Timeseries Forecasting & Meta-Learning

William Trouleau, Florian Ravasi, Sebastien Gay, Gael Grosch, Dennis Bader, Antoine Madrona

Darts

Unit8™ | SDS2023

Florian Ravasi

- MSc Data Science - EPFL
- Data Scientist @ Unit8

Antoine Madrona

- MSc Life Science Engineering - EPFL
- Data Engineer @ Unit8

Sebastian Gay

- BSc Physics and Probability - UniMelb
- Data Engineer @ Unit8

Dennis Bader

- MSc Mechanical Engineering - ETH
- Data Scientist @ Unit8
- Darts Lead

Gael Grosch

- MSc computer science - EPFL
- Data Scientist @ Unit8

William Trouleau

- PhD computer science - EPFL
- Data Scientist @ Unit8

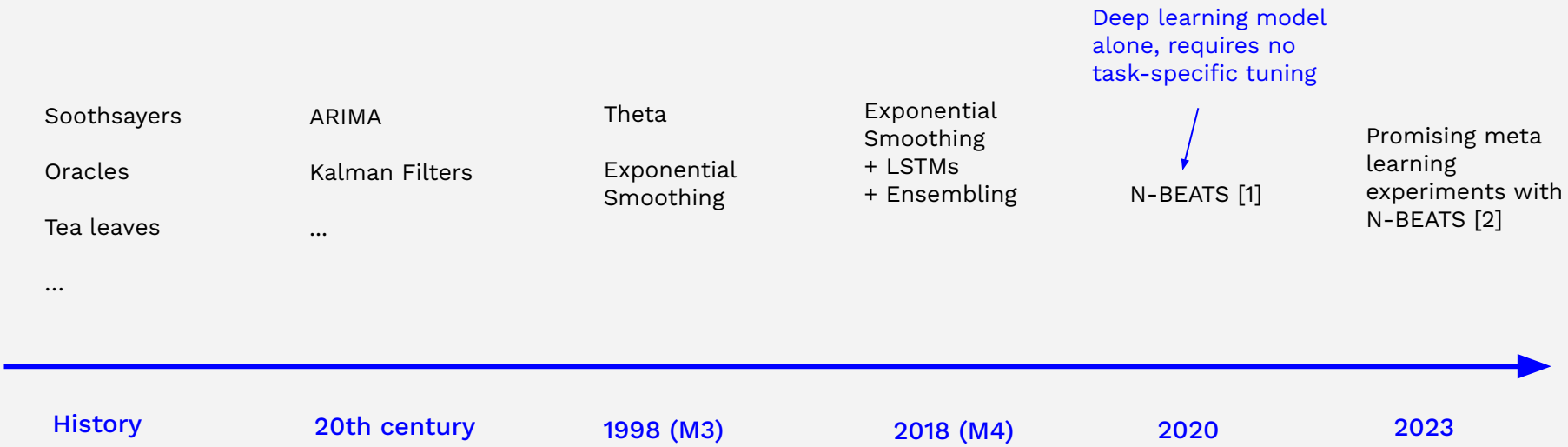
Workshop
Overview

Motivation

- Time series are everywhere
- In Forecasting models need to be handcrafted for each separate applications
- ML is making its entrance in the field
  - Need larger datasets
  - But many problems are still small data problems
- Meta-Learning - definition: applying a model trained on a task (or dataset) onto a different task (or dataset)
  - Open question: Can we use forecasting models learned on certain diverse datasets for other problems?
  - Comparable to what happened to NLP in recent years

Why Forecasting & Meta-learning?

A (brief) history of forecasting techniques...



Unit8

[1] Oreshkin, Boris N., et al. "N-BEATS: Neural basis expansion analysis for interpretable time series forecasting". *arXiv preprint arXiv:1905.10437*

[2] Oreshkin, Boris N., et al. "Meta-learning framework with applications to zero-shot time-series forecasting". *arXiv preprint arXiv:2002.02887*

Objectives

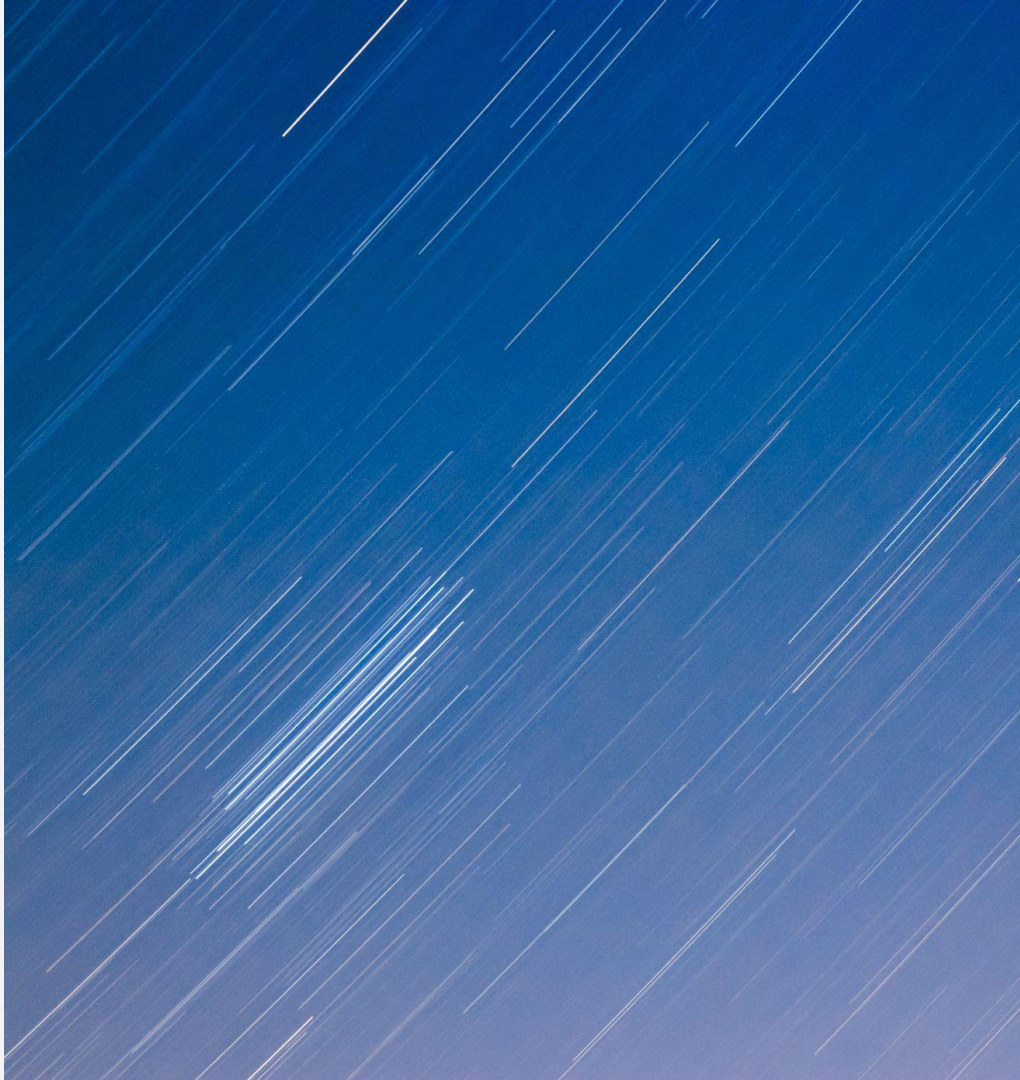
- Provide an introduction to forecasting with Darts
- Understanding how to forecast multiple time series
- Understanding how ML can be applied to forecasting
- Provide some exposure to meta learning / transfer learning for forecasting
- Build models that get performance close to winning famous M3/M4 forecasting competitions
- Experimenting, playing and having fun!
- ~~Dive in the details of how the models work~~

Pre-requisites

- Basic knowledge of Python for data science
- Basic knowledge of machine learning
- A way to run the provided notebooks, for instance:
  - **[recommended]** Colab environment (https://research.google.com/colaboratory/)
  - Your own computer
    - If possible: Nvidia GPU
    - Fresh Python 3.8+ venv

Agenda

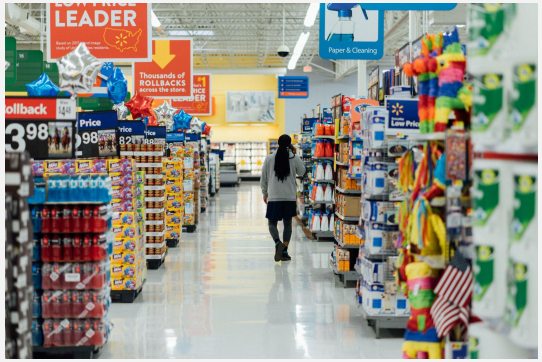
1. Workshop overview
2. Introduction to forecasting with Darts
3. Introduction to ML-based forecasting
4. <hands on>
5. Meta-learning overview
6. <hands on>
7. Take home messages



Short intro to Forecasting

Why Forecasting?

Retail



How can we ensure that the consumer demand for every product is met without overspending and creating waste?

Energy

How much energy should we produce?

How much energy will we produce given a certain setup?

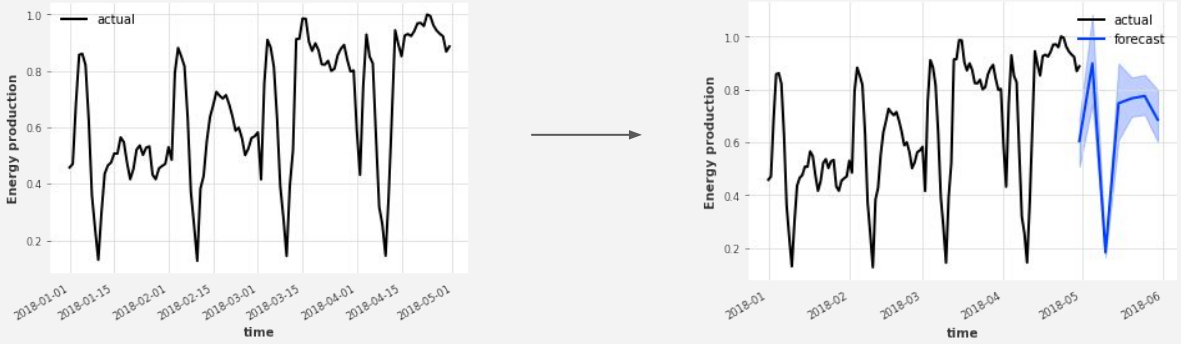
What will the electricity price be tomorrow?

Telecommunications

Will our current infrastructure support the traffic in the near and far future?

Unit8

Time Series Forecasting



Future = *f*(history + external data)

Consideration: Do we have a valuable signal in the data?

Unit8

Darts 

Darts is a Python library for easy manipulation and forecasting of time series.

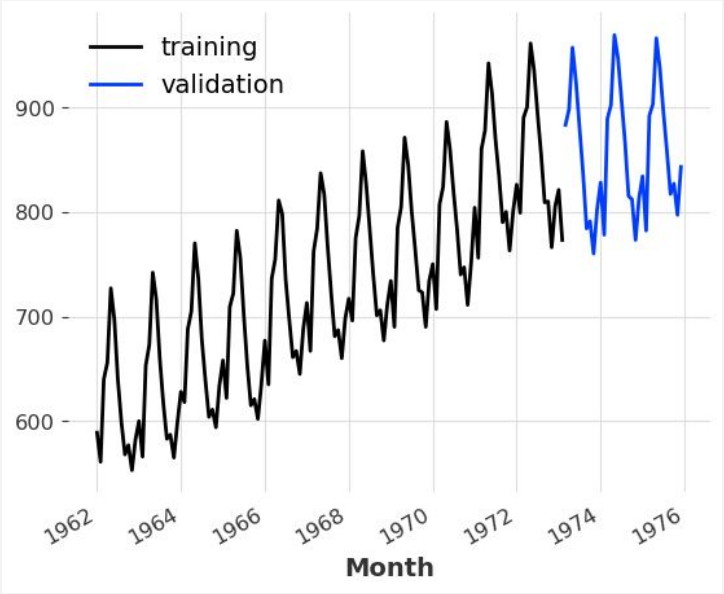
- Immutable `TimeSeries` class as basic building block.
- Unified `fit()`, `predict()` interface.
- Classical models & state-of-the-art ML approaches for forecasting and anomaly detection
- User-friendliness: intuitive API and reasonable defaults.

