Electricity Transformer Temperature and Exchange Rate Forecasting

Thanina Ait Ferhat, Kamilia Benlamara, Iness Fendes, Lynda Benkerrou
University of Paris Cité, France

22 April 2024

Abstract

This study investigates time series forecasting within electricity transformer temperatures and exchange rates using advanced machine learning techniques. Our research focuses on predicting future values based on historical data, addressing both univariate and multivariate time series challenges. Our study particularly aimed to explore the efficacy of machine learning algorithms in managing temporal dependencies and missing values. Additionally, this paper highlights the strengths and limitations of these methods, providing insights that enhance predictive accuracy in essential economic and industrial applications.

1 Introduction

Time series forecasting is crucial across numerous sectors, particularly in areas such as power and energy consumption, stock market analysis, and weather prediction. The selection of an appropriate forecasting model is vital, as it greatly influences the precision of the forecasts.

The aim of this project is twofold :

- Firstly, we address a univariate forecasting problem involving the prediction of the 100 values of the 'OT' variable within the ETTh1 dataset. This dataset captures the temporal evolution of electricity transformer temperatures spanning from July 2016 to July 2018 in China.
- Secondly, we tackle a more complex multivariate time series forecasting issue with the Exchange Rate dataset . This dataset chronicles the daily fluctuations in exchange rates across eight countries over the period spanning from 1990 to 2010.

Our study assesses the effectiveness of machine learning algorithms in handling temporal dependencies and missing data. The paper outlines the strengths and limitations of these techniques, offering valuable insights to improve predictive accuracy in critical economic and industrial settings.

2 Univariate Time Series forecasting problem

In this section, we explore our approach to addressing a key univariate forecasting problem: predicting future values of the 'OT' (Oil Temperature) variable in the ETTh1 dataset. The focus is on predicting 100 future values based solely on past observations of the 'OT' variable. In the following subsections, we will delve deeper into the model architecture, training process, and the detailed results of our forecasting results.

2.1 Long Short Term Memory (LSTM)

Long Short-Term Memory (LSTM) networks, introduced by Hochreiter and Schmidhuber in 1997, address the vanishing and exploding gradient problems encountered in recurrent neural networks (RNNs). Unlike traditional RNNs, LSTMs utilize memory cells to retain information over long sequences of data, enabling them to capture complex temporal dependencies effectively.

A typical LSTM cell consists of two main components: the internal state and gates. The internal state acts as the short-term memory, storing information from previous time steps. New information is continually added to this internal state at each time step and passed on to subsequent cells. Additionally, the internal state undergoes regularization through gates, which control the flow of information by selectively adding or removing relevant information [6].

Various LSTM variants have been proposed to enhance performance in different domains, such as time series forecasting. Despite these variations, the fundamental structure of LSTMs remains widely used due to their ability to capture intricate temporal dependencies and effectively model sequential data.

2.2 Related Work

This subsection explore notable research efforts that have contributed to advancing the field of time series forecasting using LSTM networks and that we found interesting for our subject. Suilin contributes to the field of time series forecasting by demonstrating the applicability and effectiveness of LSTM neural networks in predicting web traffic patterns, which can be valuable for optimizing resource allocation and improving user experience in web services. Elsworth and Güttel explores the use of LSTM neural networks for time series forecasting in renewable energy production. It investigates the impact of different input features, network architectures, and training strategies on the accuracy of forecasting models. Han et al. discusses the application of deep learning models, including LSTM, in time series forecasting across different domains. It highlights the strengths and limitations of LSTM networks and discusses important factors such as data preprocessing, model architecture, and hyperparameter tuning.

2.3 Database and data pre-processing

Dataset Exploration

The first dataset used in this project is `ETTh1_without_missing.csv`, which contains measurements of various parameters of an electrical transformer. The main features of the dataset are:

- The dataset contains 17,420 observations with 2 variables:
  - **date**: The date and time of the observations.
  - **OT**: The oil temperature of the transformer.
- All variables are of numeric type.
- There are no missing values in this dataset.

