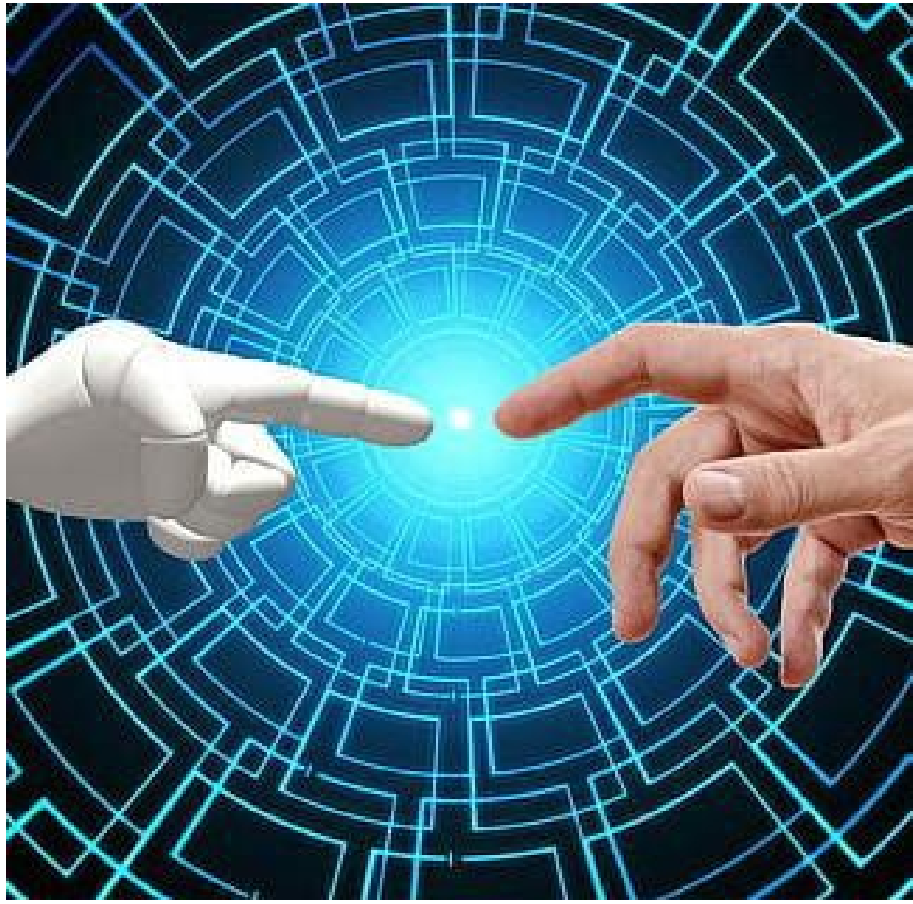
Artificial Intelligence (AI) for Engineering

COS40007

Dr. Afzal Azeem Chowdhary

Lecturer, SoCET, Swinburne University of Technology

Seminar 7: 15th April 2025



Overview

❑Basics of Time-series

❑Forecasting Model

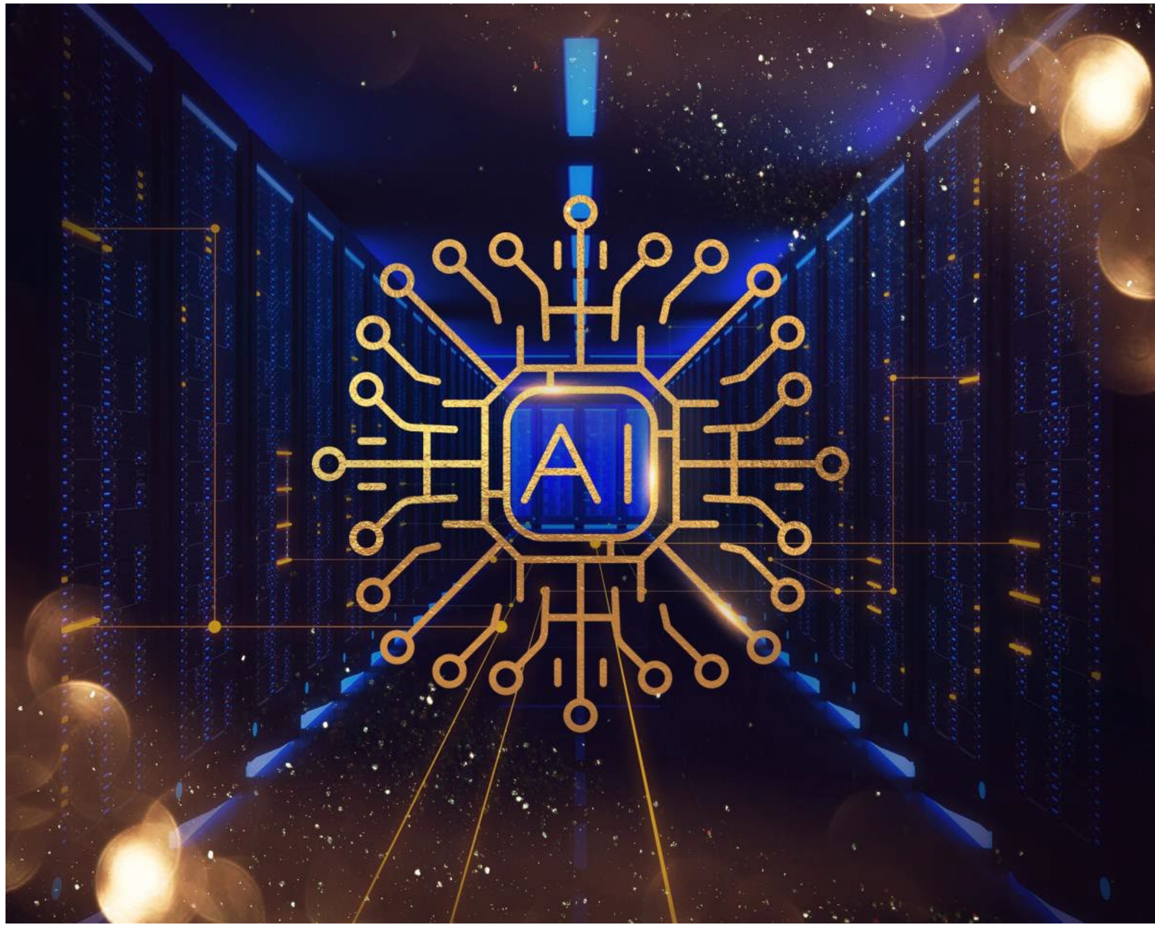
❑Regression for Forecasting

❑Forecasting Algorithms

❑Forecasting using LSTM

Required Reading

–Chapter 9, 15 of “Machine Learning with PyTorch and Scikit- Learn”



At the end of this you should be able to

- Understand time-series data
- Understand time-series data pre-processing
- Understand time-series forecasting models
- Understand deep learning based time series forecasting using Long Short-Term Memory (LSTM)

SWINBURNE UNIVERSITY OF TECHNOLOGY

Time-Series Forecasting

Time-series

A *time-series* is a set of observations on a quantitative variable collected over time.

Examples

- Production plant: machine settings update every second
- Biomedical: heart rate, ECG
- Economics: Interest rates, GDP, and employment etc.
- Energy (Electricity, Gas, Oil, and Solar) demands and prices etc.
- Weather: e.g., local and global temperature etc.
- Sensors: Internet-of-Things

Businesses are often very interested in forecasting time series variables.

In time series analysis, we analyze the past behavior of a variable in order to predict its future behavior.

Time-series

Time series analysis can be divided into two main types: *descriptive and inferential*. ***Descriptive*** analysis aims to summarise and visualise the time series data and identify its main features and components. ***Inferential*** analysis seeks to build and test models that can explain the behavior of the time series data and forecast its future values.

Time series analysis is different and complex from other types of data analysis for several reasons, such as:

1. **Time series data is ordered and sequential:** Time series data has a natural order and a temporal dimension. This means that the time series data is not independent and identically distributed (i.i.d.) but somewhat dependent and correlated.
2. **Time series data is dynamic and stochastic:** Unlike static or deterministic data, time series data is constantly changing and subject to randomness and uncertainty. This means data may have non-stationarity, heteroscedasticity, or nonlinearity that needs to be addressed. It also means that the time series data may have outliers, missing values, or regime changes that must be handled.
3. **Time series data is high-dimensional and complex:** It may have many variables and features that interact and influence each other. This means that the time series data may have multicollinearity, sparsity, or dimensionality that must be reduced. It also means that the time series data may have hidden patterns, structures, or factors that must be extracted.

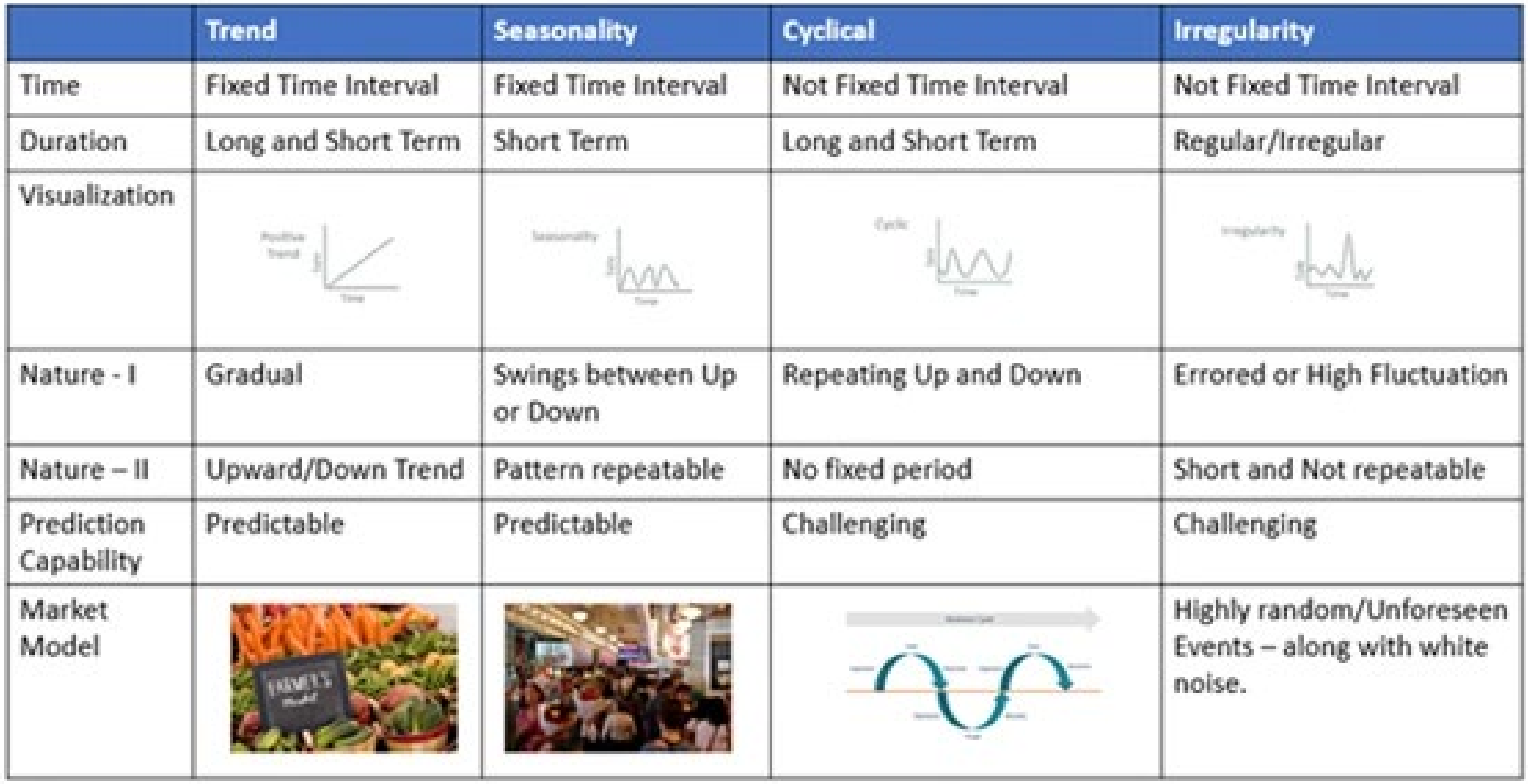
How to Approach Time Series Analysis?

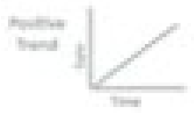
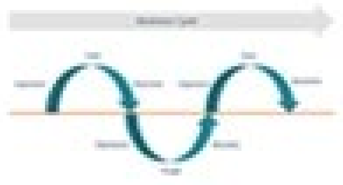
1. Define the problem and the objective
2. Collect and clean the data
3. Explore and visualise the data
4. Model and evaluate the data
5. Interpret and communicate the results

With the help of “Time Series,” we can prepare numerous time-based analyses and results.

- **Forecasting:** Predicting any value for the future.
- **Segmentation:** Grouping similar items together.
- **Classification:** Classifying a set of items into given classes.
- **Descriptive Analysis:** Analysis of a given dataset to find out what is there in it.
- **Intervention Analysis:** Effect of changing a given variable on the outcome.

