AIR QUALITY TIME SERIES ANALYSIS REPORT

Author: Vaibhav Saran

UB ID: 50615031

E-Mail: vsaran@buffalo.edu

Introduction

The goal of this project was to analyse the time series data for Air Quality dataset provided by UCI Machine Learning Repository. The dataset contains the responses of a gas multisensory device deployed on the field in an Italian city. Hourly responses averages are recorded along with gas concentrations references from a certified analyser. Apart from time series analysis, the bias variance trade-off is also explored using the Iris dataset available in the scikit-learn module. The project is divided into 3 parts as follows:

Part 1: Data Pre-Processing

Dataset Overview

The UCI Air Quality dataset contains **9,358 instances** of **hourly averaged responses** from **5 metal oxide chemical sensors**. The **data spans** from **March 2004 to February 2005**, representing the longest freely available recordings of on-field deployed air quality chemical sensor devices.

Ground Truth hourly averaged concentrations for CO, Non Methanic Hydrocarbons, Benzene, Total Nitrogen Oxides (NOx) and Nitrogen Dioxide (NO2) and were provided by a co-located reference certified analyser.

The dataset is loaded using the pandas *read_csv()* function. Based on the data card information available from the dataset home page, **missing values** had been **replaced with -200** in the source dataset.

Handling Missing Values

In order to correctly understand the dataset, All the values with -200 were replaced with **NumPy Nan** values. The basic *info()* of the dataset post having null values was as follows:

Total rows	9357		
Column Name	**Non-Null Count**	**Null Count**	**Data Type**
Date	9357	0	datetime64[ns]
Time	9357	0	object
CO(GT)	7674	1683	float64
PT08.S1(CO)	8991	366	float64
NMHC(GT)	914	8443	float64
C6H6(GT)	8991	366	float64
PT08.S2(NMHC)	8991	366	float64
NOx(GT)	7718	1639	float64
PT08.S3(NOx)	8991	366	float64
NO2(GT)	7715	1642	float64
PT08.S4(NO2)	8991	366	float64
PT08.S5(O3)	8991	366	float64
T	8991	366	float64
RH	8991	366	float64
AH	8991	366	float64

Based on the above info, NMHC(GT) column has almost all the values null so it is dropped as it will not be able to contribute much in the dataset and the remaining missing values are filled using *ffill* method available in pandas.

Feature Selection

Now for doing feature selection, the following correlation matrix was used with the target feature as **PT08.S4(NO2):**

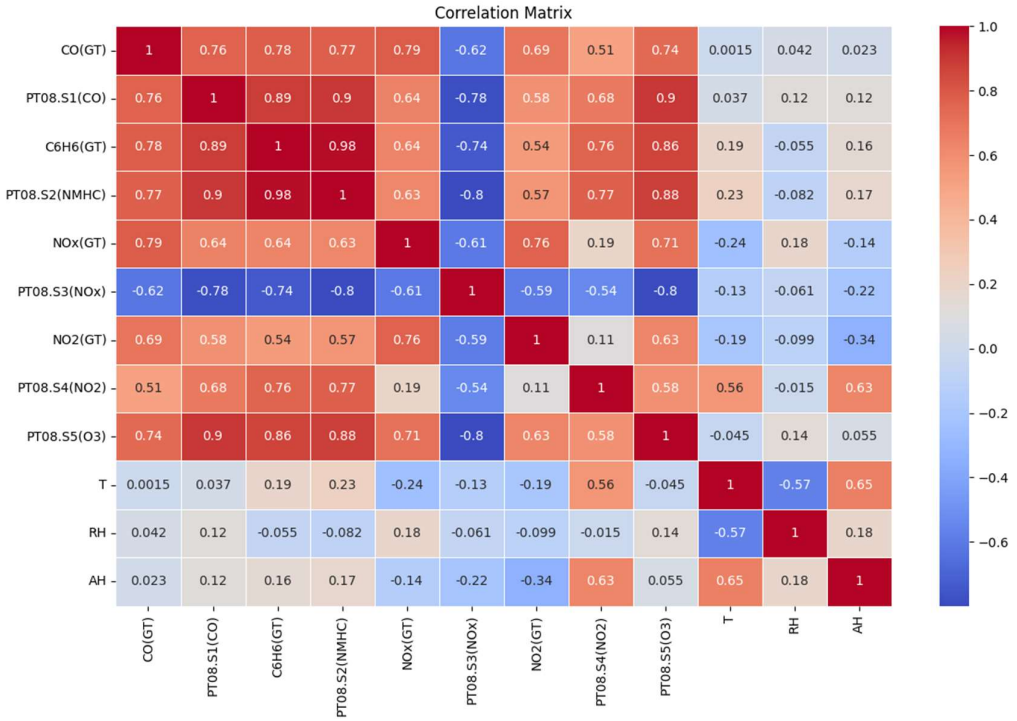

Figure 1

Based on the correlation heatmap in Figure 1, the **top 4 features with highest correlation** with the target column selected for further building process are: **PT08.S2(NMHC), C6H6(GT), PT08.S1(CO), AH**

Time Series Analysis of Best Features

The best features selected for the target column **PT08.S4(NO2)** are analysed using a time series plot as shown below:

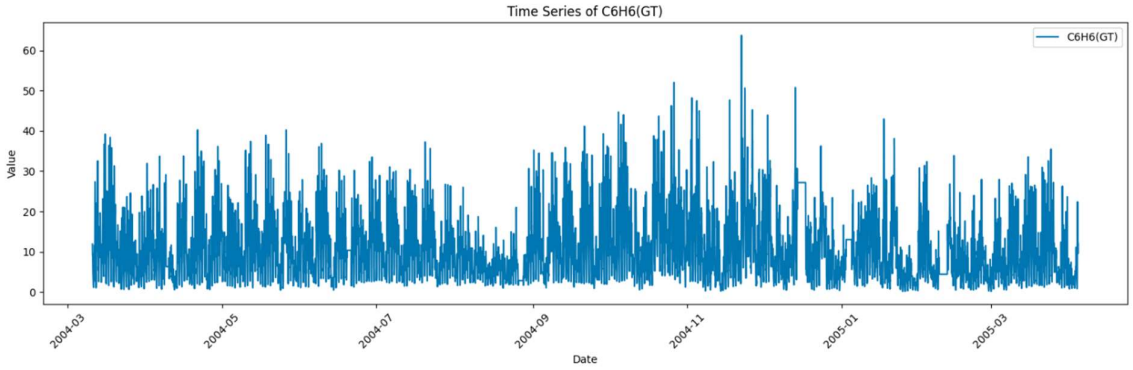

Figure 2

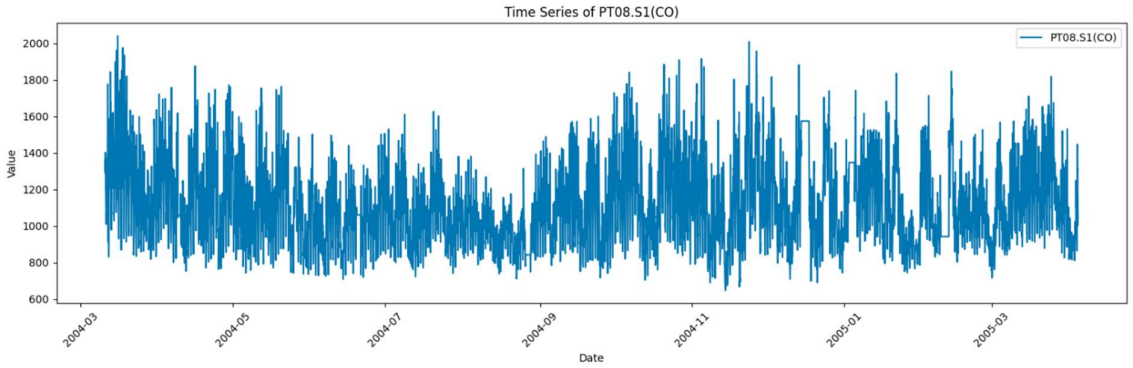

Figure 3

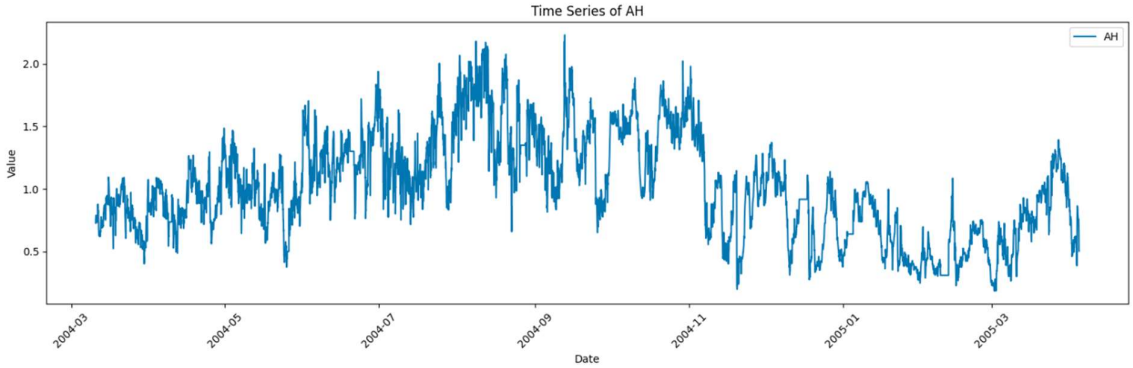

Figure 4

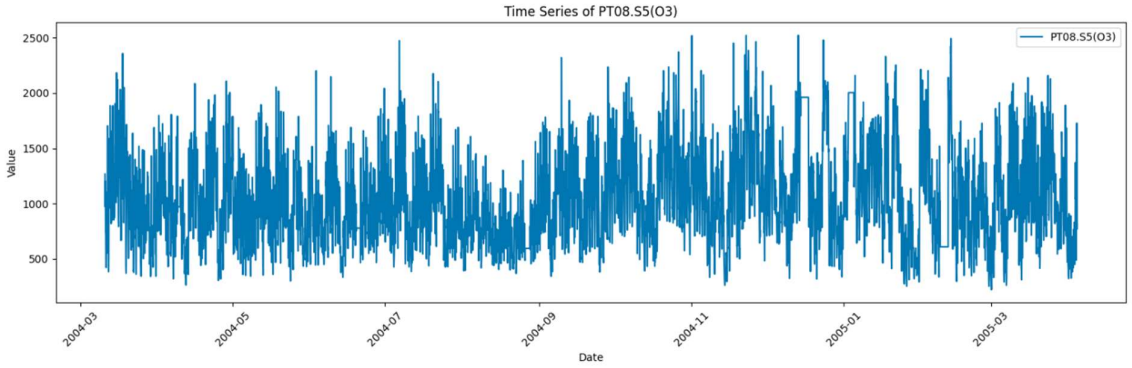

Figure 5

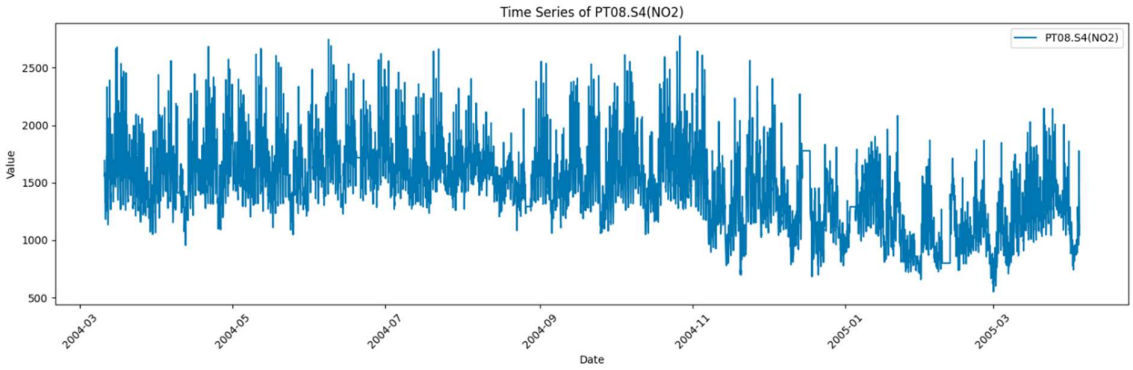

Figure 6

Now directly commenting on a time series plot can be a bit tricky for a time series plot, so in order to better understand and get confirmed observations, Dickey Fuller Test was used on all the selected features including the target feature as well; The test assumes a hypothesis that the given feature is non-stationary and tries to reject the null hypothesis, which gave the following results:

Column Name	Test Statistic	p-value	Number of Lags Used	Number of Observations Used	Critical Value (1%)	Critical Value (5%)	Critical Value (10%)
C6H6(GT)	-1.00E+01	1.63E-17	3.70E+01	9.32E+03	-3.43E+00	-2.86E+00	-2.57E+00
PT08.S1(CO)	-9.88E+00	3.88E-17	3.70E+01	9.32E+03	-3.43E+00	-2.86E+00	-2.57E+00
AH	-5.094234	1.4E-05	25	9331	-3.431051	-2.86185	-2.566935
PT08.S5(O3)	-1.08E+01	2.25E-19	3.60E+01	9.32E+03	-3.43E+00	-2.86E+00	-2.57E+00
PT08.S4(NO2)	-6.23E+00	4.96E-08	3.70E+01	9.32E+03	-3.43E+00	-2.86E+00	-2.57E+00

Based on the above plots and Dickey Fuller test values we observe the following:

- All features showed stationarity (**p-values < 0.05**)
- Strongest stationarity: **PT08.S5(O3) (-10.78 test statistic)**
- Target variable **PT08.S4(NO2) showed moderate stationarity** (**-6.23 test statistic**)

Now to visualize the stationarity claims of Dickey fuller test, rolling statistics for each of the column is plotted:

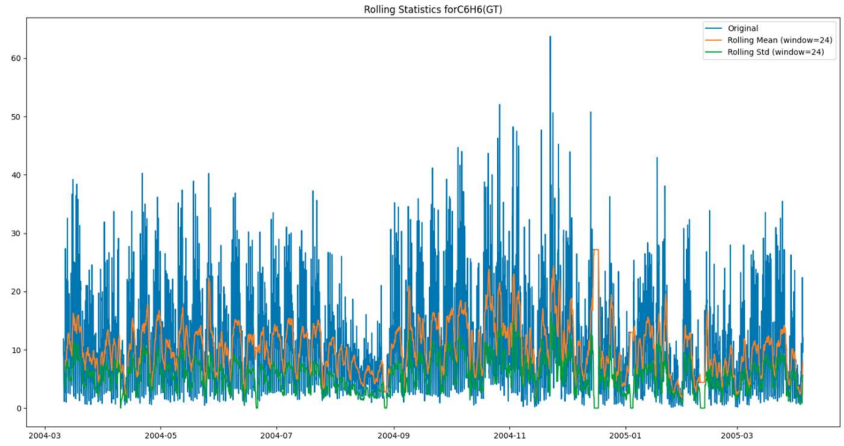

Figure 7

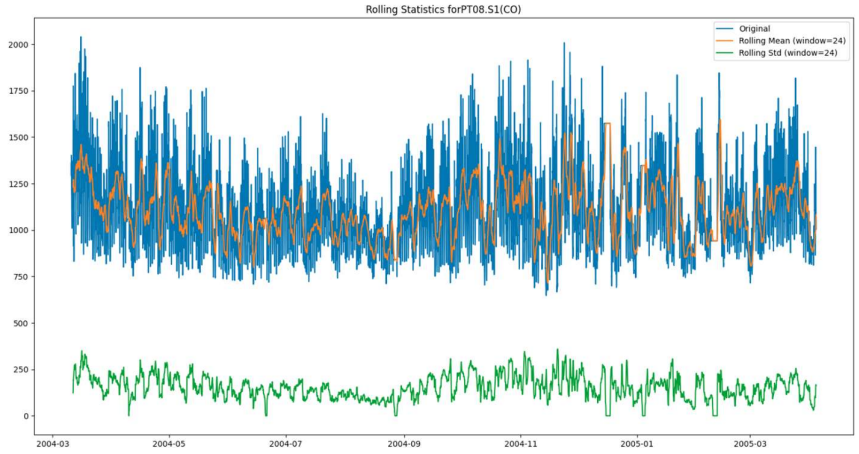

Figure 8

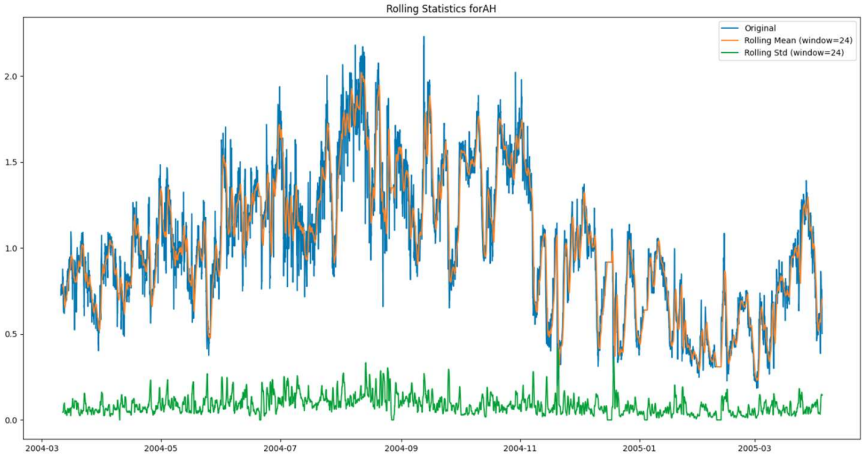

Figure 9

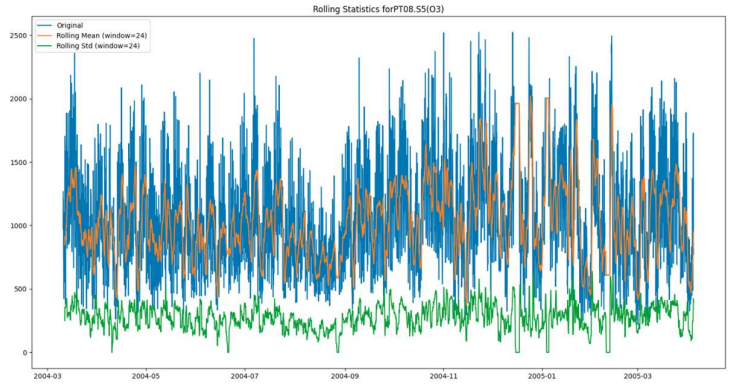

Figure 10

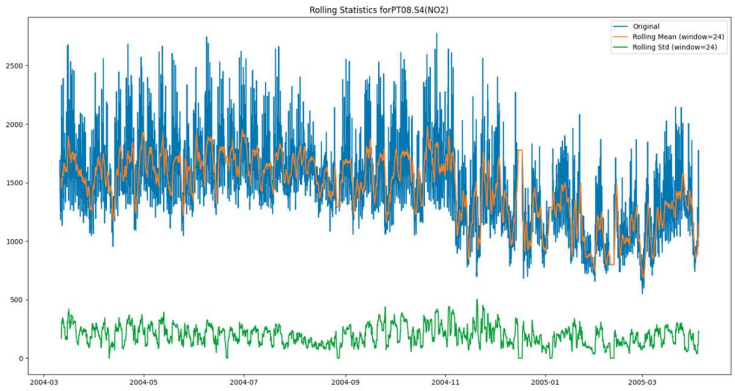

Figure 11

Based on the rolling statistics plots, dickey fuller tests and time series plots its observed that:

- **Most Stationary Features (based on test statistic values)**:
    - PT08.S5(O3): *-10.78*
    - C6H6(GT): *-10.02*
    - PT08.S1(CO): *-9.87*
- **Moderately Stationary Feature**
    - PT08.S4(NO2) (**target**): *-6.23*
- **Least Stationary (but still stationary)**:
    - AH: *-5.09*
- All features are stationary at all confidence levels (1%, 5%, and 10%)
- The feature **AH** may not look stationary in the time series plot, but by looking at the p-value from the dickey fuller test and rolling statistics plot, its proved that AH is a stationary feature.

So, this concludes that the features are stationary and no other transformations are necessary to make them stationary, as a result this makes the dataset suitable for time series analysis going forward as the high overall stationarity suggests good predictability and forecasting potential as well as the consistent sample size of data indicate that the test results should reliable enough.

Preparing The Data for Model Building

For the selected features, first the data is scaled using the scikit-learn min-max scaler, followed by converting the dataset into sequences. The data was transformed into sequential windows of 10-time steps each for two key reasons:

1. **Temporal Dependencies**
    - Air quality parameters show both immediate and lagged relationships
    - Current pollution levels are influenced by conditions from previous hours
    - Sequential structure preserves these time-based relationships

2. **Model Requirements**
    - RNN and LSTM architectures are designed for sequential data processing
    - 10-step sequences allow models to learn patterns across time
    - This structure enables detection of both short-term fluctuations and longer trends

The effectiveness of this approach is demonstrated by our models' ability to capture complex temporal dynamics in air quality parameters, leading to more accurate predictions. The sequence length of 10 was selected to balance temporal pattern capture with computational efficiency.

After this, since the dataset is large enough, it is split into train and test set of 80-20.

Part 2: Modelling and Evaluation

In this section, following 2 models were built and to ensure that the results are reproducible in future, the random seed was initialized to 42, and the architectures which utilized the seeds are:

- RNN Architecture
- LSTM Architecture

RNN Architecture

RNN (Recurrent Neural Network) is a neural network designed to handle sequential data by maintaining an internal memory state, allowing it to process each element in a sequence while considering information from previous elements through a feedback loop mechanism.

The model architecture summary for the given problem is as shown in the figure:

Model Summary:
Model: "sequential"

Layer (type)	Output Shape	Param #
simple_rnn (SimpleRNN)	(None, 10, 64)	4,480
simple_rnn_1 (SimpleRNN)	(None, 32)	3,104
dense (Dense)	(None, 16)	528
dense_1 (Dense)	(None, 1)	17

Total params: 8,129 (31.75 KB)
Trainable params: 8,129 (31.75 KB)
Non-trainable params: 0 (0.00 B)

Figure 12

Also, the RNN model comprised of callbacks containing early stopping with a patience level of 5 while observing validation scores as well as creating a model checkpoint for the best model of trained RNN.

The RNN architecture was trained for 20 epochs, and the following plots show the training curve and evaluation of the RNN architecture along with its statistics:

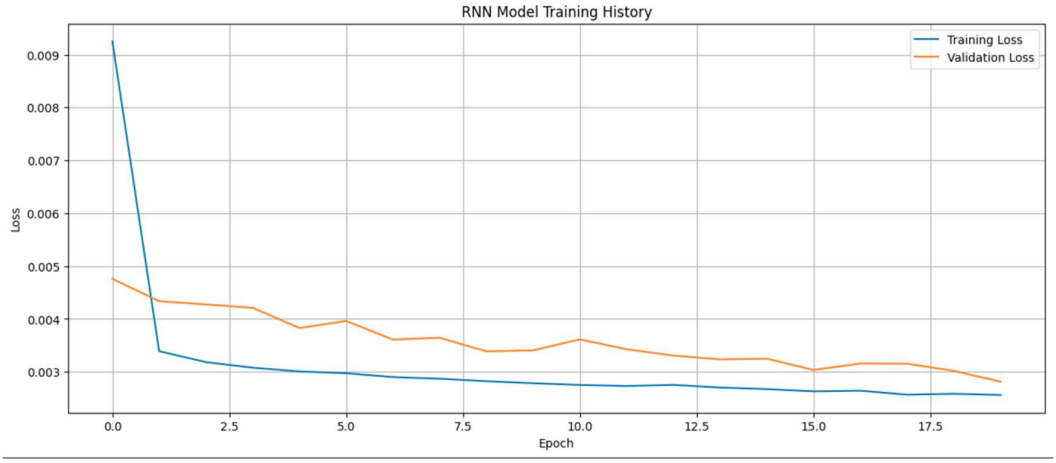

Figure 13

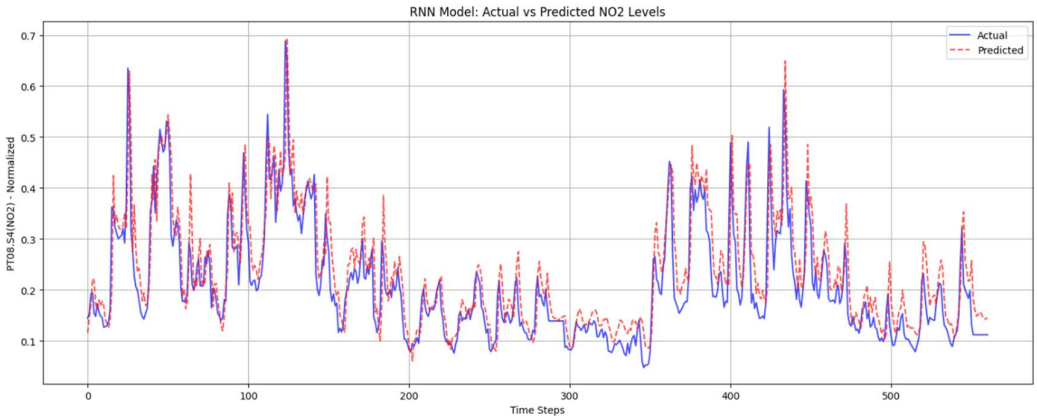

Figure 14

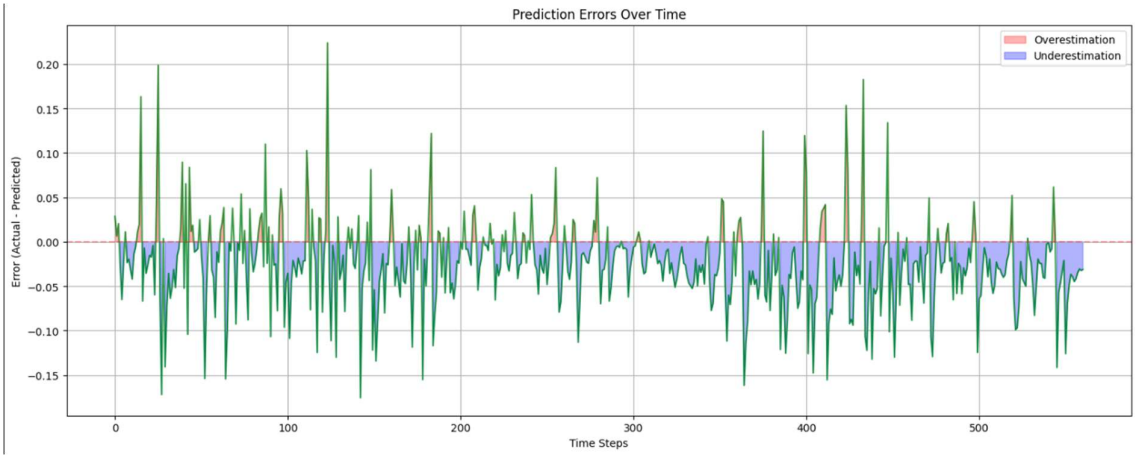

Figure 15

And the evaluation metrics gave the following results:

- **MSE: 0.0030**
- **RMSE: 0.0547**
- **MAE: 0.0409**
- **MAPE: 22.23%**
- **R^2 Score: 0.7572**

Based on the above plots and evaluation statistics, we can observe the following for RNN architecture:

1. The model uses a sequential architecture with **8,129 trainable parameters**.
    - **2** Simple RNN layers (64 and 32 units) with **ReLU activation** function have been used.
    - **2** Dense layers (16 units and 1 output unit)
    - This type of shallow RNN is on a lighter end as the task at hand is not very demanding, however for future scaling purposes, a more complex and deep architecture can be explored.

2. **RNN Training Loss Curve Analysis**
    - *Rapid initial convergence* in the *first 2-3 epochs*, with *training loss dropping* sharply from *0.009 to about 0.003*.
    - *Stable learning after epoch 5*, with both training and validation loss showing gradual improvement.
    - Small *gap between training and validation loss* indicates *good generalization* and suggests optimal convergence.

3. **RNN Prediction Analysis**
    - *Strong tracking of major trends in NO2 levels* as seen in Actual V/S Predicted Plot

- Particularly *accurate in predicting*:
    - Peak values (around 0.6-0.7 range)
    - General pattern transitions
    - Low-value regions (0.1-0.2 range)
- RNN maintains prediction accuracy across different value ranges with good consistency

4. **Error Analysis for RNN Architecture**
    - Error Distribution
        - *Mean error* of **-0.0249** indicates a **slight systematic overestimation bias**, i.e. the model is slightly overestimating in all its predictions.
        - Error *standard deviation* of **0.0487** shows consistent prediction accuracy.
        - *Error range* **[-0.1756, 0.2238]** is relatively symmetric around zero.
    - Temporal Error Patterns
        - Larger errors tend to occur during rapid value changes
        - More stable predictions during periods of steady NO2 levels
        - Both overestimation and underestimation errors are present, with slightly more overestimation

5. **Quantitative Performance**
    - Strong overall performance with $R^2 = 0.7572$ (75.72% variance explained).
    - Low MSE (0.0030) and RMSE (0.0547) indicate *good prediction accuracy*.
    - MAPE of 22.23% suggests *reasonable relative accuracy*.
    - MAE of 0.0409 shows *good absolute accuracy* in normalized scale.