We started by plotting the time series graph of the transformer oil temperature ('OT') to get a first overview of the data Figure [1]:

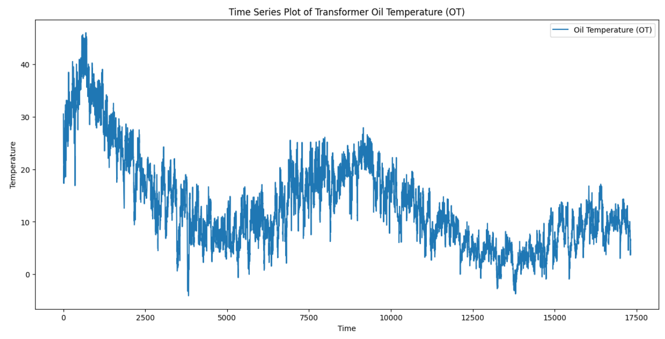


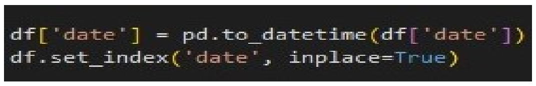
Figure 1: Data visualisation

This visualization allowed us to observe the trends and patterns present in the data [fig 1].

Data Preprocessing

Before moving on to modeling, we performed the following preprocessing steps:

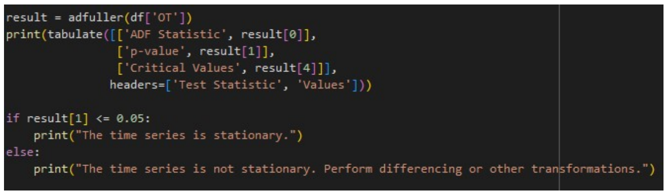
- Conversion of the 'date' column to datetime format:



```
df['date'] = pd.to_datetime(df['date'])
df.set_index('date', inplace=True)
```

Figure 2: Conversion of the 'date' column

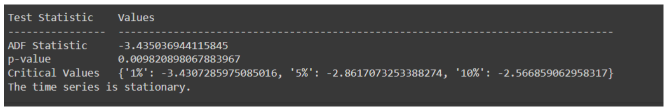
- Checking the stationarity of the 'OT' time series: the results indicate that the 'OT' time series is stationary, so we do not need to apply any additional transformations



```
result = adfuller(df['OT'])
print(tabulate([['ADF Statistic', result[0]],
                ['p-value', result[1]],
                ['Critical Values', result[4]]],
               headers=['Test Statistic', 'Values']))

if result[1] <= 0.05:
    print("The time series is stationary.")
else:
    print("The time series is not stationary. Perform differencing or other transformations.")
```

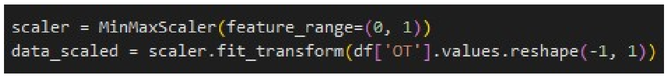
Figure 3: Checking the stationarity



```
Test Statistic    Values
----------------  --------------------------------------------------------------------------------
ADF Statistic     -3.435036944115845
p-value           0.009820898067883967
Critical Values   {'1%': -3.4307285975085016, '5%': -2.8617073253388274, '10%': -2.566859062958317}
The time series is stationary.
```

Figure 4: Results of the stationarity test

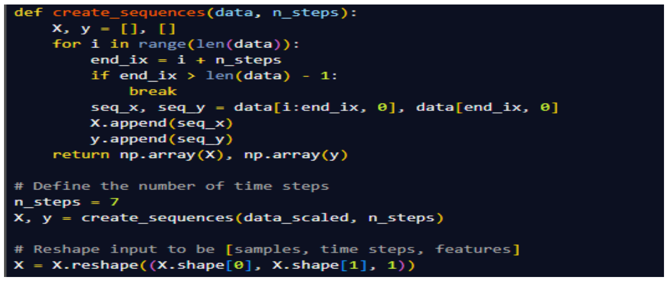
- Scaling the data: we normalized the data by scaling it between 0 and 1 to facilitate model learning.



```
scaler = MinMaxScaler(feature_range=(0, 1))
data_scaled = scaler.fit_transform(df['OT'].values.reshape(-1, 1))
```

Figure 5: Scaling Data

- Creating input-output sequences: we created input-output sequences using a sliding window of 7 time steps.