Components of Time Series Analysis

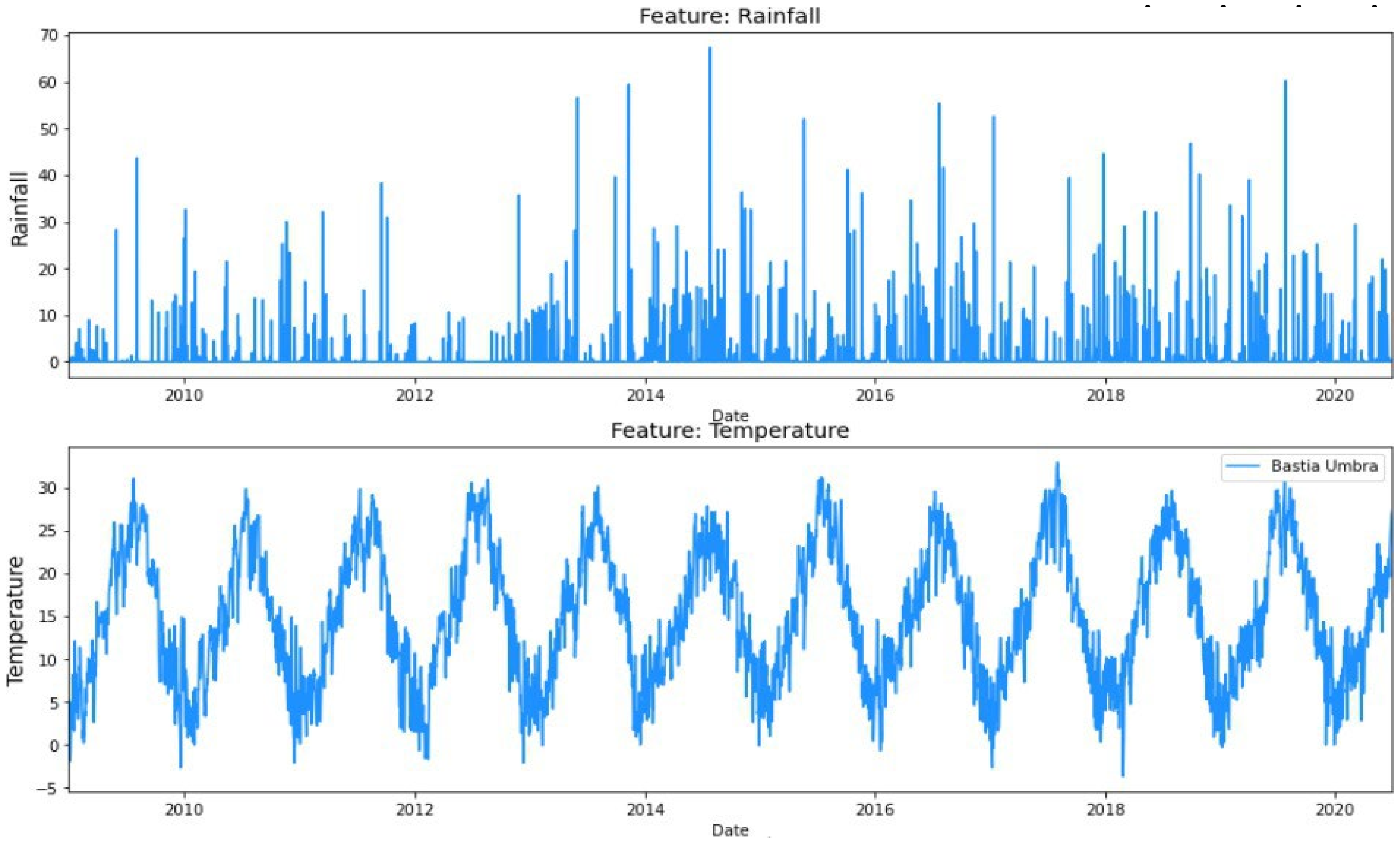


	Trend	Seasonality	Cyclical	Irregularity
Time	Fixed Time Interval	Fixed Time Interval	Not Fixed Time Interval	Not Fixed Time Interval
Duration	Long and Short Term	Short Term	Long and Short Term	Regular/Irregular
Visualization				
Nature - I	Gradual	Swings between Up or Down	Repeating Up and Down	Errored or High Fluctuation
Nature – II	Upward/Down Trend	Pattern repeatable	No fixed period	Short and Not repeatable
Prediction Capability	Predictable	Predictable	Challenging	Challenging
Market Model				Highly random/Unforeseen Events – along with white noise.

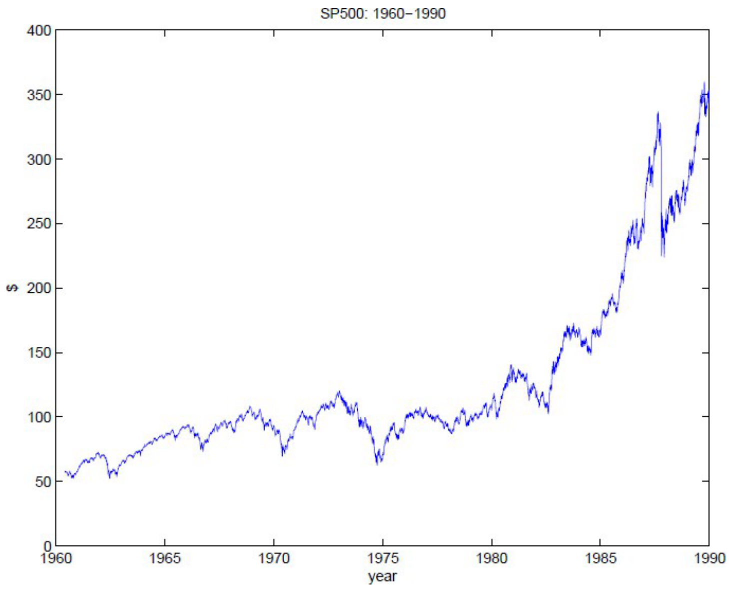
Designed by Author (Shanthababu)

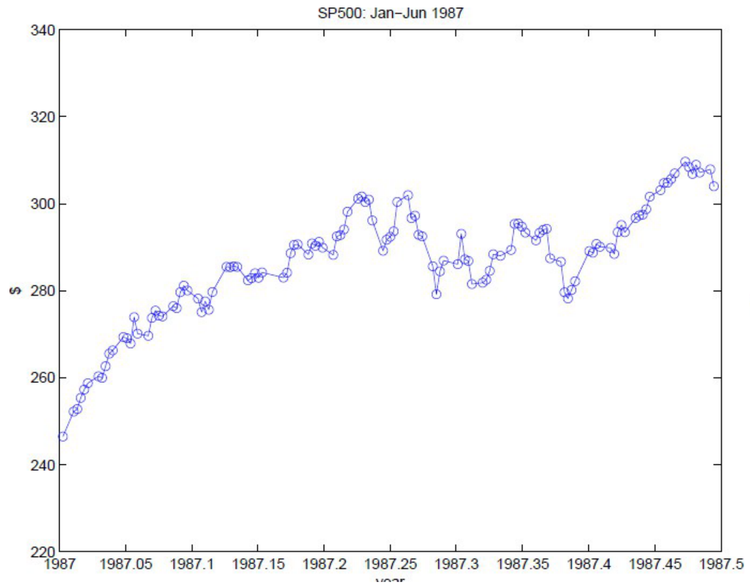
Time-series

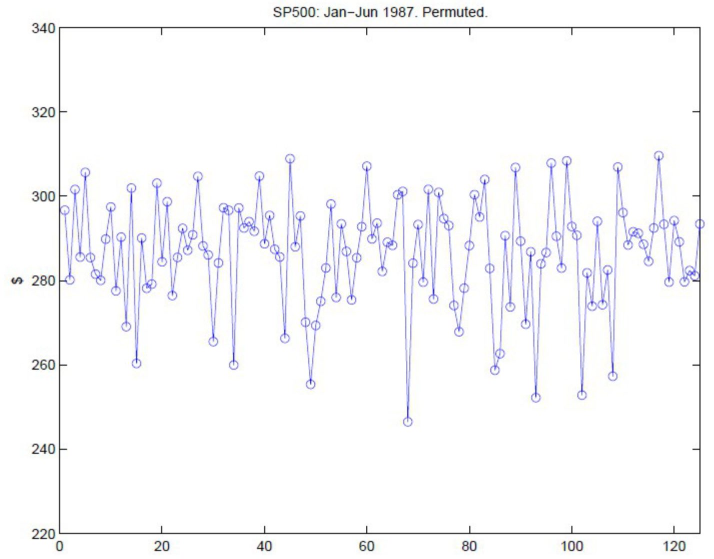
The data should be in **chronological order** and the **timestamps should be equidistant (1 sec / 1 min / 1 Hour / 1 day)** in time series.



Example of Time-Series







- Stationary time series have the best linear predictor.
- Nonstationary time series models are usually slower to implement for prediction.

Data Types of Time Series

1. Stationary: A dataset should follow the thumb rules below without having the time series' trend, seasonality, cyclical, and irregularity components.
• Their mean value should be constant in the data during the analysis.

•The **variance** should be constant with respect to the time-frame

•**Covariance** measures the relationship between two variables.

2. Non-stationary: A dataset is called non-stationary if the mean-variance or covariance changes concerning time.

There are two tests available to test if the dataset is stationary: the Augmented Dickey-Fuller (ADF) Test and the Kwiatkowski-Phillips-Schmidt-Shin (KPSS) Test
The Augmented Dickey-Fuller (ADF) or Unit Root Test is done with the following assumptions:
•Null Hypothesis (H0): Series is non-stationary

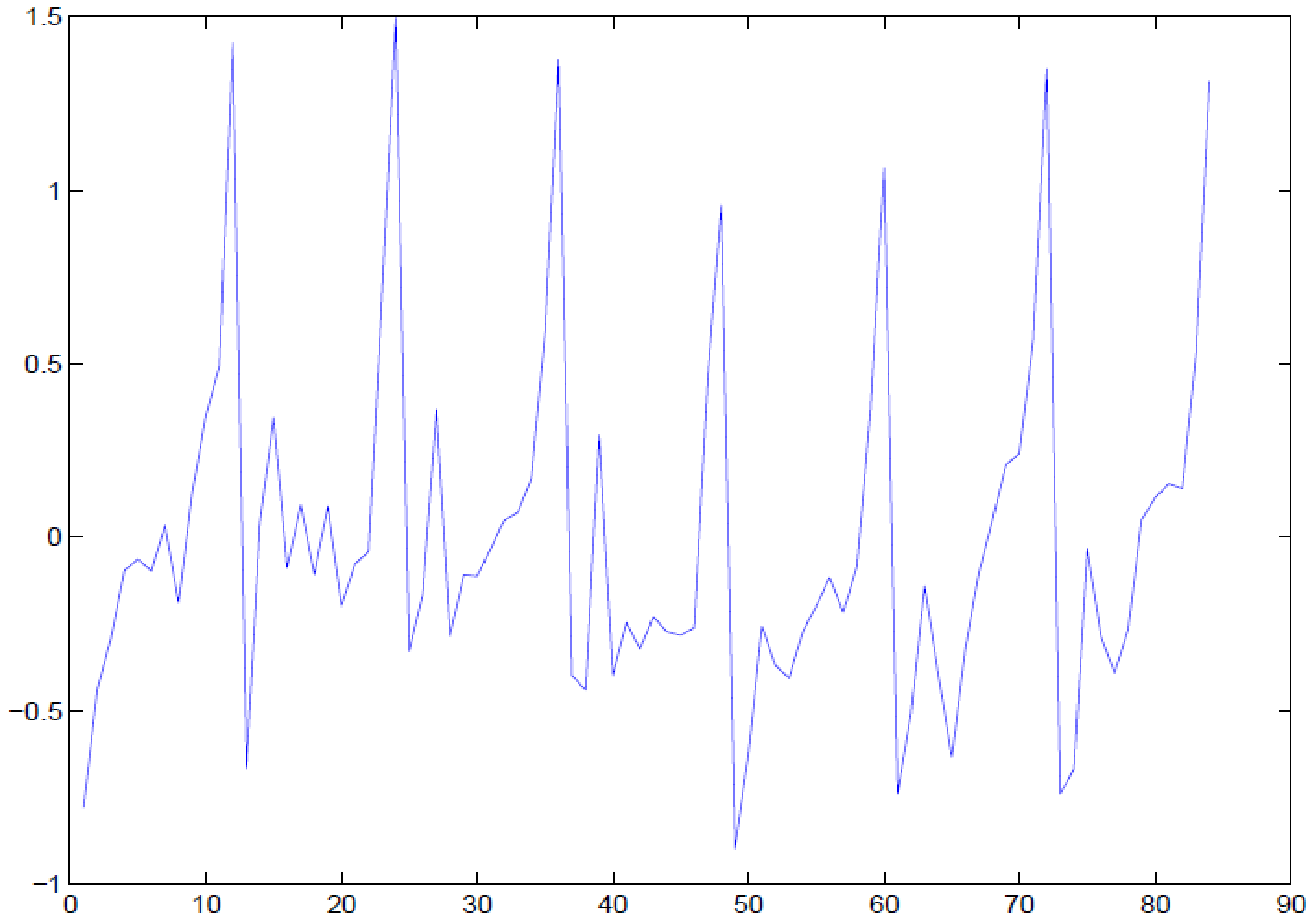
•Alternate Hypothesis (HA): Series is stationary

- ❖ p-value >0.05 Fail to reject (H_0)
- ❖ p-value <= 0.05 Accept (H_1)

