

Predicting Beer Production for the next month

Jussi Juvonen, Niko Laurén, Isuru Mulle Gamage

Initial data visualization

Figure 1 displays the raw monthly beer production series. The data exhibits strong, regular annual seasonality together with an upward long-term trend through the 1970s and a subsequent levelling-off. Variability increases in the later decades. Seasonal peaks become larger and the scatter around the trend widens. There are no obvious gaps in the record, therefore continuity is preserved.

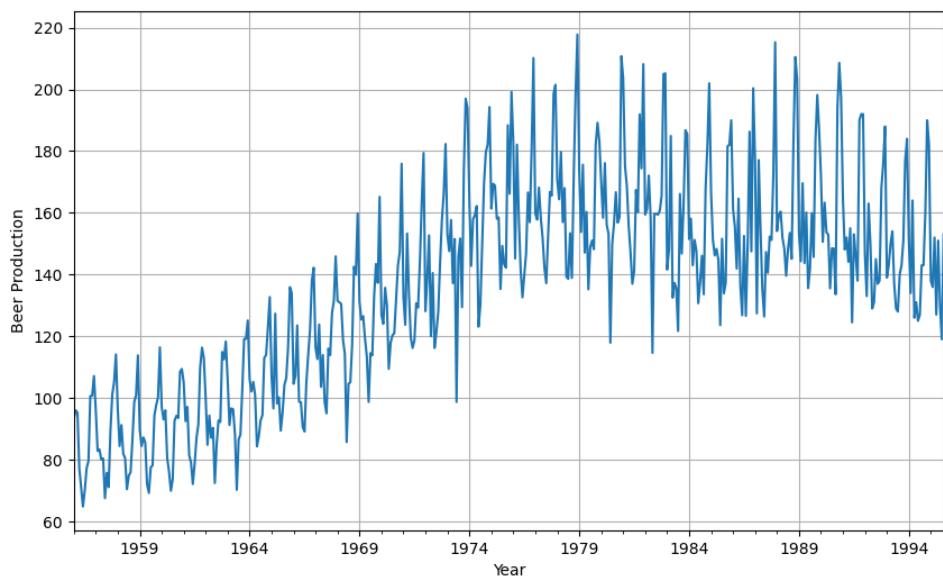


Figure 1: Raw data visualized

Time series exploration with visualization and decomposition of trend, seasonality, and residuals

The time series data represents monthly beer production in Australia, with no missing months or duplicate timestamps. The dataset spans from January 1956 to August 1995. Figure 2 shows a Seasonal-Trend decomposition using LOESS (STL) analysis of the series.

The observed data shows annual seasonality and a gradual long-term increase in production until the late 1970s, followed by a slight decline. The trend component captures the long-term pattern, and both robust and non-robust decompositions produce nearly identical trends, indicating stable estimates with minimal outlier influence. The seasonal component shows a consistent, repeating yearly pattern with roughly constant amplitude. The residuals fluctuate

around zero, which means that most systemic variation is explained by the trend and seasonal terms.

