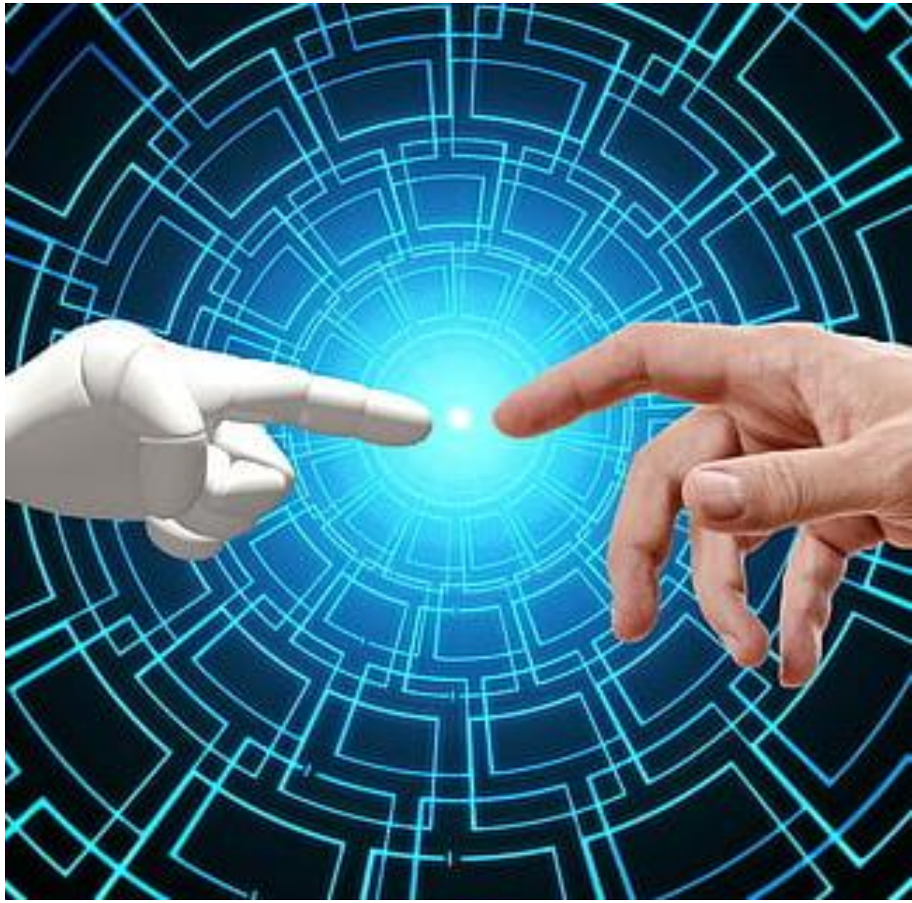
Artificial Intelligence (AI) for Engineering