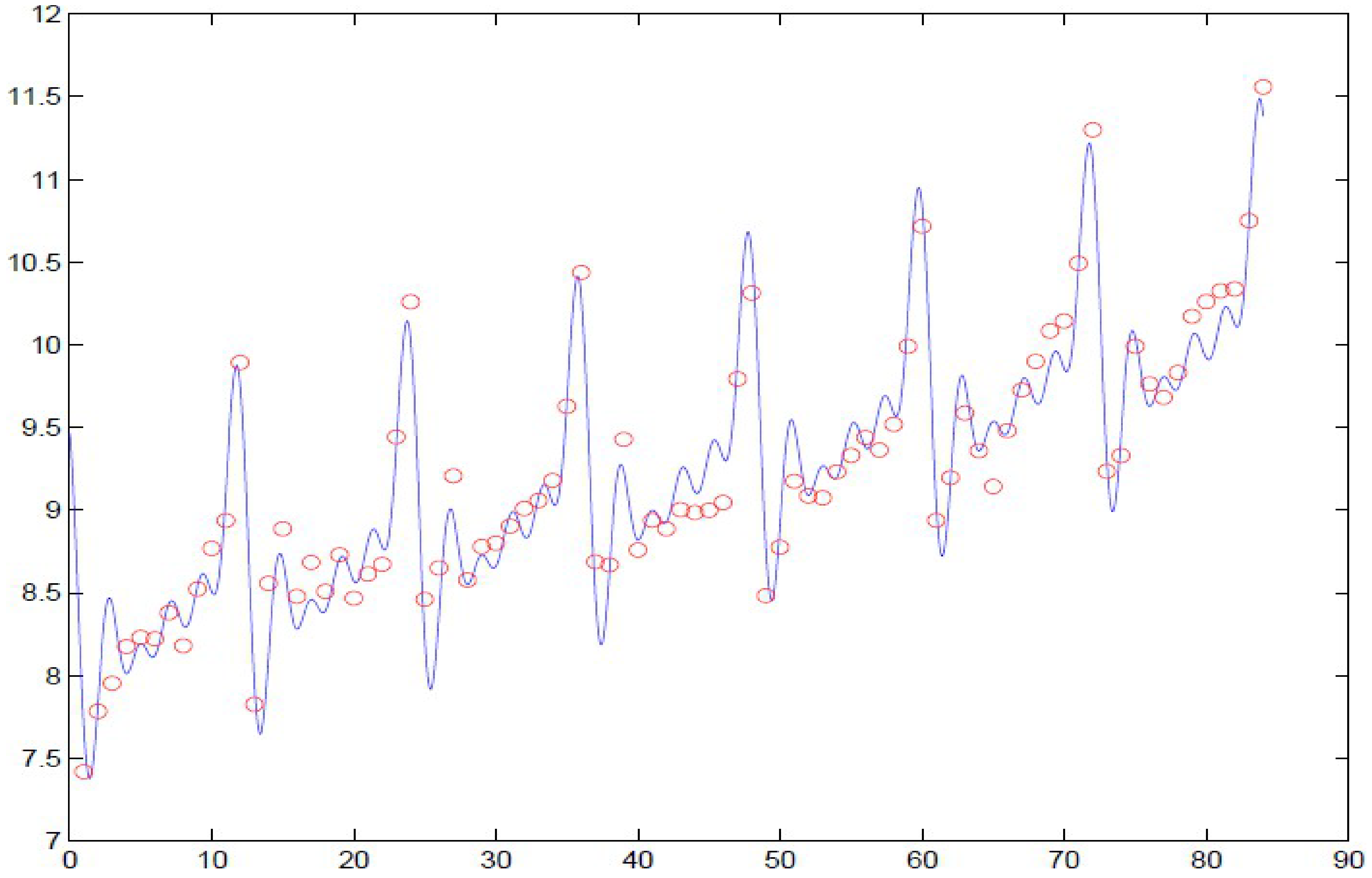
Converting Nonstationary Time Series to Stationary Time Series

- Remove deterministic factors
  - Trends (detrending)
  - Polynomial regression fitting
  - Exponential smoothing
  - Transformation (Power Transform, Square Root, and **Log Transfer**)
  - Moving average smoothing
  - Differencing (B is a back-shift operator)
- After conversion, the remaining data points are called residuals.
- If residuals are IID (Independent and Identically Distributed), then no more analysis is necessary since its mean value will be the best predictor

Residuals

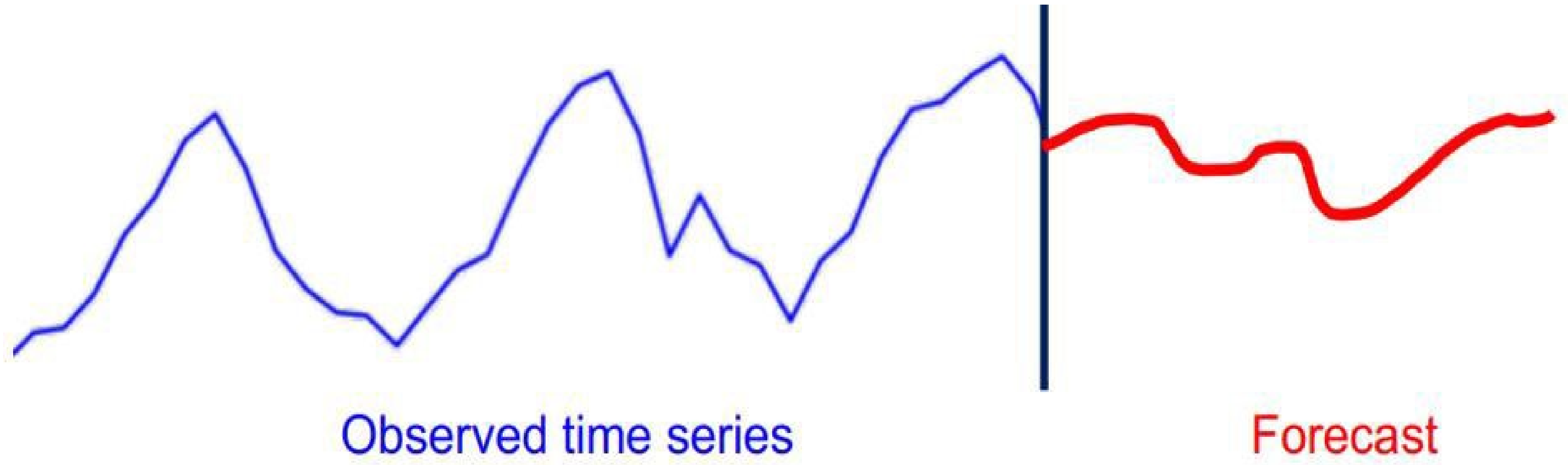


Trend and Seasonal variation



Forecasting

The goal of forecasting is just not to predict what is in the future, but this also helps to take meaningful action in the present



Data preparation for time-series foresting (Train, Test Set)

- To demonstrate the predictive power, the time series is split into training and test sets.
- Unlike other datasets, time series data are usually split without shuffling. The training set is the first half of the time series, and the remaining will be used as the test set.

```
# train-test split for time series
train_size = int(len(timeseries) * 0.67)
test_size = len(timeseries) - train_size
train, test = timeseries[:train_size], timeseries[train_size:]
```

Lookback

A parameter that indicates how many previous time series values are used to predict the current time point

On a long enough time series, multiple overlapping windows can be created.

It is convenient to create a function to generate a dataset of fixed window from a time series.

```python
import torch

def create_dataset(dataset, lookback):
    """Transform a time series into a prediction dataset

    Args:
        dataset: A numpy array of time series, first dimension is the time steps
        lookback: Size of window for prediction (number of time steps to use as input)

    Returns:
        X: Input features of shape (num_samples, lookback, num_features)
        y: Target values of shape (num_samples, num_features)
    """
    X, y = [], []
    for i in range(len(dataset)-lookback):
        feature = dataset[i:i+lookback]  # Input window
        target = dataset[i+lookback]     # Single target value (next time step)
        X.append(feature)
        y.append(target)
    return torch.tensor(X, dtype=torch.float32), torch.tensor(y, dtype=torch.float32)

# Example usage:
lookback = 1
X_train, y_train = create_dataset(train, lookback=lookback)
X_test, y_test = create_dataset(test, lookback=lookback)
```

Types of missing value

- **MCAR (Missing Completely at Random)**, when the probability of missing values in a variable is the same for all samples. E.g, whether to include house numbers in a survey during data collection
- **MAR (Missing at Random)**, when a variable has missing values that are randomly distributed. However, they are related to some other variables' values. For example, if people are more likely to hide their income level, income is MAR.
- **MNAR (Missing not at Random)**, when the missingness of data is related to events or factors which the researcher does not measure. For example, people with high income levels are less likely to respond to income level questions due to the fear of higher taxes.

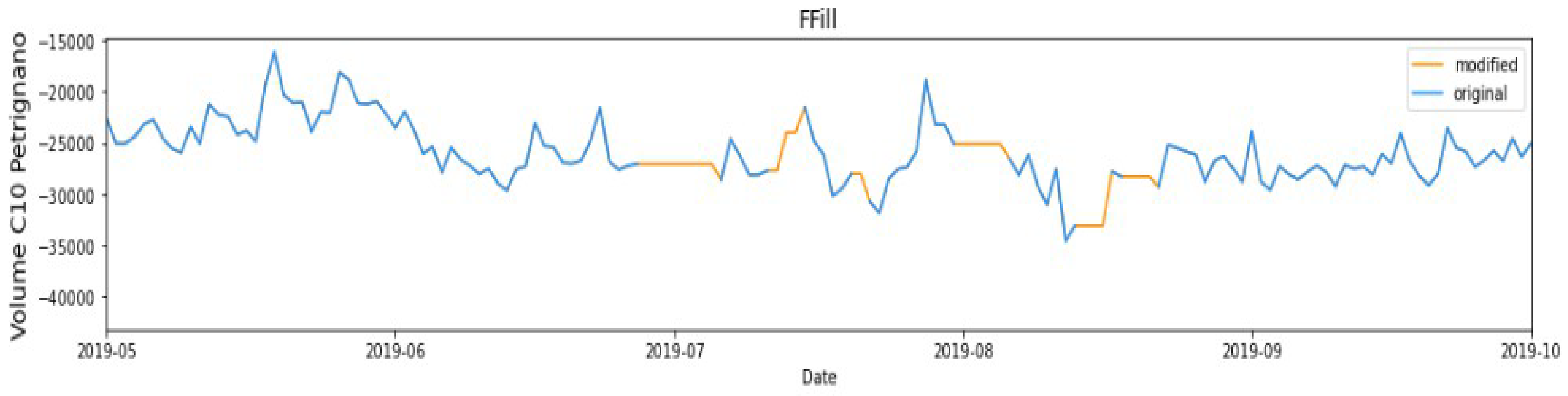
https://medium.com/@aaabulkhair/data-imputation-demystified-time-series-data-69bc9c798cb7

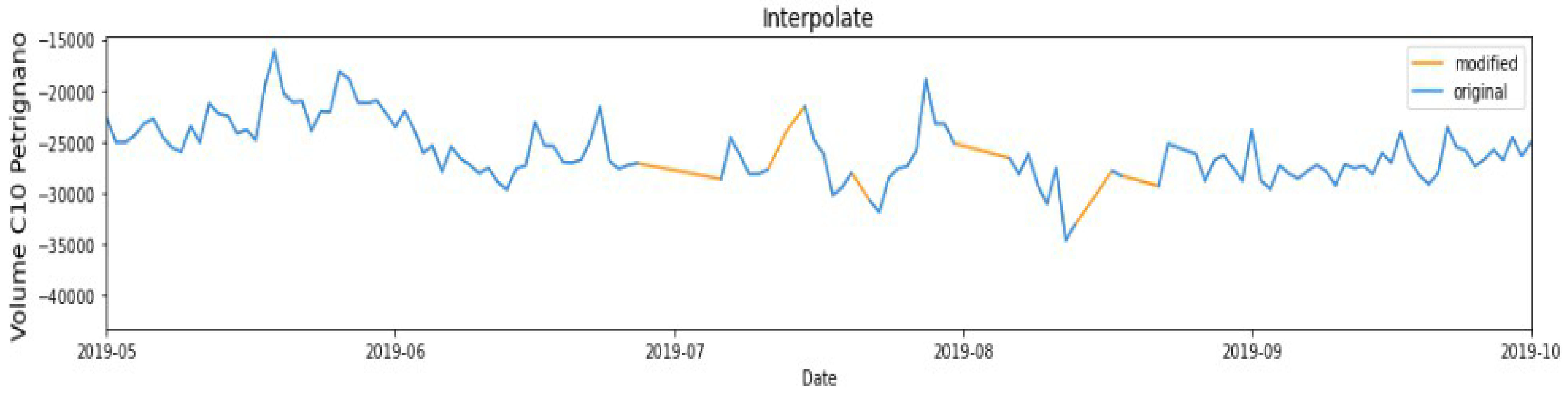
Methods to handle missing value

- Deletion
- Constant Imputation - substitutes all missing values with a constant
- Last Observation Carried Forward (LOCF) and Next Observation Carried Backward (NOCB) - methods replace missing values either with the immediately preceding observed value (LOCF) or the subsequent observed value (NOCB)
- Mean/Median/Mode Imputation
- Rolling Statistics Imputation
- Linear Interpolation - replaced based on a linear equation derived from the available values
- Spline Interpolation - missing values are replaced based on a spline interpolation of the available values
- K-Nearest Neighbors (KNN) Imputation
- STL Decomposition for Time Series - breaks down the time series into trend, seasonality, and residuals, then imputes missing values before reassembling the components.

https://medium.com/@aaabulkhair/data-imputation-demystified-time-series-data-69bc9c798cb7