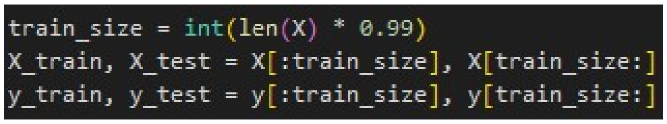
```
def create_sequences(data, n_steps):
    X, y = [], []
    for i in range(len(data)):
        end_ix = i + n_steps
        if end_ix > len(data) - 1:
            break
        seq_x, seq_y = data[i:end_ix, 0], data[end_ix, 0]
        X.append(seq_x)
        y.append(seq_y)
    return np.array(X), np.array(y)

# Define the number of time steps
n_steps = 7
X, y = create_sequences(data_scaled, n_steps)

# Reshape input to be [samples, time steps, features]
X = X.reshape((X.shape[0], X.shape[1], 1))
```

Figure 6: Input-Output sequences

- Splitting the data into training and test sets: we divided the data into a training set (99%) and a test set (1%). With these preprocessing steps, we are now ready to build and train our time series forecasting model.

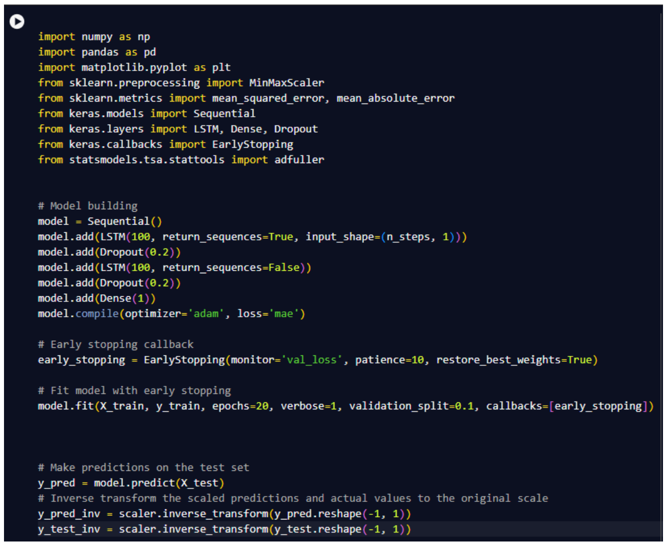


```
train_size = int(len(X) * 0.99)
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]
```

Figure 7: Train-Test Split

2.4 Training parameters

Before we discuss the training parameters of this model, it's important to highlight that several variants of LSTM models have been developed to improve performance and efficiency across different types of tasks. Our model implements the Stacked LSTM. This variant involves stacking multiple LSTM layers on top of each other, making the architecture deeper. Each layer's output is sequenced into the next layer's input, helping to learn higher level temporal dependencies in the sequence.



```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error
from keras.models import Sequential
from keras.layers import LSTM, Dense, Dropout
from keras.callbacks import EarlyStopping
from statsmodels.tsa.stattools import adfuller


# Model building
model = Sequential()
model.add(LSTM(100, return_sequences=True, input_shape=(n_steps, 1)))
model.add(Dropout(0.2))
model.add(LSTM(100, return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mae')

# Early stopping callback
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

# Fit model with early stopping
model.fit(X_train, y_train, epochs=20, verbose=1, validation_split=0.1, callbacks=[early_stopping])



# Make predictions on the test set
y_pred = model.predict(X_test)
# Inverse transform the scaled predictions and actual values to the original scale
y_pred_inv = scaler.inverse_transform(y_pred.reshape(-1, 1))
y_test_inv = scaler.inverse_transform(y_test.reshape(-1, 1))
```

Figure 8: Building-Training Stacked LSTM Model

To implement a Stacked Long Short-Term Memory (LSTM) model, one of the most crucial steps is to define the model's architecture, as it directly influence the model's ability to capture and learn from the temporal dependencies within the data.