Unit8

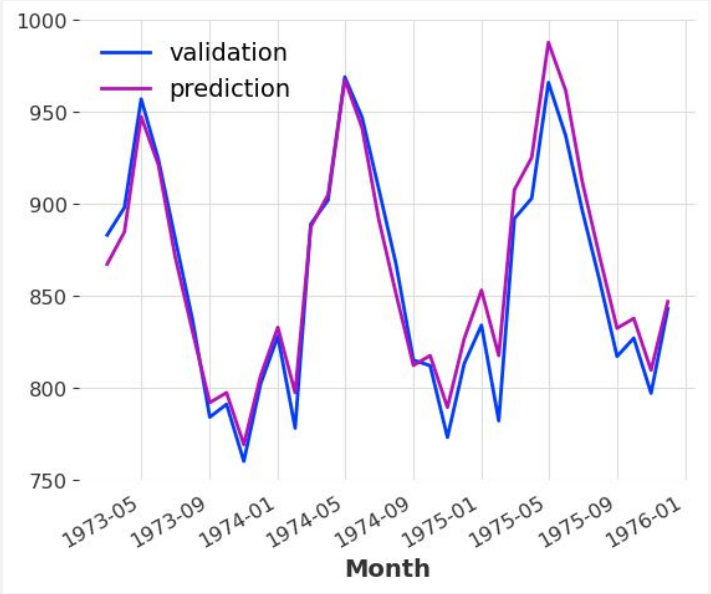
https://github.com/unit8co/darts

A simple Forecasting example (one series)

Goal



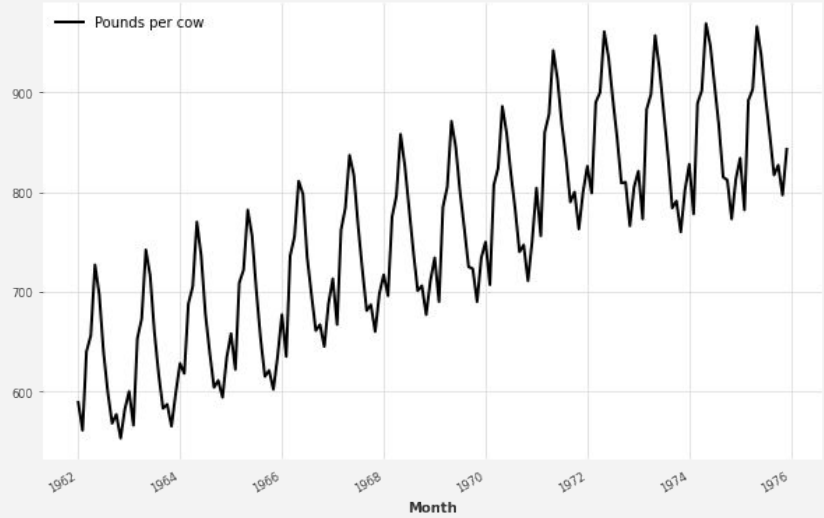
Goal



TimeSeries | Processing | Forecasting | Evaluating

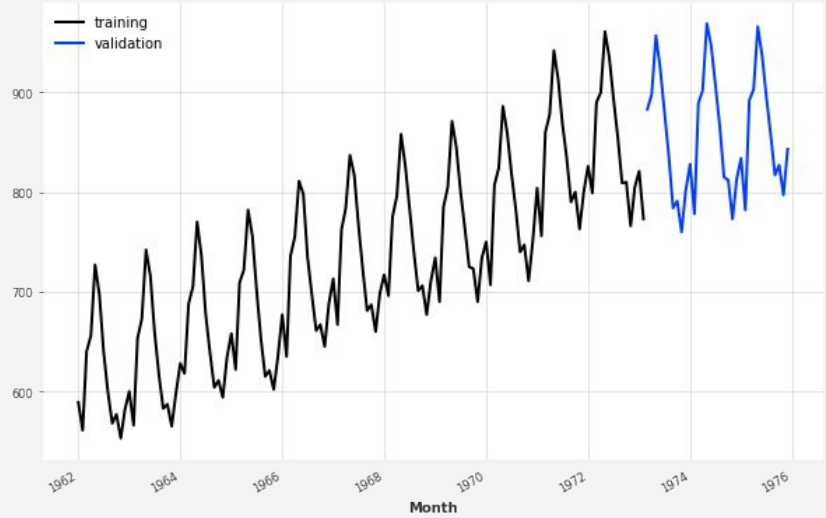
The TimeSeries object

```python
from darts import TimeSeries

# Read time series
series = TimeSeries.from_csv('milk.csv', time_col='Month')
series.plot()
```



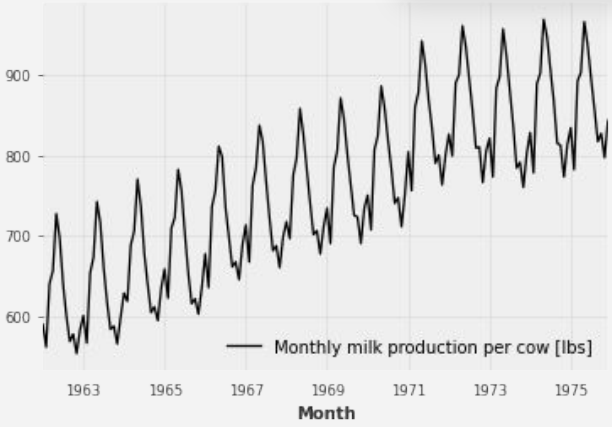
Training / validation split

```python
from darts import TimeSeries

# Read time series
series = TimeSeries.from_csv('milk.csv', time_col='Month')
series.plot()

# Split train/val
train, test = series.split_after(0.8)
```

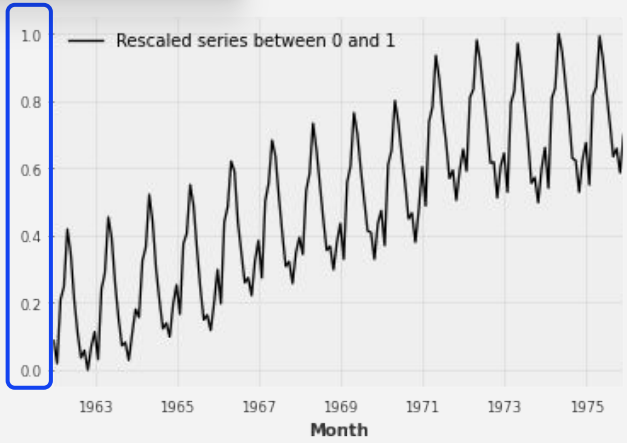


Normalizing TimeSeries

```
scaler = Scaler()
scaler.fit(series)  # Learn min and max from the data
rescaled = scaler.transform(series)
```



Normalizing

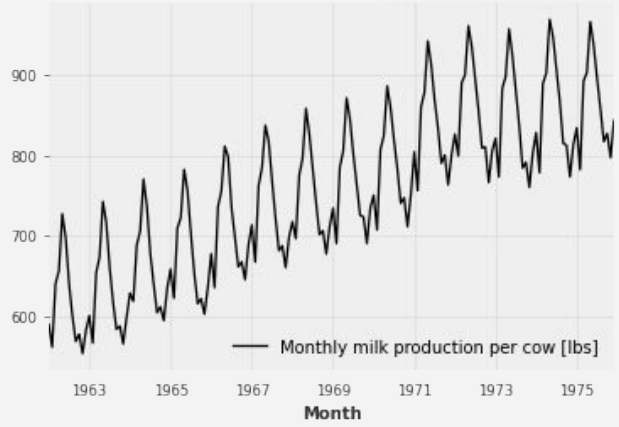


Unit8

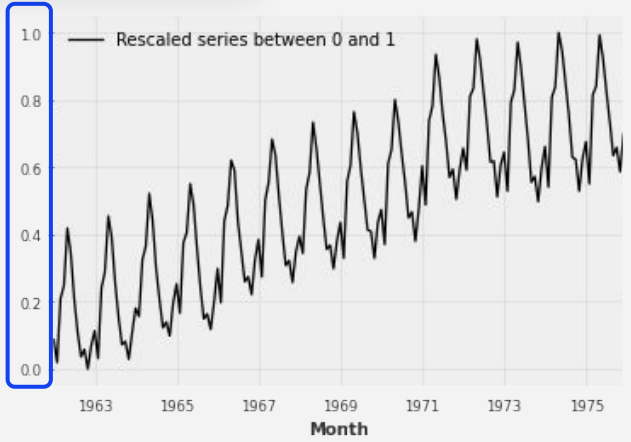
TimeSeries | **Processing** | Forecasting | Evaluating

Normalizing TimeSeries

```
original_series = scaler.inverse_transform(rescaled)
```



Re-scaling back

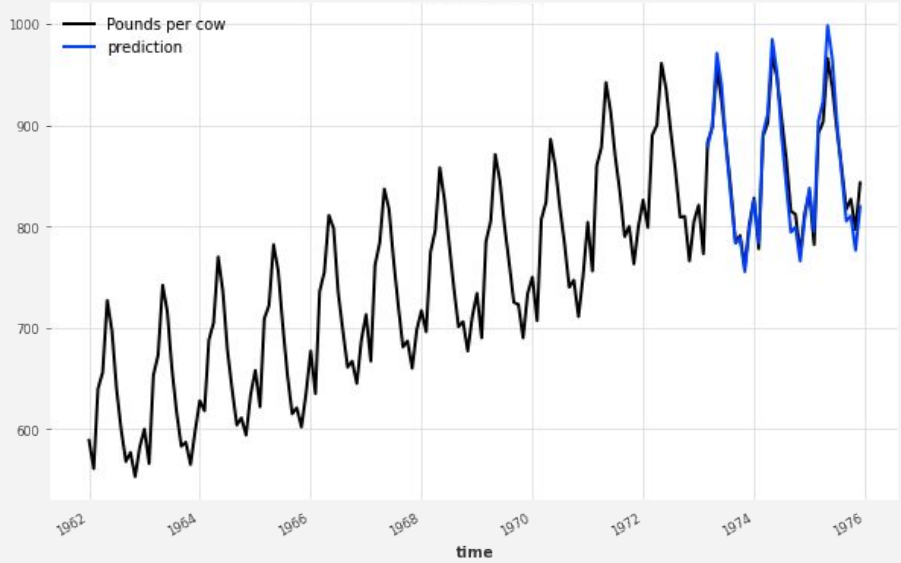


Unit8

TimeSeries | **Processing** | Forecasting | Evaluating

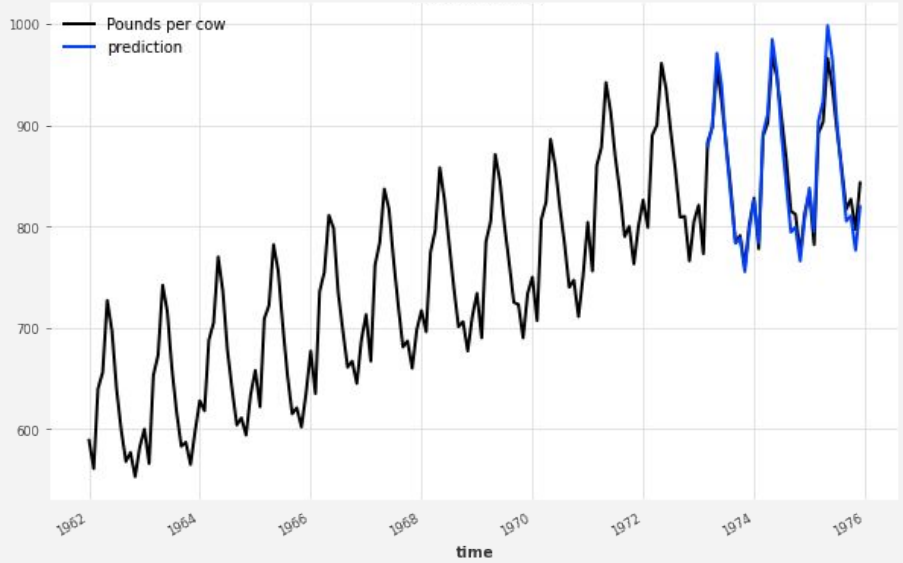
Forecasting – Exponential Smoothing

```python
from darts.models import ExponentialSmoothing

model = ExponentialSmoothing()
model.fit(train)
pred = model.predict(n=len(test))
```