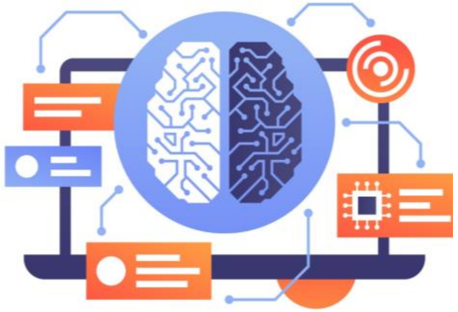
COS40007

Dr. Abdur Forkan

Senior Research Fellow, AI and Machine Learning

Digital Innovation Lab

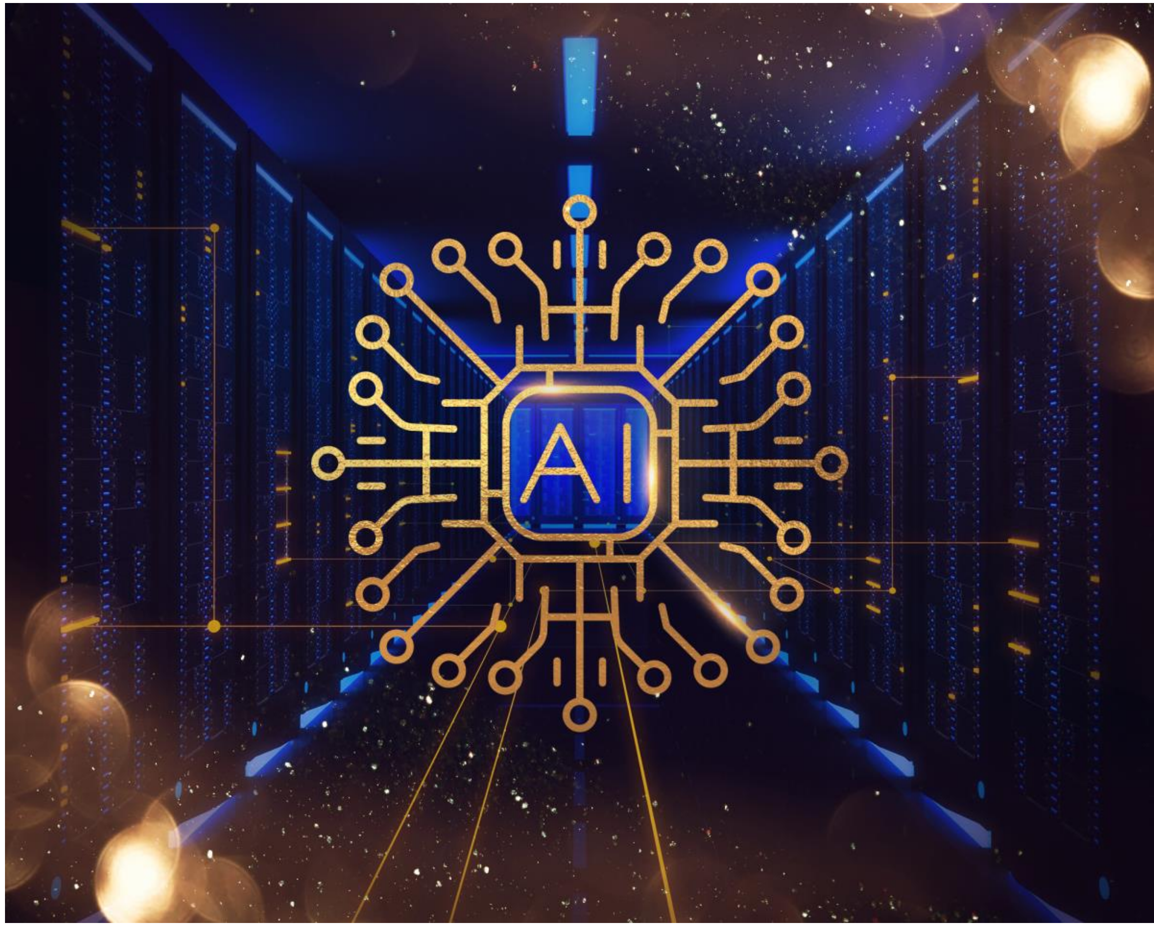
Seminar 7: 16th September 2024



SWINBURNE UNIVERSITY OF TECHNOLOGY

Overview

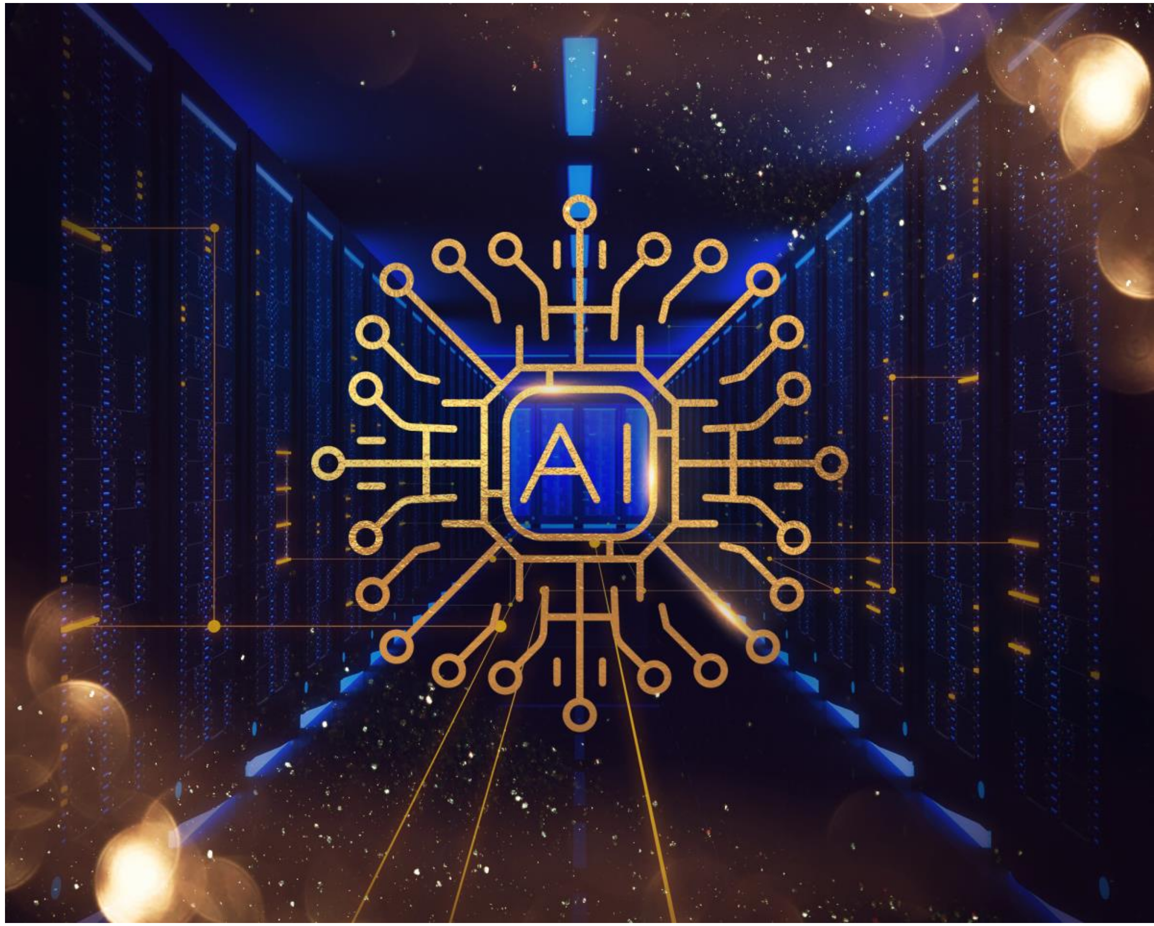
- ❑Basics of time-series
- ❑Forecasting model
- ❑Regression for forecasting
- ❑Forecasting algorithms
- ❑Forecasting using LSTM



SWINBURNE UNIVERSITY OF TECHNOLOGY

Required Reading

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



SWINBURNE UNIVERSITY OF TECHNOLOGY

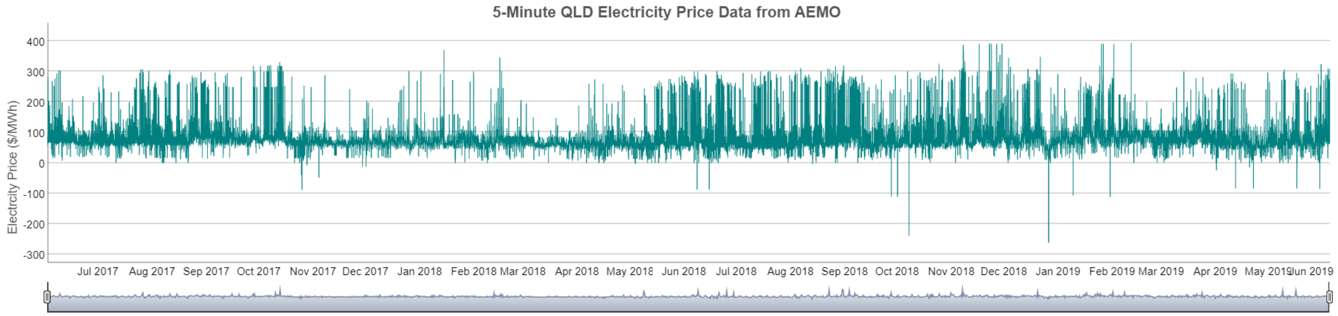
At the end of this you should be able to

- Understand about 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)

