



■ ■ Polars

■ ■ ■ 9: Time Series Analysis

■ ■ ■ ■ ■ : 02/11/2025



1. **Signal Processing Fundamentals**
2. **Discrete-Time Signals and Systems**
 - 2.1 **Discrete-Time Signals**
 - 2.2 **Discrete-Time Systems**
 - 2.3 **Block Diagrams**
 - 2.4 **Interconnection of Systems**
 - 2.5 **Stability**
 - 2.6 **Frequency Response**
3. **Discrete-Time Fourier Transform (DTFT) (Rolling Windows)**
 - 3.1 **DTFT Pairs**
 - 3.2 **DTFT Properties**
 - 3.3 **DTFT of Discrete-Time Signals**
 - 3.4 **DTFT of Discrete-Time Systems**
4. **Discrete-Time Fourier Series (DTFS) (Resampling)**
 - 4.1 **DTFS Pairs (Downsampling)**
 - 4.2 **DTFS Pairs (Upsampling)**
 - 4.3 **DTFS Properties**
 - 4.4 **DTFS of Discrete-Time Signals**
5. **Discrete-Time Fourier Transform (DTFT)**

1. ■■■■■ ■■■■■■■■■■ ■■■■■■■■■■

■■■■■ ■■ ■■■■ ■■■■■ ■■■■■■ ■■■■■■ ■■■■■■ ■■■■ (Time Series Analysis)
■■■■■■■ ■■■■■■ Polars - ■■■■■■ ■■■■■■ ■■■■■■ ■■■■■■ ■■■■■■
■■■■■■■.

■■■■■■■ ■■■■■■:

Python 3.8+	■■■■■ ■■■■■■ ■■■■■■■■■■
Polars	■■■■■■■ ■■■■■■ ■■■■■■
Datetime	■■■■■■■ ■■■■■■■■■■ (■■■■■■■)
■■■■■■■ ■■■■ ■■■■■■	toronto_weather.csv

■■■■■■■ ■■■■■■■■■■:

```
import polars as pl
lf = pl.scan_csv('../data/toronto_weather.csv')
lf = lf.with_columns(pl.col('temperature')-273.15)
```

2. **Polars Datetime Operations**

Polars provides a powerful set of tools for working with datetime data. This section covers the basic operations for extracting components from a datetime column. Polars uses the `dt` accessor for these operations.

2.3 **Extracting Date and Time Components:**

Operation	Result	Example
<code>.dt.year()</code>	Year	2024
<code>.dt.month()</code>	Month	1-12
<code>.dt.day()</code>	Day	1-31
<code>.dt.hour()</code>	Hour	0-23
<code>.dt.weekday()</code>	Weekday	0-6
<code>.dt.date()</code>	Date	2024-01-15
<code>.dt.time()</code>	Time	14:30:00

```
lf.select(
    'datetime',
    pl.col('datetime').dt.year().alias('year'),
    pl.col('datetime').dt.month().alias('month'),
    pl.col('datetime').dt.day().alias('day')
)
```

3. Rolling Windows (Rolling Windows)

Rolling windows are a way to calculate statistics over a moving window of data. They are useful for smoothing out noise in time series data and for identifying trends. Rolling windows can be applied to any data series, but they are most commonly used on time series data. Rolling windows can be applied to any data series, but they are most commonly used on time series data.

rolling_mean()		
rolling_min()		
rolling_max()		
rolling_sum()		
rolling_std()		
rolling_map()		

```
# Rolling mean of temperature
pl.col('temperature').rolling_mean(3)

# Rolling min_periods of temperature
pl.col('temperature').rolling_mean(
    window_size=7,
    min_periods=1
)
```

4. Resampling (Resampling)

Resampling is the process of changing the frequency of the data. It is used to convert data from one frequency to another. For example, you can convert data from daily to hourly or from monthly to quarterly.