Handling missing value



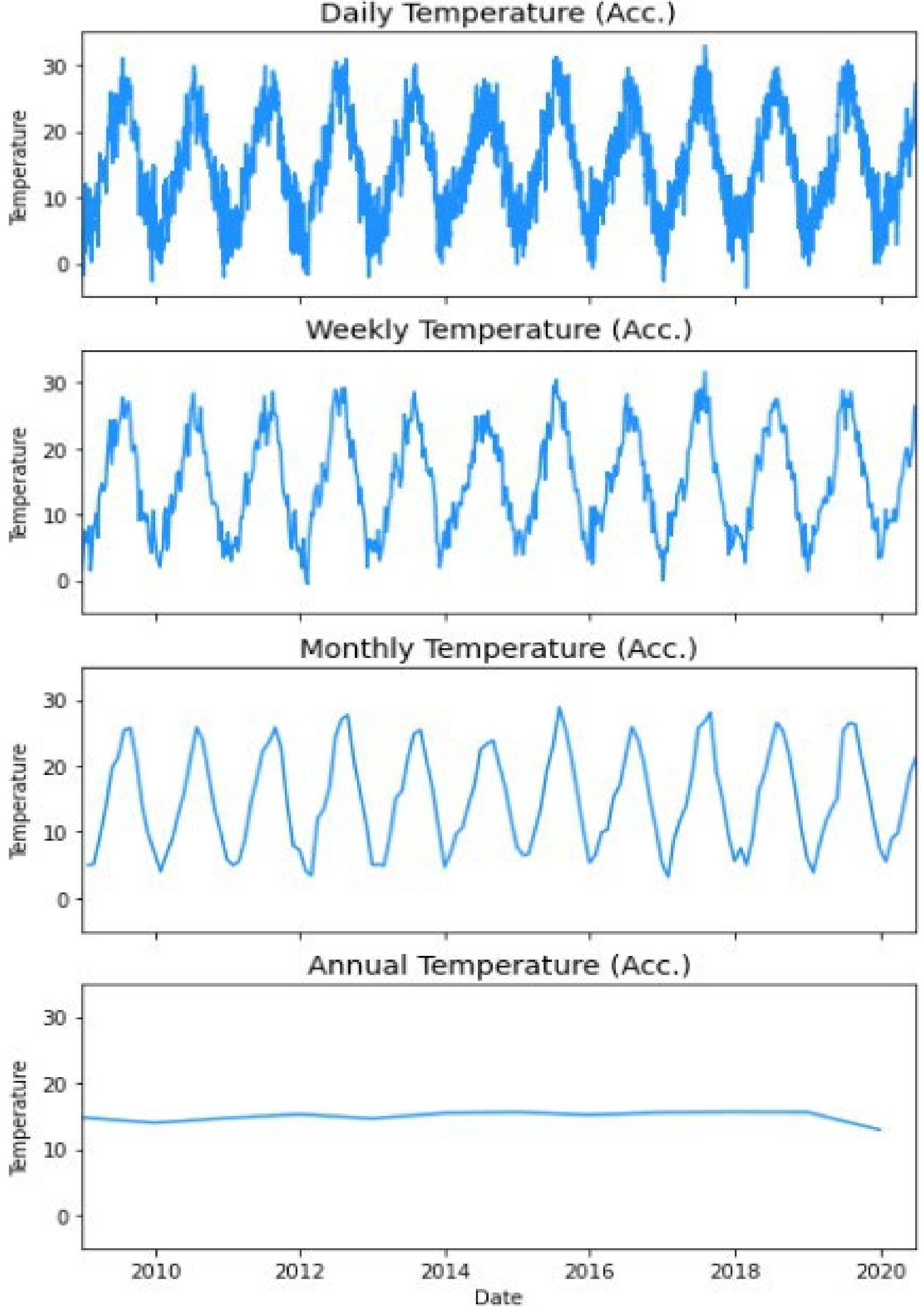


Resampling

When timesteps are not constants and values aren't acquired at a fixed time stamp, we need to resample the data.
We have to work here to align the time series to get fixed timestamps and resample the time series to get regular time steps.

- *Fixed Timestamps:* Align the Time Series
- *Regular Time Steps:* Resample the Time Series

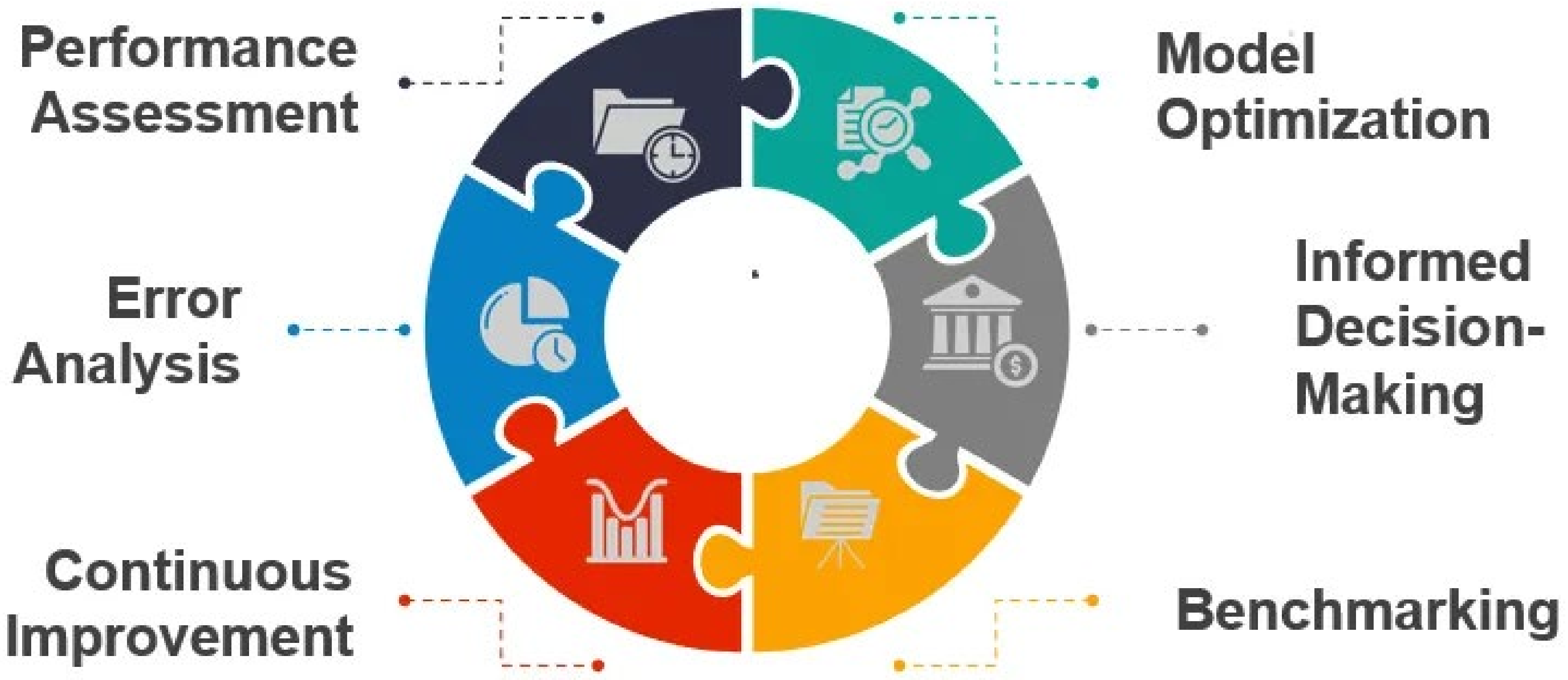
There are different resampling strategies:
- Under sampling: take a bigger granularity
- Over sampling: take a smaller granularity (and so create new values)



Time-series Prediction Evaluation

- Root Mean Square Error (RMSE): **= √((1 / n) * Σ(actual — forecast)²)**
- Mean Average Error (MAE) **= (1 / n) * Σ(|(actual — forecast) / actual|)**
- Mean Average Percentage Error (MAPE) **= (1 / n) * Σ(|(actual — forecast) / actual|) * 100%**
- Mean Absolute Error (MAE) **= (1 / n) * Σ |actual — forecast|**
- Symmetric Mean Absolute Percentage Error SMAPE **= (1 / n) * Σ(2 * |actual — forecast| / (|actual| + |forecast|)) * 100%**

Importance of Accuracy Metrics in Time Series Forecasting



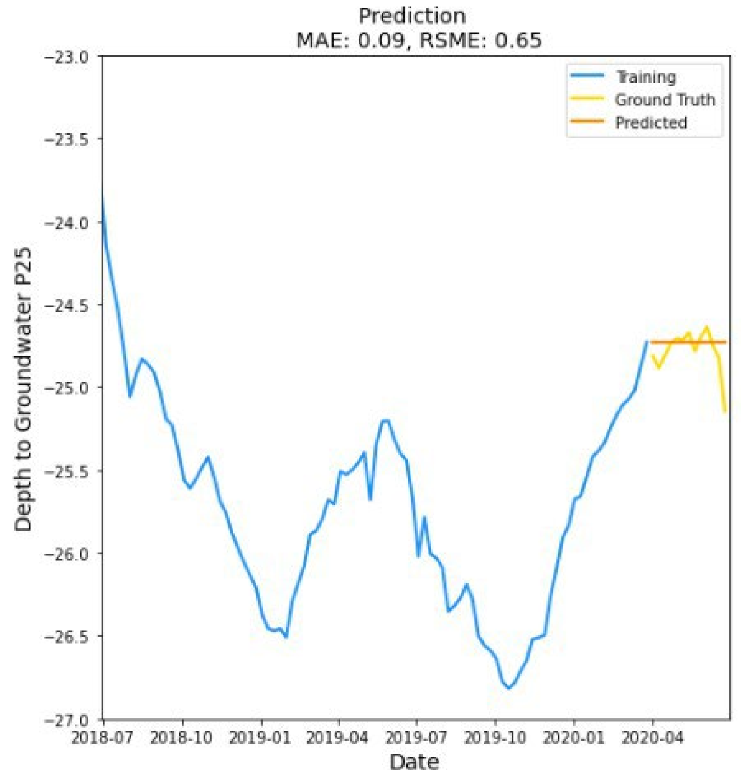
Forecasting Models

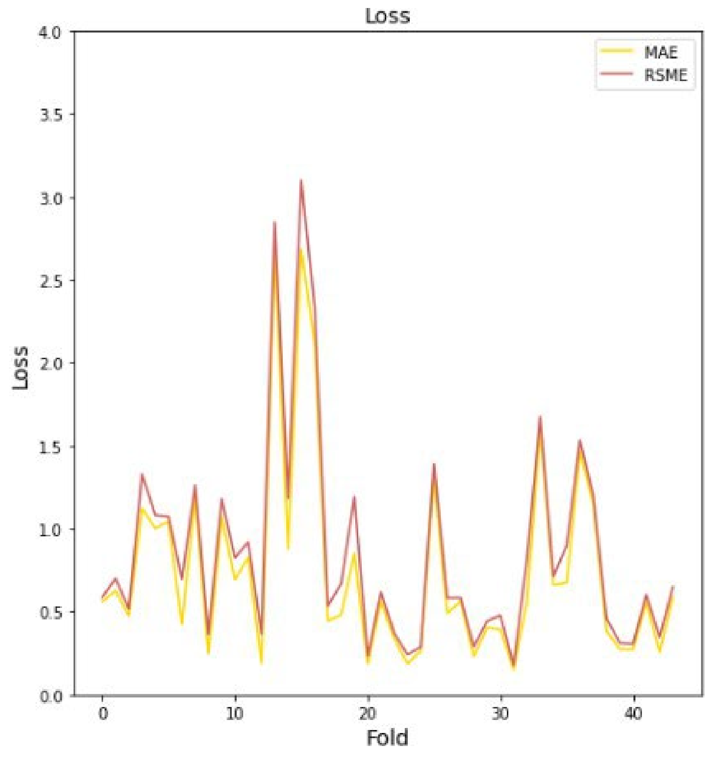
Naïve approach

For naïve forecasts, we set all forecasts as the value of the last observation. That is,

y^t+1=yt

Trend and seasonality are reduced during this transformation. It works well for many economic and financial time series.