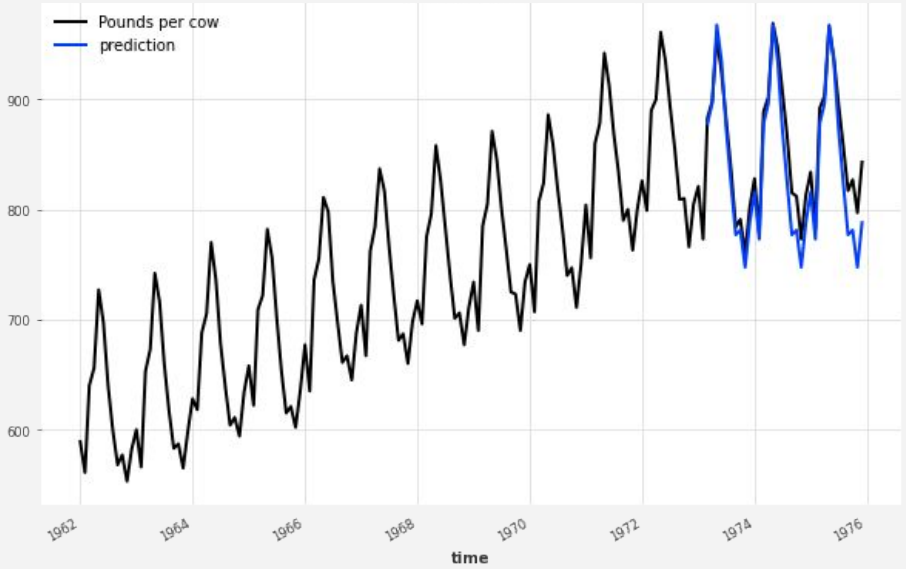
Forecasting – Theta

```python
from darts.models import Theta

model = Theta()
model.fit(train)
pred = model.predict(n=len(test))
```



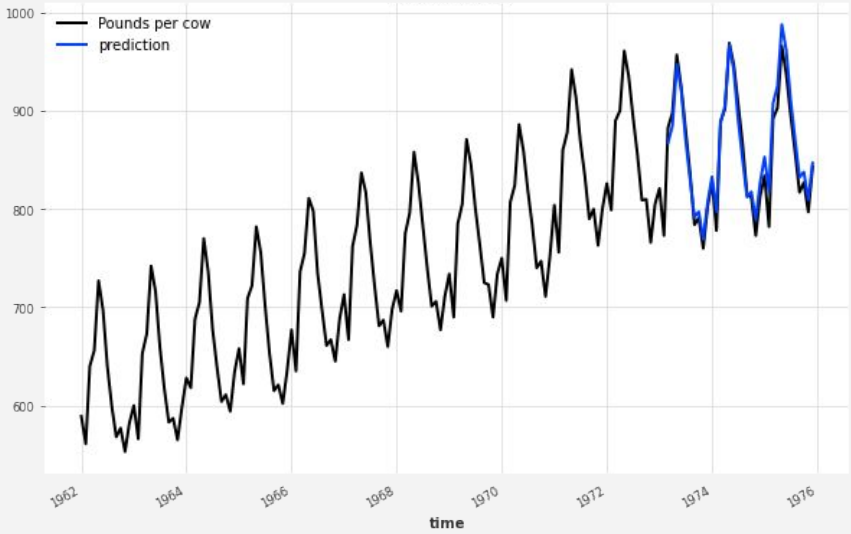
Specifying parameters

```
from darts.models import Theta

model = Theta(theta=1)
model.fit(train)
pred = model.predict(n=len(test))
```

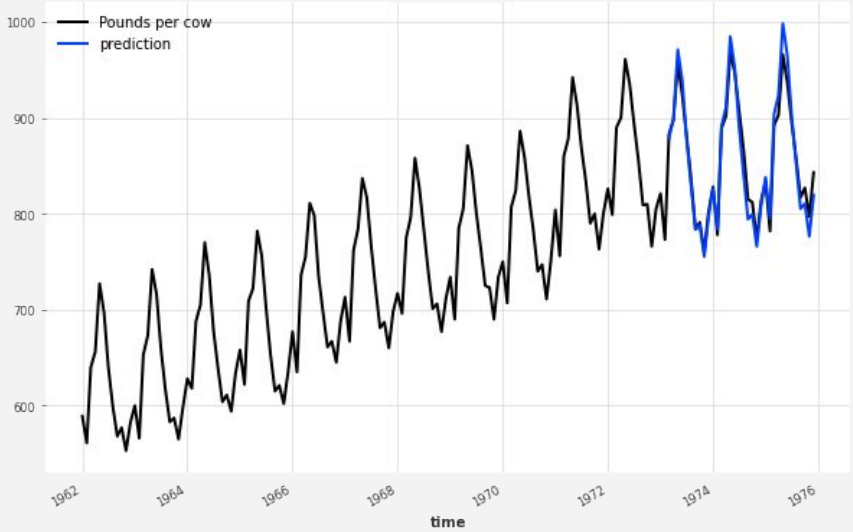


Evaluating predictions – Which one is better?

Exponential Smoothing



Theta



Metrics

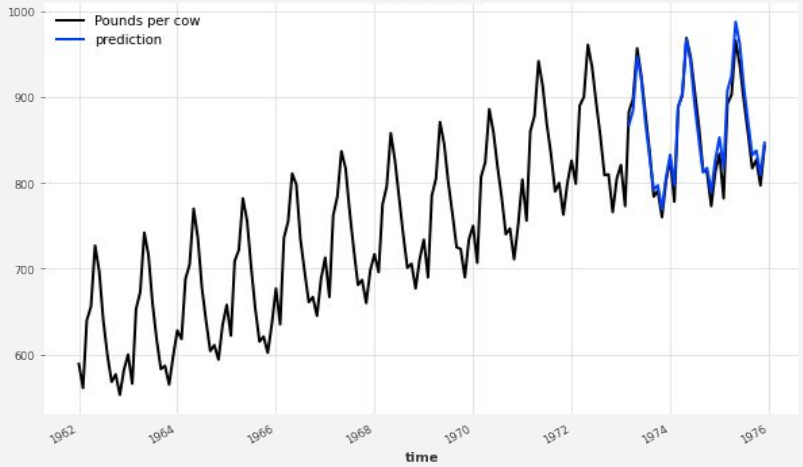
Many different scores can be computed – Darts lets you import the one you need.

```python
from darts.metrics import mape

score = mape(test, pred)
```

```python
from darts.metrics import mase

score = mase(test, pred, train)
```

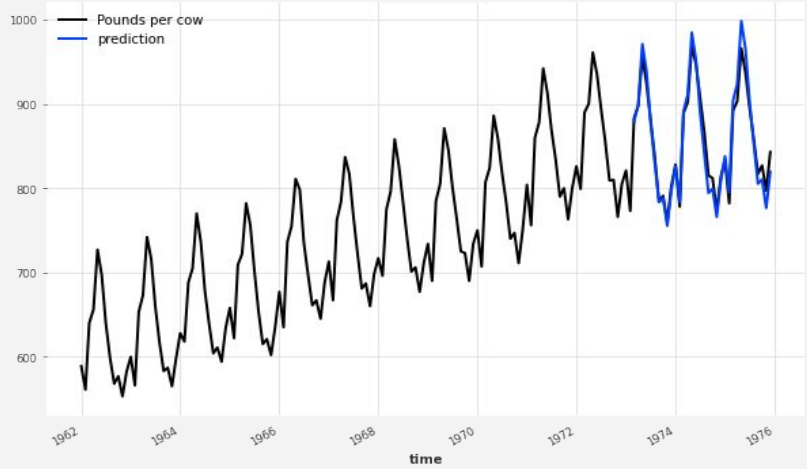
Which one is better?

Exponential Smoothing



MAPE: ~1.39%

Theta



MAPE: ~1.28%

TimeSeries → Processing → Forecasting → **Evaluating**

Our Metric: sMAPE

$$\text{MAPE} = 100\%/N \sum_{t=1}^{N} \frac{|A_t - F_t|}{|A_t|}$$

⬅️ *Difference between Actual and Forecast*

⬅️ *Normalize by taking Actual*

$$\text{sMAPE} = 100\% * 2/N \sum_{t=1}^{N} \frac{|A_t - F_t|}{|A_t| + |F_t|}$$

⬅️ *Normalize by taking sum of Actual and Forecast*

- sMAPE between 0% and 200%
- MAPE degenerates into positive infinity as soon as any of the actual values is zero

Recap

End-to-end example

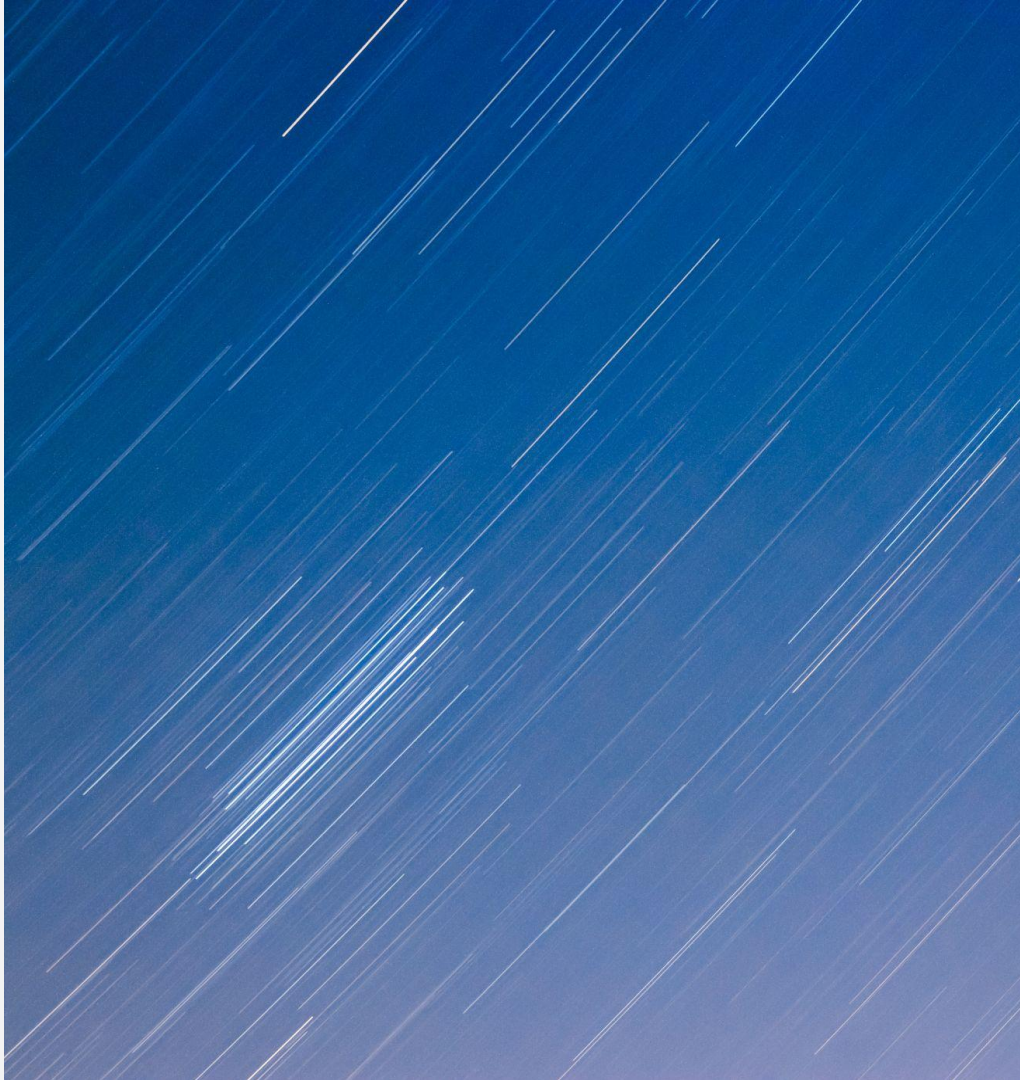
```python
from darts import TimeSeries
from darts.models import Theta  # or others
from darts.metrics import smape
from darts.dataprocessing.transformers import Scaler

# read series, split train/val
series = TimeSeries.from_xxx(...)
train, test = series[:-HORIZON], series[-HORIZON:]

# scale train and val
scaler = Scaler()
train = scaler.fit_transform(train)
test = scaler.transform(test)

# build model with some parameters
model = Theta(...)

# train model
model.fit(train)

# get a prediction
pred: TimeSeries = model.predict(n=HORIZON)

# compute sMAPE error
smape_val = smape(pred, test)
```