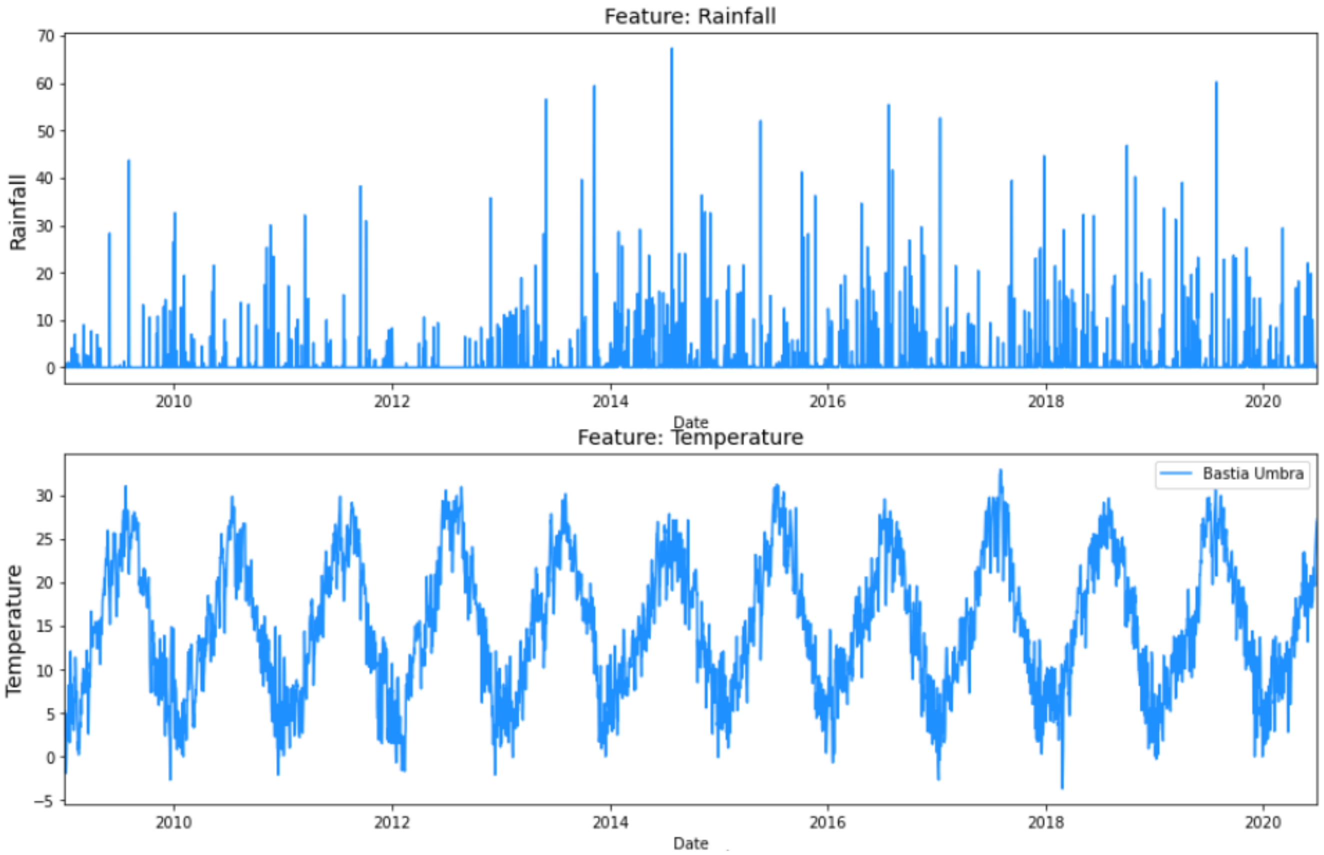
Time –Series Forecasting

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

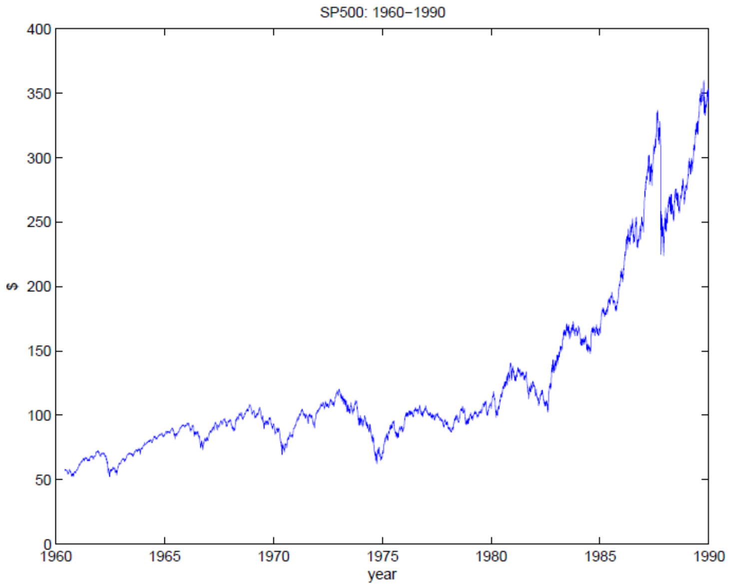


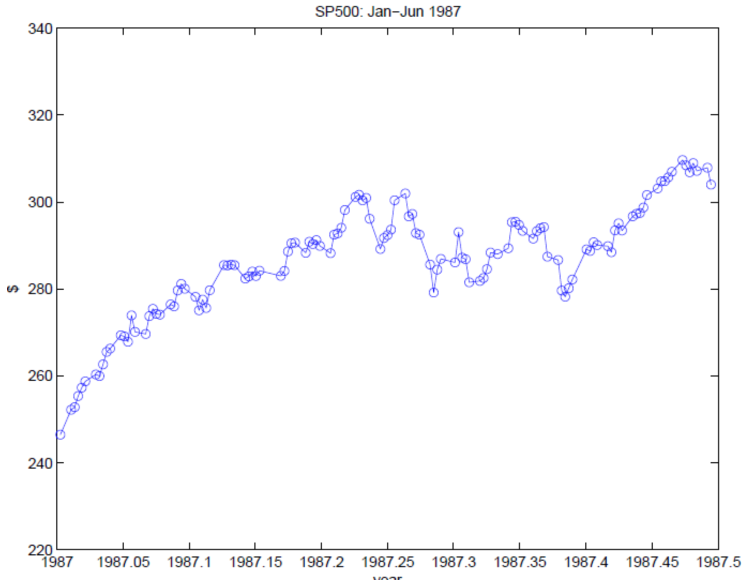
Time-series

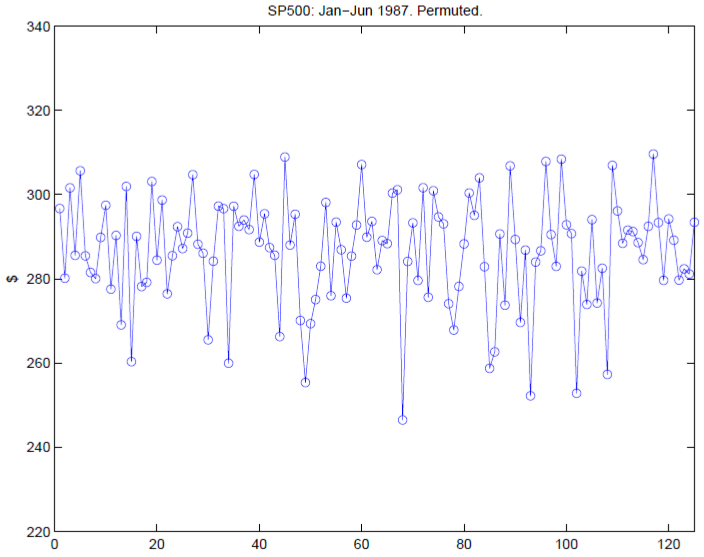


The data should be in **chronological order** and the **timestamps should be equidistant (1 sec/1 min/1 hour/1 day)** in time series.

Example of time-series







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

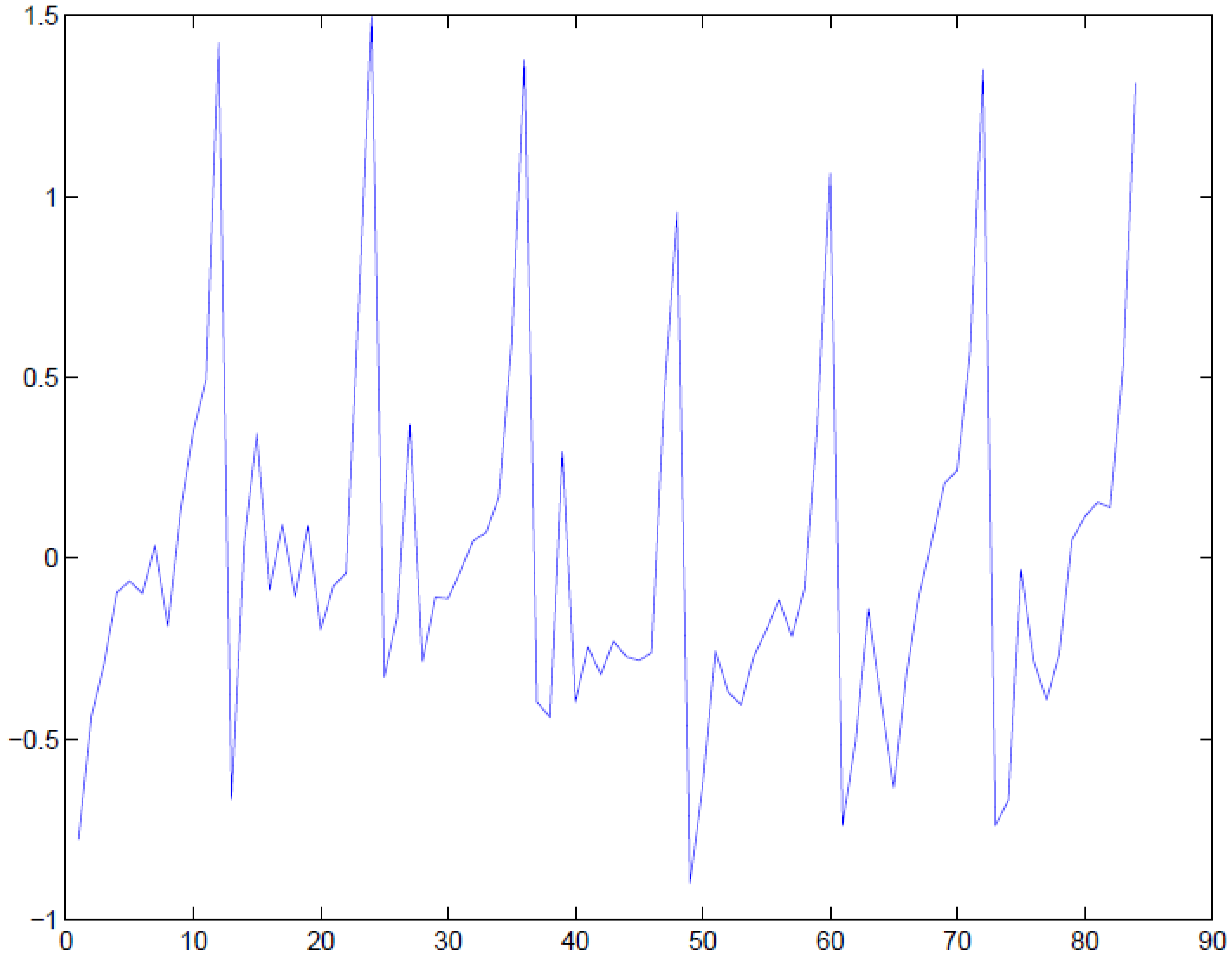
Converting Nonstationary Time Series to Stationary Time Series