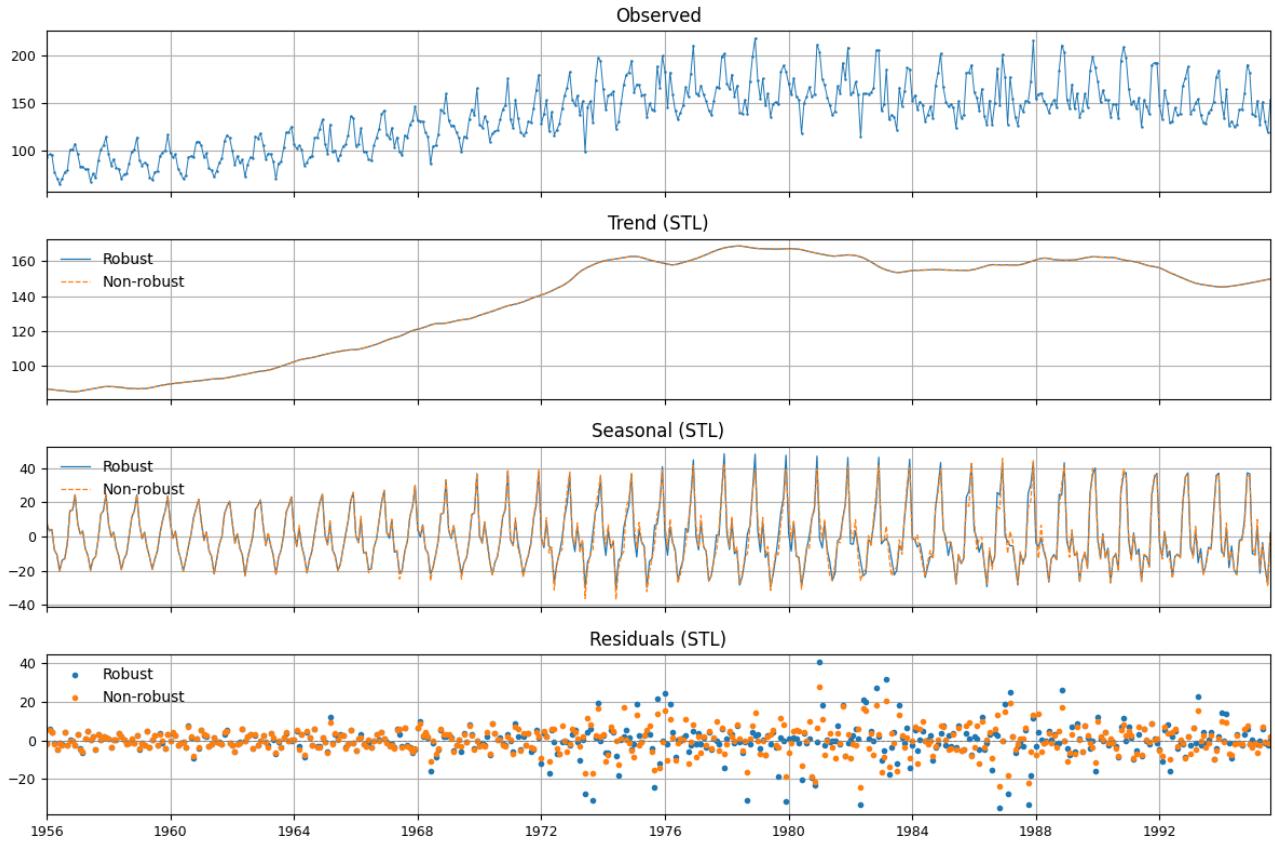


Figure 2: STL decomposition of monthly beer production time series showing the observed data, long-term trend, seasonal component, and residuals.

Figure 3 visualizes the seasonal component from the STL decomposition grouped by calendar month. The boxplots show a cyclical pattern in beer production, with the lowest seasonal values occurring during winter months (June - July) and the highest during summer (November - December). The relatively small spread of most boxes suggests that this seasonal pattern is stable across years, with only minor year-to-year variation.

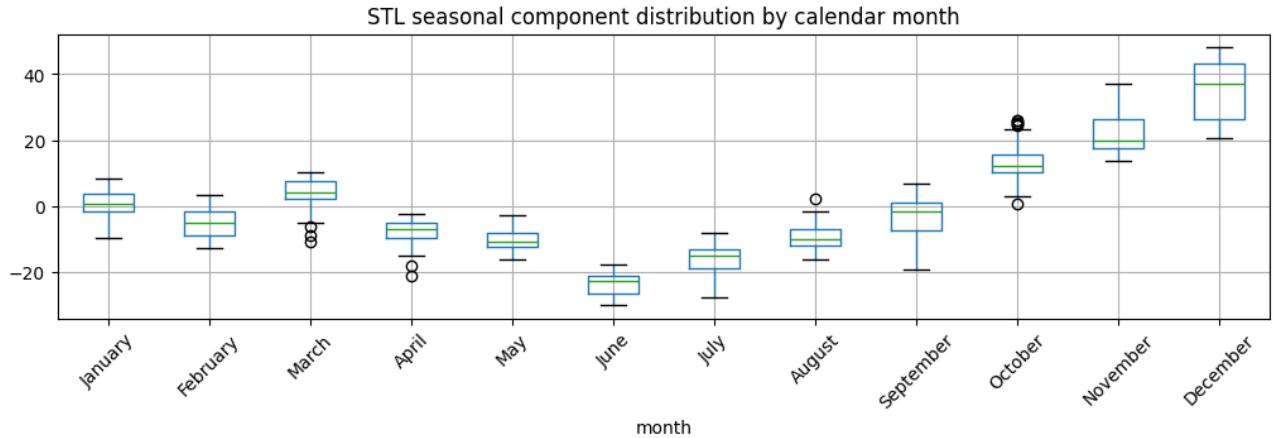


Figure 3: Distribution of the STL seasonal component by calendar month.

Autocorrelation analysis

Figure 4 shows the autocorrelation structure of the time series. The Autocorrelation (ACF) plot displays a strong positive correlation that gradually decays over many lags, with a noticeable seasonal pattern repeating approximately every 12 months, confirming yearly seasonality in the data. The Partial Autocorrelation (PACF) plot shows significant spikes at lag 1 and around lag 12. This suggests short-term dependence and annual cyclical effects. Together the plots confirm that beer production is highly autocorrelated and strongly seasonal over time.