Loading, splitting and scaling is taken care of for you in the notebook

Unit8

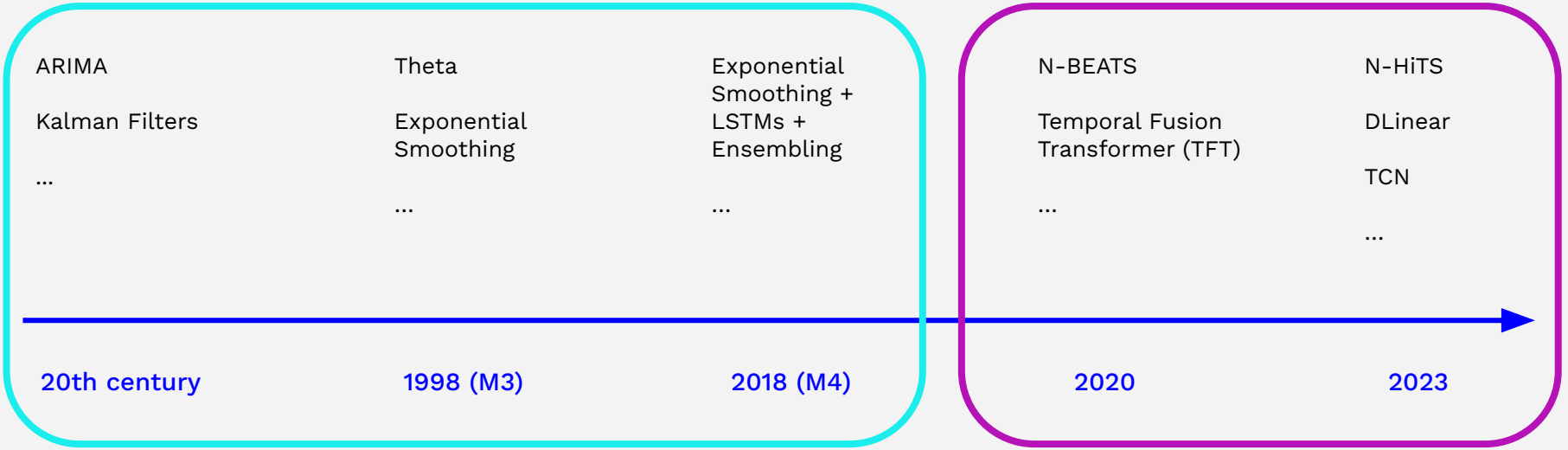
Agenda

1. Workshop overview
2. Introduction to forecasting with Darts
3. **Introduction to ML-based forecasting**
4. <hands on>
5. Meta-learning overview
6. <hands on>
7. Take home messages



Machine Learning
for Forecasting

Back to the forecasting techniques history



Carefully selected, series specific, parameters
→ **“Local models”, trained with one series**

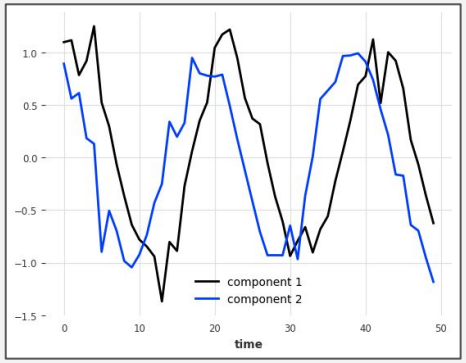
“Generic” architecture
→ **“Global model”, trained with one or more series**

Unit8

Multivariate -vs- multi-series

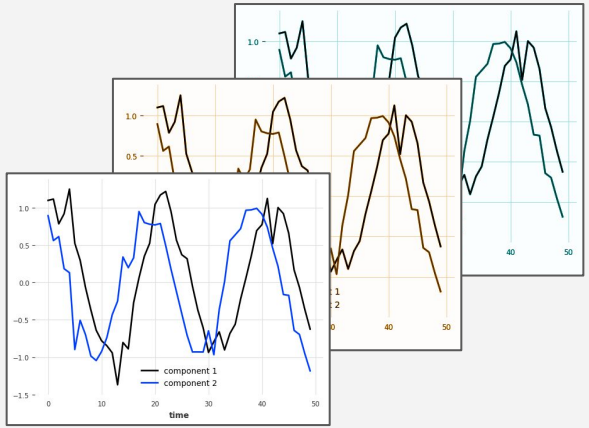
In Darts, one `TimeSeries` can be:

- Univariate - i.e. scalar values over time
- Multivariate - i.e., vector values over time



← one multivariate `TimeSeries` with two components

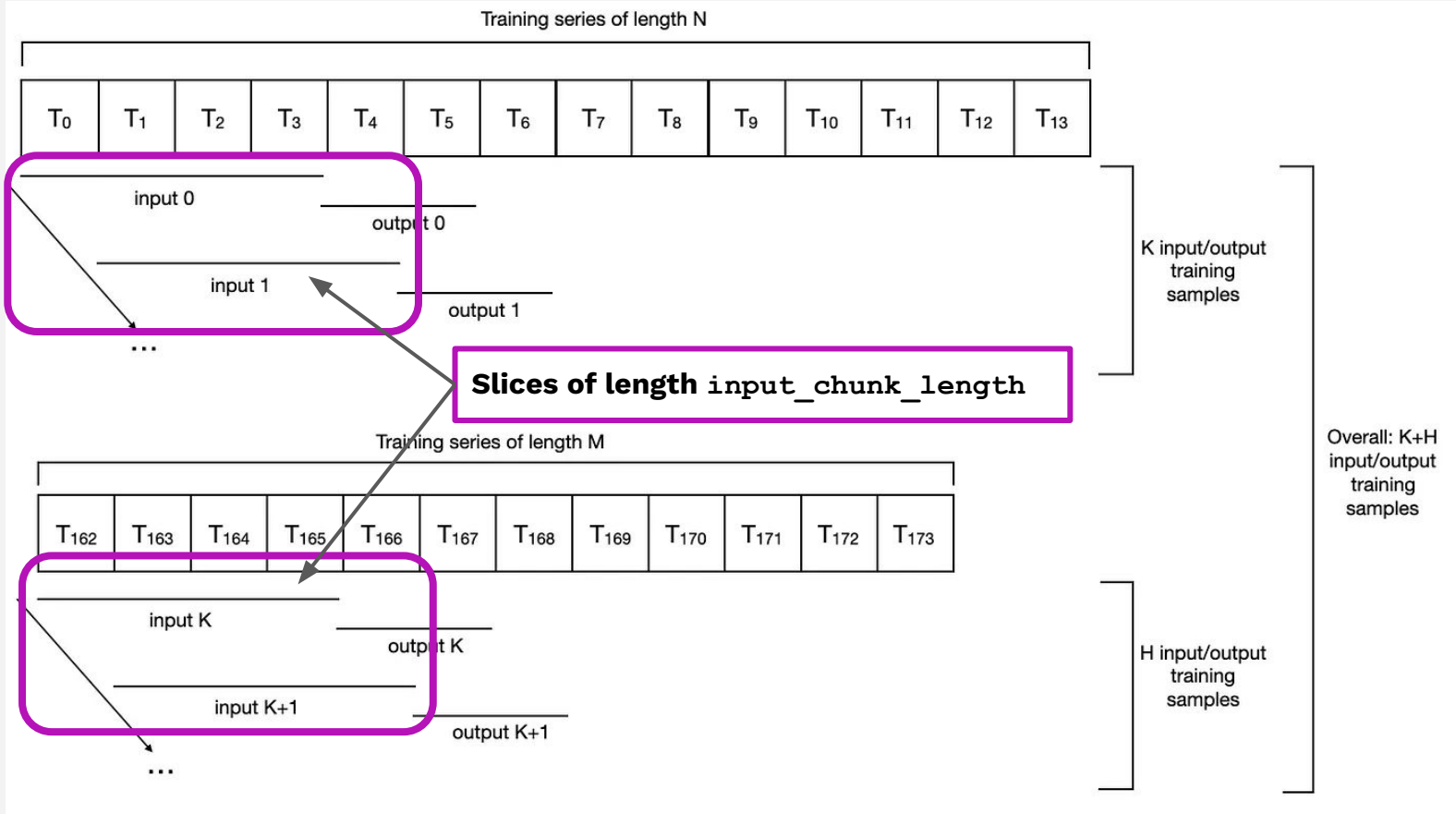
Multiple series : several series, either univariate or multivariate, with potentially different time axes, scales, trends, ...



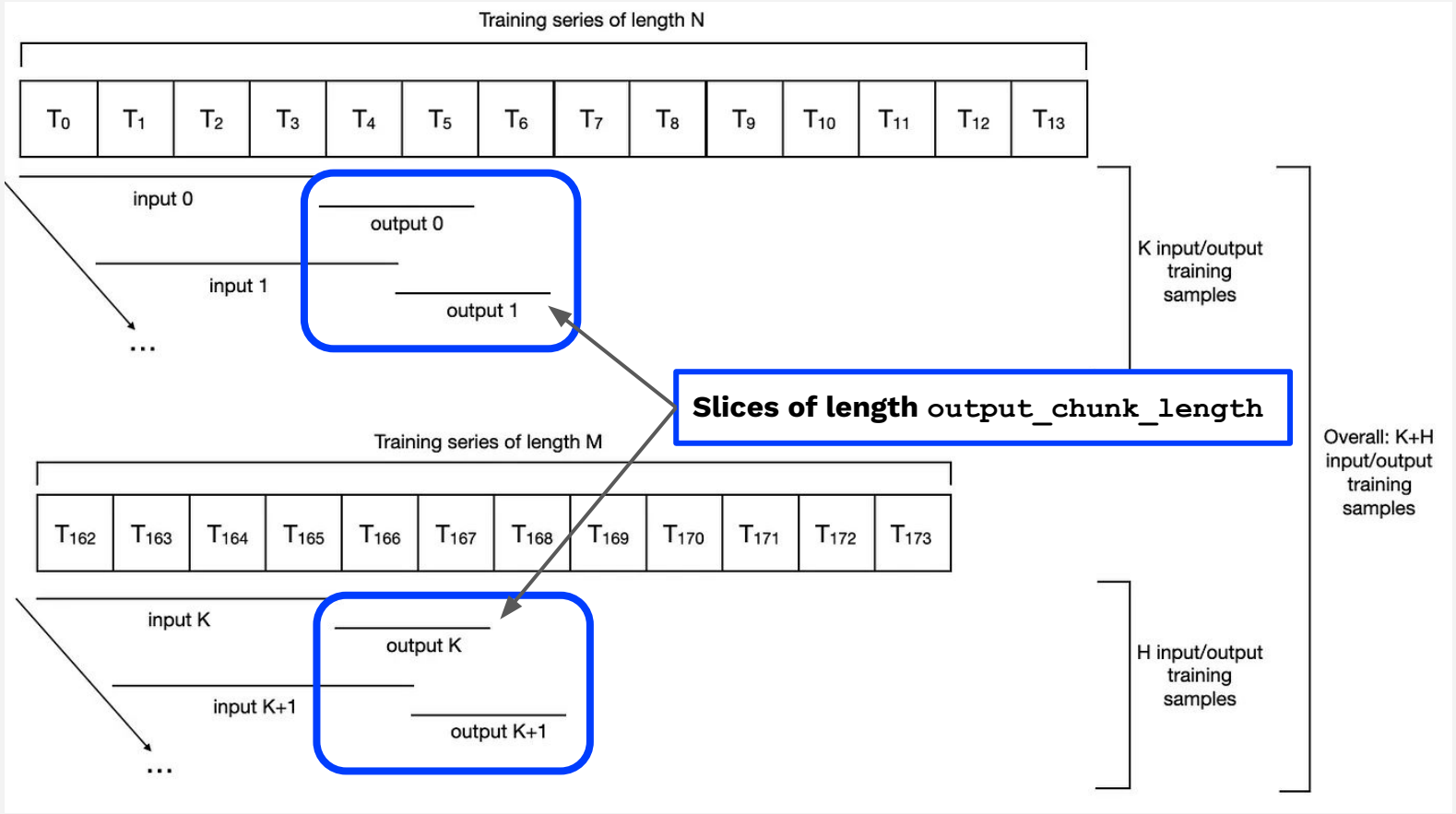
← multiple `TimeSeries`, each corresponding to a different phenomenon