So overall for RNN we were able to get convergent training along with a good generalization capability. It is able to track general trends and the prediction across different time steps are quite stable.

LSTM Architecture

LSTM (Long Short-Term Memory) is an advanced version of RNN that solves the vanishing gradient problem by using specialized gates (input, forget, output) to selectively remember or forget information, making it better at capturing both long-term and short-term patterns in sequential data.

The LSTM architecture was designed and trained similar to RNN in order to have a fair comparison between the two and the details for which are as shown in the figure:

LSTM Model Summary:

Model: "sequential_1"

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 10, 64)	17,920
lstm_1 (LSTM)	(None, 32)	12,416
dense_2 (Dense)	(None, 16)	528
dense_3 (Dense)	(None, 1)	17

Total params: 30,881 (120.63 KB)

Trainable params: 30,881 (120.63 KB)

Non-trainable params: 0 (0.00 B)

Figure 16

Also, the LSTM model comprised of callbacks containing early stopping with a patience level of 5 while observing validation scores as well as creating a model checkpoint for the best model of trained LSTM.

The LSTM architecture was trained for 20 epochs, and the following plots show the training curve and evaluation of the LSTM architecture along with its statistics:

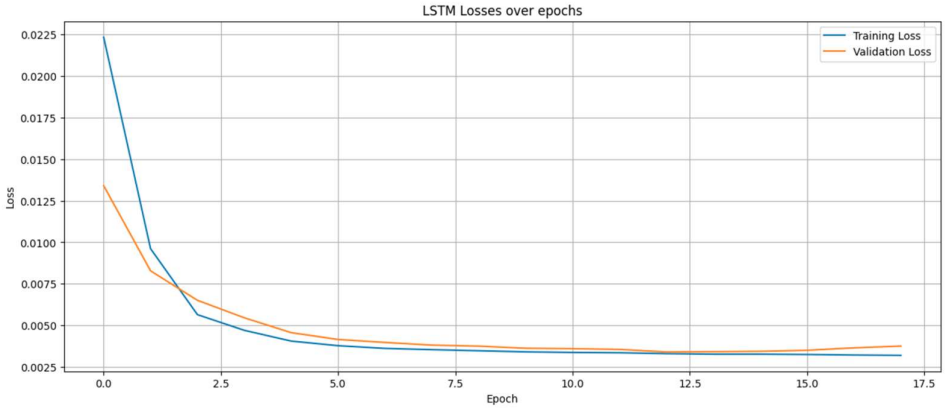

Figure 17

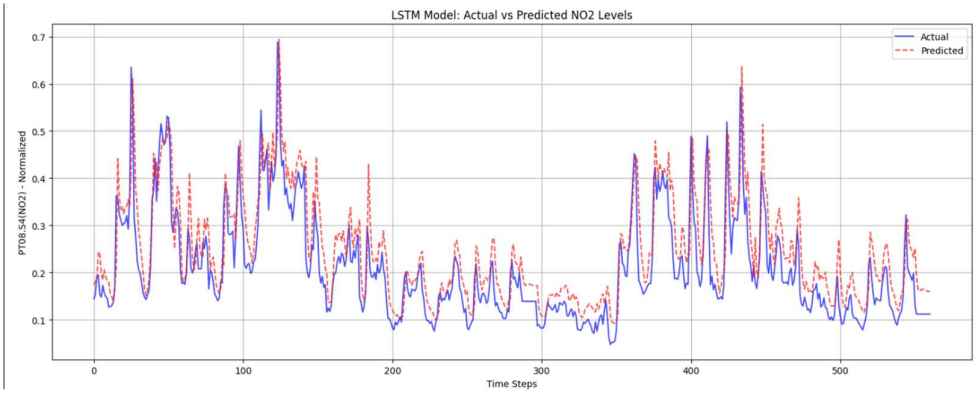

Figure 18

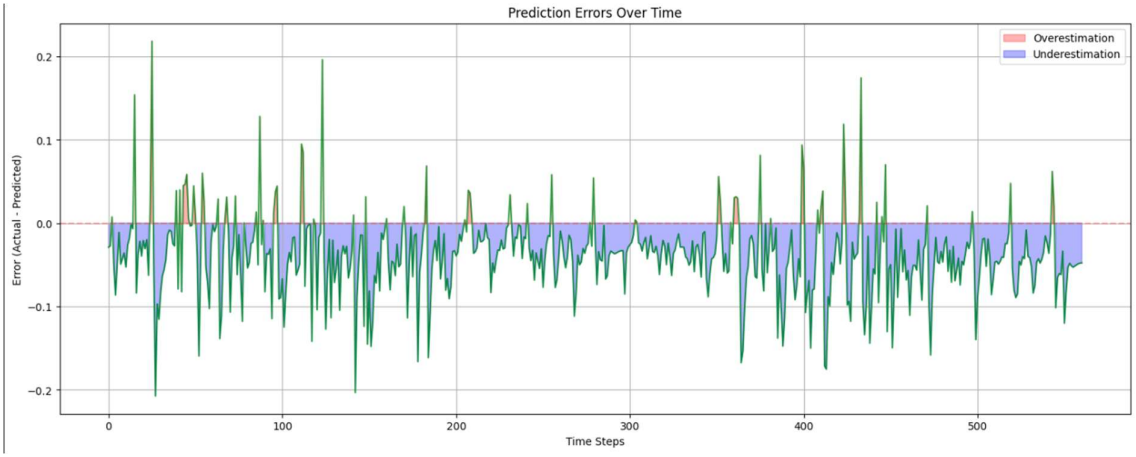

Figure 19

And the evaluation metrics for LSTM gave the following results:

- **MSE: 0.0042**
- **RMSE: 0.0648**
- **MAE: 0.0529**
- **MAPE: 30.16%**
- **R^2 Score: 0.6588**