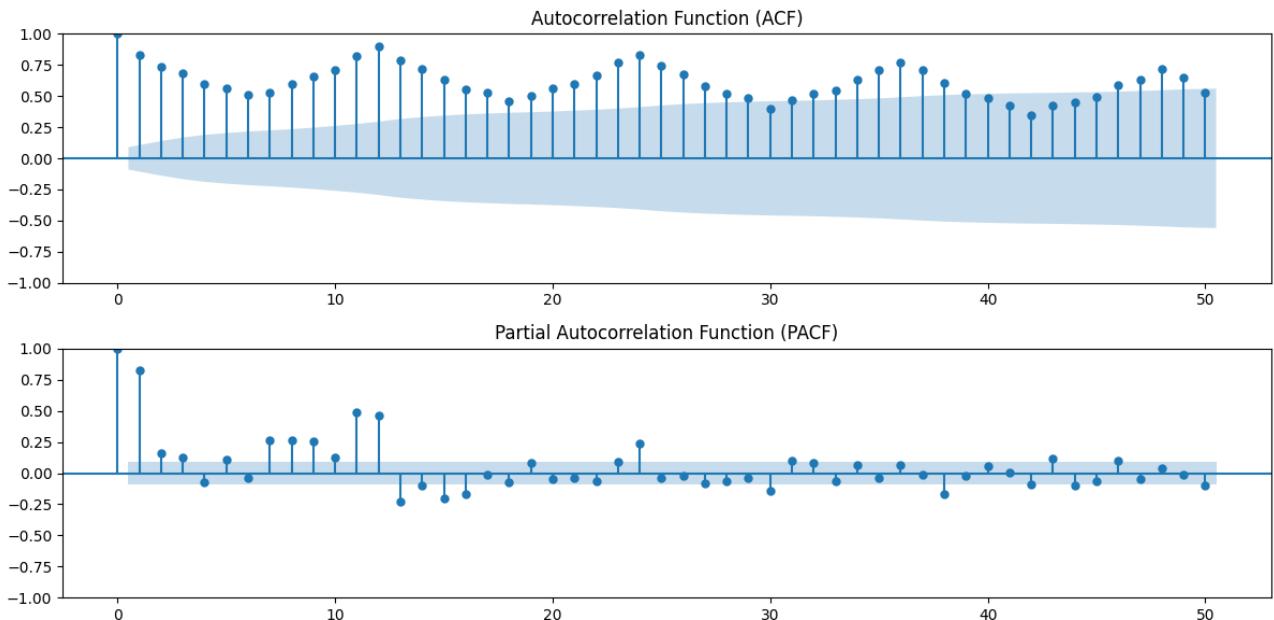


Figure 4: ACF and PACF plots of monthly beer production

Partitioning time series data

The dataset is partitioned chronologically into three subsets. The training subset is ~70% of the data covering period from January 1956 to September 1983 consisting of 333 samples. This

subset is used to train the model and identify seasonal patterns in beer production. The validation subset is the following 71 samples or ~15% of the data from October 1983 to August 1989. The validation subset is used to adjust and fine-tune the parameters of the model. The remaining 72 samples (~15%) from September 1989 to August 1995 is for the test subset. The test subset is used to evaluate the model's predictive performance on unseen future data.

Data preprocessing and quality assessment

The dataset consists of uninterrupted monthly observations from January 1956 to August 1995, showing no missing or duplicate timestamps. The measurements are uniformly sampled at a constant monthly frequency. For irregular series, a uniform rate can be restored by resampling or interpolation. For missing data, short gaps can be interpolated, while longer gaps can be addressed through model-based smoothing or segment-wise estimation that preserves the temporal structure, such as Kalman filtering or spline-based local regression.

Only one variable, monthly beer production, is recorded, making temporal alignment across variables unnecessary. In multivariate settings, synchrony can be achieved by aligning all variables to a shared time index and interpolating values where data is missing or offset.

Outlier identification using STL decomposition

STL decomposition first separates the time series into trend, seasonal, and residual components. After removing the systematic trend and seasonality, the remaining residuals capture irregular variations in the data. Outliers can then be detected by identifying residuals that deviate strongly from zero using statistical measures such as standard deviation or median absolute deviation. These points are usually flagged rather than removed to preserve the temporal structure.

LSTM

Long Short-Term Memory (LSTM) is a Recurrent Neural Network (RNN) that was developed to solve standard RNN disability to learn long term dependencies. In standard RNN models the error signals decay exponentially in backpropagation. This makes it difficult to connect relevant events between long timespan. The LSTM architecture has specialized memory cells that contain a mechanism called Constant Error Carousel (CEC). In CEC the constant error flows inside the cell preventing the information from degrading. To regulate the information flow, the LSTM has multiplicative input and output gates that protect the information from irrelevant noise and when the internal state can be used to influence the model output. (Hochreiter & Schmidhuber, 1997)

Preparing Long Time Series for LSTM Forecasting: Sub Sequences, Seasonality, and Trend Handling

Dividing a long time series into shorter sub sequences is a standard preprocessing step for LSTM models. Instead of using the full sequence, the data is transformed into many fixed length sliding windows, each containing past observations and a future prediction target. This

approach increases the number of training samples and helps the model learn local temporal patterns more effectively (Hewamalage et al., 2021). [1]

Seasonality strongly influences how these windows should be constructed. If the data contains annual or monthly seasonal cycles, the window length must be long enough to cover at least one full seasonal period; otherwise, the LSTM cannot learn recurring patterns (Lemke & Gabrys, 2019) [2]. Seasonal decomposition methods, such as STL, may also be used before windowing to simplify the series (Hyndman & Athanasopoulos, 2021). [3]

LSTMs can model trend and seasonality directly from raw sequences or through explicit preprocessing, such as deseasonalization or adding seasonal indicators as extra features (Bandara et al., 2020) [4].