Unit8

Slicing Multiple Series into a Training Set

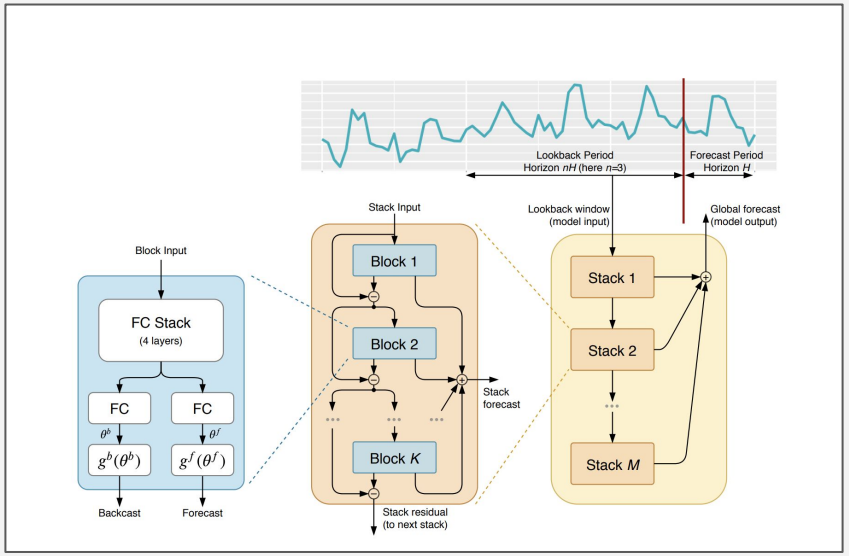


Slicing Multiple Series into a Training Set



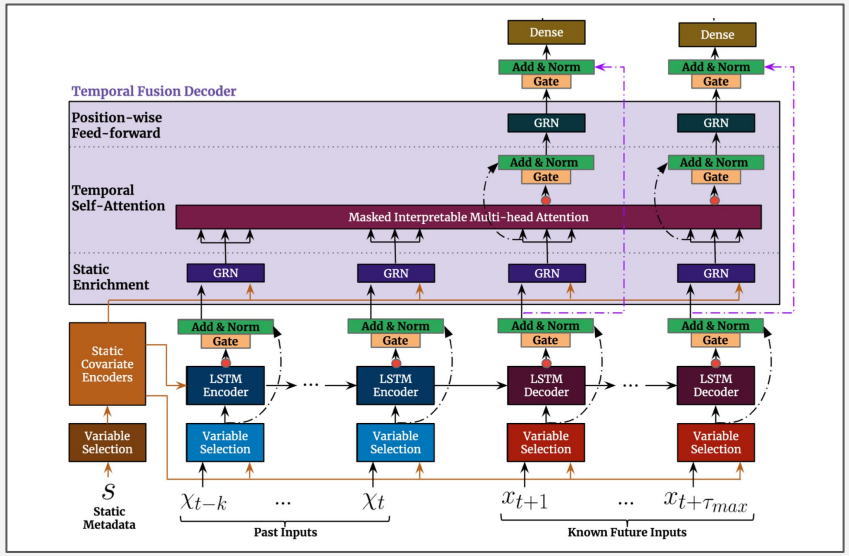
How are these complex architectures trained?

N-BEATS



Input : Tensor [k+h, input_chunk_length]
Target : Tensor [k+h, output_chunk_length]

Temporal Fusion Transformer



Input : Tensor [k+h, input_chunk_length]
Target : Tensor [k+h, output_chunk_length]

Unit8

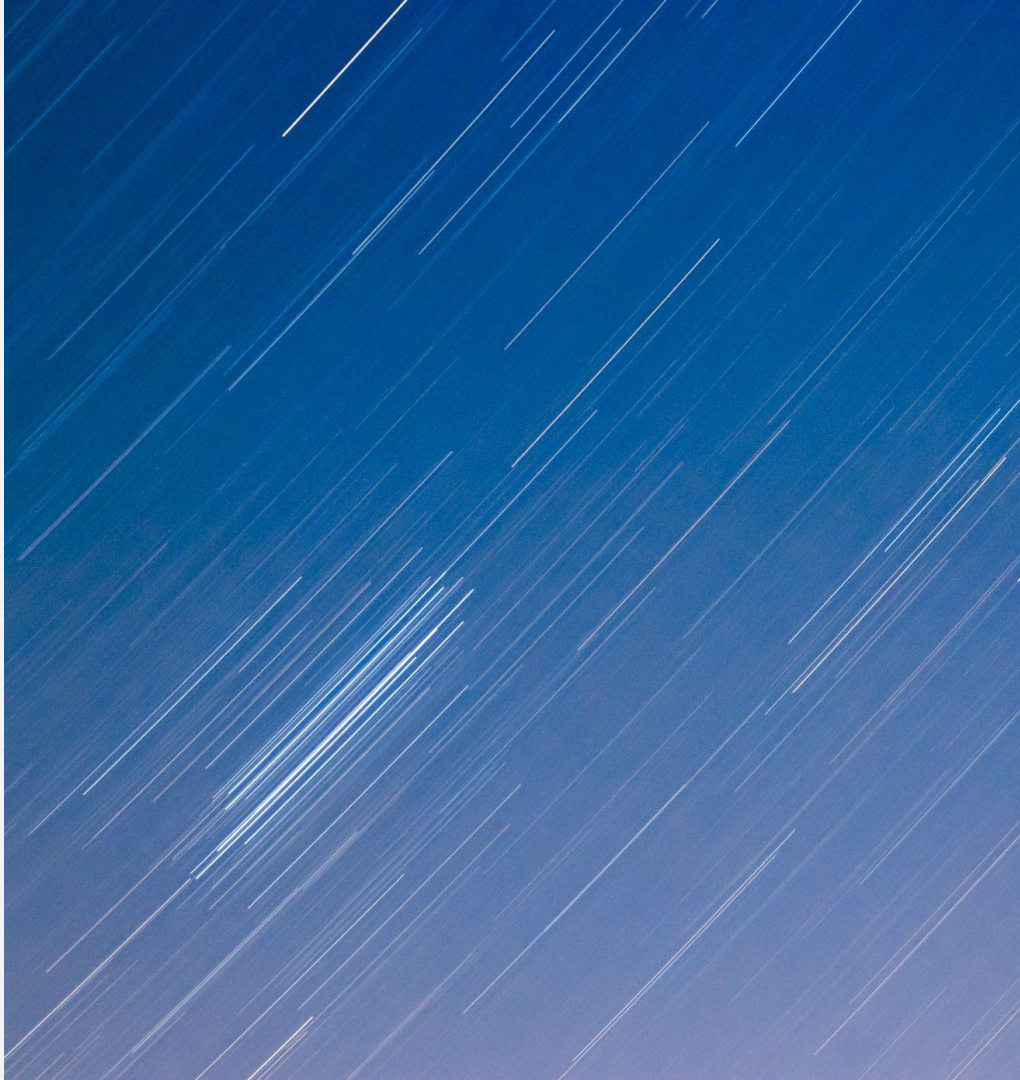
In Practice

Fitting and forecasting on collections of series

```python
from darts.models import SomeGlobalModel

model = SomeGlobalModel(...)
model.fit(train_series)

# get one forecast TimeSeries per series to forecast:
preds: List[TimeSeries] = model.predict(n=HORIZON,
                                        series=train_series)
```

Agenda

1. Workshop overview
2. Introduction to forecasting with Darts
3. Introduction to ML-based forecasting
4. **<hands on>**
5. Meta-learning overview
6. <hands on>
7. Take home messages



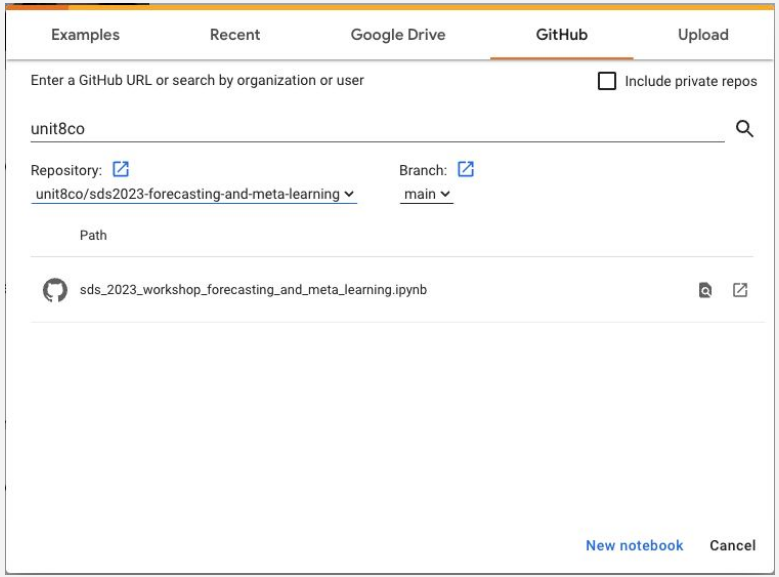
Hands-On Session 1

- Getting Started
- Forecasting multiple series one-by-one
- Global forecasting models to forecast airlines traffic

Setup: Google Colab

We recommend using Google Colab or your local machine for this workshop

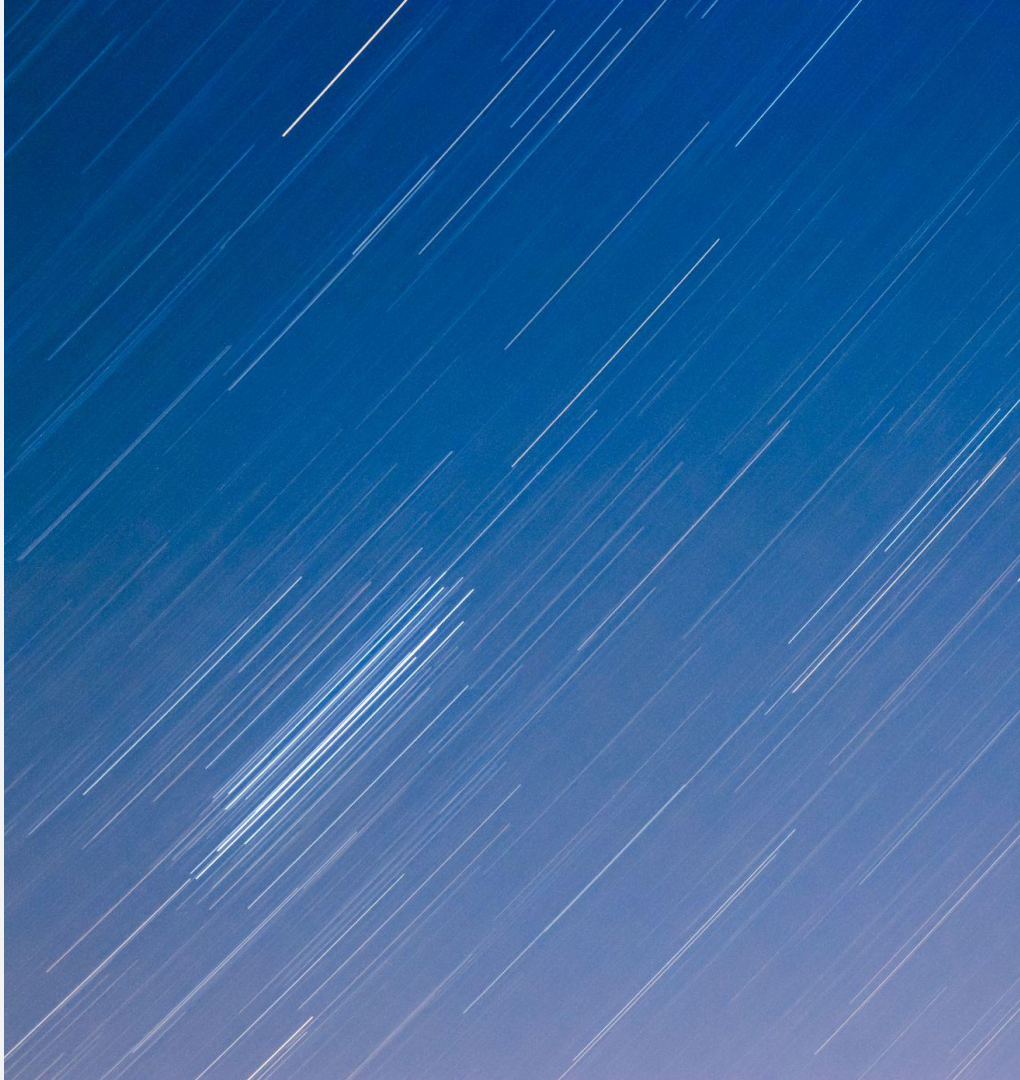
- **Go to: https://colab.research.google.com/**
- **"File -> Import notebook -> Github"**
- **Paste the following link: https://github.com/unit8co/sds2023-forecasting-and-meta-learning**
- **Load the relevant notebook 🎊**



Unit8

Agenda

1. Workshop overview
2. Introduction to forecasting with Darts
3. Introduction to ML-based forecasting
4. <hands on>
5. Meta-learning overview
6. <hands on>
7. Take home messages



Meta Learning Overview

Objective: given a set of “Tasks” build a model that can adapt to a new “Task”

“Task”:

- Training dataset
- Validation dataset
- A loss function

“Model”:

- Regular Parameters: Θ
- Meta-Parameters: Φ

Parameters trained using 2 loops:

- **Inner Loop**: given task i create a set of optimised model parameters Θ_i
- **Outer Loop**: based on the Θ_i seen till now update the meta parameters Φ

[1] Oreshkin, Boris N., et al. "Meta-learning framework with applications to zero-shot time-series forecasting". *arXiv preprint arXiv:2002.02887*

Meta Learning in this Workshop

Question: Can we train a model on a “representative” dataset of time series (M4) and then use it to forecasts time series coming from other datasets (e.g. airline traffic)?

Why would we care?