- Remove deterministic factors
  - Trends
    - Polynomial regression fitting
    - Exponential smoothing
    - Moving average smoothing
    - Differencing (B is a back shift operator

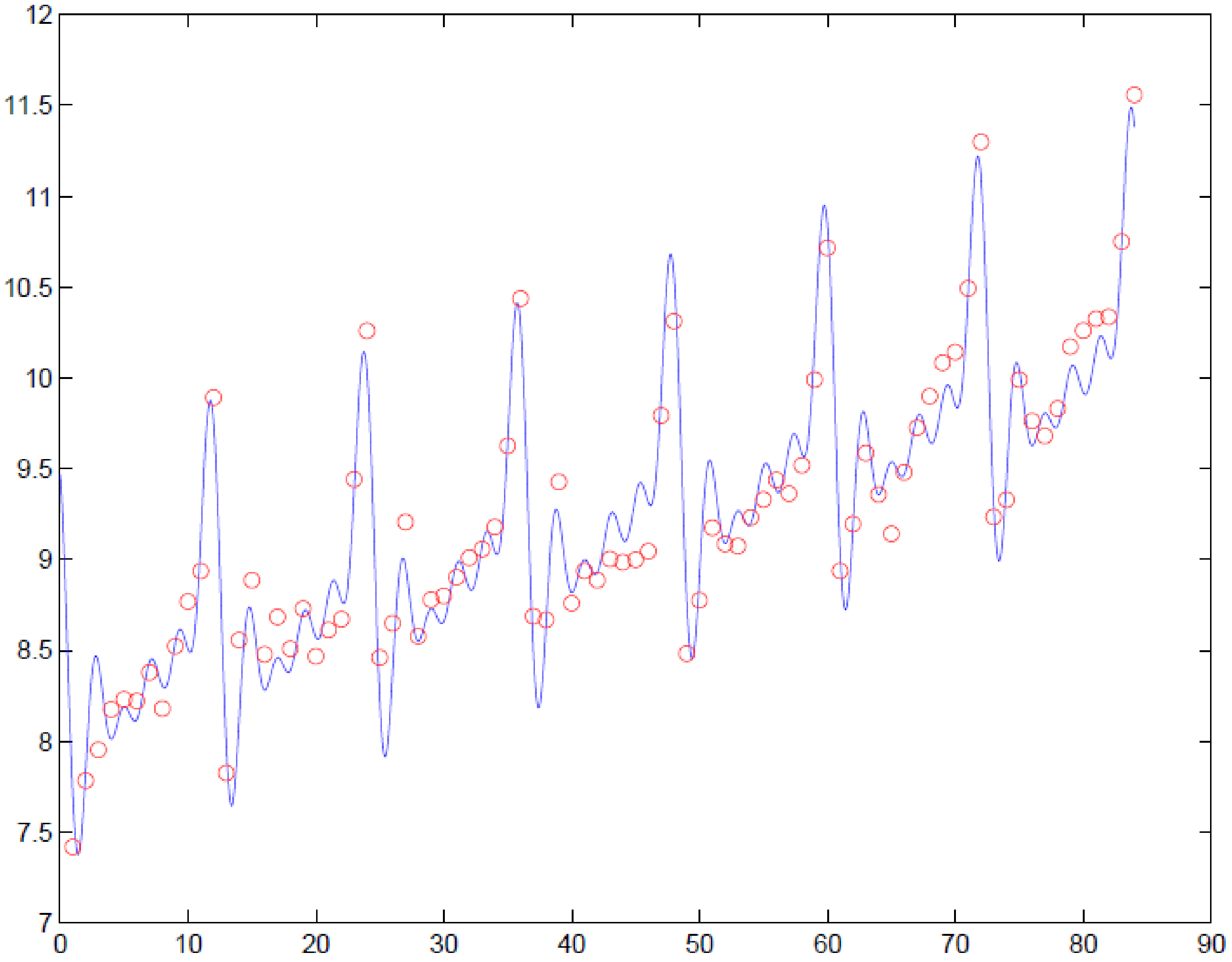
- After conversion, remaining data points are called residuals
- If residuals are IID, then no more analysis is necessary since its mean value will be the best predictor

SWINBURNE UNIVERSITY OF TECHNOLOGY

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 splitted into training and test sets.
- Unlike other dataset, usually time series data are splitted without shuffling. That is, the training set is the first half of 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:]
```

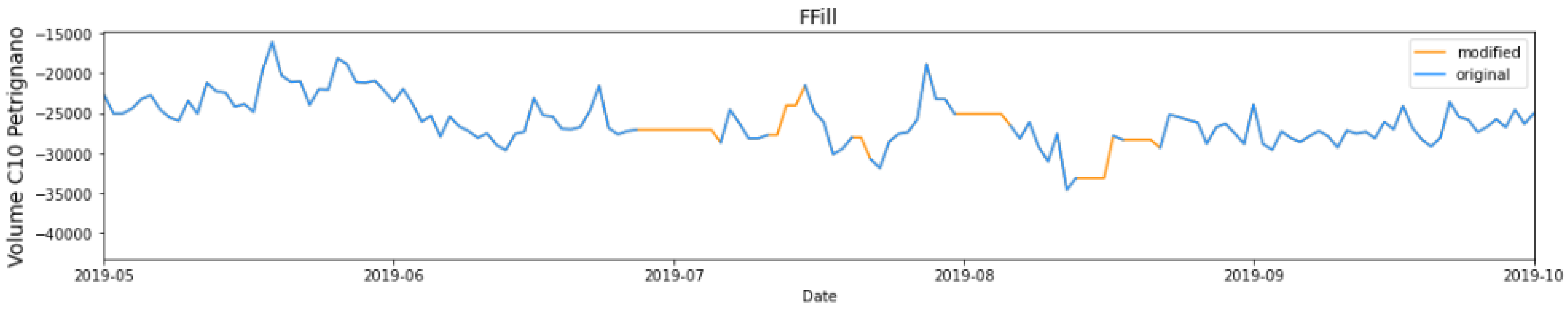
Lookback

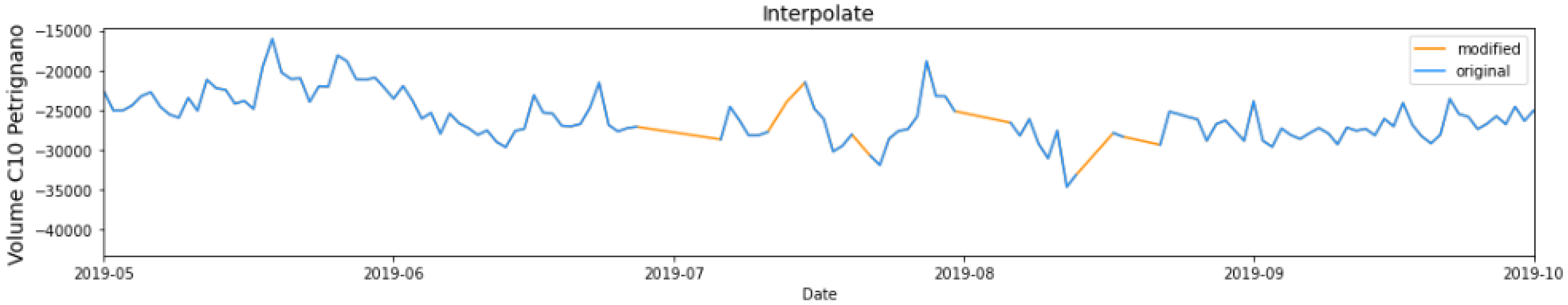
On a long enough time series, multiple overlapping window can be created. It is convenient to create a function to generate a dataset of fixed window from a time series.