We initiated our model building by employing the Keras Sequential API. We created a sequential model which is a linear stack of layer. For the compilation of our model, we chose the Adam optimizer and the mean absolute error (MAE) as the loss function.During the training phase, we integrated an EarlyStopping callback to monitor the validation loss.The model was trained over multiple epochs to refine its accuracy, taking advantage of the iterative nature of training deep learning models to progressively minimize the loss function and improve prediction accuracy on the training data.Once training was complete, the model's effectiveness was evaluated by deploying it to make predictions on the test set.

2.5 Experiments

We have made over **47** Kaggle submissions, each time striving to improve the model's performance. It is worth noting that we conducted these various executions on a Google Colab environment. The table 1 summarizes the different parameters and their ranges of variation over all the tests performed.

	Parameters	**Value Range**
Model definition phase	Number of units	[10,200]
	Number of times steps	[5,15]
	Dropout Layer	[0.1,0.3]
	Dense Layer	[0.1,0.3]
Training	patience	10,15
	epochs	[10,50]

Table 1: Parameter Settings of Tested Models

After conducting various tests, we were able to improve our score according to the **MAE** metric to a value of **0.65**. Our worst score had an MAE of 6.79.
If we want to analyze the different models we obtained, we can clearly conclude that with an LSTM trained with a high number of units and/or a high number of epochs, we achieve a very low MAE on the training data, but a poor MAE score in the prediction part. This result can be explained by overfitting of the obtained models. The model that achieved the best score was trained with 100 units, 20 epochs, 7 time steps and 0.1 for the Dropout Layer.

2.6 Summary

In Part A of the project, we addressed the challenge of forecasting the next 100 values of the 'OT' (Oil Temperature) variable from July 2016 to July 2018 using the ETTh1 dataset. We chose Long Short-Term Memory (LSTM) networks for their ability to handle long-term dependencies and overcome the vanishing gradient issue common in traditional recurrent neural networks.

Throughout the project, we refined the model's accuracy through over 47 Kaggle submissions. Each submission helped us to better understand and optimize the model's performance, enhancing its predictive capabilities.

This phase demonstrated the effectiveness of LSTM networks in forecasting significant temperature metrics with high precision. The strategic data preprocessing and fine-tuning of the model were crucial in achieving robust forecasting results.

Utilizing LSTMs proved advantageous for processing long data sequences essential for accurate forecasts. However, the complexity of these models suggests a need for further research into more efficient computational strategies or alternative modeling approaches. Future work might explore integrating additional data features or experimenting with different neural network architectures to improve forecasting accuracy.

3 Multivariate Time Series Forecasting

In this section, we address the problem of multivariate time series forecasting using the Exchange Rate dataset. The goal is to predict the next 100 values of the target variable ’6’, based on past observations of 8 variables representing exchange rates in different countries between 1990 and 2010. In the following subsections, we will delve deeper into the model architecture, training process, and provide a detailed analysis of the forecasting results.

3.1 Random Forest Regressor

The Random Forest Regressor is a supervised learning model based on the Random Forests algorithm for solving regression problems. It consists of an ensemble of multiple decision trees trained on random subsets of the training data. [2]

The operation of the Random Forest Regressor can be summarized as follows:

1. **Forest Creation:** a collection of decision trees is built from bootstrap samples of the training data. Each tree is trained on a random subset of the data. Random Feature
2. **Selection:** for each decision node in a tree, a random subset of features is selected to determine the best split.
3. **Prediction:** to make a prediction, the input data are passed through each tree in the forest. The final prediction is the average of the predictions from all individual trees.

The Random Forest Regressor is well known of its robustness against noise and is adept at managing correlated features, thanks to its ensemble approach. Its training can be parallelized, which boosts efficiency. Nonetheless, the model’s complexity as a ’black box’ makes it challenging to interpret, and without careful tuning, it may be prone to overfitting, potentially compromising its effectiveness on new data Breiman.