- “Zero-shot” forecasting - no training/engineering/tuning required
- Avoid “cold start” issues on problems with not enough data
- Inference speed

M-Competitions

40 Years of M-Competitions

	Year	# Timeseries	Particularity
M-Competition	1982	1001	- First Competition - Simple vs Complex
M2-Competition	1993	29	- Larger call - 4 actual company data
M3-Competition	2000	3'003	- larger - NN based methods
M4-Competition	2020	100'000	- Much larger set of real life time series - All methods welcomed = ML vs Stat
M5-Competition	2021	42'000	- Kaggle platform - Walmart data
M6-Competition	2022	100	- 50 S&P500 stocks / 50 ETFs - Forecast + investment decision

M4-Competition Dataset

48,000 “diverse” monthly series

Unit8

Spyros Makridakis et al., “The M4 Competition: 100,000 time series and 61 forecasting methods”, International Journal of Forecasting, 2020.

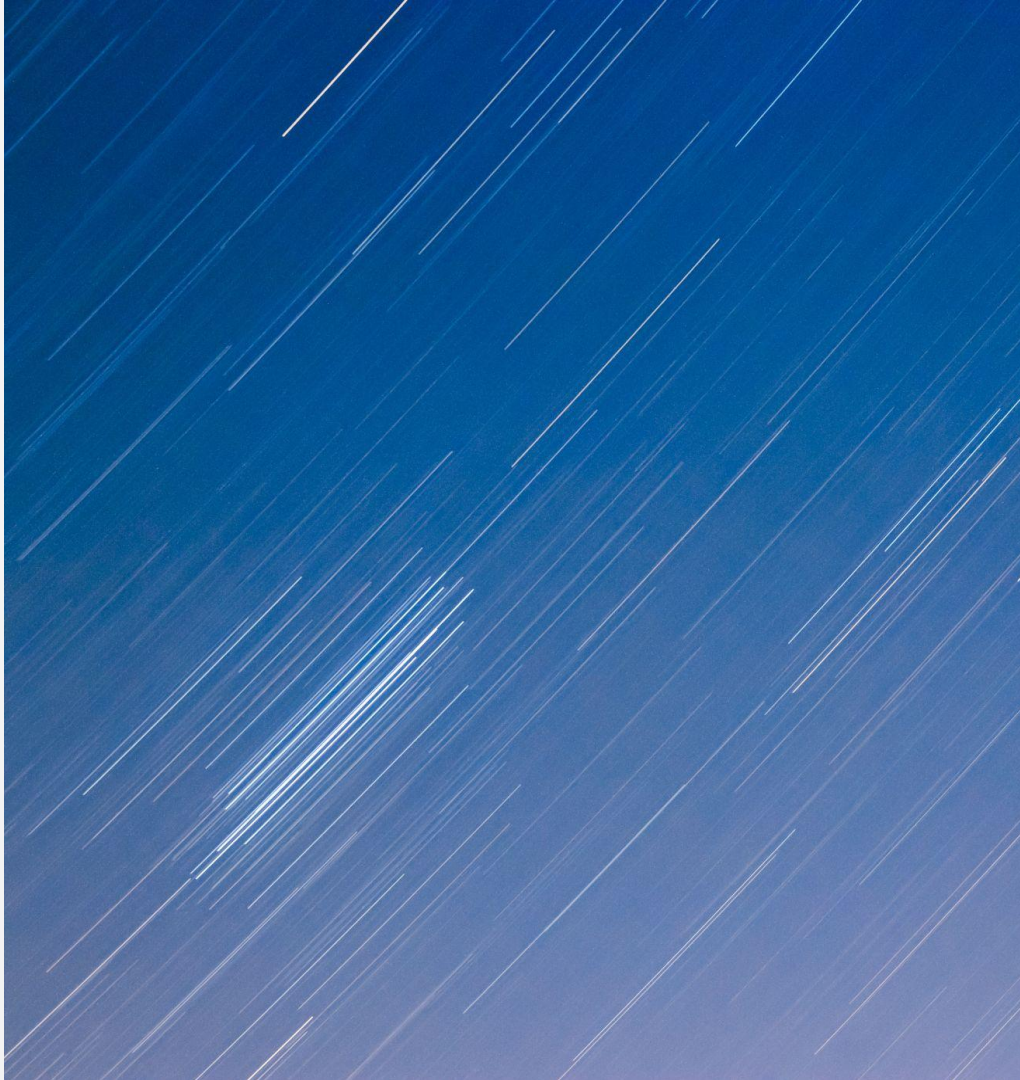
Recap

Training on one dataset and testing on another

```python
from darts.models import SomeGlobalModel

# train_series is a Sequence[TimeSeries] (e.g., a list)
model = SomeGlobalModel(...)
model.fit(train_series)

# get one forecast TimeSeries per series to forecast from another dataset
preds: List[TimeSeries] = model.predict(n=HORIZON,
                                        series=series_to_forecast)
```

Agenda

1. Workshop overview
2. Introduction to forecasting with Darts
3. Introduction to ML-based forecasting
4. <hands on>
5. Meta-learning overview
6. **<hands on>**
7. Take home messages

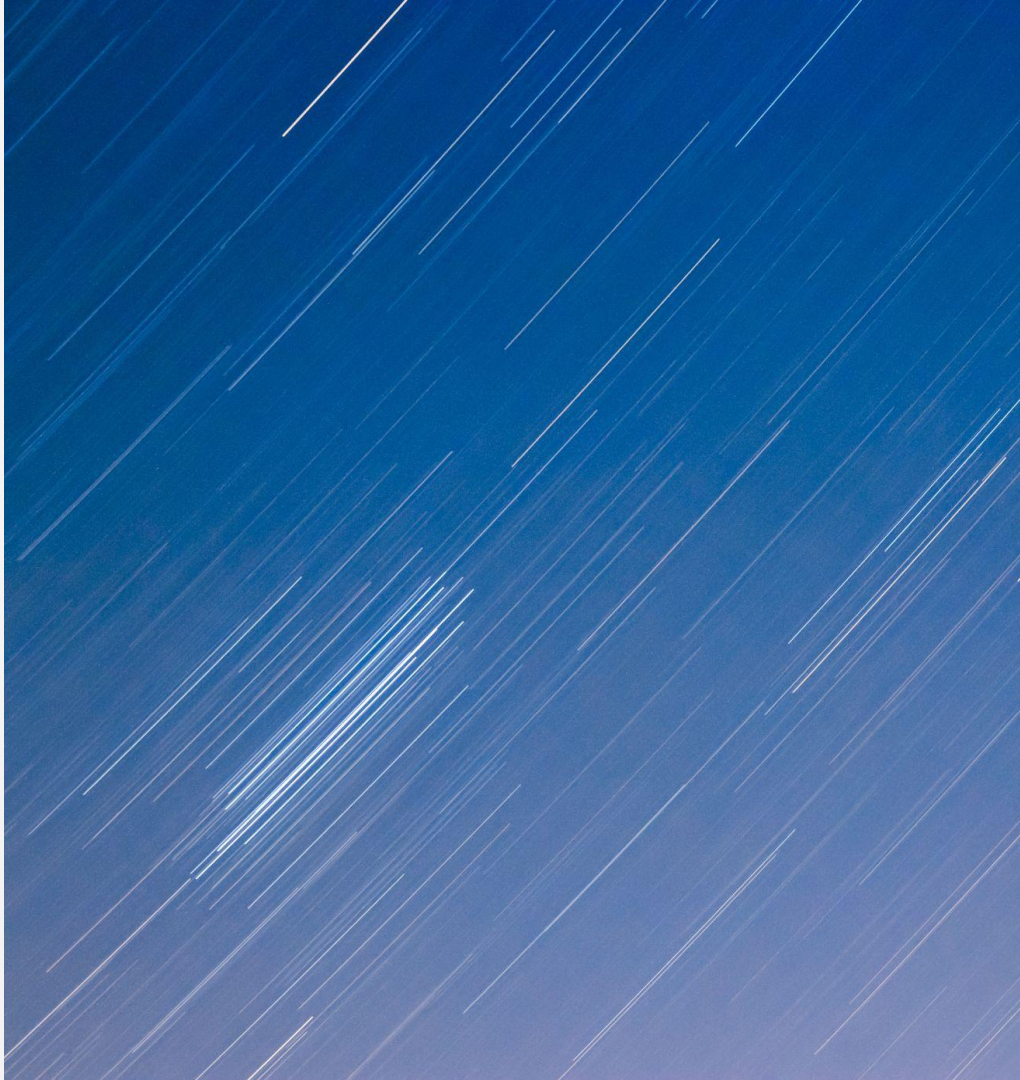


Hands-On Session 2

- Use a model trained on M4 dataset to forecast other series

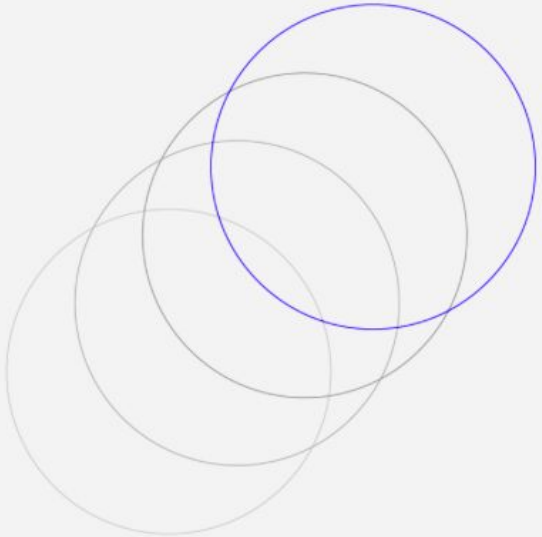
Agenda

1. Workshop overview
2. Introduction to forecasting with Darts
3. Introduction to ML-based forecasting
4. <hands on>
5. Meta-learning overview
6. <hands on>
7. Take home messages



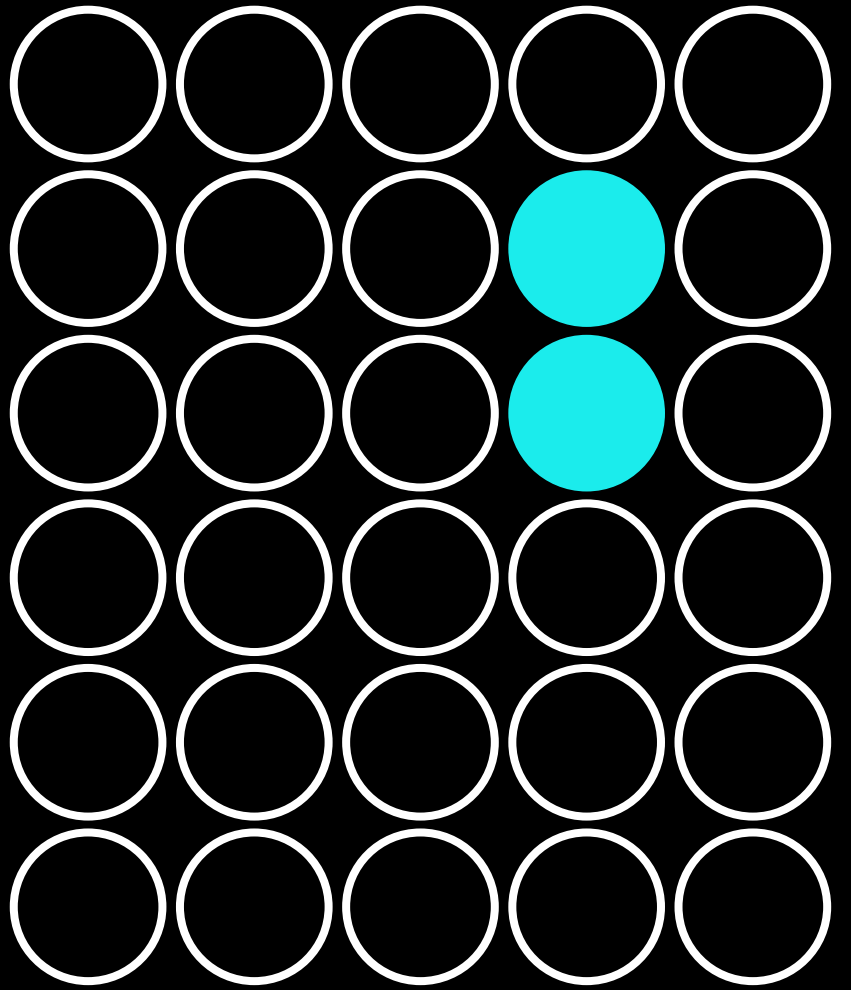
Take home messages

- Darts can be used to very quickly use and compare different models
- “Global models” represent the new way of fitting models on collections of series
- Transferring / meta-learning models from one dataset to another can in some cases reach performance comparable to other models but without requiring training



Unit8

unit8.com



thank you!