LSTM standardization methods

LSTM has two typical standardization methods called Min-Max normalization and Z-score standardization. Min-Max transforms data into a fixed range and it works well with sigmoid and tanh functions. Z-score centers the data around zero mean with unit variance and it's suitable for dealing with outliers and varying feature scales. Min-Max produces lower Round Mean Square Error (RMSE) in LSTM-based multivariate time series forecasting when compared to Z-score. Min-Max enhances training stability and prediction accuracy in LSTM models (Pranolo et al., 2024).

Baseline model

A autoregressive baseline model was made by converting the series into sliding windows of length $T = 12$ months. Each sample uses the previous 12 monthly observations to predict the next month. The model itself is a single layer linear predictor mapping the 12-dimensional input to scalar forecast and trained to minimize mean squared error on the training set. The data was split so that all samples up to September 1983 were used for training and in the remaining months formed a single holdout set for validation.

In Figure 5, the training and validation losses of the baseline autoregressive model are shown across 40 epochs. The training loss decreases rapidly during the first 10 epochs and then gradually stabilizes. The validation loss follows a similar trajectory but begins to plateau and fluctuate after roughly 20-30 epochs, suggesting some mild overfitting. Both losses remain within the same order of magnitude, implying that the model generalizes reasonably well given its simplicity.

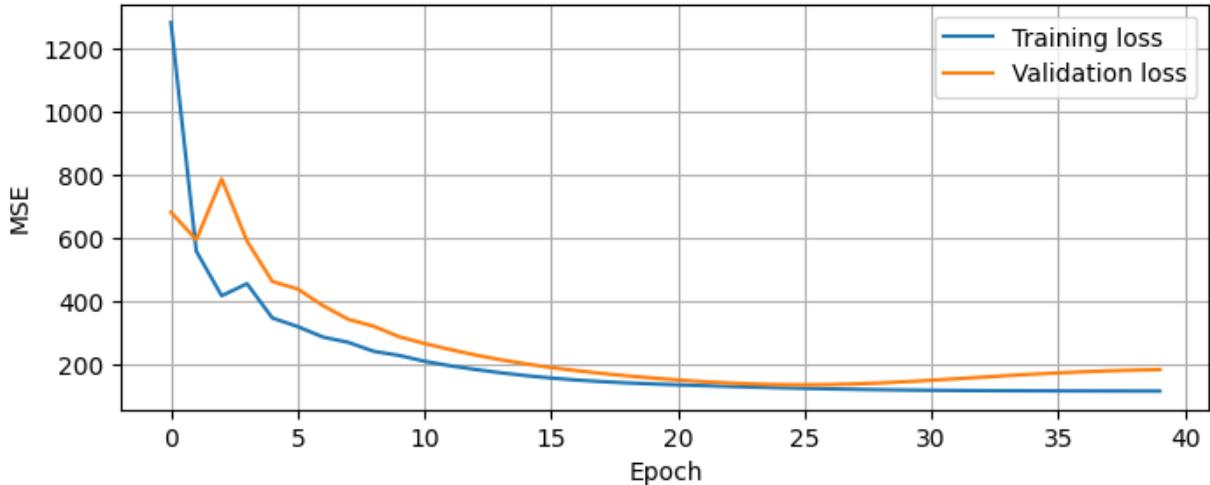


Figure 5: Training and validation MSE across epochs for the autoregressive baseline model.

In Figure 6, the iterative forecast is compared against the actual validation target over time. The iterative forecast shows moderate accuracy at the beginning of the validation data but tends to drift as the predictions progress. This drift might happen because each step's prediction is recursively fed back as input, allowing small errors to accumulate. The model occasionally underestimates or overestimates the overall trend, resulting in systematic bias near the end of the forecast. In this run the forecast overestimates the trend and achieved $MAE \approx 69.3$ and $RMSE \approx 82.3$, suggesting that while the baseline model captures the long-term pattern, its simplicity limits its ability to maintain stable trend estimation across multiple forecasted steps.