3.2 Related Work

This subsection highlights significant research contributions to the field of time series forecasting using Random Forest Regressor, which are relevant to our study.Rana, Koprinska, and Aalstdemonstrated the effective use of Random Forests for predicting the lifespan of drainage pumps in urban sanitation systems, underscoring the importance of feature selection and hyperparameter optimization. Zhu, Baesens, and Vanden Broucke analyzed the application of Random Forests for credit scoring in the financial sector, comparing its performance with other machine learning techniques and outlining its strengths and limitations. Aznarte focused on the use of Random Forests to forecast electricity demand in multivariate time series, highlighting their capability to handle missing data and correlated variables, and the need for strategic feature selection and model tuning.

These studies illustrate the robustness of Random Forests in noise management and their ability to handle correlated variables. They also note challenges like model interpretability and the risk of overfitting without careful hyperparameter management. These insights are crucial for optimizing the use of Random Forests in time series forecasting, emphasizing the importance of thorough data preprocessing, careful feature selection, and precise hyperparameter adjustments.

3.3 Database and data pre-processing

Dataset Exploration

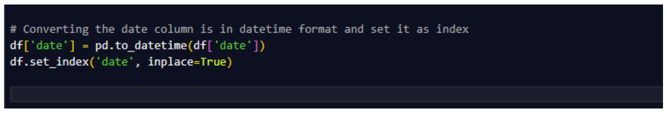
The Exchange Rate dataset contains daily fluctuations in exchange rates across 8 countries between 1990 and 2010. The dataset initially contains missing values.

- The file `exchange_rate_with_missing.csv` contains the raw data with missing values (NaN).
- There are 8 columns, one for each country, with the date as the index.
- The dataset has a total of 7588 observations covering the period from January 1, 1990 to December 31, 2010.
- The values represent the daily exchange rates for each country.

Data Preprocessing

The following preprocessing steps were performed on the Exchange Rate dataset:

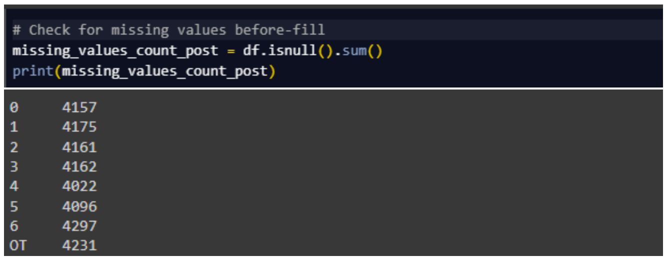
- Convert date column to datetime format and set as index:



```
# Converting the date column is in datetime format and set it as index
df['date'] = pd.to_datetime(df['date'])
df.set_index('date', inplace=True)
```

Figure 9: Date Conversion

- Check for missing values in the dataset: Figure [10] illustrates the number of missing values for each feature:

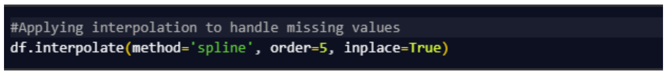


```
# Check for missing values before-fill
missing_values_count_post = df.isnull().sum()
print(missing_values_count_post)
```

```
0     4157
1     4175
2     4161
3     4162
4     4022
5     4096
6     4297
OT    4231
```

Figure 10: Checking for missing values

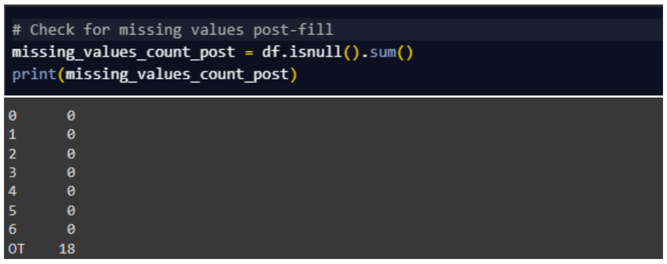
- Interpolate missing values using spline method: Missing values are interpolated using the 'spline' method with order 5. This method uses 5th-degree polynomials to interpolate the missing values smoothly.



```
#Applying interpolation to handle missing values
df.interpolate(method='spline', order=5, inplace=True)
```

Figure 11: Interpolation

- Check for remaining missing values after interpolation:: Figure[12] illustrates the remaining missing values in the dataset.



