



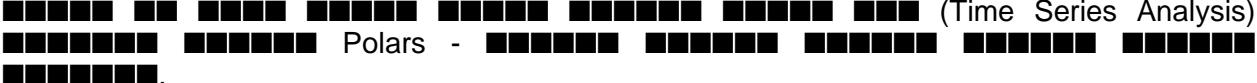
■■■ 9: Time Series Analysis

■■■■■: 02/11/2025



1. [REDACTED]
2. [REDACTED]
- 2.1 [REDACTED]
- 2.2 [REDACTED]
- 2.3 [REDACTED]
- 2.4 [REDACTED]
- 2.5 [REDACTED]
- 2.6 [REDACTED]
3. [REDACTED] (Rolling Windows)
 - 3.1 [REDACTED]
 - 3.2 [REDACTED]
 - 3.3 [REDACTED]
 - 3.4 [REDACTED]
4. [REDACTED] (Resampling)
 - 4.1 [REDACTED] (Downsampling)
 - 4.2 [REDACTED] (Upsampling)
 - 4.3 [REDACTED]
 - 4.4 [REDACTED]
5. [REDACTED]

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. [REDACTED]

[REDACTED] [REDACTED] [REDACTED] [REDACTED] [REDACTED] [REDACTED] [REDACTED] [REDACTED] [REDACTED] [REDACTED] [REDACTED] [REDACTED]. Polars [REDACTED]
[REDACTED] [REDACTED] [REDACTED], [REDACTED] [REDACTED] [REDACTED] [REDACTED].

2.3 [REDACTED] [REDACTED]:

[REDACTED]	[REDACTED]	[REDACTED]
.dt.year()	[REDACTED]	2024
.dt.month()	[REDACTED]	1-12
.dt.day()	[REDACTED]	1-31
.dt.hour()	[REDACTED]	0-23
.dt.weekday()	[REDACTED]	0-6
.dt.date()	[REDACTED]	2024-01-15
.dt.time()	[REDACTED]	14:30:00

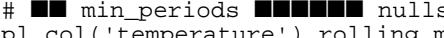
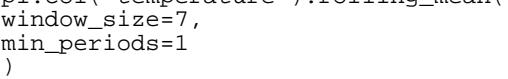
```
lf.select(  
    'datetime',  
    pl.col('datetime').dt.year().alias('year'),  
    pl.col('datetime').dt.month().alias('month'),  
    pl.col('datetime').dt.day().alias('day')  
)
```

3. (Rolling Windows)

Rolling windows are used to calculate moving statistics over a series of data points. They are particularly useful for time-series analysis where you want to analyze data in a sliding window fashion.

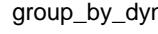
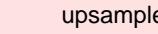
		
rolling_mean()		
rolling_min()		
rolling_max()		
rolling_sum()		
rolling_std()		
rolling_map()		

```
# 
pl.col('temperature').rolling_mean(3)

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

4. (Resampling)

Resampling  - .

			
Downsampling		group_by_dynamic	 → 
Upsampling		upsample	 → 
Interpolation		interpolate	 nulls

- Downsampling:

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

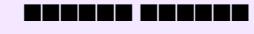
- Upsampling + Interpolation:

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

5. Functime

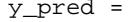
 . Functime            Polars.

 :

		
linear_model		
lightgbm	LightGBM	
xgboost	XGBoost	
knn	K-Nearest Neighbors	
auto_lightgbm	AutoML	

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

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

# 
y_pred = forecaster.predict(fh=3)

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

 :

		
MASE	Mean Absolute Scaled Error	 ()
SMAPE	Symmetric MAPE	 0
MAE	Mean Absolute Error	
RMSE	Root Mean Squared Error	

6. Polars

Polars is a fast, column-oriented DataFrame library for Python. It is designed to be easy to use and efficient, making it a great choice for data processing tasks.

1	df.set_sorted()	
2	df.head().collect()	
3	df.rolling_map	
4	upsample DataFrame (LazyFrame)	
5	df.min_periods=1 nulls	
6		
7		
8	df.maintain_order=True group_by	

(Pitfalls):

datetime is not sorted	df.set_sorted('datetime')
upsample requires DataFrame	df.collect()
df nulls	min_periods=1
rolling_map	
	replace vs convert
-String	-Datetime

Resources:

- Official Website: <https://pola-rs.github.io/polars/>
- API Reference: <https://pola-rs.github.io/polars/py-polars/html/reference/>
- Discord: <https://discord.gg/4UfP5cfBE7>
- GitHub: <https://github.com/pola-rs/polars>

That's all for now! See you in the next video!