```
lookback = 1
X_train, y_train = create_dataset(train,
lookback=lookback)
X_test, y_test = create_dataset(test,
lookback=lookback)
```

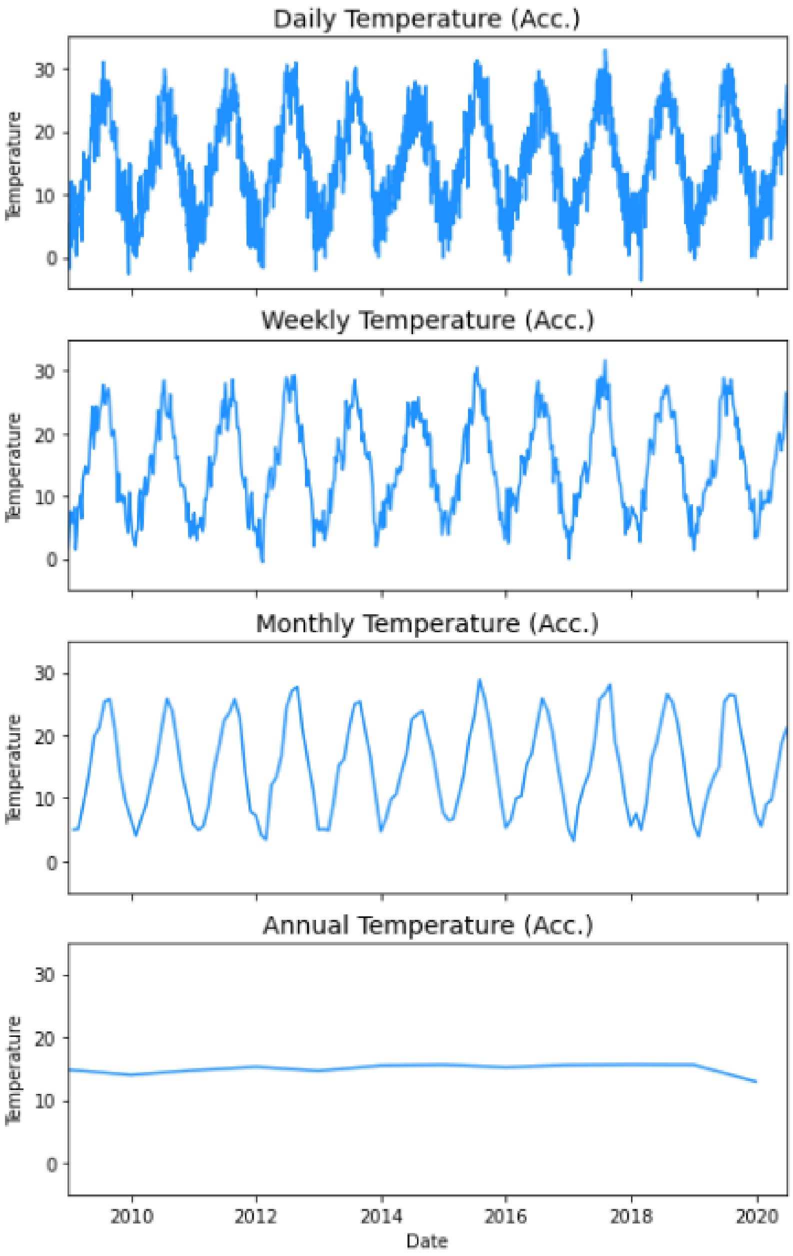
```
import torch