	Downsampling	Upsampling	Interpolation
Downsampling	group_by_dynamic	group_by_dynamic	group_by_dynamic
Upsampling	upsample	upsample	upsample
Interpolation	interpolate	interpolate	interpolate

Downsampling:

```
df.set_sorted('datetime').group_by_dynamic(
    'datetime',
    every='1w' # 1 week
).agg(
    pl.col('humidity').mean()
)
```

Upsampling + Interpolation:

```
df.upsample(
    time_column='datetime',
    every='30m'
).with_columns(
    pl.col('humidity').interpolate()
)
```

5. Functime

Functime is a Python library for time series forecasting. It is built on top of Polars. Functime provides a simple and intuitive API for forecasting time series data.

Example:

Model Name	Model Class	Model Type
linear_model	LinearModel	Linear
lightgbm	LightGBM	Tree
xgboost	XGBoost	Tree
knn	K-Nearest Neighbors	Distance
auto_lightgbm	AutoML	Tree

```
from functime.forecasting import linear_model
from functime.metrics import mase

# Create a linear model
forecaster = linear_model(lags=24, freq='1mo')
forecaster.fit(y=y_train)

# Predict
y_pred = forecaster.predict(fh=3)

# Calculate MASE
scores = mase(y_true=y_test, y_pred=y_pred, y_train=y_train)
```

Example:

Metric	Metric Name	Description
MASE	Mean Absolute Scaled Error	Mean Absolute Scaled Error (MASE)
SMAPE	Symmetric MAPE	Symmetric Mean Absolute Percentage Error (SMAPE)
MAE	Mean Absolute Error	Mean Absolute Error (MAE)
RMSE	Root Mean Squared Error	Root Mean Squared Error (RMSE)

6. Polars Rolling Windows

Polars has a powerful rolling window API that allows you to perform a wide range of operations on your data. In this section, we will explore some of the most common operations and their corresponding Polars syntax.

Rolling Window Operation	Polars Syntax	Notes
1. Rolling Mean	<code>df.rolling_window(5).mean()</code>	Calculates the mean of the data in the rolling window.
2. Rolling Sum	<code>df.rolling_window(5).sum()</code>	Calculates the sum of the data in the rolling window.
3. Rolling Min	<code>df.rolling_window(5).min()</code>	Calculates the minimum value in the rolling window.
4. Rolling Max	<code>df.rolling_window(5).max()</code>	Calculates the maximum value in the rolling window.
5. Rolling Std	<code>df.rolling_window(5).std()</code>	Calculates the standard deviation of the data in the rolling window.
6. Rolling Var	<code>df.rolling_window(5).var()</code>	Calculates the variance of the data in the rolling window.
7. Rolling Correlation	<code>df.rolling_window(5).corr()</code>	Calculates the correlation between two columns in the rolling window.
8. Rolling Regression	<code>df.rolling_window(5).fit()</code>	Calculates the linear regression coefficients for the data in the rolling window.

Rolling Window Pitfalls (Pitfalls):

Pitfall	Solution
datetime is not sorted	<code>df.sort_by('datetime')</code>
upsample requires DataFrame	<code>df.to_pandas().upsample().to_polars()</code>
rolling_map requires min_periods=1	<code>df.rolling_window(5).min_periods(1).rolling_map()</code>
rolling_map requires DataFrame	<code>df.to_pandas().rolling_map().to_polars()</code>
rolling_map requires replace vs convert	<code>df.to_pandas().rolling_map().to_polars()</code>
rolling_map requires String	<code>df.to_pandas().rolling_map().to_polars()</code>

Rolling Window Examples:

- Rolling Mean: `df.rolling_window(5).mean()`
- API Reference: <https://pola-rs.github.io/polars/py-polars/html/reference/>
- Discord: <https://discord.gg/4UfP5cfBE7>
- GitHub: <https://github.com/pola-rs/polars>

Polars is a powerful data manipulation library that allows you to perform a wide range of operations on your data. In this section, we have explored some of the most common operations and their corresponding Polars syntax. We have also discussed some of the pitfalls of the rolling window API and provided solutions for each. We hope this section has been helpful to you and that you will be able to apply these concepts to your own data.