time series generally
  - better understanding problems during time series analysis and forecasting



	CO2	trend	seasonal	resid
datestamp				
1959-04-04	317.7	315.759615	1.235242	0.705142
1959-04-11	317.1	315.760577	1.412344	-0.072921
1959-04-18	317.6	315.765385	1.701186	0.133429
1959-04-25	318.3	315.773077	1.950694	0.576229
1959-05-02	318.2	315.787500	2.032939	0.379561
1959-05-09	318.7	315.811538	2.445506	0.442955
1959-05-16	318.0	315.840385	2.535041	-0.375426
1959-05-23	318.4	315.872115	2.662031	-0.134147
1959-05-30	318.5	315.899038	2.837948	-0.236987
1959-06-06	318.5	315.914423	2.786137	-0.200560
1959-06-13	318.1	315.930769	2.897139	-0.727908

```python
from statsmodels.tsa.seasonal import seasonal_decompose

# Plot the decomposition for multiplicative series
decomposition_plot = seasonal_decompose(df)
```

Series Decomposition

procedure DLINEAR(x)

$\boxed{seasonal\_init}, \boxed{trend\_init} \leftarrow \text{SeriesDecomposition}(x)$

$seasonal\_output \leftarrow \text{Linear\_Seasonal}(seasonal\_init)$

$trend\_output \leftarrow \text{Linear\_Trend}(trend\_init)$

$result \leftarrow seasonal\_output + trend\_output$

return $result$

end procedure

Zeng, Ailing, et al. "Are transformers effective for time series forecasting?." *Proceedings of the AAAI conference on artificial intelligence*. Vol. 37. No. 9. 2023.

Domain-Specific Features

- Incorporating domain-specific features can **significantly enhance the performance**.

- Domain-specific features are **derived from expert knowledge** in the relevant field and can **provide valuable information that is not present in the raw time series data**.

For example in finance:

- P/E, P/E, EPS

- Earnings Before Interest and Taxes (EBIT): Lợi nhuận trước lãi vay và trước thuế
  `EBIT = Lợi nhuận trước thuế + Chi phí lãi vay`

- Enteprise Value (EV): Giá trị doanh nghiệp
  `EV = Market Cap + Tổng nợ - Tiền mặt và các khoản tương đương tiền`