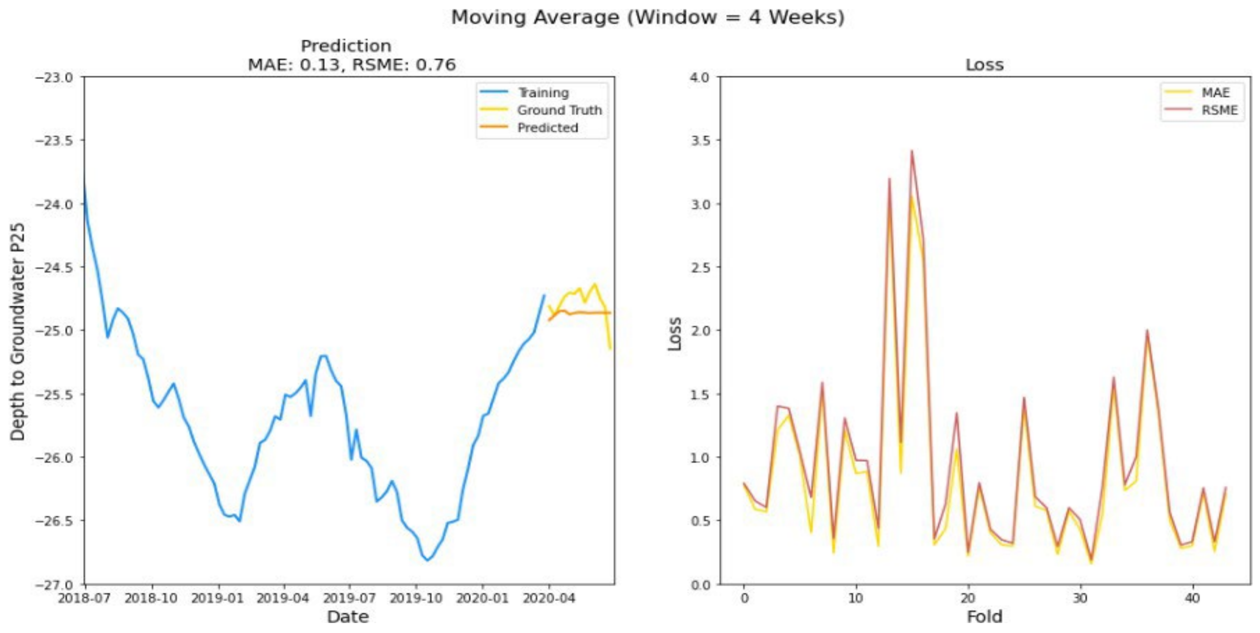




Moving Average

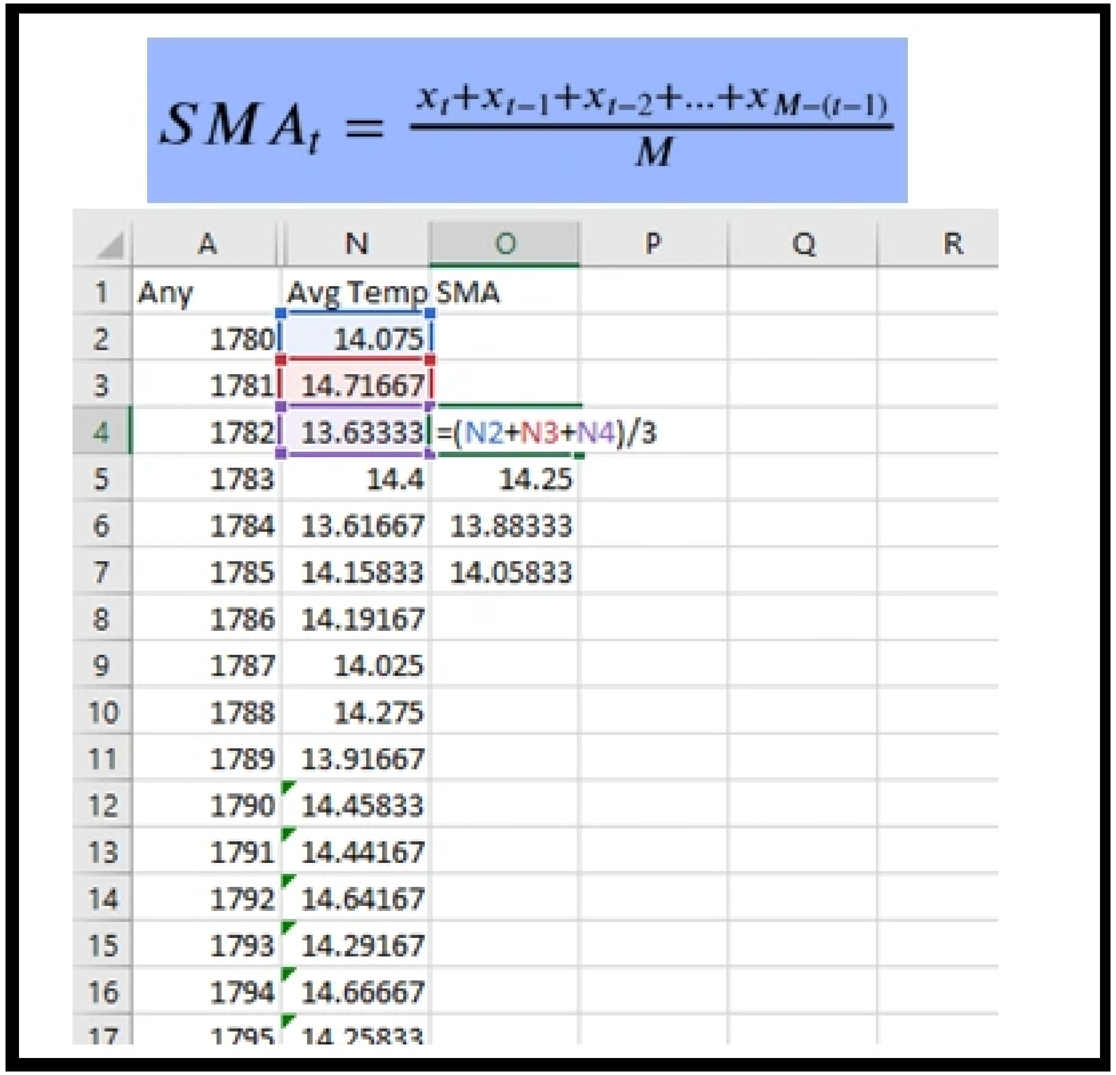
Is a common approach for modelling univariate time series
It involves smoothing data by averaging values over a specific period, revealing underlying trends and patterns while reducing short-term fluctuations.

$$MA(t) = (X(t) + X(t-1) + \dots + X(t-n+1)) / n$$



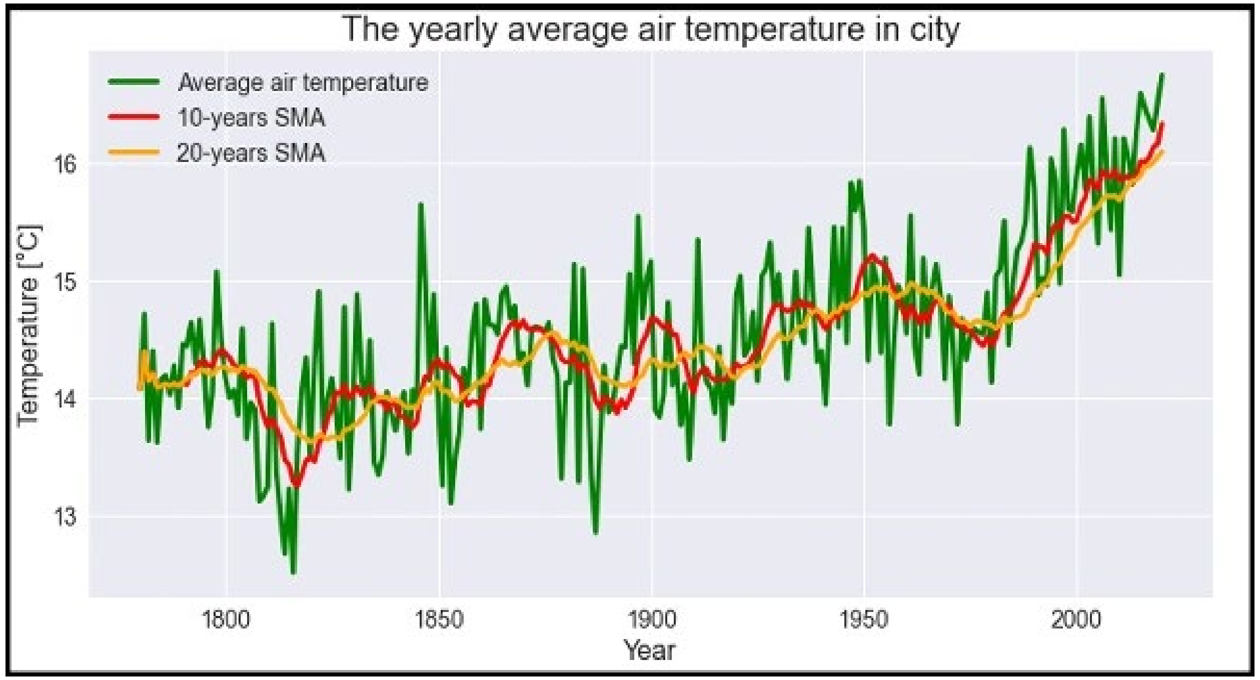
https://medium.com/@anuj_chavan/forecasting-the-future-a-comprehensive-guide-to-moving-averages-in-time-series-analysis-880fa17e877a

Moving Average



$$SMA_t = \frac{x_t + x_{t-1} + x_{t-2} + \ldots + x_{M-(t-1)}}{M}$$

	A	N	O	P	Q	R
1	Any	Avg Temp	SMA			
2	1780	14.075				
3	1781	14.71667				
4	1782	13.63333	=(N2+N3+N4)/3			
5	1783	14.4	14.25			
6	1784	13.61667	13.88333			
7	1785	14.15833	14.05833			
8	1786	14.19167				
9	1787	14.025				
10	1788	14.275				
11	1789	13.91667				
12	1790	14.45833				
13	1791	14.44167				
14	1792	14.64167				
15	1793	14.29167				
16	1794	14.66667				
17	1795	14.25833				



Example code

```python
import numpy as np
import pandas as pd
from sklearn.metrics import mean_absolute_error, mean_squared_error
import math

def rolling_mean_predict(train_series, test_series, window=4):
    """Generate predictions using a rolling mean."""
    predictions = []
    history = list(train_series)  # Initialize with training data
    for i in range(len(test_series)):
        # Use last 'window' observations (or all available if fewer than window)
        pred = pd.Series(history).rolling(window, min_periods=1).mean().iloc[-1]
        predictions.append(pred)
        history.append(test_series.iloc[i])  # Update history with true values (simulating real-world use)

    return pd.Series(predictions, index=test_series.index)

# Time-series cross-validation
score_mae, score_rmse = [], []
for fold, valid_quarter_id in enumerate(range(N_SPLITS)):
    # Split data
    train_mask = df['quarter_idx'] < valid_quarter_id
    valid_mask = df['quarter_idx'] == valid_quarter_id
    X_train, X_valid = X[train_mask], X[valid_mask]
    y_train, y_valid = y[train_mask], y[valid_mask]

    # Predict validation set
    y_valid_pred = rolling_mean_predict(y_train, y_valid, window=4)

    # Compute metrics
    score_mae.append(mean_absolute_error(y_valid, y_valid_pred))
    score_rmse.append(math.sqrt(mean_squared_error(y_valid, y_valid_pred)))
# Predict test set (assuming X_test is available)
y_test_pred = rolling_mean_predict(y, pd.Series(np.zeros(len(X_test))), window=4)
# Plot results
plot_approach_evaluation(y_test_pred, score_mae, score_rmse, 'Moving Average (Window = 4)')
```

ARIMA

The Auto-Regressive Integrated Moving Average (ARIMA) model describes the autocorrelations in the data. The model assumes that the time series is stationary.
It consists of three main parts that involve analysing the autocorrelation function (ACF) and partial autocorrelation function (PACF) plots of the time series data to identify which

1. *Auto-Regressive (AR) filter (long term):*

$$y_t = c + \alpha_1 y_{t-1} + \ldots \alpha_p y_{t-p} + \epsilon_t = c + \sum_{i=1}^{p} \alpha_i y_{t-i} + \epsilon_t \rightarrow \mathrm{p}$$

2. *Integration filter (stochastic trend)* $\rightarrow$ *d (*determined by the number of times the data needs to be differenced to achieve stationarity)

3. *Moving Average (MA) filter (short term):*

$$y_t = c + \epsilon_t + \beta_1 \epsilon_{t-1} + \cdots + \beta_q \epsilon_{t-q} = c + \epsilon_t + \sum_{i=1}^{q} \beta_i \epsilon_{t-i} \rightarrow \mathrm{q}$$

Components of ARIMA

AR (autoregressive term), I (differencing term) and **MA (moving average term)**. Let us understand each of these components –

1. AR term refers to the past values used for forecasting the next value. The AR term is defined by the parameter ‘p’ in arima. The value of ‘p’ is determined using the PACF plot.
2. MA term is used to defines number of past forecast errors used to predict the future values. The parameter ‘q’ in arima represents the MA term. ACF plot is used to identify the correct ‘q’ value.
3. Order of differencing specifies the number of times the differencing operation is performed on series to make it stationary. Test like ADF and KPSS can be used to determine whether the series is stationary and help in identifying the d value.

ARIMA Model

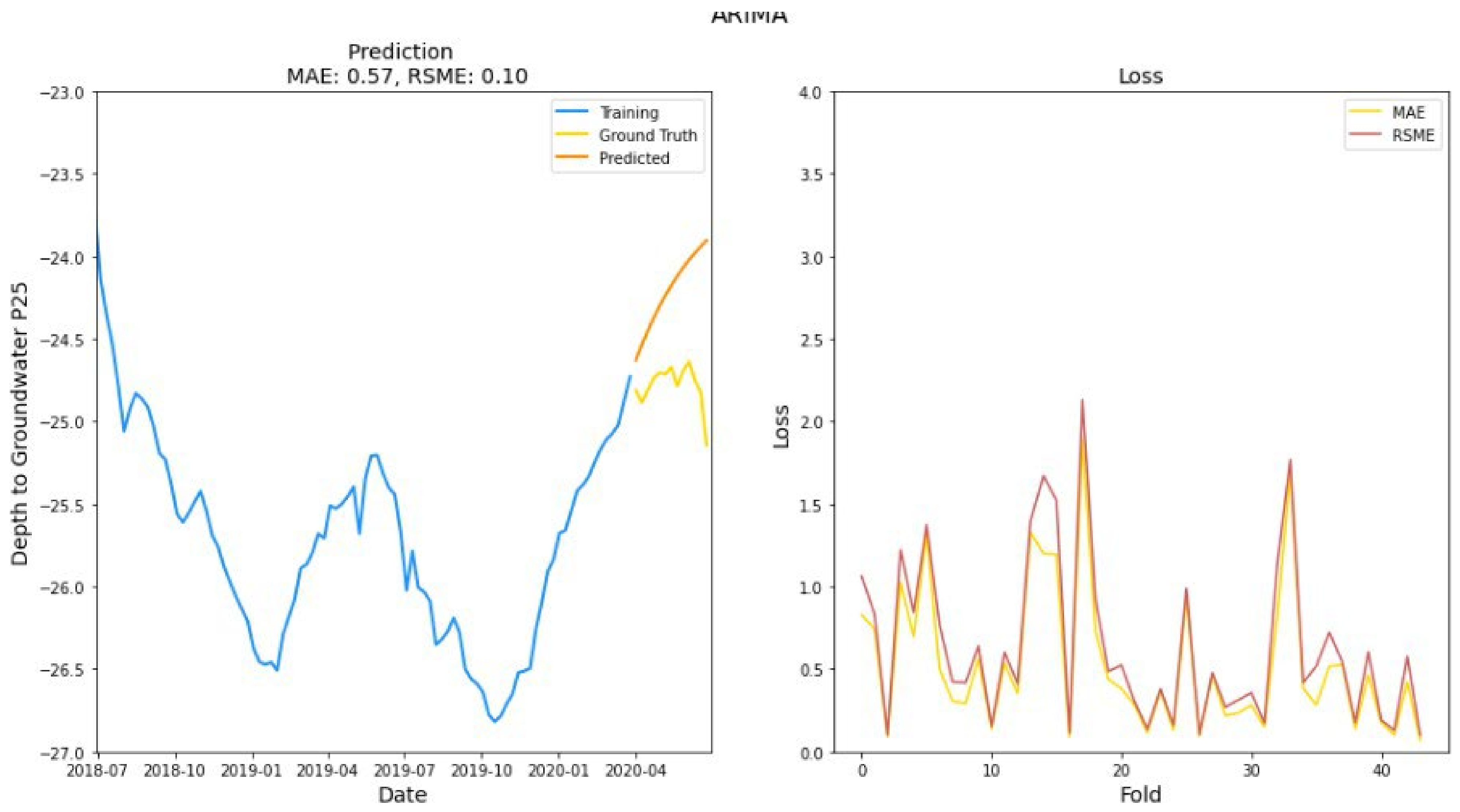
Pros

- Good for short-term forecasting
- Only needs historical data
- Models non-stationary data

