# Modeling and Analysis of Time Series Data
# Chapter 10: Forecasting

STATS 531, Winter 2026

Edward Ionides

# Model-based forecasts

- Data, $y^*_{1:N}$, and a model $Y_{1:N+h}$ with joint density $f_{Y_{1:N+h}}(y_{1:N+h}|\theta)$ can be used to *forecast* future values $y_{N+1:N+h}$ up to a *horizon*, $h$.
- A model-based *probabilistic forecast* of the not-yet-observed values $y_{N+1:N+h}$ is

$$f_{Y_{N+1:N+h}|Y_{1:N}}(y_{N+1:N+h}|y^*_{1:N};\hat{\theta}), \tag{1}$$

  where $\hat{\theta}$ is a point estimate such as an MLE.
- A model-based *point forecast* of $y_{N+1:N+h}$ is

$$\mathbb{E}\left[Y_{N+1:N+h}\middle|Y_{1:N}=y^*_{1:N};\hat{\theta}\right]. \tag{2}$$

- Point forecasts and probabilistic forecasts have many applications in business and elsewhere.

# Evaluating forecasts

- Point forecasts could be evaluated by squared error, absolute error, relative squared error, relative absolute error, etc.
- Probabilistic forecasts are naturally evaluated by the forecast log-density,

$$\log f_{Y_{N+1:N+h}|Y_{1:N}}(y_{N+1:N+h}|y^*_{1:N};\hat{\theta}), \tag{3}$$

  evaluated at the data, $y^*_{N+1:N+h}$, once it is collected.
- Due to time dependence, and limited amounts of data, it can be problematic to evaluate by cross-validation.
- Note that log-likelihood can be written as a sum of one-step forecast log-densities:

$$\log f_{Y_{1:N}}(y^*_{1:N};\theta) = \sum_{n=1}^{N} \log f_{Y_n|Y_{1:n-1}}(y^*_n|y^*_{1:n-1};\theta) \tag{4}$$