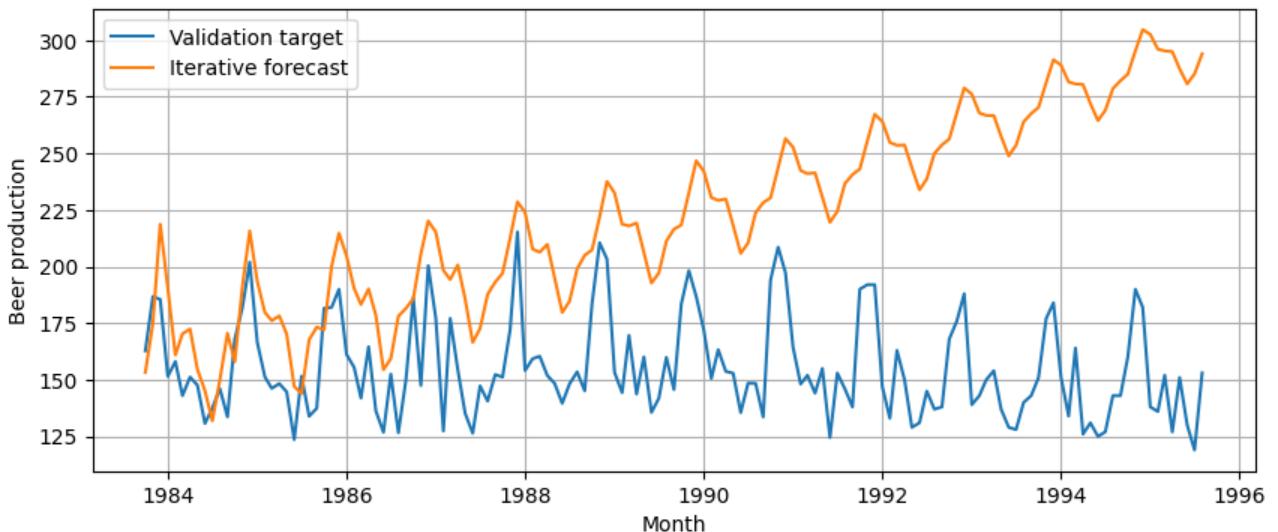


Figure 6: Iterative multi-step forecast compared with the validation target for the baseline autoregressive model.

LSTM model

The LSTM implementation uses a chronological train and holdout split with data shuffling disabled during training. Z-score scalers were fitted on the training set and then applied to the validation and holdout folds to prevent data leakage. Early stopping with patience 15 is used to limit overfitting and iterative multi-step forecasting performed recursively by appending one-step predictions to the input window for subsequent forecasts. Training minimized mean squared error (MSE) using the Adam optimizer. The model weights with the best validation loss were used for final evaluation on the holdout set.

Training and validation errors decrease rapidly during the initial epochs and then converge to a low, stable level, as shown in Figure 7. Validation loss closely tracks training loss. The eventual plateau suggests reasonable convergence under the chosen stopping criterion. Several hyperparameter combinations were inspected manually and then the reported model corresponds to the configuration that produced the most stable training behavior.

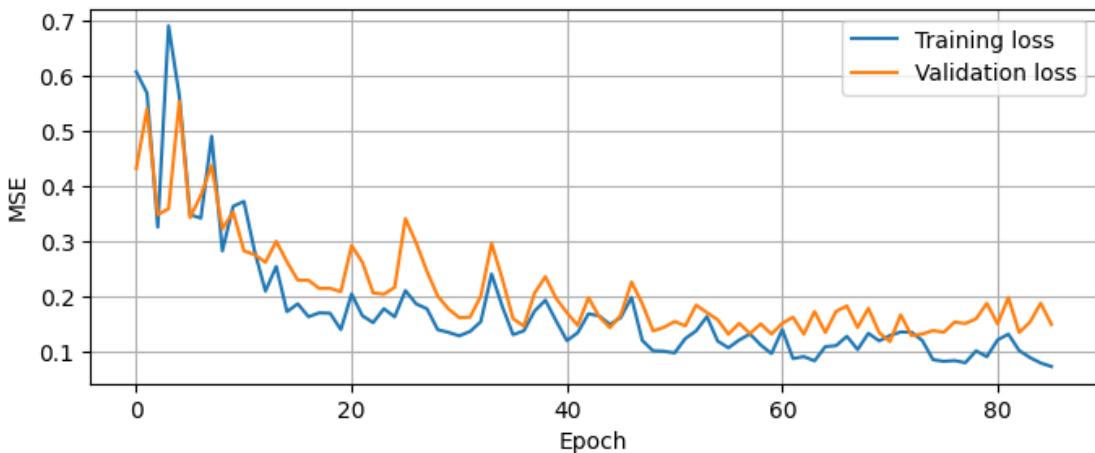


Figure 7: Training and validation MSE across epochs for the LSTM model.

The LSTM forecast captures the dominant annual pattern and broadly follows the shape of the holdout series, but the predicted curve consistently reaches its peaks slightly later than the true data, this is shown in Figure 8. This small lag becomes more noticeable toward the end of the plot, where the recursive multi-step procedure allows timing errors to accumulate. The model also underestimates peak heights, producing smoother and slightly lower seasonal maxima in some years.