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
    """
    X, y = [], []
    for i in range(len(dataset)-lookback):
        feature = dataset[i:i+lookback]
        target = dataset[i+1:i+lookback+1]
        X.append(feature)
        y.append(target)
    return torch.tensor(X), torch.tensor(y)
```

Handling missing value





Resampling



Daily Temperature (Acc.)
Weekly Temperature (Acc.)
Monthly Temperature (Acc.)
Annual Temperature (Acc.)
Temperature
30
20
10
0
2010
2012
2014
2016
2018
2020
Date

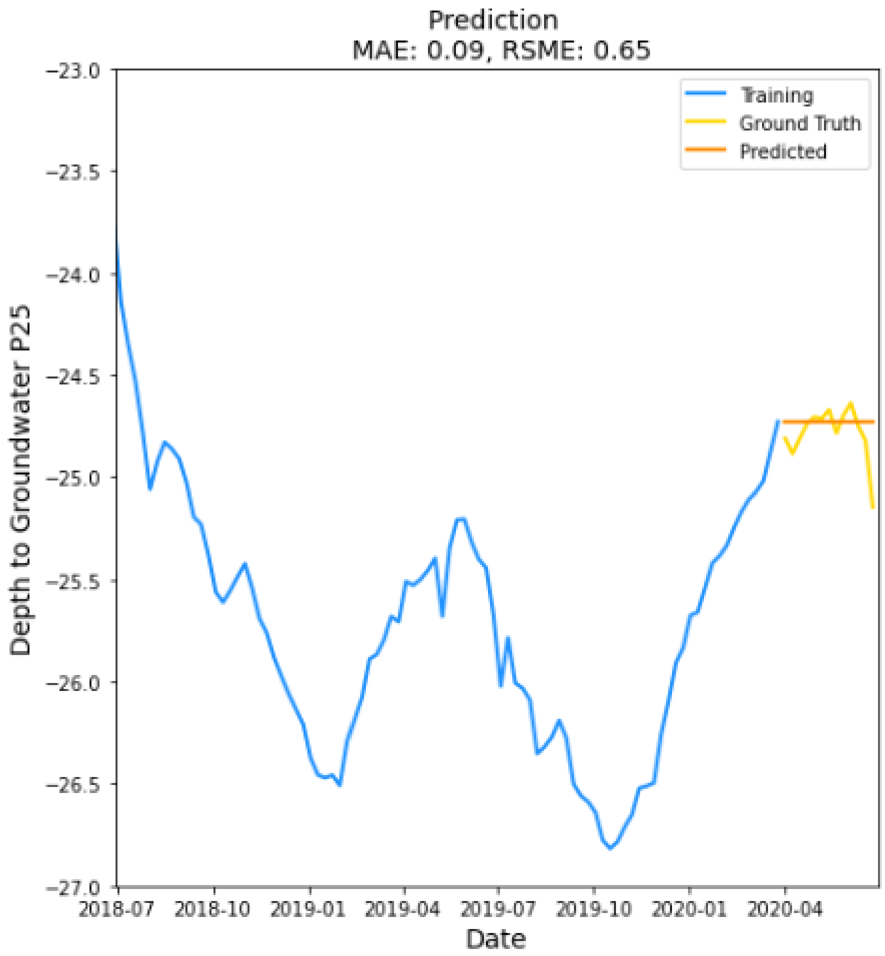
Time-series prediction evaluation

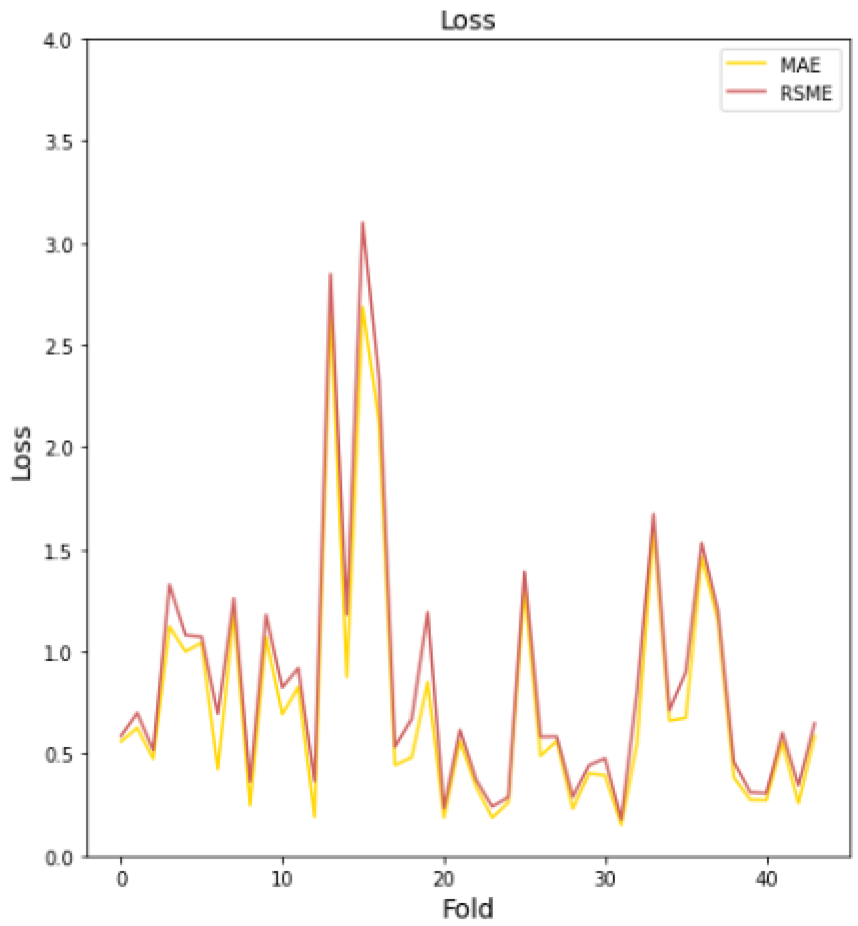
- Root mean square error (RMSE)
- Mean average error (MAE)

Forecasting models

Naïve approach

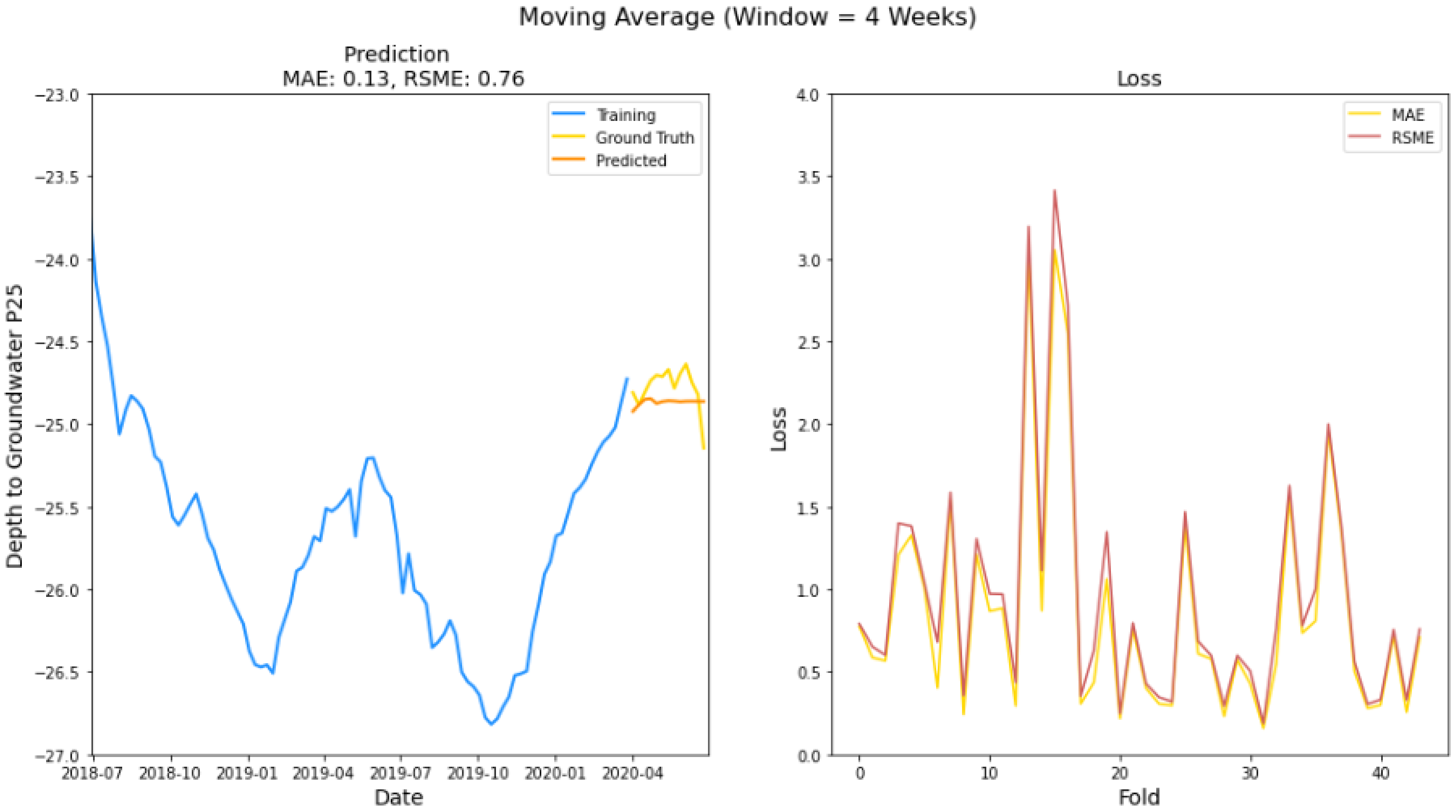
y^t+1=yt





SWINBURNE UNIVERSITY OF TECHNOLOGY

Moving average



Example code

```
score_mae = []
score_rsme = []
for fold, valid_quarter_id in enumerate(range(2, N_SPLITS)):
    # Get indices for this fold
    train_index = df[df.quarter_idx < valid_quarter_id].index
    valid_index = df[df.quarter_idx == valid_quarter_id].index

    # Prepare training and validation data for this fold
    #X_train, X_valid = X.iloc[train_index], X.iloc[valid_index]
    y_train, y_valid = y.iloc[train_index], y.iloc[valid_index]

    # Initialize y_valid_pred
    y_valid_pred = pd.Series(np.ones(len(y_valid)))



    for i in range(len(y_valid_pred)):
        y_valid_pred.iloc[i] = y_train.append(y_valid_pred.iloc[:(i)]).reset_index(drop=True).rolling(4).mean().iloc[-1]

    # Calcuate metrics
    score_mae.append(mean_absolute_error(y_valid, y_valid_pred))
    score_rsme.append(math.sqrt(mean_squared_error(y_valid, y_valid_pred)))

y_pred = pd.Series(np.zeros(len(X_test)))

for i in range(len(y_pred)):
    y_pred.iloc[i] = y.append(y_pred.iloc[:(i)]).reset_index(drop=True).rolling(4).mean().iloc[-1]

