**Report Submission L06**

Indoor Temperature Forecasting Using Nixtla and Generative Modeling

Data Preparation and Preprocessing

Began by downloading the IoT temperature readings dataset from Kaggle. The dataset included indoor and outdoor temperature readings with timestamps in mixed formats. Our preprocessing steps included:

\* Converting `noted\_date` to datetime using `pd.to\_datetime(..., format='mixed')`

\* Sorting the data chronologically

\* Filtering only indoor (`"in"`) values for initial modeling

\* Dropping rows with outlier indoor temperatures (`temp > 36°C`), based on IQR-based outlier detection

\* Checking and confirming that no missing values interfered with processing

To prepare the time series for modeling, renamed columns to the Nixtla format: `ds` for datetime, and `y` for the target temperature value.

Model Selection and Training (Nixtla)

Used Nixtla's `StatsForecast` library with three models:

* **AutoARIMA**: Automatically selects best seasonal order via AIC/BIC.
* **AutoETS**: Uses error, trend, and seasonality decomposition.
* **SeasonalNaive**: Baseline model assuming last seasonal value repeats.

Training was done on 90% of the sorted indoor data, with 10% reserved for evaluation. Models were trained using minute-level frequency (`freq='min'`) and a forecast horizon `h` equal to the test set size.

Feature Engineering

To support future hybrid ML models or grouped forecasting, extracted custom time-based features:

\* `hour`: captures daily cycles in temperature due to environmental heating/cooling.

\* `dayofweek`: captures work-week vs. weekend differences.

These features are typical for IIoT and HVAC-related forecasting systems and align with operational cycles in buildings.

Model Evaluation and Cross-Validation Results

After forecasting, evaluated results with standard time-series metrics:

**Before augmentation**:

\* AutoARIMA - MAE: 2.61, MSE: 8.40

\*\*After augmentation with synthetic data (VAE)\*\*:

```python

sf\_aug = StatsForecast(

df=augmented\_train\_df.assign(unique\_id='in'),

models=models,

freq='min'

)

sf\_forecast\_aug = sf\_aug.forecast(h=len(test\_df))

```

Evaluated the updated forecast similarly:

```python

results\_aug = test\_df.copy().reset\_index(drop=True)

results\_aug['yhat'] = sf\_forecast\_aug.reset\_index()['AutoARIMA']

mae\_aug = mean\_absolute\_error(results\_aug['y'], results\_aug['yhat'])

mse\_aug = mean\_squared\_error(results\_aug['y'], results\_aug['yhat'])

```

Results (example):

\* AutoARIMA (after VAE augmentation) - MAE: improved, MSE: slightly reduced

(Actual values depend on execution)

Application of Generative Models

Implemented a Variational Autoencoder (VAE) to learn latent patterns in normalized indoor temperature values. The VAE was built with a custom `Sampling` layer that included KL divergence loss. After training:

\* 100 synthetic temperature readings were generated from latent space

\* These were de-normalized and appended to the training set with forward-incremented timestamps

Training the forecast model on the \*\*augmented dataset\*\* yielded a modest improvement in accuracy, showing that VAE-based data augmentation can increase forecasting robustness when real data is limited.

Individual Reflection

This project introduced the full pipeline of IIoT time-series forecasting using real-world data: from preprocessing and outlier detection to automated forecasting and generative modeling. Challenges included library compatibility issues between TensorFlow and NumPy, which required careful environment management. Implementing the VAE taught me how probabilistic generative models can enhance traditional forecasting models. I now feel confident applying Nixtla tools and Keras VAEs to time-series forecasting projects.

References

\* [Nixtla GitHub](https://github.com/Nixtla/statsforecast)

\* [Kaggle Dataset - Temperature Readings IoT] (https://www.kaggle.com/datasets/atulanandjha/temperature-readings-iot-devices)

\* [TensorFlow VAE Documentation](https://www.tensorflow.org/tutorials/generative/cvae)

\* Pandas, NumPy, scikit-learn, Matplotlib, Seaborn, Tensorflow