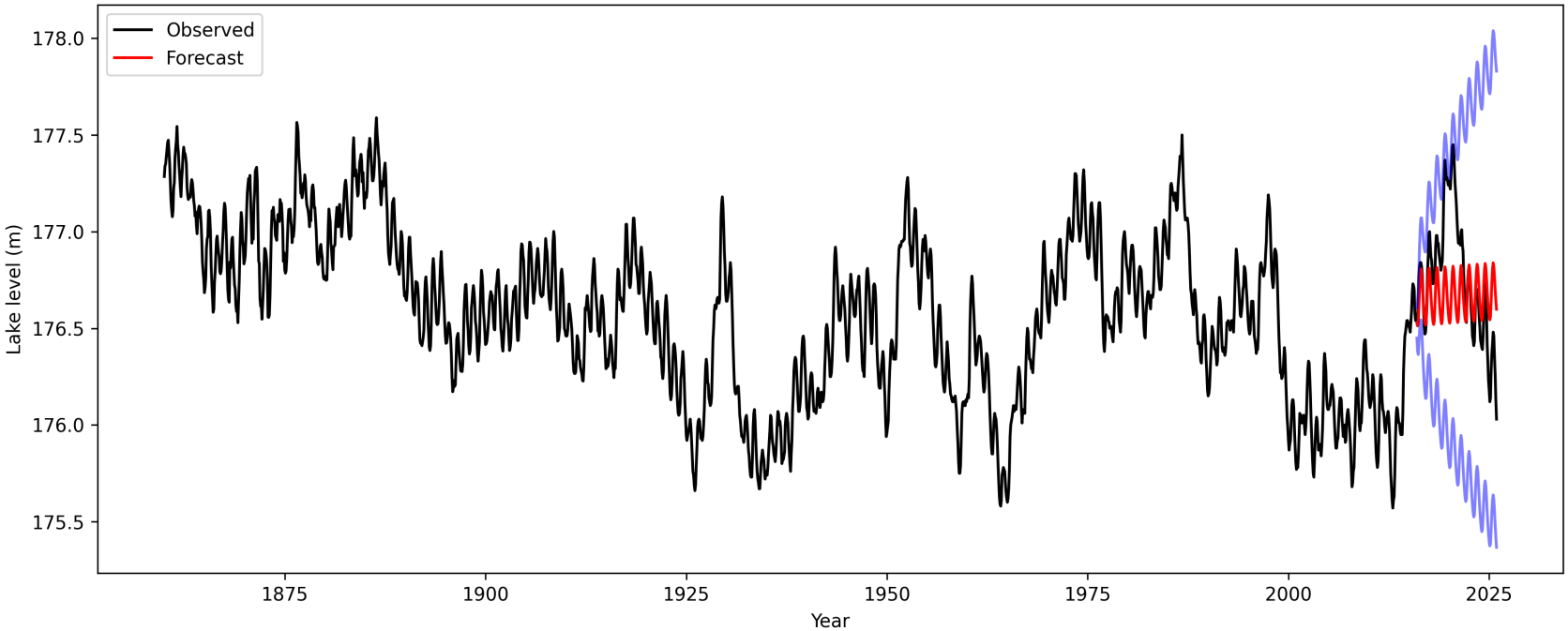
# ARIMA forecasting

The `.forecast()` method computes the conditional Gaussian distribution for forecasting an ARIMA model.

```
dat = pd.read_csv("huron_level.csv", comment='#')
huron_level = dat.iloc[:, 1:13].values.flatten()
year = np.repeat(dat['Year'].values, 12)
month = np.tile(np.arange(12), len(dat))
time = year + month / 12

# Use data through end of 2014 and forecast 10 years ahead
huron_old = huron_level[:len(huron_level) - (2024 - 2014) * 12]
time_old = time[:len(time) - (2024 - 2014) * 12]
sarma = SARIMAX(huron_old, order=(0, 1, 1),
    seasonal_order=(0, 1, 1, 12)).fit()
forecast_df = sarma.get_forecast(steps=120)
summary_df = forecast_df.summary_frame()
f_val = summary_df['mean']
f_se = summary_df['mean_se'] # Standard error of the forecast
f_time = time_old[-1] + (np.arange(1, 121)) / 12
```

# 95% prediction interval from December 2014




- Here, we use SARIMA$((0,1,1) \times (0,1,1)_{12}))$.
- The fitted model is quite similar to SARIMA$((1,0,1) \times (1,0,1)_{12}))$.