plot_approach_evaluation(y_pred, score_mae, score_rsme, 'Moving Average (Window = 4 Weeks)')
```

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:

Auto-Regressive (AR) filter (long term):

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

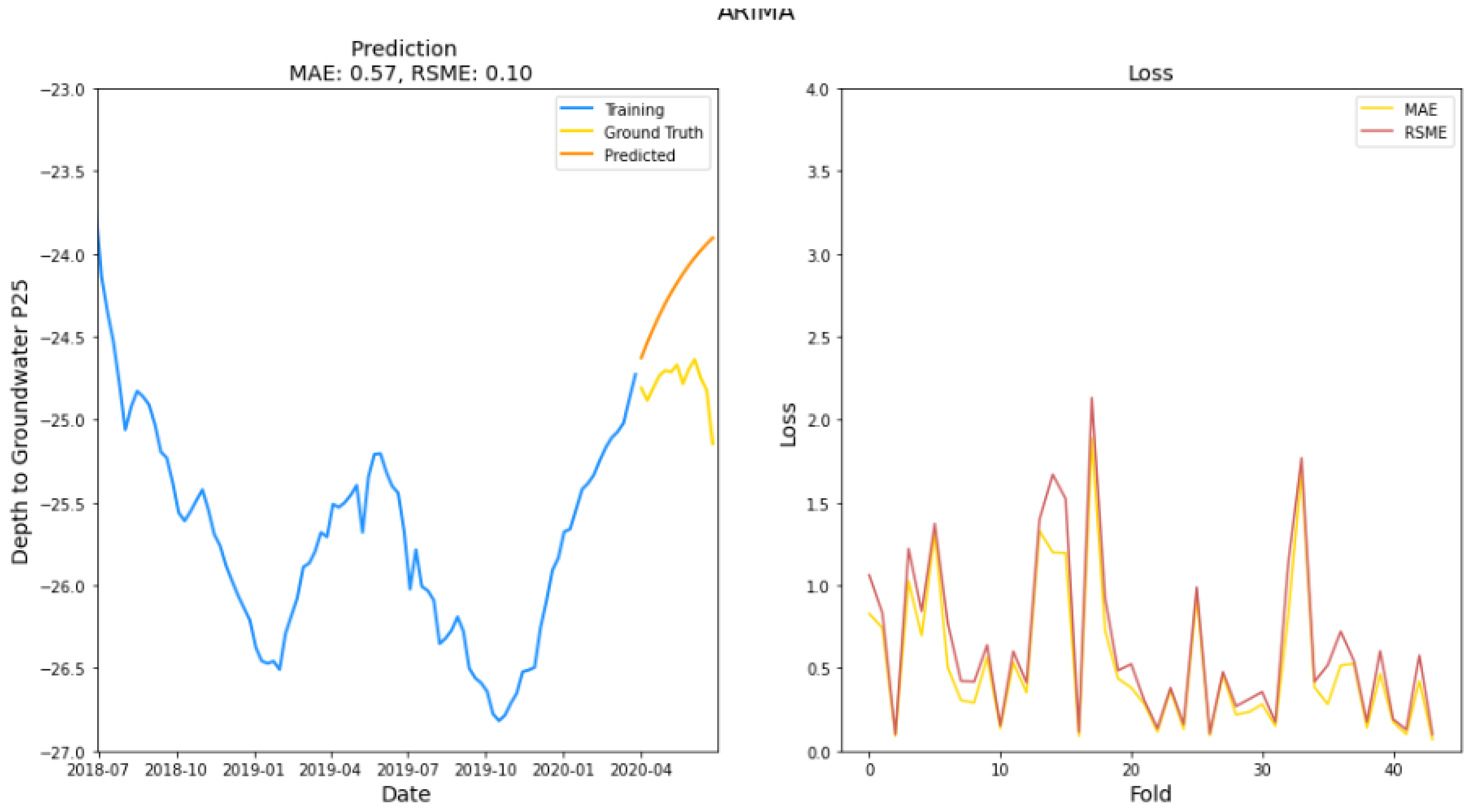
Integration filter (stochastic trend)

-> d

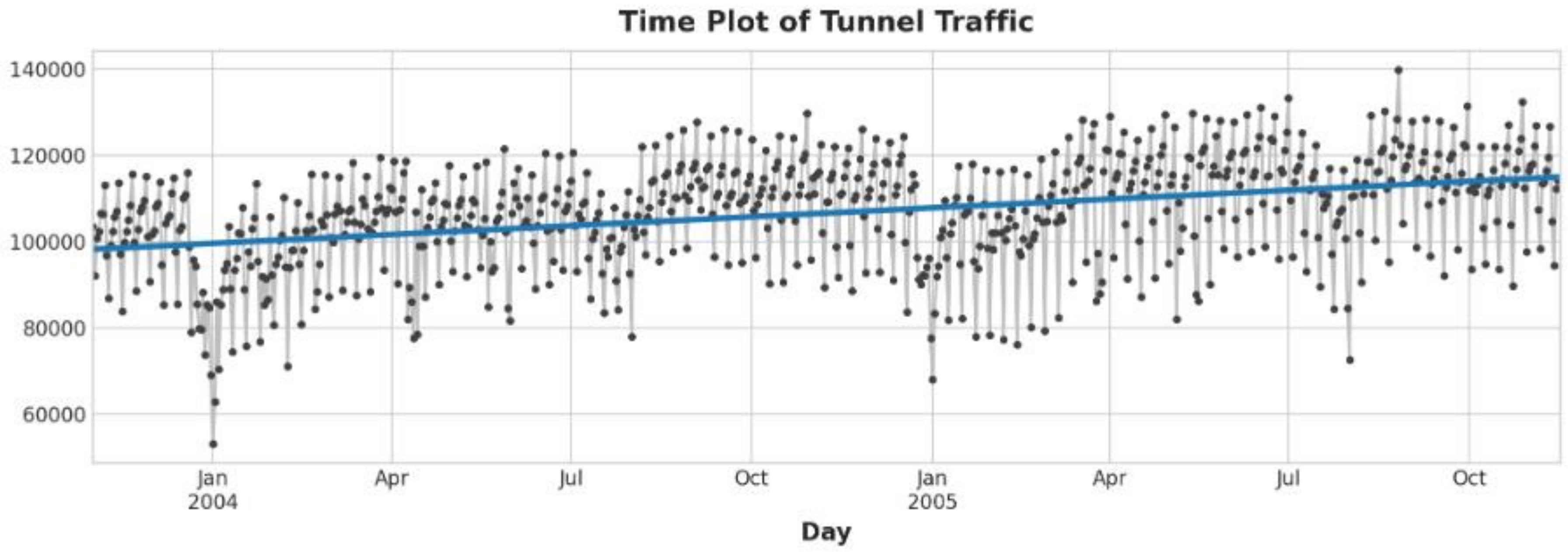
Moving Average (MA) filter (short term):

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

ARIMA model

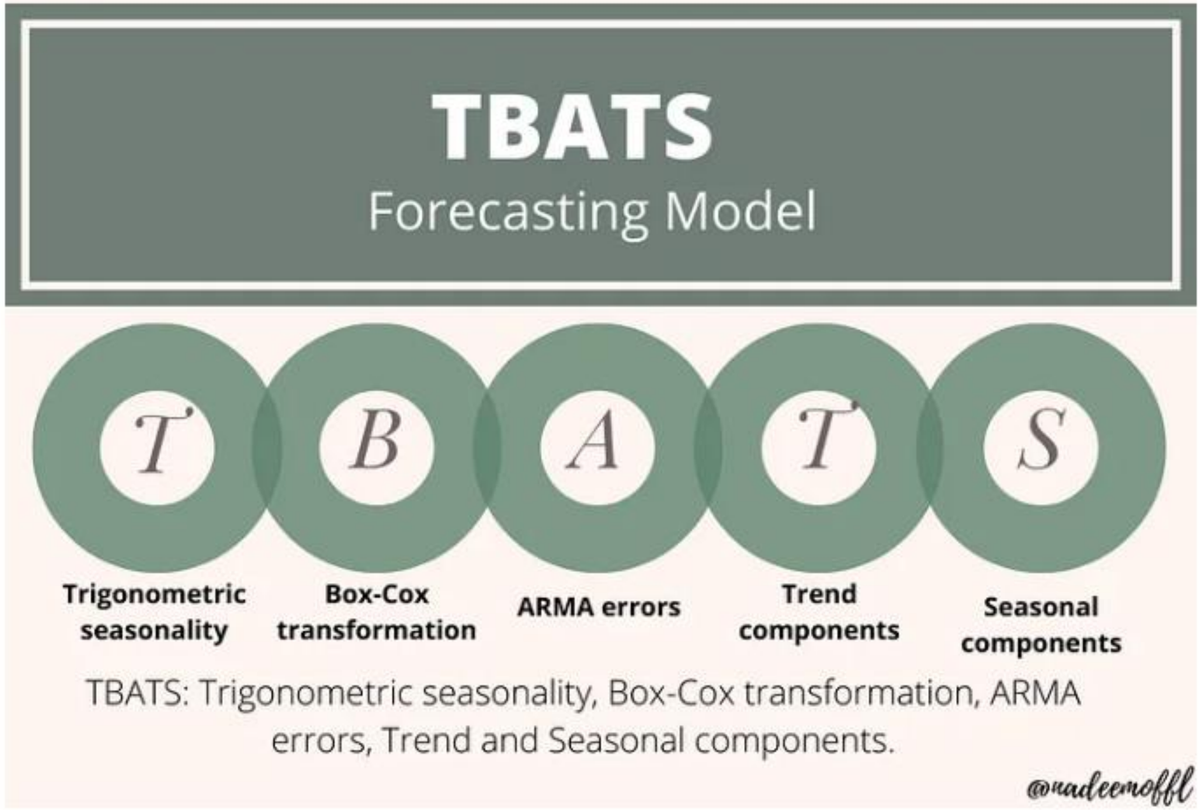

ARIMA
Prediction
MAE: 0.57, RSME: 0.10
Training
Ground Truth
Predicted
Depth to Groundwater P25
−23.0
−23.5
−24.0
−24.5
−25.0
−25.5
−26.0
−26.5
−27.0
2018-07
2018-10
2019-01
2019-04
2019-07
2019-10
2020-01
2020-04
Date
Loss
MAE
RSME
Loss
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
10
20
30
40
Fold

Prediction with Liner Regression



TBATS