The holdout performance is $MAE = 21.49$ and $RMSE = 26.43$, indicating moderate accuracy relative to typical monthly production levels. Overall, the results show that the model learns the seasonal structure but exhibits systematic phase lag and reduced peak amplitude, suggesting that improvements in the model or preprocessing could improve the forecasting.

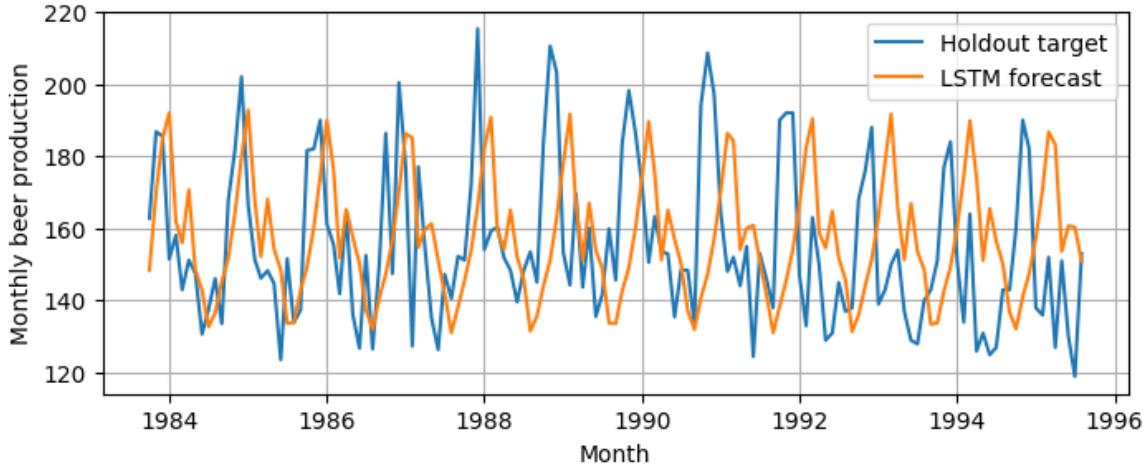


Figure 8: Iterative multi-step LSTM forecast compared with the holdout target.

The LSTM model can be optimized through a focused hyperparameter search. Key parameters to refine include the LSTM hidden size, learning rate, batch size and window length. These will be explored over moderate ranges. LSTM units from a few dozen units to a few hundred units, learning rate over a logarithmic scale, and window lengths that test shorter and longer seasonal contexts.

Further improvements will be investigated by comparing preprocessing strategies such as Z-score and Min-Max scaling. Candidate models will be evaluated consistently on the chronological holdout set, and the best configuration will be re-trained with several random seeds to ensure stability and reproducibility.

Transformer model

The transformer model is implemented as a one-ahead forecasting model with a chronological train and holdout split. Z-score scalers are fitted on the training fold and then applied to validation and holdout to avoid data leakage. The model projects a single input feature into a dimensional embedding, adds sinusoidal fixed positional encodings, and passes the sequence through an encoder before using the final time-step encoder output to predict the next value with a linear head. Training minimizes MSE using the Adam optimizer. Gradient norm clipping is used to stabilize updates. Early stopping is used to stop training when validation loss did not improve for 8 epochs.

A random hyperparameter search was performed over 200 trials, Table 1 shows all 8 hyperparameters tuned and their search ranges or distributions. For each trial we trained until early stopping and kept the model state with the best validation loss. The same chronological split and scaler practice is used as in the LSTM model.

Hyperparameter	Search range / distribution
Model dimension d_{model}	16, 32, 48, 64, 128, 256
Attention heads	1, 2, 4, 8, 16
Encoder layers	1-5
Feedforward dimensions	$d_{model} \times \{2, 4\}$
Dropout	Uniform on [0.0, 0.3]
Learning rate	Log-uniform on $[10^{-4}, 5 \times 10^{-3}]$
Input window	6, 12, 24, 36
Batch size	16, 32, 64, 128

Table 1: Tunable hyperparameters for the transformer model.

Figure 9 shows the training and validation loss curves for the chosen best trial. Training and validation losses seem to both decrease early and quickly flatten. The model is performing worse on the validation dataset with small peaks in loss values.