# Facebook Prophet

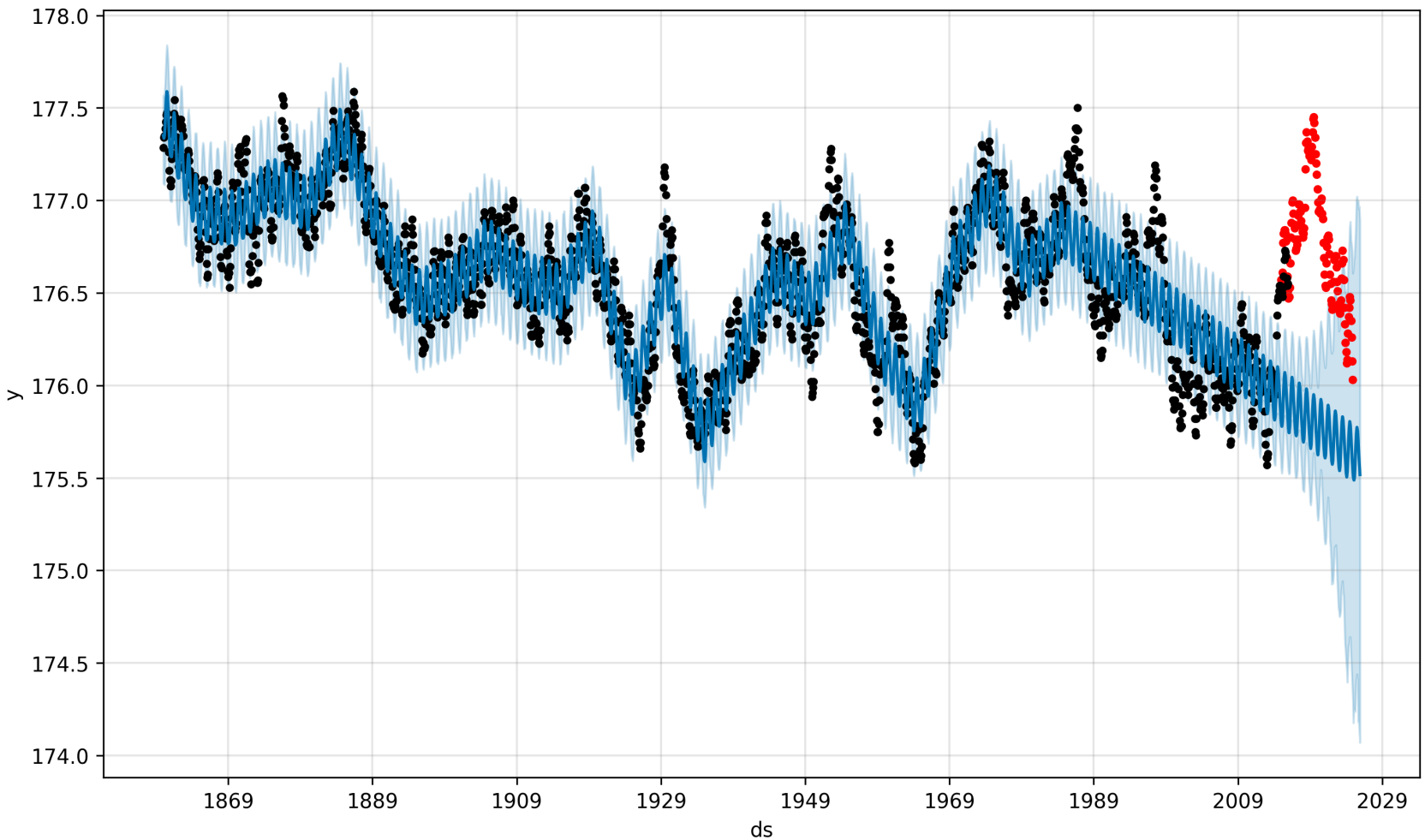
- ARIMA models are good for relatively short time series.
- SARIMA is good for monthly and quarterly data, but less so for daily or hourly.
- You may have already experienced this. Large-scale forecasting competitions confirm it [4].
- Prophet was designed for high-frequency (daily, hourly) business forecasting tasks at Facebook, and is widely used for similar tasks elsewhere.
- Prophet does penalized regression estimating trend and seasonality components. It can also do Bayesian fitting.
- Unlike ARIMA, Prophet cannot describe general covariance structures.
- Prophet prediction intervals are based on forward simulations of a piecewise linear model with fitted rate of breakpoints Taylor and Letham [6].

```
from prophet import Prophet
# Data for Prophet requires 'ds' and 'y' columns
history = pd.DataFrame({
    'ds': pd.date_range(start='1860-01-01',
        periods=len(huron_old), freq='MS'),
    'y': huron_old
})

# Fit Prophet model
fit = Prophet()
fit.fit(history)

# Create future dataframe for 10 years (120 months)
future = fit.make_future_dataframe(periods=10*12, freq='MS')
forecast = fit.predict(future)
```

```
fig = fit.plot(forecast)
# Add actual future values in red
plt.scatter(
    pd.date_range(start='2015-01-01',
    periods=len(huron_level)-len(huron_old), freq='MS'),
    huron_level[len(huron_old):], color='red', s=10)
plt.tight_layout(); plt.show()
```



# An artificial neural net approach

- The most popular neural net architecture for time series is long short-term memory (LSTM).
- LSTM is a type of recurrent neural network (RNN) in which new data is incorporated into a hidden state (i.e., memory) used to predict future data.
- LSTM has a gating mechanism to allow information into and out of this state.
- In some situations, especially if combined with SARIMA methods, this can give very competititive forecast results Makridakis, Spiliotis, and Assimakopoulos [4].
- Neural networks are good for addressing nonlinear, non-Gaussian, high-dimensional behavior when there is a large amount of training data. They are not a natural method for small or medium sized time series datasets.

# Implementing LSTM for the Huron level data

**Preprocessing for LSTM:** LSTM requires explicit preprocessing to handle the (d=1) (trend) and (D=1,m=12) (seasonal) components. Scaling is done to help training.

```
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
from sklearn.preprocessing import MinMaxScaler

torch.manual_seed(531) # random seed for reproducibility
data = pd.Series(huron_old)
diff_seasonal = data.diff(12).dropna()
diff_data = diff_seasonal.diff(1).dropna()
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data=scaler.fit_transform(diff_data.values.reshape(-1, 1))
```

**For compatibility with the Pytorch `DataLoader`**, we create a custom `Dataset` capturing the seasonal relationship $(m = 12)$. The LSTM input should look back at least 12-24 steps.

```
class TimeSeriesDataset(Dataset):

    def __init__(self, data, lookback):
        self.data = torch.FloatTensor(data).squeeze()
        self.lookback = lookback

    def __len__(self):
        return len(self.data) - self.lookback

    def __getitem__(self, idx):
        # Input: lookback window, Target: next value
        x = self.data[idx:idx + self.lookback]
        y = self.data[idx + self.lookback:idx + self.lookback + 1]
        return x, y

lookback = 12  # Use 12 months of history
dataset = TimeSeriesDataset(scaled_data, lookback)
dataloader = DataLoader(dataset, batch_size=32, shuffle=True)
```