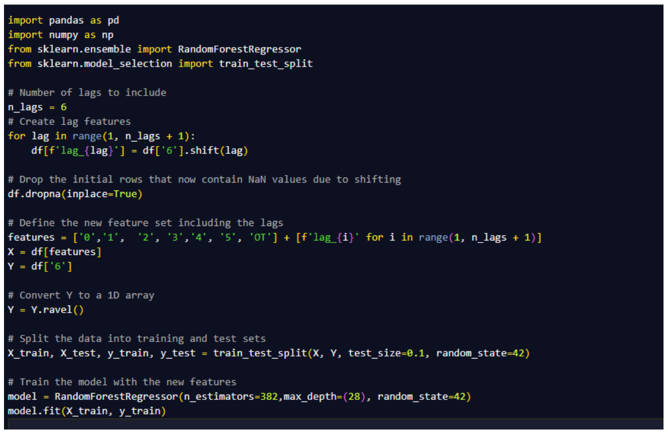
```
# Check for missing values post-fill
missing_values_count_post = df.isnull().sum()
print(missing_values_count_post)
```

```
0      0
1      0
2      0
3      0
4      0
5      0
6      0
OT    18
```

Figure 12: Checking for remaining missing values

3.4 Training parameters

We chose the RandomForestRegressor for our modeling needs because it offers excellent performance on regression tasks involving complex and non-linear relationships. RandomForestRegressor is particularly robust to overfitting, which is essential in handling our dataset's potentially intricate patterns.
In our study, we enhanced our model by integrating six lagged features to capture historical influences on our target variable '6'. We, then, added our lagged features to the initial set to ensure that both current and past data points were considered. Our dataset was preprocessed by removing rows with missing values and splitting the data into training (90%) and test (10%) sets with a fixed random seed for reproducibility. The model was configured



```
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split

# Number of lags to include
n_lags = 6
# Create lag features
for lag in range(1, n_lags + 1):
    df[f'lag_{lag}'] = df['6'].shift(lag)

# Drop the initial rows that now contain NaN values due to shifting
df.dropna(inplace=True)

# Define the new feature set including the lags
features = ['0','1',  '2', '3','4', '5', 'OT'] + [f'lag_{i}' for i in range(1, n_lags + 1)]
X = df[features]
Y = df['6']

# Convert Y to a 1D array
Y = Y.ravel()

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.1, random_state=42)

# Train the model with the new features
model = RandomForestRegressor(n_estimators=382,max_depth=(28), random_state=42)
model.fit(X_train, y_train)
```

Figure 13: Building-Training RandomForestRegressor Model

with two (2) parameters: number of trees and a maximum depth to balance accuracy and computational efficiency. This setup, including the careful selection of features and hyperparameters, was strategically chosen to optimize the model's performance, enhancing its ability to generalize well to new data while mitigating the risk of overfitting.

3.5 Experiments

For the experimentation and testing phase, our initial focus was on exploring different techniques for handling missing values. We conducted multiple trainings, varying these methods to identify the most effective approach for our problem. Then, we attempted to enhance the performance of the trained model by starting with a basic random forest regressor. Subsequently, we adopted a more sophisticated approach, utilizing lagged features derived from the values of the target '6' feature. These lagged features were created to capture temporal dependencies and patterns in the data, thereby improving the predictive capabilities of the model.

Additionally, adjusting the number of trees in the forest and the maximum depth of the trees contributed significantly to refining the final model.
The table below 2 summarizes the latest results with the best parameters contributing to the improvement of the predictor's performance.

Trees numbre	Depth	MAE
380	20	0.0103
384	22	0.00947
385	22	0.00941
390	22	0.0101

Table 2: Parameter Settings and scores of best Tested Models

We conducted a total of **119 submissions**, resulting in a remarkable MAE score that reached as low as **0.009**.

3.6 Summary

In part B of the project, we tackled the problem of multivariate time series forecasting using the dataset of exchange rates.