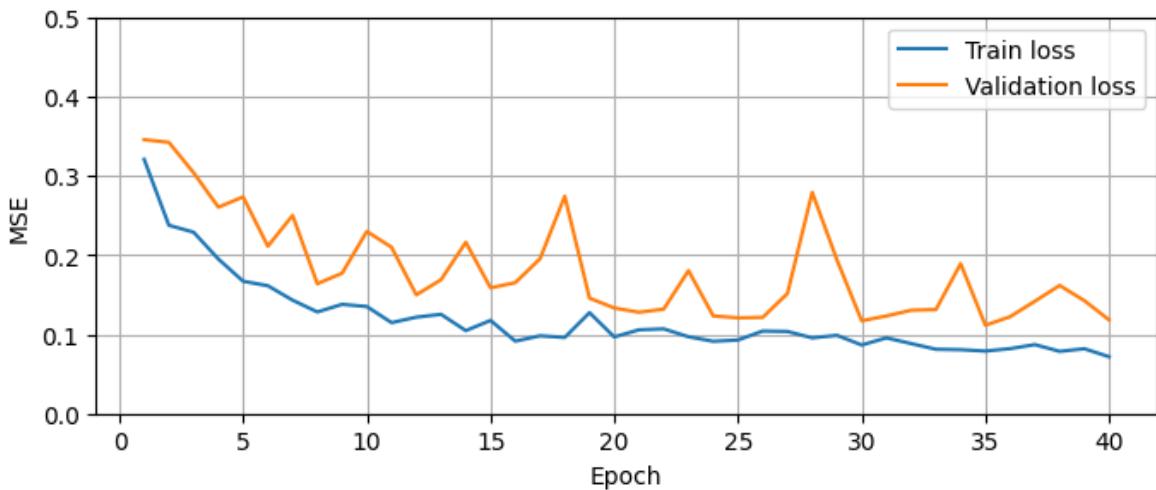


Figure 9: Training and validation MSE across epochs for the transformer model.

Figure 10 shows the best model's 1-ahead forecasts plotted against the holdout series. The transformer captures the dominant seasonal pattern and timing. The forecast has a smaller amplitude for some seasonal peaks when compared to the target. The forecasts plotted are 1-ahead predictions produced by applying the trained model to each holdout input window and then inverse-transformed back to the original scale for plotting.

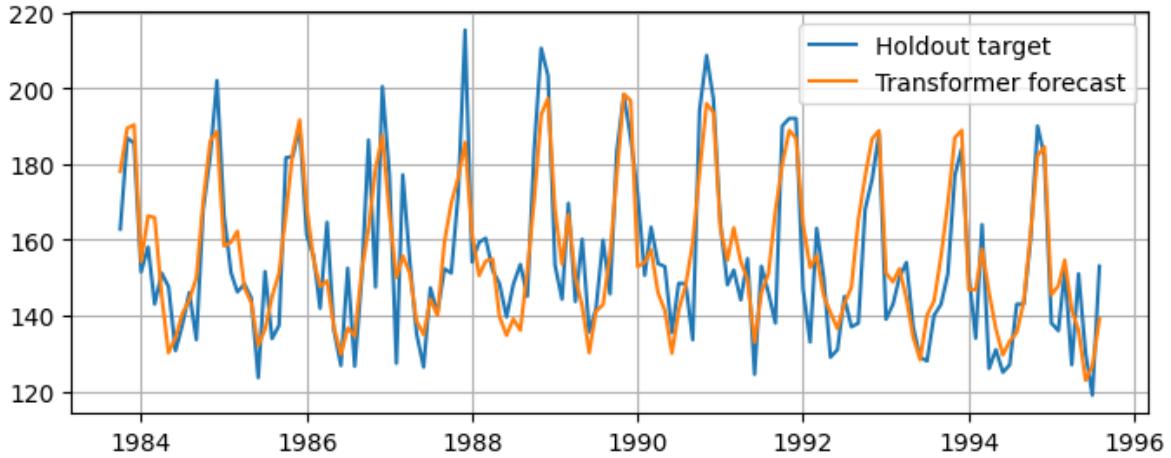


Figure 10: Transformer 1-ahead forecast.

Figure 11 presents boxplots of holdout MAE grouped by hyperparameter value. These summarize how choice of each hyperparameter affected validation MAE across the 200 random trials.

The hyperparameter analysis indicates broadly stable performance across several settings with some parameters showing modest advantages. Model dimensions between 32 and 128 perform similarly well. Batch size has little effect except for 128, which gives higher errors. Encoder depth shows minimal differences, through 3-4 layers perform best. Attention heads behave similarly overall. Feedforward dimensions show uneven variability due to interactions with model dimensions, with 256 and 1024 showing tighter, more stable error distributions than 512.

The most effective dropout values fall in the lower and mid-upper ranges of the tested range. Learning rates around $0.6 - 1.8 \times 10^{-3}$ yields the lowest errors. A window size of 12 provides the lowest error rate with 24 performing comparably. The best model found in the 200-trial search used 48 model dimensions, 16 attention heads, 1 encoder layer, 192 feedforward dimensions, dropout of roughly 0.04, learning rate 7.7×10^{-4} , window size of 24 and a batch size of 16, Achieving these validation results $MAE = 9.07$ and $RMSE = 11.30$.

Across the top three models, the hyperparameters vary noticeably, suggesting that multiple configurations can achieve similar performance. This variance may arise because several of the hyperparameters interact, allowing different combinations to compensate for one another, and because the optimization contains several nearby minima rather than a single clear optimum.