# Set up the LSTM as a PyTorch `nn.Module`

```
class LSTMModel(nn.Module):

    def __init__(self, input_size=1, hidden_size=50, num_layers=1,
            output_size=1):
        super(LSTMModel, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers,
            batch_first=True)
        self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size)
        c0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size)
        out, _ = self.lstm(x, (h0, c0))
        out = self.fc(out[:, -1, :])
        return out

# Initialize model, loss function, and optimizer
model = LSTMModel(input_size=1, hidden_size=50, num_layers=1)
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

## Train with standard gradient descent.

```
num_epochs = 50
model.train()
for epoch in range(num_epochs):
    epoch_loss = 0
    for batch_x, batch_y in dataloader:
        # batch_x: (batch, lookback) -> (batch, lookback, 1)
        batch_x = batch_x.unsqueeze(-1)
        outputs = model(batch_x)
        loss = criterion(outputs, batch_y)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        epoch_loss += loss.item()
    if (epoch + 1) % 10 == 0:
        print(f'Epoch [{epoch+1}/{num_epochs}]',
            f'Loss: {epoch_loss/len(dataloader):.6f}')
```

```
Epoch [10/50] Loss: 0.015177
Epoch [20/50] Loss: 0.014493
Epoch [30/50] Loss: 0.014117
Epoch [40/50] Loss: 0.014005
Epoch [50/50] Loss: 0.012753
```

## Recursively generate a 10-year forecast.

```
model.eval()

# Start with the last lookback window from training data
# scaled_data has shape (N, 1), squeeze to (N,)
current_window = torch.FloatTensor(
    scaled_data[-lookback:].squeeze())
# Reshape to (1, lookback, 1) for model input
current_window = current_window.unsqueeze(0).unsqueeze(-1)
forecast_scaled = []

with torch.no_grad():
    for i in range(120):
        pred = model(current_window)
        forecast_scaled.append(pred.squeeze().item())
        # Reshape pred from (1, 1) to (1, 1, 1) for concatenation
        pred_reshaped = pred.unsqueeze(1)
        current_window = torch.cat([current_window[:, 1:, :],
            pred_reshaped], dim=1)
```

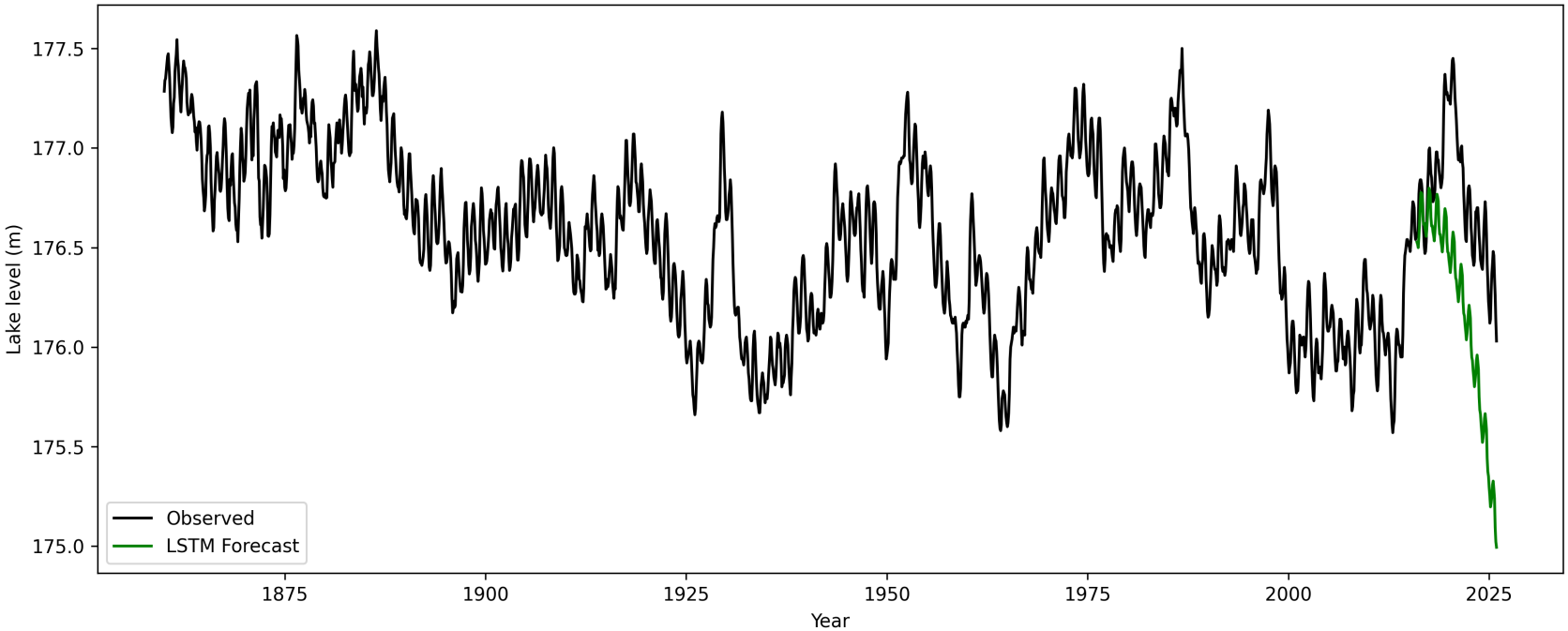
## Reverse all transformations to get back to original scale.