Additional Material

Some definitions

- ***Time series*** - sequence of values over time
  - ***Univariate time series*** - a sequence of scalars
  - ***Multivariate time series*** - a sequence of vectors. Not to be confused with a collection of multiple time series.
- ***Forecasting*** - making predictions about future values of some series
- ***Local forecasting model*** - a model trainable on one series to forecast this one series
- ***Global forecasting model*** - a model trainable on multiple series, to forecast these or other series
- ***Meta learning*** - applying a model trained on a task (or dataset) onto a different task (or dataset)

Code
Snippets

Train like in the NBEATs paper with HorizonBasedDataset

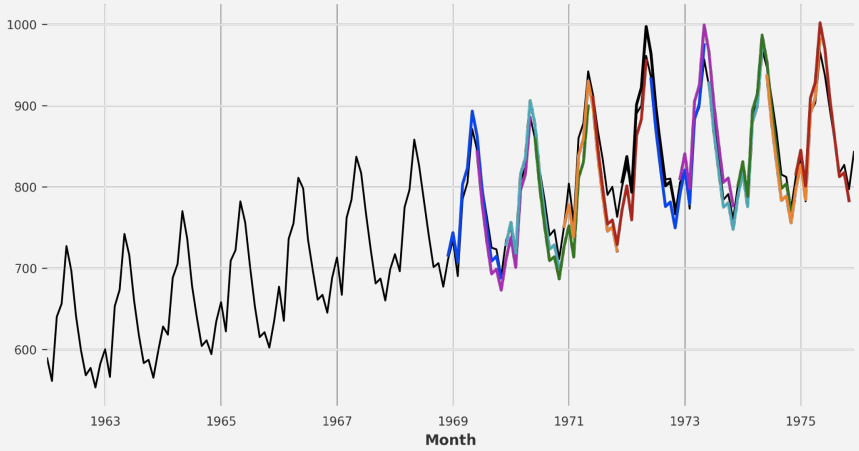
```
train_ds = HorizonBasedDataset(all_train_series, output_chunk_length=12, lh=(1, 3), lookback=7)
val_ds = HorizonBasedDataset(all_val_series, output_chunk_length=12, lh=(1, 3), lookback=7)
print('training set length: {}'.format(len(train_ds)))

_ = nbeats_model.fit_from_dataset(
    train_ds,
    val_dataset=val_ds,
    ...
)
```

Unit8

https://unit8co.github.io/darts/generated_api/darts.utils.data.horizon_based_dataset.html

Historical simulations using Darts

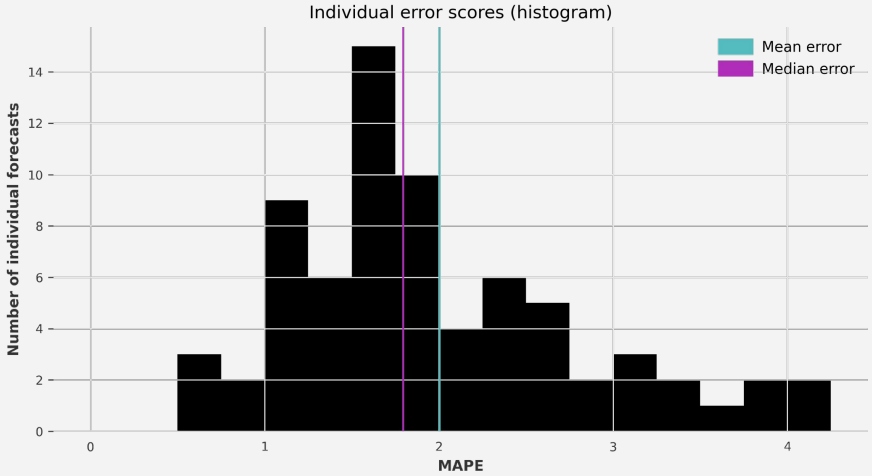
```
forecasts = model.historical_forecasts(
    series=series,
    start=0.5,
    forecast_horizon=12,
    stride=6,
    last_points_only=False
)
```



Unit8

Backtesting models on historical data

```python
import numpy as np

backtest_errors = model_es.backtest(
    series=series,
    start=0.5,
    forecast_horizon=12,
    stride=6,
    last_points_only=False,
    metric=mape,
    reduction=np.mean
)
```



Unit8

Early Stopping for Pytorch models

```python
early_stopper = EarlyStopping('val_loss', min_delta=0., patience=3, verbose=True)

nbeats_model = NBEATSModel(
    ...
    pl_trainer_kwargs={"enable_progress_bar": True,
                       "accelerator": "gpu",
                       "gpus": -1,
                       "auto_select_gpus": True,
                       "callbacks": [early_stopper]},
    ...
)


train_ds = HorizonBasedDataset(all_train_series, output_chunk_length=OUT_LEN, lh=LH, lookback=LOOKBACK)
val_ds = HorizonBasedDataset(all_val_series, output_chunk_length=OUT_LEN, lh=LH, lookback=LOOKBACK)

_ = nbeats_model.fit_from_dataset(
    train_ds,
    val_dataset=val_ds,
    num_loader_workers=2,
    epochs=NUM_EPOCHS
)
```

Unit8

https://pytorch-lightning.readthedocs.io/en/stable/extensions/generated/pytorch_lightning.callbacks.EarlyStopping.html

Custom Loss in Pytorch (eg. sMAPE)

```python
class SmapeLoss2(nn.Module):
    """
    sMAPE loss as defined in https://robjhyndman.com/hyndsight/smape/ (Makridakis 1993)
    """
    def __init__(self):
        super().__init__()

    def forward(self, inpt, tgt):
        num = torch.abs(tgt - inpt)
        with torch.no_grad():
            denom = torch.abs(tgt) + torch.abs(inpt)
        return torch.mean(_divide_no_nan(num, denom))

...

nbeats_model = NBEATSModel(
    ...
    loss_fn=SmapeLoss2(),
    ...
)
```

https://pytorch.org/docs/stable/generated/torch.nn.Module.html

Unit8

Hyperparameter search using Gridsearch

```
parameters = {
    'theta': [0.5, 1, 1.5, 2, 2.5],
    'season_mode': [SeasonalityMode.MULTIPLICATIVE,
                    SeasonalityMode.ADDITIVE]
}

best_model, best_parameters = Theta.gridsearch(
    parameters=parameters,
    series=training,
    start=0.5,
    forecast_horizon=12
)
```

Unit8

https://optuna.org/#code_examples

Hyperparameter search using Optuna

```python
import optuna

def objective(trial):
    # Configure parameters
    params = {}
    for param in [YOUR_CONTINUOUS_HYPERPARAMETERS]:
        params[param] = trial.suggest_uniform(param, bounds[param][0], bounds[param][1])

    for param in [YOUR_INT_HYPERPARMETERS]:
        params[param] = trial.suggest_int(param, bounds[param][0], bounds[param][1])

    seasonality = ['additive', 'multiplicative']
    params['seasonality_mode'] = seasonality[trial.suggest_int('seasonality_mode',
                                                               bounds['seasonality_mode'][0],
                                                               bounds['seasonality_mode'][1])]

    ....

    # Train
    m = YOUR_DARTS_MODEL(**params)
    m.fit(train_ts)
    pred = model.predict(val_ts)

    # Return the metric
    return smape(pred, val_ts)


study = optuna.create_study()
study.optimize(objective, n_trials=100)
```

Unit8

https://optuna.org/#code_examples

Probabilistic Forecasting in Darts

```
model = TCNModel(likelihood=GaussianLikelihoodModel(),
                 **kwargs)
model.fit(train, covariates)
pred = model.predict(n, covariates=covariates,
                     num_samples=100)
```

You can find all the likelihood available in the darts documentation:
https://unit8co.github.io/darts/generated_api/darts.utils.likelihood_models.html

Unit8

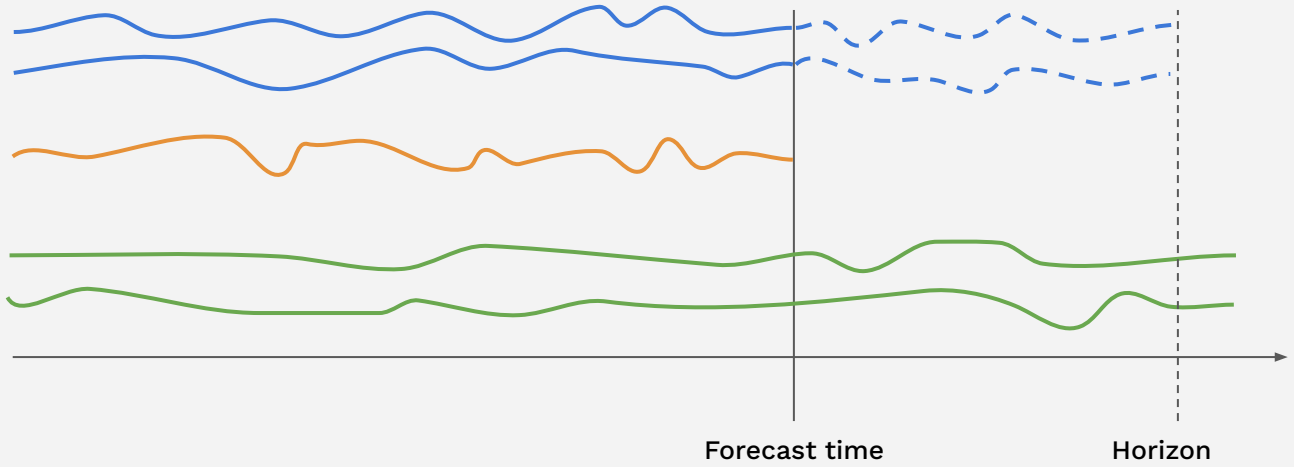
Covariates

Including Past & Future External Data

Target series: what we want to forecast

Past covariates: Unknown into the future e.g. measurements

Future covariates: Known into the future e.g. calendar, weather forecasts, actions taken



- `fit()` and `predict()` can accept `past_covariates` and/or `future_covariates`, depending on model.
- If `future_covariates` are given, future values will be required at inference time.
- Alignment of covariates with target is automatic.

Unit8

Training Datasets

[Behind the scenes] Darts Datasets

- “Datasets” classes specify how to slice several series and make a training dataset.
- Out-of-the-box models will build their own datasets (usually simple sequential).
- Datasets wrap around Python Sequence types, so that lazy loading can be used on bigger datasets.

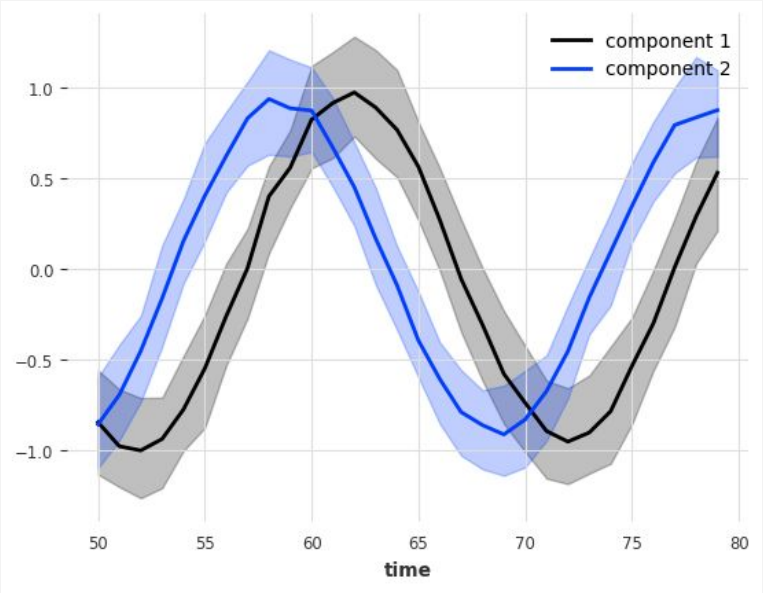
```python
class MyDataset(TrainingDataset):
  def __init__(
    self,
    series: Sequence[TimeSeries],
    past_covariates: Optional[Sequence[TimeSeries]],
    ...
  )

  def __len__(self,):
    return len(self.series)

  def __getitem__(self, i: int):
    # Return i-th (input, output) example
```

Probabilistic Forecasting

Probabilistic Forecasting

```python
from darts.models import KalmanForecaster

model = KalmanForecaster(dim_x=2)
model.fit(series)
pred = model.predict(n=30, num_samples=100)

pred.plot()
```