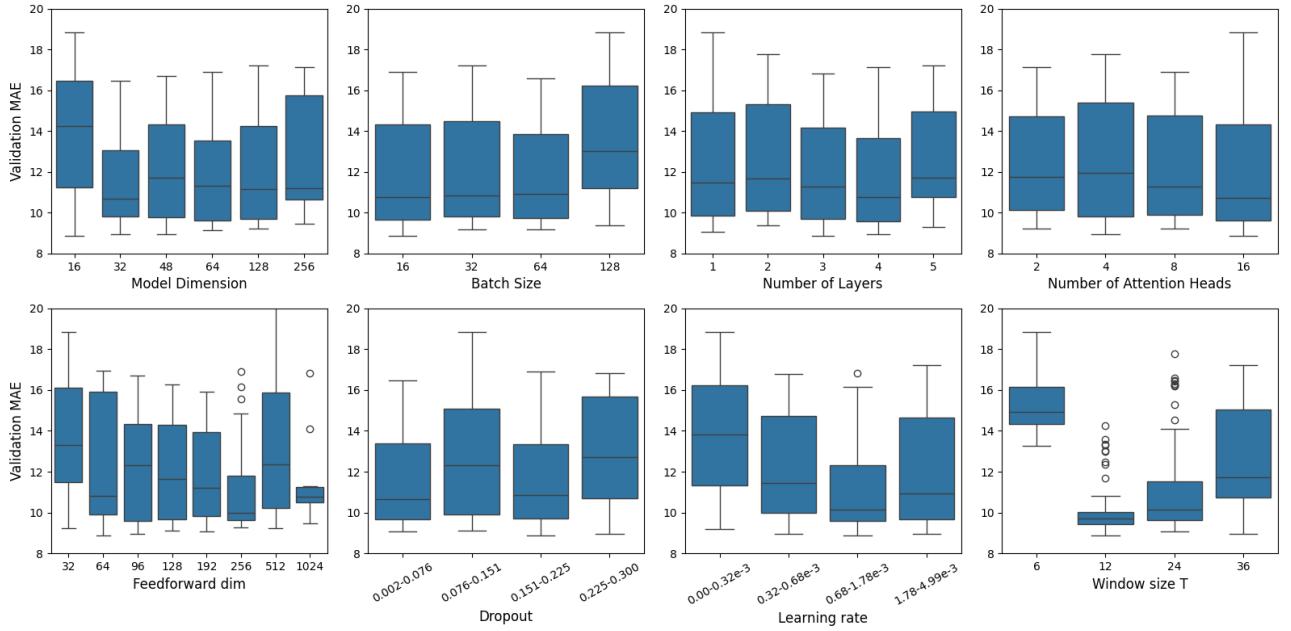


Figure 11: Validation MAE distributions across trials grouped by hyperparameter values.

Next, we will tune the LSTM using the same evaluation protocol and using a random search over the main LSTM settings for a better comparison of the models.

References

- Bandara, K., Bergmeir, C. & Smyl, S. (2020), "Forecasting across time series databases using recurrent neural networks on groups of similar series: A clustering approach", *Expert Systems with Applications*, 140, pp. 112896.
- Bianchi, F.M., Livi, L., Rizzi, A. & Sadeghian, A. (2022), *Recurrent Neural Networks for Short-Term Load Forecasting*, Springer.
- Fawaz, H.I., Forestier, G., Weber, J., Idoumghar, L. & Muller, P.A. (2019), "Deep learning for time series classification: A review", *Data Mining and Knowledge Discovery*, 33(4), pp. 917–963.
- Hewamalage, H., Bergmeir, C. & Bandara, K. (2021), "Recurrent neural networks for time series forecasting: Current status and future directions", *International Journal of Forecasting*, 37(1), pp. 388–427.
- Hyndman, R. & Athanasopoulos, G. (2021), *Forecasting: Principles and Practice*, 3rd edn., OTexts.
- Lemke, C. & Gabrys, B. (2019), "Meta-learning for time series forecasting and forecast combination", *Neurocomputing*, 353, pp. 1–16.
- Pranolo, A., Setyaputri, F.U., Paramarta, A.K.I., Triono, A.P.P., Fadhilla, A.F., Akbari, A.K.G., Putra Utama, A.B., Prasetya Wibawa, A. & Uriu, W. (2024) 'Enhanced Multivariate Time Series Analysis Using LSTM: A Comparative Study of Min-Max and Z-Score Normalization Techniques', *ILKOM Jurnal Ilmiah*, 16(2), pp. 210-220. DOI: 10.33096/ilkom.v16i2.2333.210-220.
- Hochreiter, S. & Schmidhuber, J. (1997) 'Long Short-Term Memory', *Neural Computation*, 9(8), pp. 1735–1780.