Based on the above plots and evaluation statistics, we can observe the following for LSTM architecture:

1. **Model Architecture Analysis**
    - LSTM model is more complex with **30,881 trainable parameters** compared to **RNN's 8,129**.
    - The layer structure is same as RNN but with LSTM cells replacing SimpleRNN cells.
    - We observe a significant increase in parameters due to LSTM's additional gates.
    - Model size: **120.63 KB** (larger than RNN due to more complex architecture)
2. **LSTM Training Performance**
    - *Very rapid initial convergence*:
        - First epoch: *validation loss dropped* from 0.0499 to 0.0134
        - By epoch 5 the network stabilized around *0.0042 (training)* and *0.0046 (validation)*
    - Training completed 19 epochs before early stopping.
    - Small gap between training and validation loss indicates good generalization
    - Final training loss (0.0032) slightly better than validation loss (0.0040)
3. **LSTM Prediction Performance**
    - Generally good tracking of NO2 level trends

- Particularly accurate in:
    - Capturing peak values (around 0.6-0.7 range)
    - Following major trend changes in the data
- Slight overestimation tendency in lower value regions
- More pronounced lag in rapid transitions compared to RNN

4. **Error Analysis for LSTM Architecture**
    - Error Distribution
        - *Mean error* of **-0.0438** indicates **stronger overestimation bias than RNN (-0.0249)**
        - *Similar error standard deviation (0.0478) to RNN (0.0487)*.
        - *Error range* **[-0.2147, 0.2123]**
    - Temporal Error Patterns
        - *Larger errors during rapid transitions*
        - *More consistent overestimation pattern*
        - *Error magnitude increases with value volatility*
5. **Performance Comparison of RNN V/S LSTM**

Metric	LSTM	RNN	Conclusion
MSE	0.004	0.003	RNN is better
RMSE	0.065	0.055	RNN is better
MAE	0.053	0.041	RNN is better
MAPE	0.302	0.222	RNN is better
R^2 Score	0.659	0.757	RNN is better

So based on the above pointers and comparisons we observe that:

- RNN outperformed LSTM on all metrics for this specific dataset
- *LSTM shows stronger overestimation bias (-0.0438 vs -0.0249)*
- *LSTM required more **computational resources** (**3.8x more parameters**)*
- Both models show good convergence but **RNN achieved better final accuracy**

The RNN likely outperformed the LSTM on this specific dataset due to the dataset's characteristics and the models' complexities. The Air Quality dataset consists of hourly measurements, which means the relevant temporal dependencies are relatively short-term. RNNs are well-suited for capturing short-term dependencies, while LSTMs excel at capturing long-term dependencies due to their gating

mechanisms. Given the nature of the data, the additional complexity of LSTM cells may not provide significant benefits over simple RNN cells.

Moreover, the RNN model has fewer parameters (8,129) compared to the LSTM model (30,881). The increased complexity of the LSTM model might lead to slight overfitting on this particular dataset, resulting in lower performance. The simpler RNN architecture seems sufficient to capture the necessary patterns without introducing excess complexity.

Part 3: Bias and Variance

Bias and variance are essential concepts in machine learning that helps understand model performance on a given problem statement and dataset as well as guide model optimization. These concepts are fundamental to achieving reliable predictions while avoiding common pitfalls in model development.

Bias refers to the error that occurs due to overly simplified assumptions in a learning algorithm. **A high-bias model consistently misses relevant relations between features and target outputs**. It represents the difference between the expected predictions of our model and the actual values we aim to predict.

When a model has **high bias**, it tends to **underfit the data**, meaning it fails to capture the underlying patterns in the training data. For instance, attempting to model a complex nonlinear relationship with a simple linear model would introduce high bias.

Variance, in contrast, **measures how much the model's predictions fluctuate for different training datasets**.

A **high-variance** model is extremely sensitive to small fluctuations in the training data. Such models tend to **overfit**, meaning they capture not just the underlying patterns but also the noise in the training data.

These models perform exceptionally well on training data but fail to generalize to new, unseen data points.

The bias-variance trade-off represents a fundamental conflict in machine learning model development.

As we increase model complexity to reduce bias, we typically increase variance. Conversely, as we decrease model complexity to reduce variance, we typically increase bias.

For e.g., a simple linear regression model might have high bias but low variance, consistently making predictions that are off-target but clustered together.

On the other hand, a complex polynomial regression model might have low bias but high variance, making predictions that are sometimes accurate but highly inconsistent across different datasets.

The goal in model development is to find the sweet spot where the combined error from both bias and variance is minimized.

This typically involves careful model selection, appropriate feature engineering, and proper regularization techniques. This balance is particularly relevant in real-world applications where both prediction accuracy and model reliability are crucial.

Too much bias leads to consistently inaccurate predictions, while too much variance leads to unstable and unreliable models.

The key is to develop models that are sophisticated enough to capture true underlying patterns while remaining simple enough to generalize well to new data.

Now, the trade-off is implemented and show cased by taking the iris dataset available in scikit-learn module and dividing it into **10 subsets**.

The split of data into train and test is done on the standard 80-20 split and Decision Tree Classifier is trained on the subsets and the following results were obtained:

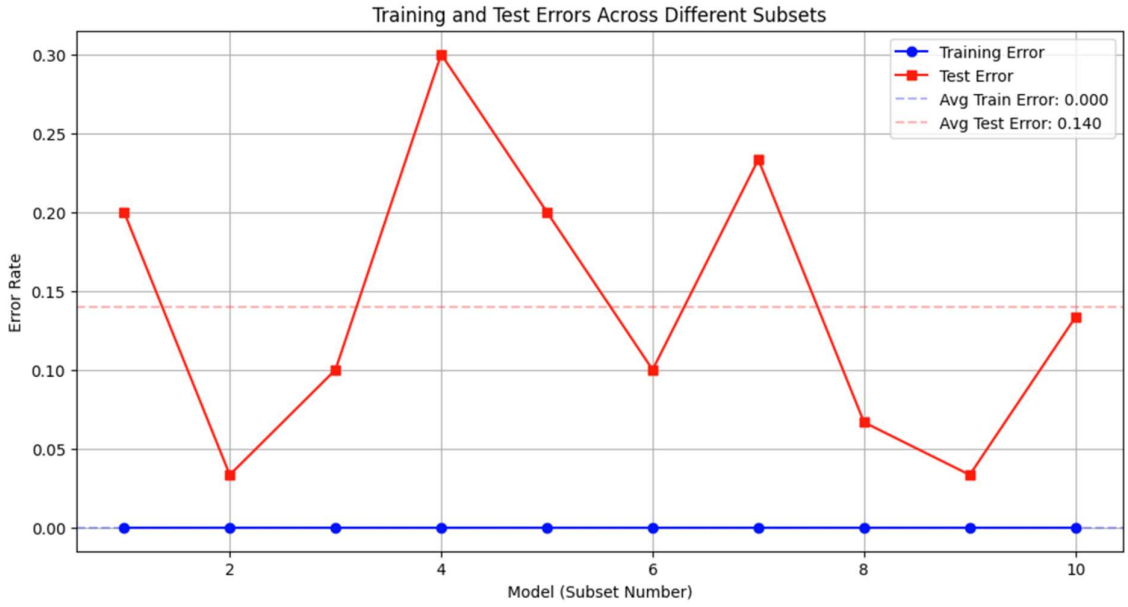

Figure 20

And the list of all the classifier models trained on subset are categorized in the below table:

Model	Tree Depth	Number of Leaves	Training Error	Test Error
1	2	3	0	0.2
2	2	3	0	0.033
3	2	3	0	0.1
4	2	3	0	0.3
5	2	3	0	0.2
6	2	3	0	0.1
7	2	3	0	0.233
8	2	3	0	0.067
9	2	3	0	0.033
10	2	3	0	0.133

Based on the above plot and table, the following can be observed:

1. **Model Performance Overview**
    - Training Error
        - Perfect consistency: **0.000 ± 0.000**
        - *Zero training error across all models*
        - Indicates *potential overfitting*
    - Test Error
        - *Average*: **0.140 ± 0.085**
        - *Range*: **0.033 to 0.300**
        - *High variability in test performance*
2. **Bias-Variance Analysis**
    - Bias (Systematic Error)
        - Measured by *average test error: 0.140*
        - **Moderate bias level**
        - Consistent gap between train and test performance
    - Variance (Prediction Variability)
        - Measured by *test error std: 0.085*
        - *High variance* indicated by:
            - Fluctuating test errors
            - Large range in test performance
            - Inconsistent model behaviour
3. **Model Complexity Analysis**
    - Structural Consistency
        - All models show identical complexity:
            - Tree Depth: 2
            - Number of Leaves: 3
        - Suggests structural stability
    - Performance Variability
        - Despite identical structure:
            - Best Test Error: 0.033 (Models 2 & 9)
            - Worst Test Error: 0.300 (Model 4)

        - *High performance variance with same architecture*

4. **Best Model Features**
    - Model 2 Performance
        - *Training Error*: *0.000*
        - *Test Error*: *0.033*
        - *Best generalization among all models*
        - **96.7% accuracy on test set**
5. **Overfitting Issues**
    - Zero training error in all models
    - Significant gap between train and test errors
    - High variance in test performance
6. **Conclusion**
    - Trade-off Assessment
        - **High Variance Problem**
            - Test error varies significantly (0.033-0.300)
            - Sensitive to training data differences
            - *Unstable generalization*
        - **Low Bias Problem**
            - Perfect training accuracy
            - Simple model structure
            - *Potential memorization of training data*

All the decision tree models have the same complexity, with a tree depth of 2 and 3 leaves, despite varying performance on the test set. This is likely due to the simplicity of the Iris dataset and the default hyperparameters used for the DecisionTreeClassifier.

The Iris dataset consists of only 4 features and 3 classes, making it a relatively simple classification problem. With the default settings, the decision trees are able to perfectly fit the training data from each subset, resulting in zero training error. However, the models' performance on the test set varies due to the differences in the subsets and the randomness introduced during subset creation.

The consistent model complexity suggests that the decision trees are not growing overly complex and are limited by the maximum depth or other stopping criteria. Increasing the model complexity (e.g., allowing deeper trees) might lead to even higher variance and overfitting.

References

1. Air Quality - UCI Machine Learning Repository
2. GitHub - VaibhavSaran/FB-PROPHET-TESLA-TOY-DATASET
3. TimeSeries Analysis 📈A Complete Guide 📚
4. Prophet | Forecasting at scale.
5. Time Series Forecasting With Prophet in Python - MachineLearningMastery.com
6. Time Series Forecasts using Facebook's Prophet
7. Time Series Analysis with Facebook Prophet: How it works and How to use it | by Mitchell Krieger | Towards Data Science
8. Stan - Stan
9. Time Series Prediction with LSTM Recurrent Neural Networks in Python with Keras - MachineLearningMastery.com
10. How to Develop LSTM Models for Time Series Forecasting - MachineLearningMastery.com
11. LSTM Framework For Univariate Time-Series Prediction | by Joseph (Iosif) Mushailov | Towards Data Science
12. Multivariate Time Series Forecasting with LSTMs in Keras - MachineLearningMastery.com
13. Multivariate Time Series Forecasting with LSTMs in Keras
14. Hacking Time-Series Forecasting Like a Pro with FBProphet | by Nikolaus Herjuno Sapto Dwi Atmojo | Tokopedia Data | Medium
15. Gentle Introduction to the Bias-Variance Trade-Off in Machine Learning - MachineLearningMastery.com
16. Base RNN layer
17. LSTM layer
18. SimpleRNN layer
19. Working with RNNs | TensorFlow Core
20. Understanding the Bias-Variance Tradeoff | by Seema Singh | Towards Data Science
21. Lecture 12: Bias Variance Tradeoff