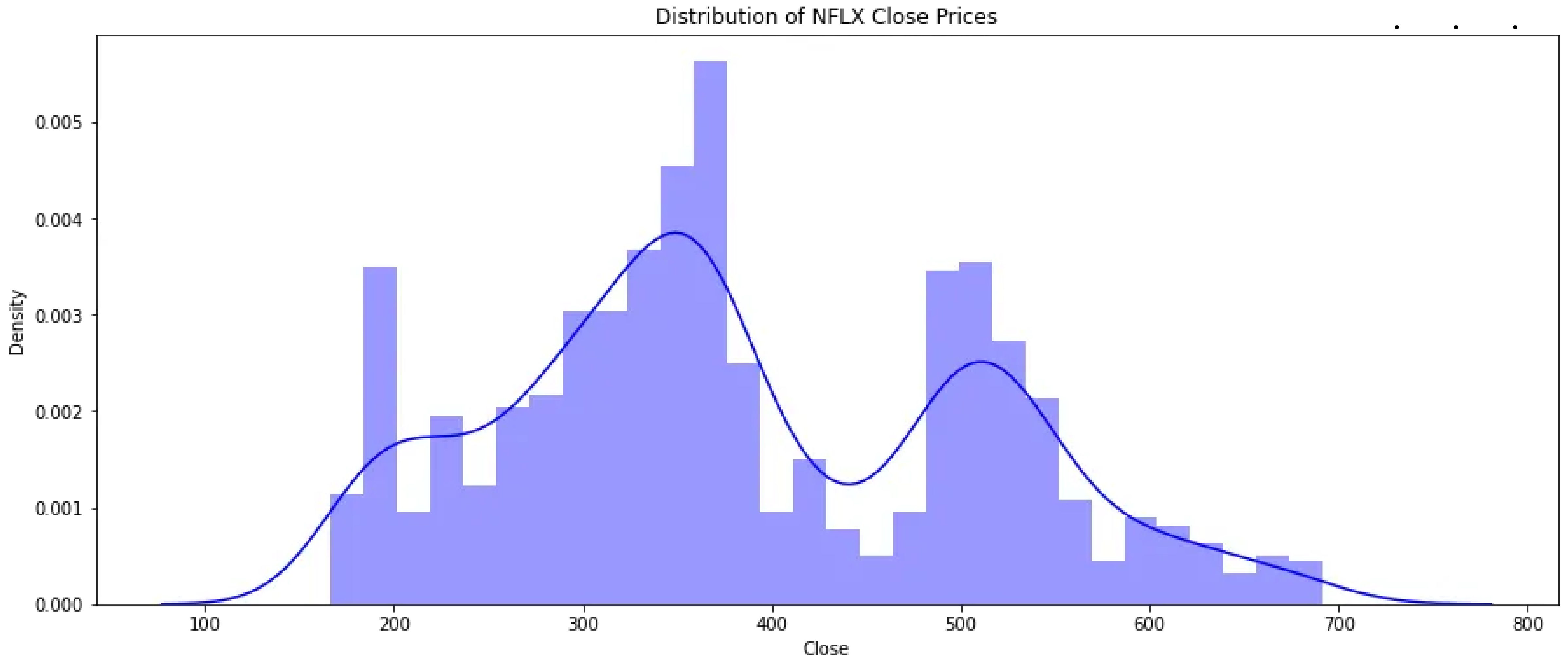
Cons

- Not built for long-term forecasting
- Poor at predicting turning points
- Computationally expensive
- Parameters are subjective

ARIMA Model


ARIMA
Prediction
MAE: 0.57, RSME: 0.10
Training
Ground Truth
Predicted
Depth to Groundwater P25
Date
2018-07
2018-10
2019-01
2019-04
2019-07
2019-10
2020-01
2020-04
−23.0
−23.5
−24.0
−24.5
−25.0
−25.5
−26.0
−26.5
−27.0
Loss
MAE
RSME
Loss
Fold
0
10
20
30
40
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

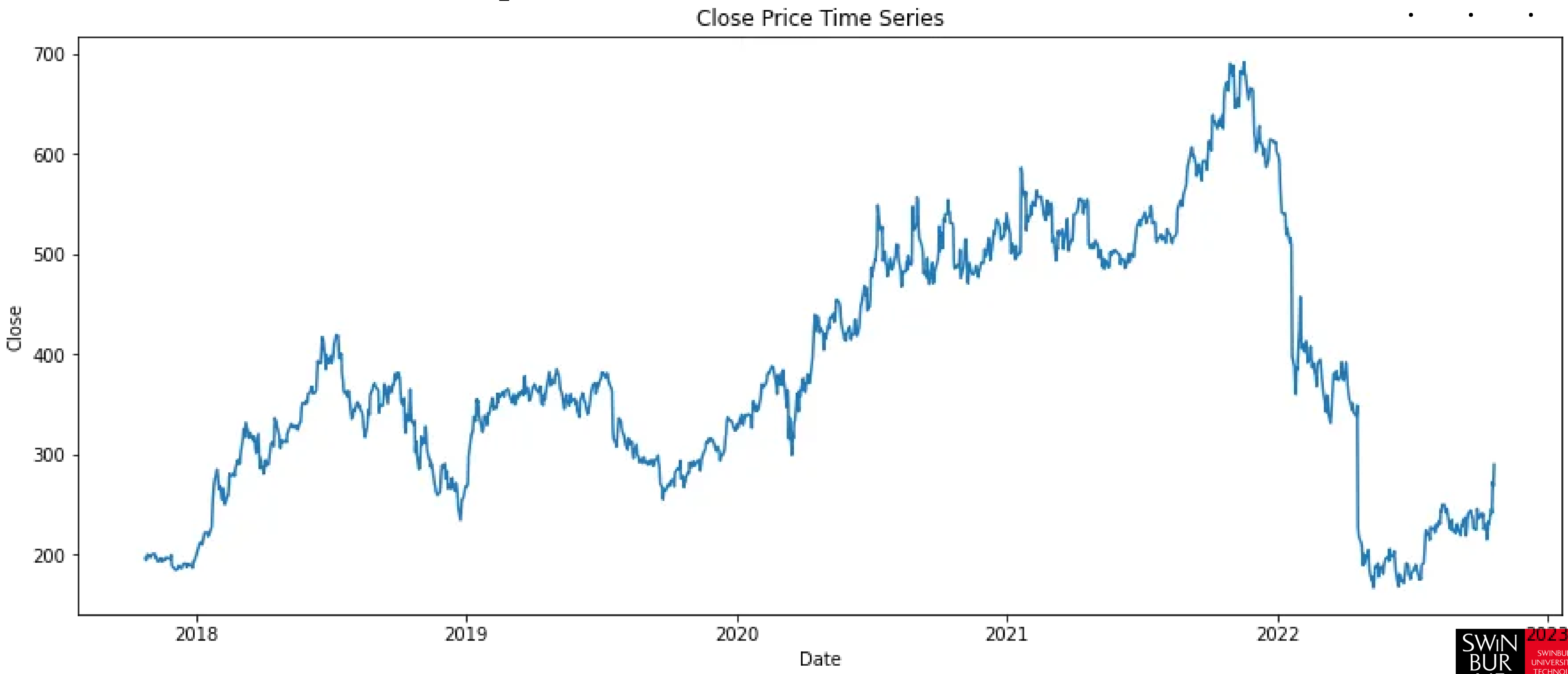
Prediction with Liner Regression



https://medium.com/@sawsanyusuf/linear-regression-with-time-series-data-9186eb1ee607

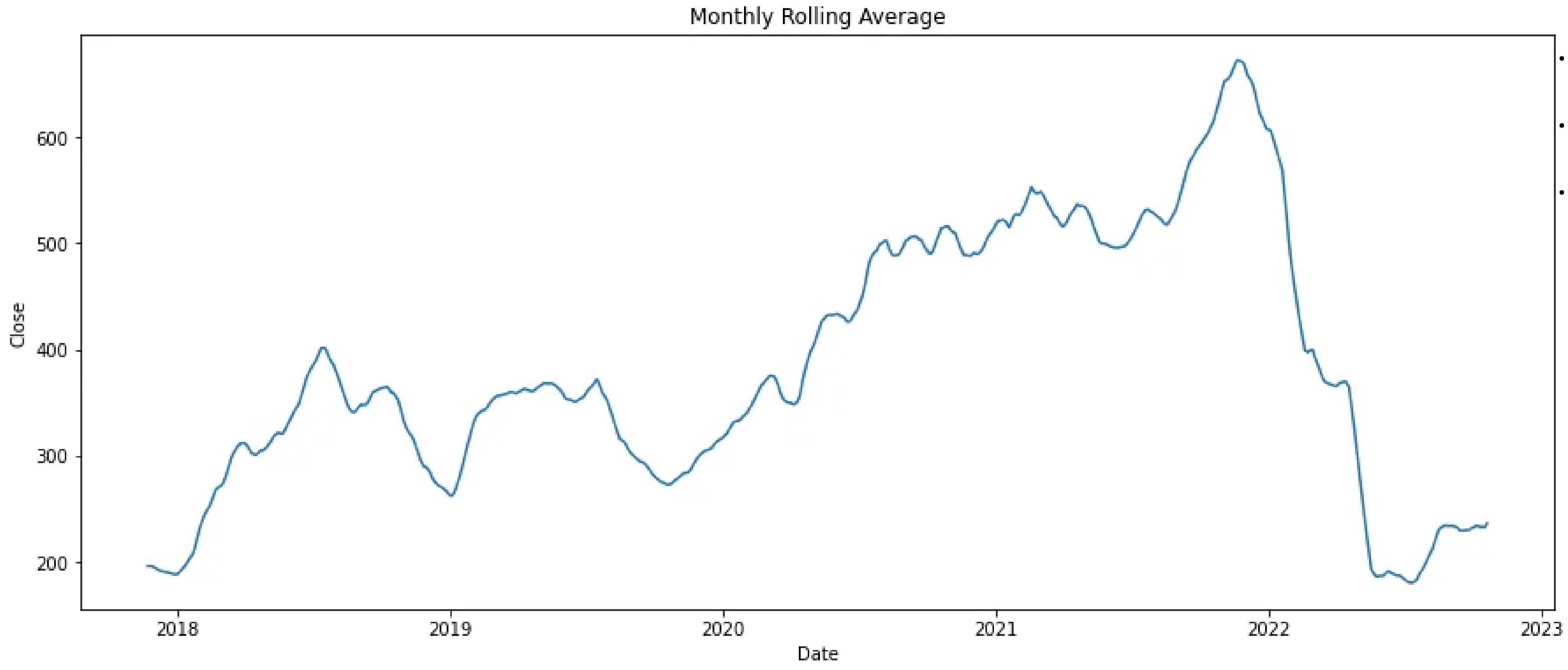
Prediction with Liner Regression

Let’s create a time series plot.



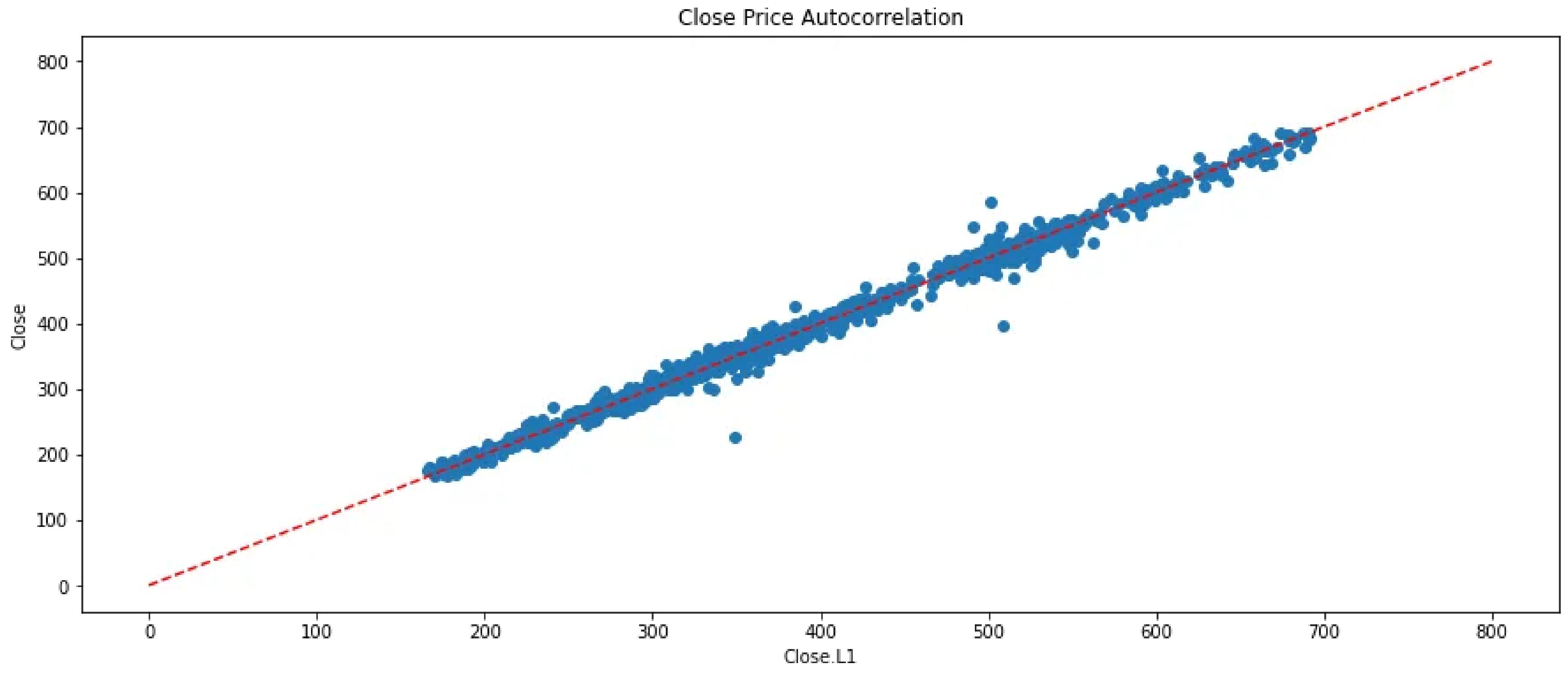
Prediction with Liner Regression

Let's find the general trend for the close price. Use the rolling average with window size of 30 (the number of days in a month).