```
forecast_diff = scaler.inverse_transform(
    np.array(forecast_scaled).reshape(-1, 1)).flatten()
last_diff_seasonal = diff_seasonal.iloc[-1]
forecast_seasonal_diff = (np.cumsum(forecast_diff)
    + last_diff_seasonal)

# Invert the seasonal differencing (D=1, m=12)
# Need the last 12 values of the original series
last_12_values = data.iloc[-12:].values
forecast_original = np.zeros(120)
for i in range(120):
    if i < 12:
        forecast_original[i] = (forecast_seasonal_diff[i] +
            last_12_values[i])
    else:
        forecast_original[i] = (forecast_seasonal_diff[i] +
            forecast_original[i - 12])

lstm_time = time_old[-1] + (np.arange(1, 121)) / 12
```

# LSTM Forecast Plot



# Forecasting versus model fitting

- A good model should imply a good model-based forecast.
- Long-term forecasting is extrapolation. The model may be unreliable far from the timeframe used to build it.
- Without evidence to support a model for long-term forecasts, uncertainty estimates should be high. Uncertainty estimates are also uncertain!
- Deep learning methods need large amounts of data. They are not yet standard for forecasting. Prophet uses automatic differentiation techniques that enable deep learning.

# Forecasting with trends and covariates

- A model with trends and covariates must project those into the future in order to forecast.
- Uncertainty about future trends may be captured by "stochastic trend" models. Prophet does this.
- We have seen the difficulty assessing stationarity vs slowly varying trend. The same issue arises with forecasting. How do we know if a trend will continue, or if it will change in future?

# Further reading

- Section 3.5 of Shumway and Stoffer [5] covers ARIMA forecasting.
- Hyndman and Khandakar [2] introduces the `forecast` R package.
- Taylor and Letham [6] presents the Facebook Prophet forecasting algorithm.
- Chapter 10 of Huang and Petukhina [1] discusses artificial neural network methods and introduces LSTM.

# Acknowledgements


- Compiled on February 2, 2026using Python.
- Licensed under the Creative Commons Attribution-NonCommercial license. Please share and remix non-commercially, mentioning its origin.
- We acknowledge previous versions of this course.
- The LSTM section was drafted in collaboration with Gemini, edited in collaboration with Claude, and then revised after reading Lim and Zohren [3].

# References I


[1] Changquan Huang and Alla Petukhina. *Applied Time Series Analysis and Forecasting with Python*. Springer, 2022.

[2] Rob J Hyndman and Yeasmin Khandakar. "Automatic time series forecasting: The forecast package for R". In: *Journal of Statistical Software* 27 (2008), pp. 1–22.

[3] Bryan Lim and Stefan Zohren. "Time-series forecasting with deep learning: A survey". In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 379.2194 (2021).

[4] Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. "The M4 Competition: 100,000 time series and 61 forecasting methods". In: *International Journal of Forecasting* 36.1 (2020), pp. 54–74.

[5] Robert H Shumway and David S Stoffer. *Time Series Analysis and its Applications: With R Examples*. 4th. Springer, 2017.

# References II

[6] Sean J Taylor and Benjamin Letham. "Forecasting at scale". In: *The American Statistician* 72.1 (2018), pp. 37–45.