The goal was to predict the next 100 values of the target variable '6', based on past observations of 8 variables representing exchange rates in different countries between 1990 and 2010. For this task, we chose to use the Random Forest Regressor model.

The choice of Random Forest Regressor was due to its robustness against noise and its ability to handle correlated variables, thanks to its ensemble approach. During the experimentation phase, we initially focused on exploring different techniques for managing missing values. We then sought to improve the performance of the trained model by using lagged variables derived from the values of the target variable '6'. These lagged variables were created to capture temporal dependencies and patterns in the data, thus enhancing the predictive capabilities of the model. Additionally, adjusting the number of trees in the forest and the maximum depth of the trees significantly contributed to refining the final model.

In summary, part B of the project demonstrated the effectiveness of the Random Forest regression model in addressing the challenge of multivariate time series forecasting, with an emphasis on managing missing values, variable selection, and hyperparameter optimization.

4 Conclusion

In this study, we have explored two distinct approaches to time series forecasting - a univariate problem involving electricity transformer temperatures, and a more complex multivariate challenge focused on exchange rate predictions.

For the univariate forecasting task, we leveraged the power of Long Short-Term Memory (LSTM) networks, which have demonstrated exceptional capabilities in capturing intricate temporal dependencies within sequential data. Through extensive experimentation and model refinement, we were able to achieve robust forecasting results, with a Mean Absolute Error (MAE) as low as 0.65 on the test set. This highlights the effectiveness of LSTM networks in predicting critical industrial metrics like transformer temperatures.

In the multivariate time series forecasting task, we employed the Random Forest Regressor, a powerful ensemble-based model known for its resilience to noise and ability to handle correlated features. By carefully managing missing values, incorporating lagged variables, and optimizing hyperparameters, we were able to achieve an impressive MAE of 0.009410 on the test set. This underscores the suitability of Random Forests for complex, real-world forecasting problems involving multiple interdependent variables.

The key takeaways from this study are the importance of strategic data preprocessing, the value of model experimentation and fine-tuning, and the complementary strengths of LSTM networks and Random Forest Regressors in tackling diverse time series forecasting challenges. These insights can inform future research and practical applications in critical economic and industrial domains.

While the results of this study are promising, there remains room for further exploration. Future work could investigate the integration of additional data features, the exploration of alternative neural network architectures, or the development of hybrid modeling approaches that leverage the unique advantages of different techniques.

References

[1] Jose Luis Aznarte. "Random forests for multivariate time series forecasting". In: *International Work-Conference on Time Series*. Springer. 2017, pp. 3–14.

[2] G'erard Biau. "Analysis of a random forests model". In: *The Journal of Machine Learning Research* 13.1 (2012), pp. 1063–1095.

[3] Leo Breiman. "Random forests". In: *Machine learning* 45.1 (2001), pp. 5–32.

[4] Steven Elsworth and Stefan Güttel. "Time Series Forecasting Using LSTM Networks: A Symbolic Approach". In: (Mar. 2020).

[5] Zhongyang Han et al. "A Review of Deep Learning Models for Time Series Prediction". In: *IEEE Sensors Journal* PP (June 2019), pp. 1–1. DOI: `10.1109/JSEN.2019.2923982`.

[6] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-term Memory". In: *Neural computation* 9 (Dec. 1997), pp. 1735–80. DOI: `10.1162/neco.1997.9.8.1735`.

[7] Masood Rana, Irena Koprinska, and Wil Van Der Aalst. "Forecasting remaining useful life of drain pumps in urban drainage systems using random forests". In: *Knowledge-Based Systems* 226 (2021), p. 107145.

[8] A. Suilin. *kaggle-web-traffic*. `https://github.com/Arturus/kaggle-web-traffic/`. Accessed on 19 November 2018. 2017.

[9] Bingqing Zhu, Bart Baesens, and Seppe KLG Vanden Broucke. “An empirical comparison of techniques for credit scoring”. In: *International Journal of Forecasting* 33.2 (2017), pp. 424–435.