```
np.random.seed(2342)
t = np.array(range(0, 160))
y = 5 * np.sin(t * 2 * np.pi / 7) + 2 * np.cos(t * 2 * np.pi / 30.5)
+ \
((t / 20) ** 1.5 + np.random.normal(size=160) * t / 50) + 10

# Create estimator
estimator = TBATS(seasonal_periods=[14, 30.5])

# Fit model
fitted_model = estimator.fit(y)

# Forecast 14 steps ahead
y_forecasted = fitted_model.forecast(steps=14)

# Summarize fitted model
print(fitted_model.summary())
```



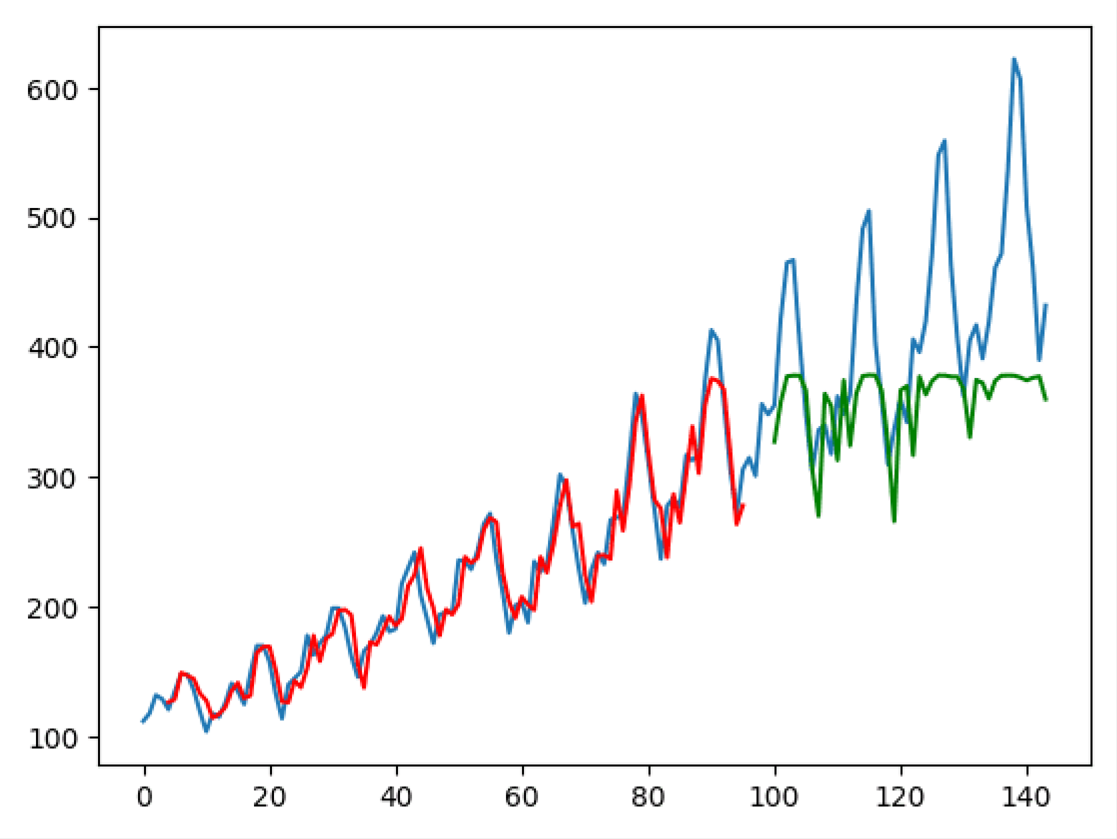
Random Forest

```
predicted_values = []

# Iterate through the test data with the moving window approach
for j in range(window_size, len(test_data), prediction_steps):
   # Create the window for the current iteration
   window = test_data[target_column].iloc[j - window_size: j]

   # Rf
   rf_model = RandomForestRegressor()
   rf_model.fit(window[:-prediction_steps].values.reshape(-1, 1),
window[prediction_steps:].values.reshape(-1, 1).ravel())

   # Forecasting/Predictions
   y_pred = rf_model.predict(window[-prediction_steps:].values.reshape(-1, 1))

   # Appending actual and predicted values to the respective lists
   actual_values.extend(test_data[target_column].iloc[j:j+prediction_steps])
   predicted_values.extend(y_pred)

# Calculate evaluation metrics
length = min(len(actual_values), len(predicted_values))
rmse = np.sqrt(mse)
mae = mean_absolute_error(actual_values[:length], predicted_values[:length])
```

SWINBURNE UNIVERSITY OF TECHNOLOGY

LSTM

```
# Split the data into training and testing sets
train_size = int(len(y) * 0.8)
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]

# Convert to PyTorch tensors
X_train = torch.from_numpy(X_train).float()
y_train = torch.from_numpy(y_train).float()
X_test = torch.from_numpy(X_test).float()
y_test = torch.from_numpy(y_test).float()
```



LSTM

```python
import torch.nn as nn

class LSTM(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers, output_size):
        super(LSTM, 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
```

LSTM

```python
# Initialize the LSTM model
input_size = 1
hidden_size = 50
num_layers = 1
output_size = 1
model = LSTM(input_size, hidden_size, num_layers, output_size)
```

```python
# Set training parameters
learning_rate = 0.01
num_epochs = 100
```

```python
# Define loss function and optimizer
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

# Train the model
for epoch in range(num_epochs):
    outputs = model(X_train.unsqueeze(-1)).squeeze()  # Add .squeeze() here
    optimizer.zero_grad()
    loss = criterion(outputs, y_train)
    loss.backward()
    optimizer.step()

    if (epoch + 1) % 10 == 0:
        print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}")
```

LSTM

```
with torch.no_grad():
    test_outputs = model(X_test.unsqueeze(-1)).squeeze()
    test_loss = criterion(test_outputs, y_test)
    print(f"Test Loss: {test_loss.item():.4f}")
```

```
def lstm_predict(model, input_data):
    input_data = torch.tensor(input_data).float().unsqueeze(0).unsqueeze(-1)
    with torch.no_grad():
        prediction = model(input_data).squeeze().item()
    return prediction
```

Learn, Practice and Enjoy the AI journey