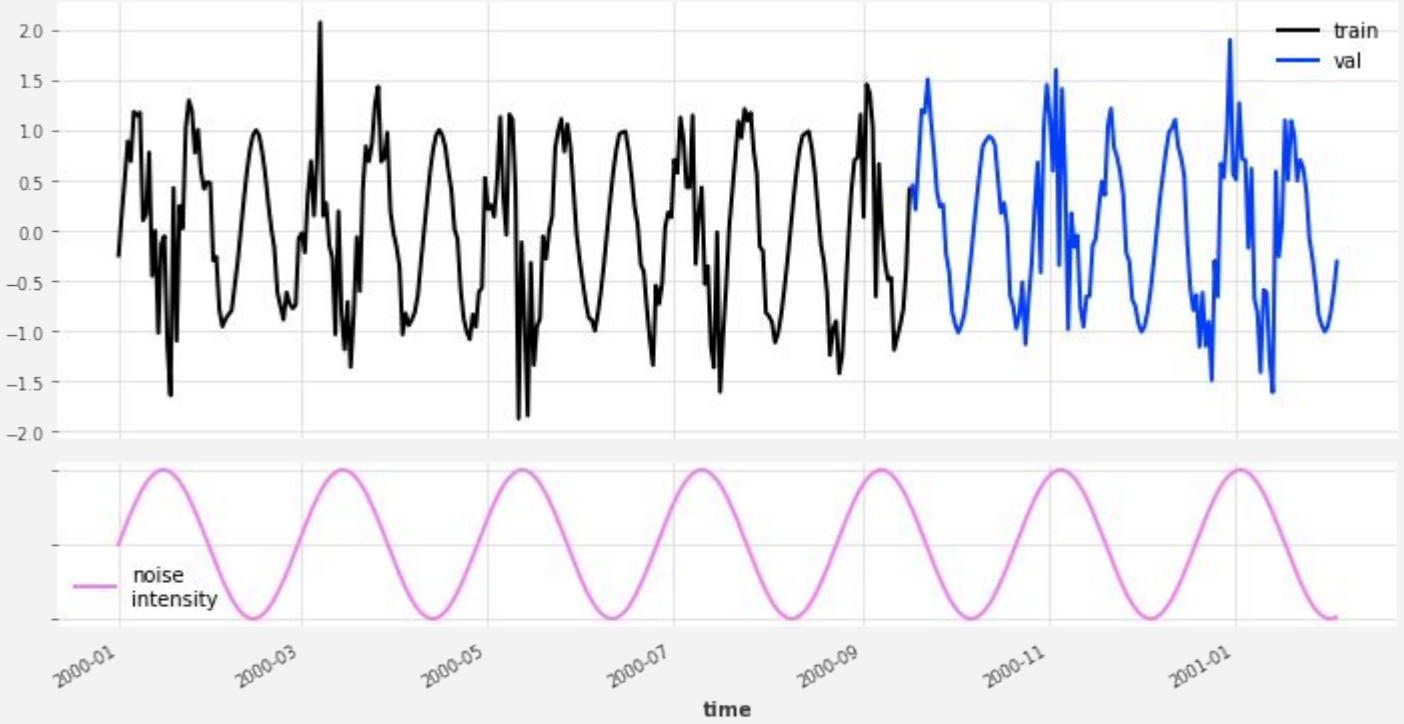
one probabilistic `TimeSeries`:



`TimeSeries` can be deterministic or stochastic.
Some models provide probabilistic forecasts.

Unit8

Probabilistic Forecasts - Example with a noisy sine wave

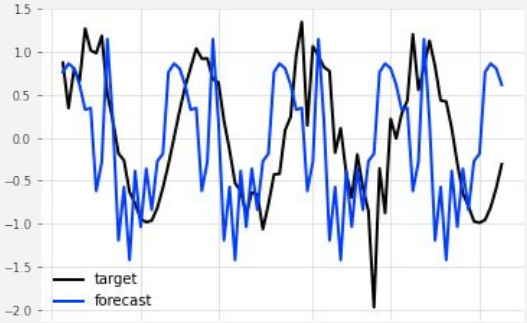


Unit8

Capturing Series Stochasticity

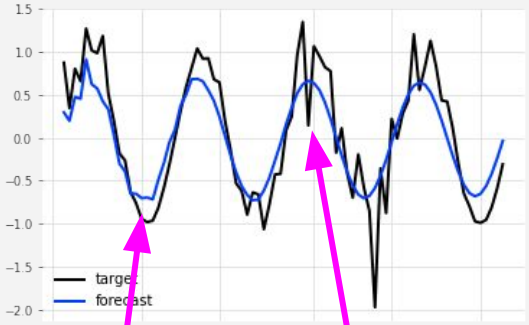
Attempt 1

```
model = NaiveSeasonal(seasonal_period)
model.fit(train)
pred = model.predict(n)
```



Attempt 2

```
model = ARIMA(seasonal_period, 0, 0)
model.fit(train)
pred =  model.predict(n)
```

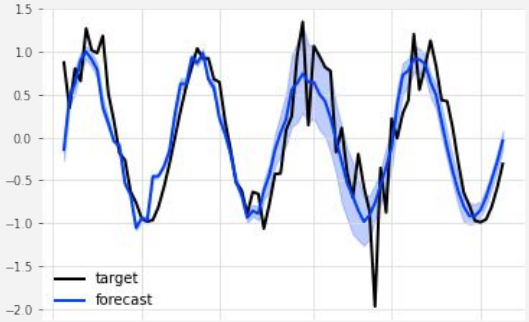


“safe” forecast

uncertain forecast

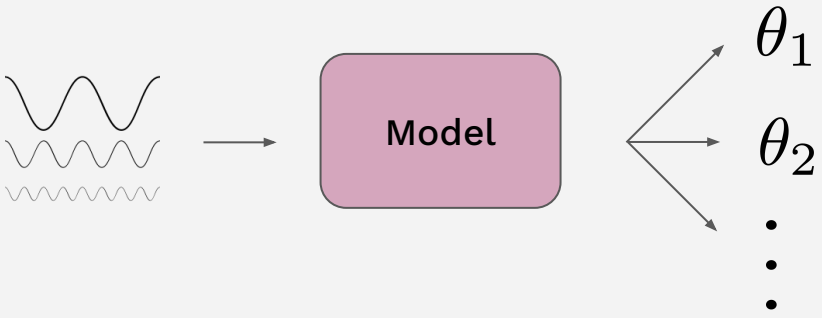
Attempt 3

```
model = TCNModel(likelihood=GaussianLikelihood(),
                 **kwargs)
model.fit(train, past_covariates=noise_intensity)
pred = model.predict(n,
                     past_covariates=noise_intensity,
                     num_samples=100)
```

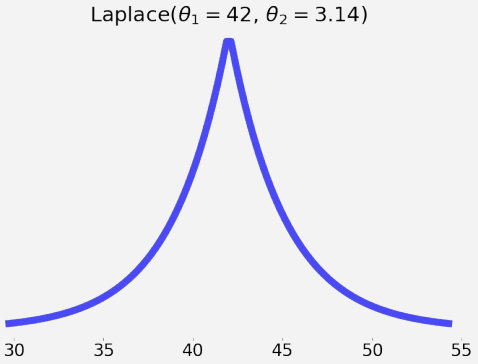


Unit8

Probabilistic forecasts

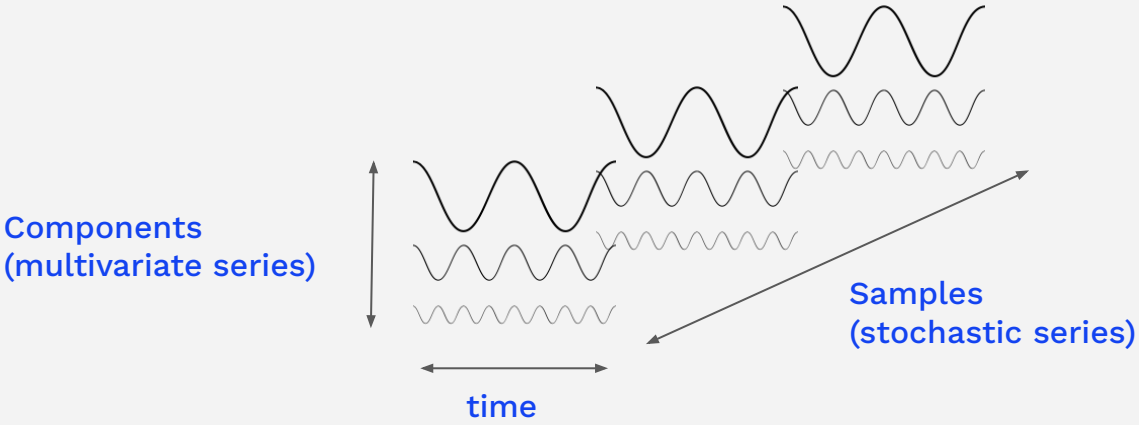


Distribution parameters

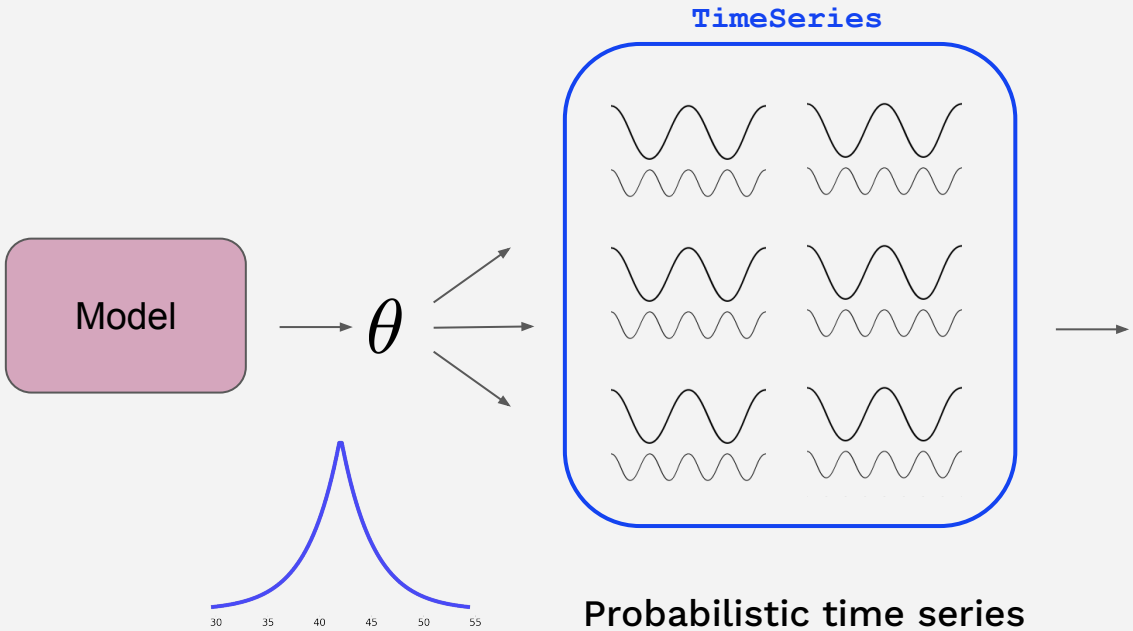


Unit8

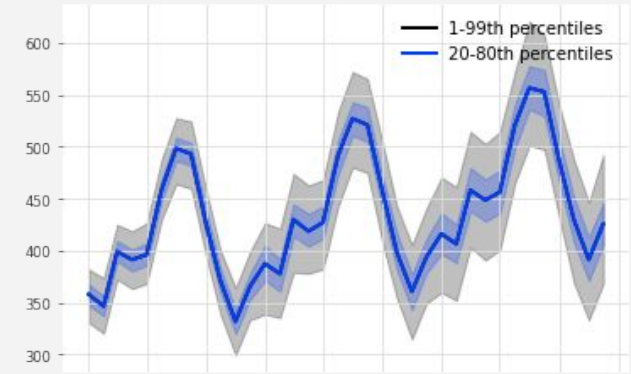
A `TimeSeries` contains 3 dimensions



Probabilistic forecasts



Probabilistic time series
(distribution-agnostic)




```
forecast.plot(low_quantile=0.01, high_quantile=0.99)
forecast.plot(low_quantile=0.2, high_quantile=0.8)
```

Confidence intervals

Unit8

Probabilistic forecasts

```
from darts.utils.likelihood_models import (
    BernoulliLikelihood,
    BetaLikelihood,
    CauchyLikelihood,
    ContinuousBernoulliLikelihood,
    DirichletLikelihood,
    ExponentialLikelihood,
    GammaLikelihood,
    GaussianLikelihood,
    GeometricLikelihood,
    GumbelLikelihood,
    HalfNormalLikelihood,
    LaplaceLikelihood,
    LogNormalLikelihood,
    NegativeBinomialLikelihood,
    PoissonLikelihood,
    QuantileRegression,
    WeibullLikelihood,
)
```

\+ Priors on distributions' parameters

Unit8