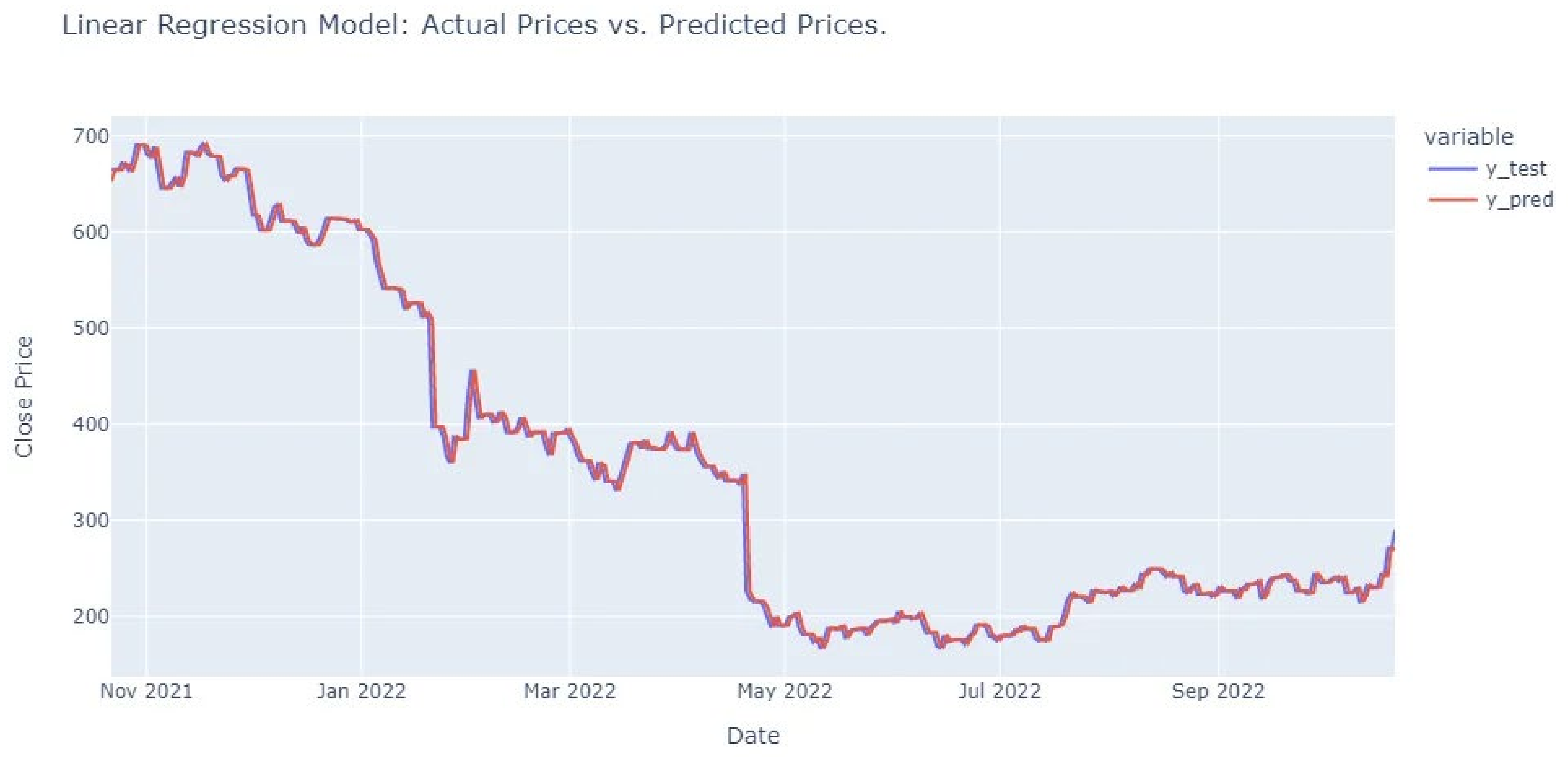
Prediction with Liner Regression

Create a lag by shifting our data, i.e., use the previous day's price to predict the present. Check correlation between target and source variable.



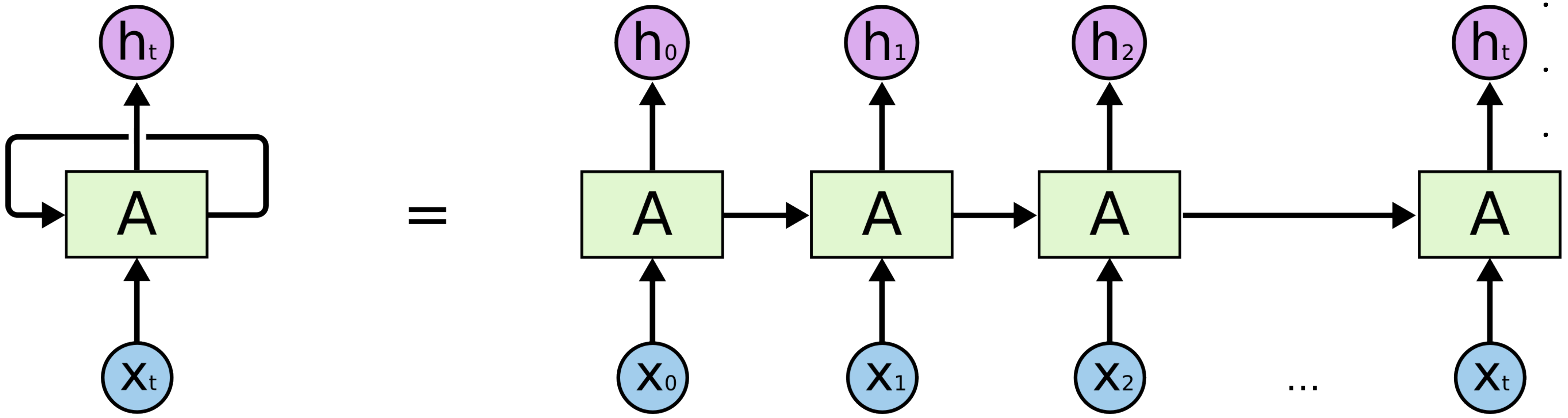
Prediction with Liner Regression

Use 80/20 split to test and train



RNNs

In the diagram, a chunk of neural network, *A*, looks at some input X_t and outputs a value h_t. A loop allows information to be passed from one step of the network to the next.

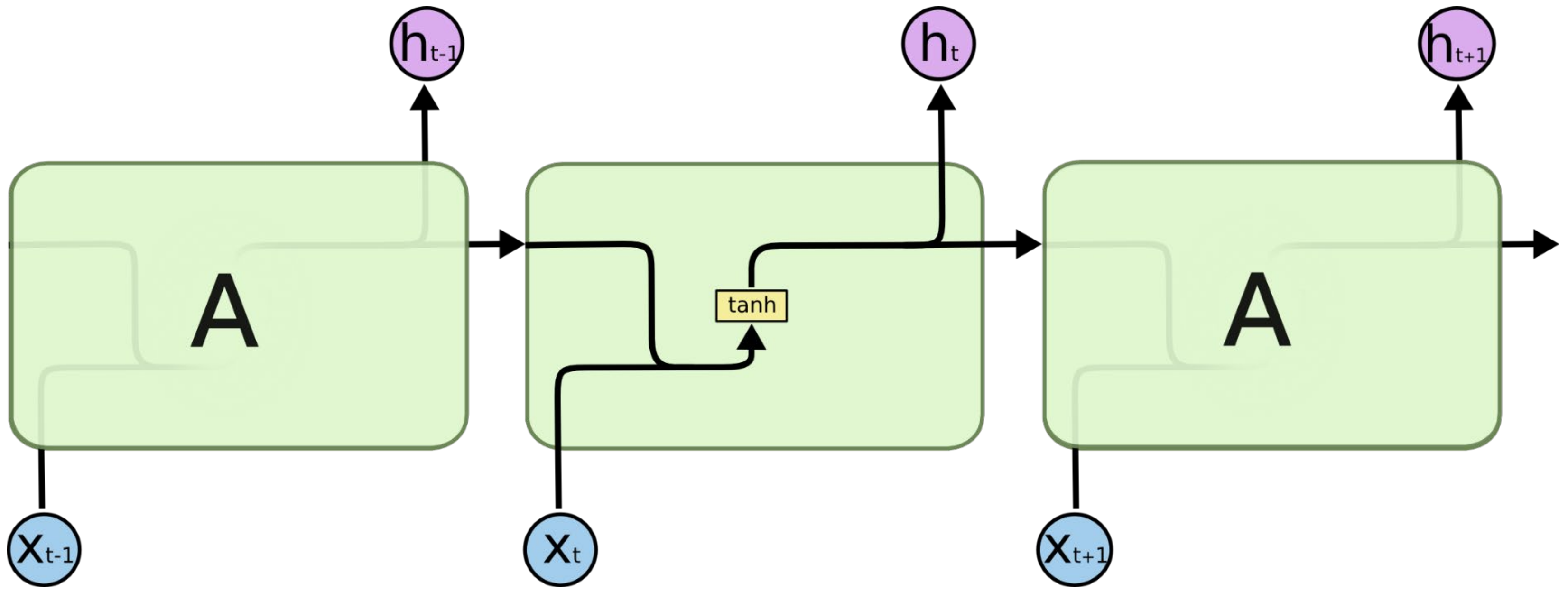


A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor.
RNNs are absolutely capable of handling such “long-term dependencies”, but they can’t.

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM

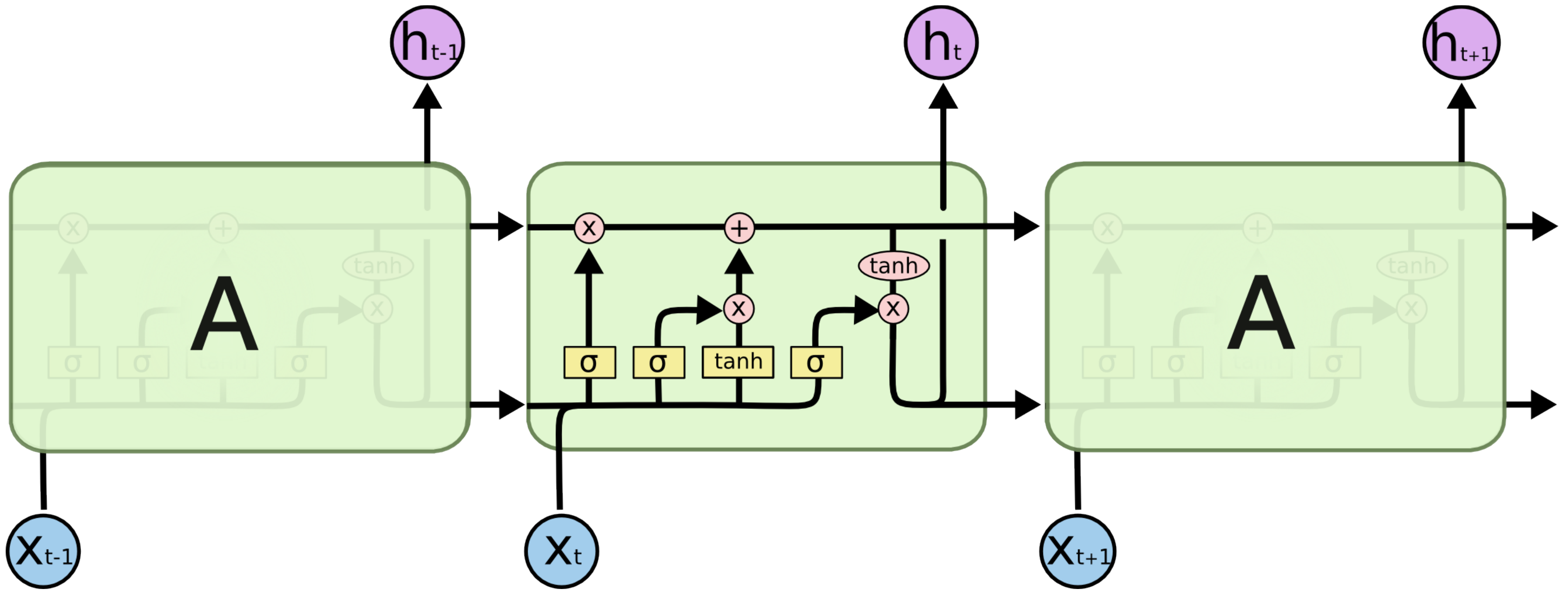
Are a special kind of RNN, capable of learning long-term dependencies. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn!

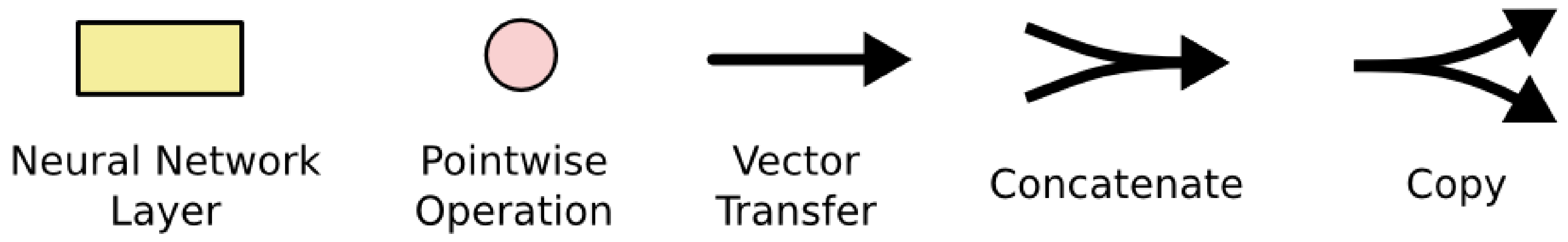


All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.

LSTM

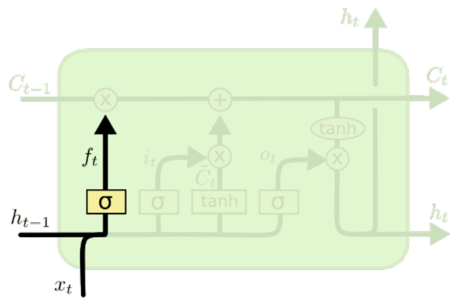
LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four



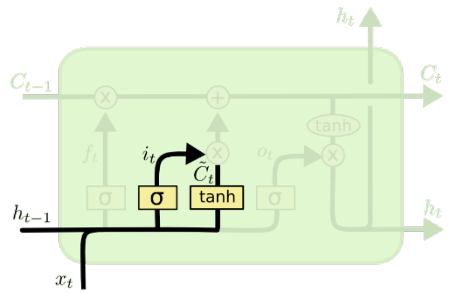


LSTM

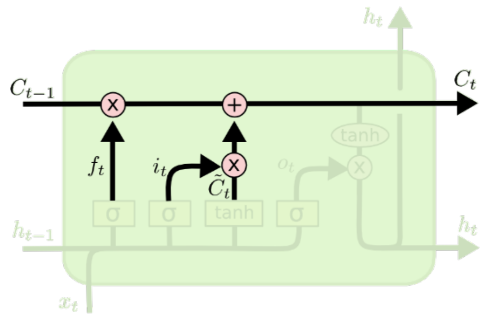
1. The first step in our LSTM is to decide what information we're going to throw away from the cell state.
2. The next step is to decide what new information we're going to store in the cell state. This has two parts. First, a sigmoid layer called the "input gate layer" decides which values we'll update. Next, a tanh layer creates a vector of new candidate values, $C_{\sim t}$, that could be added to the state. In the next step, we'll combine these two to create an update to the state.
3. It's now time to update the old cell state, C_{t-1}, into the new cell state C_t. We multiply the old state by f_t, forgetting the things we decided to forget earlier. Then we add $i_t * C_{\sim t}$. This is the new candidate values, scaled by how much we decided to update each state value.
4. This output will be based on our cell state but will be a filtered version. First, we run a sigmoid layer which decides what parts of the cell state we're going to output. Then, we put the cell state through tanh (to push the values to be between −1 and 1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.



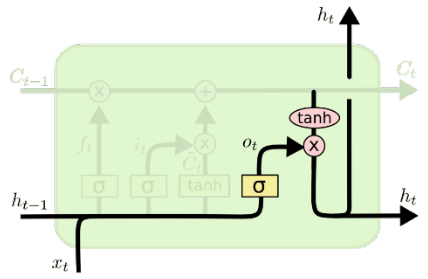
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$



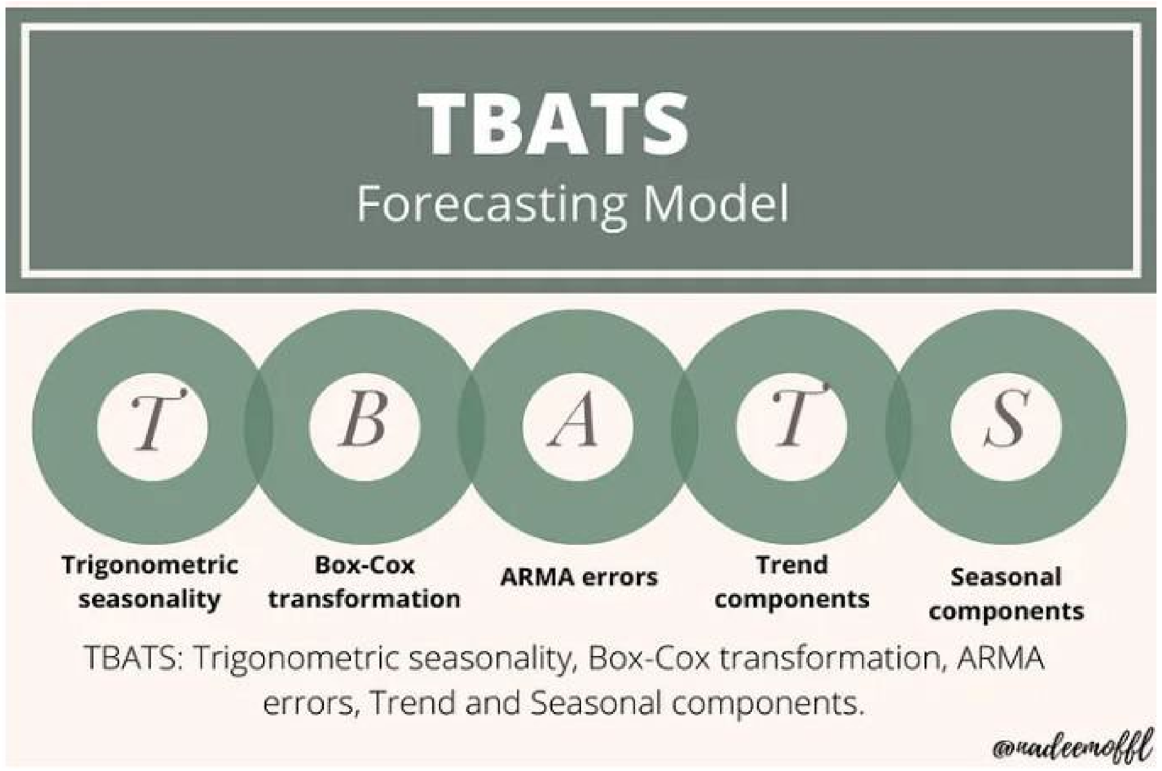
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

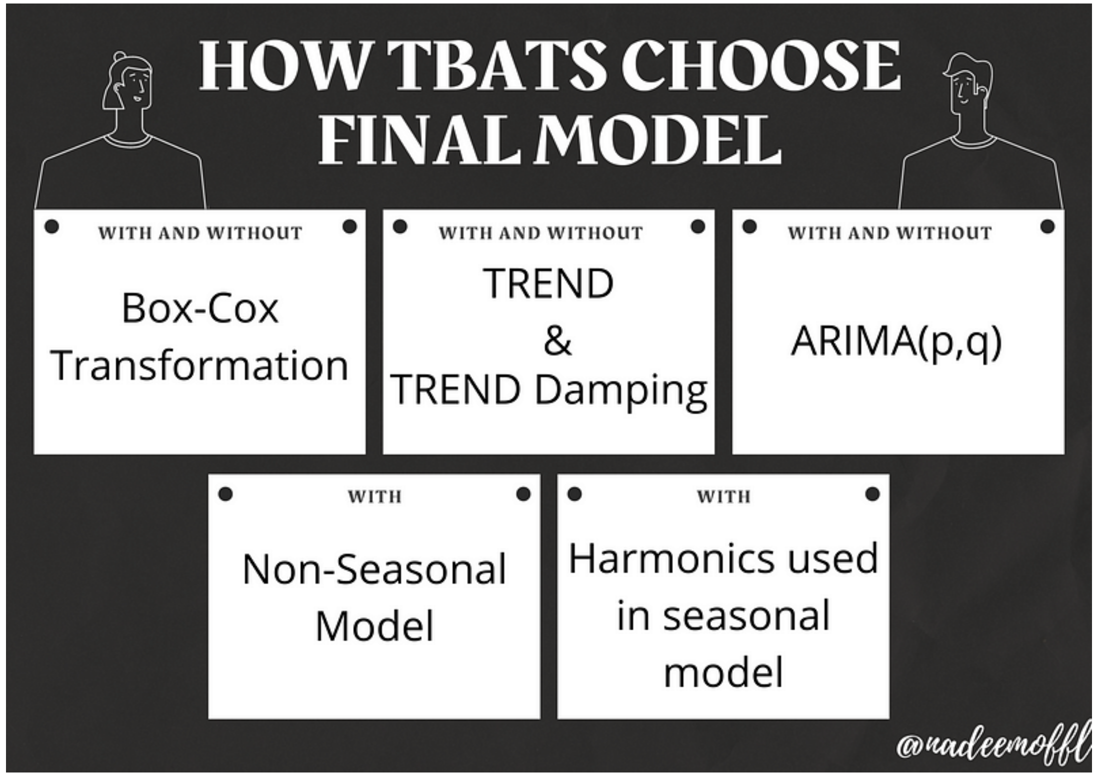


$$o_t = \sigma\left(W_o \ [h_{t-1}, x_t] \ + \ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

TBATS

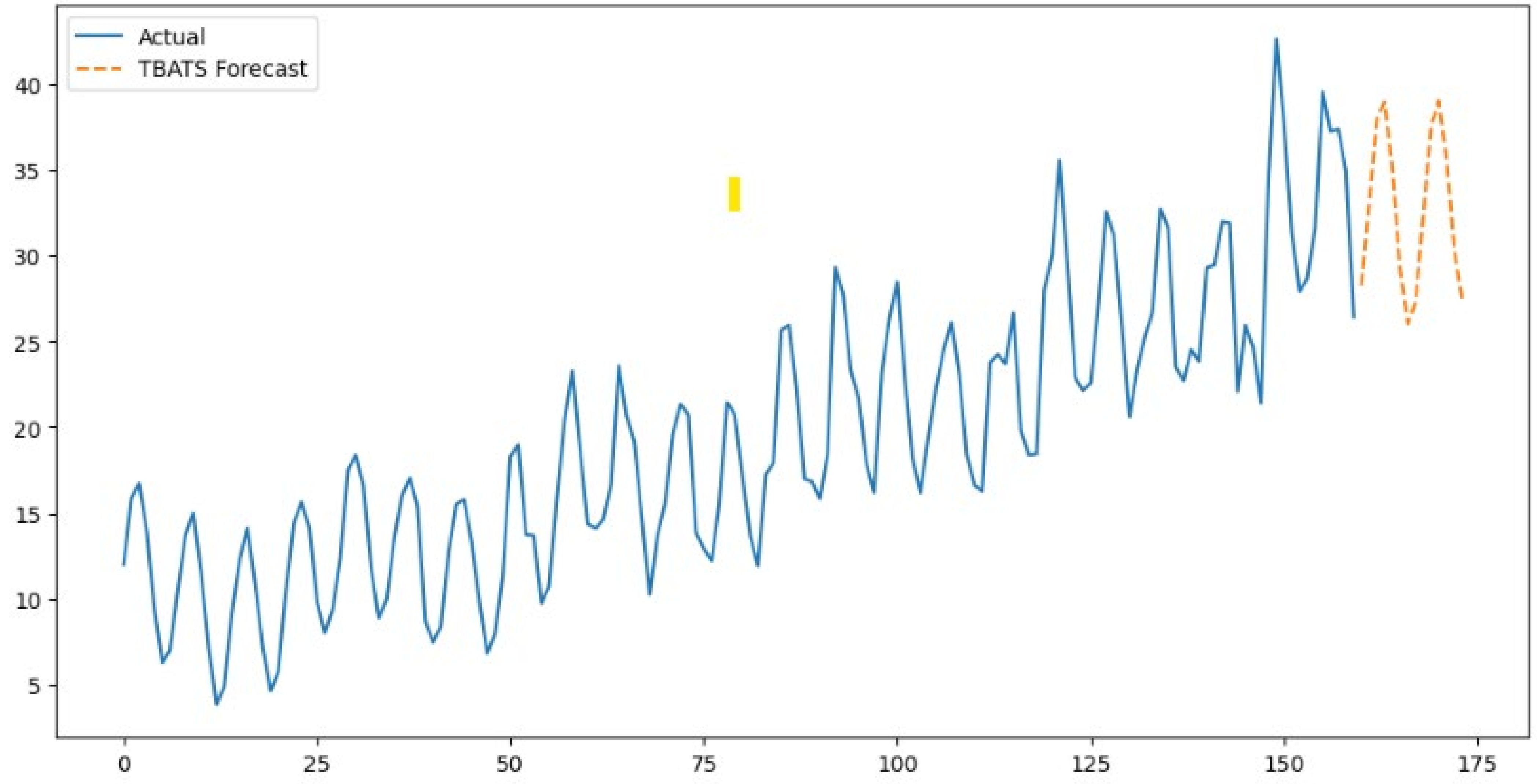
https://medium.com/analytics-vidhya/time-series-forecasting-using-tbats-model-ce8c429442a9


TBATS
Forecasting Model
T
B
A
T
S
Trigonometric seasonality
Box-Cox transformation
ARMA errors
Trend components
Seasonal components
TBATS: Trigonometric seasonality, Box-Cox transformation, ARMA errors, Trend and Seasonal components.
@nadeemoffl



```python
import numpy as np
from tbats import TBATS

# Generate synthetic data
np.random.seed(2342)
t = np.arange(0, 160)
y = (
    5 * np.sin(t * 2 * np.pi / 7) +               # Weekly seasonality (7-day)
    2 * np.cos(t * 2 * np.pi / 30.5) +            # Monthly seasonality (~30.5-day)
    (t / 20) ** 1.5 +                             # Non-linear trend
    np.random.normal(size=160) * t / 50 +          # Heteroscedastic noise
    10                                            # Offset
)

# Initialize TBATS with correct seasonal periods
estimator = TBATS(
    seasonal_periods=[7, 30.5],  # Match data generation
    use_box_cox=True,            # Handle heteroscedasticity
    use_trend=True,              # Capture trend
    use_arma_errors=True         # Model residuals
)

# Fit model
fitted_model = estimator.fit(y)
# Forecast 14 steps ahead
y_forecast = fitted_model.forecast(steps=14)

# Summarize model
print(fitted_model.summary())
# Plot results
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 6))
plt.plot(t, y, label='Actual')
plt.plot(np.arange(160, 174), y_forecast, label='TBATS Forecast', linestyle='--')
plt.legend()
plt.show()
```

TBATS Plot


Actual
TBATS Forecast
40
35
30
25
20
15
10
5
0
25
50
75
100
125
150
175

Learn, Practice and Enjoy the AI journey