Machine Learning Models for Outdoor Air Quality Prediction

ENPH 455 Final Report

Mack Sell

Queen's University

Kingston, Ontario, Canada

April 2020

A thesis submitted to the

Department of Physics, Engineering Physics and Astronomy

in conformity with the requirements for

the degree of Bachelor of Applied Science

Abstract

In many cities around the world, concern is growing over air pollution and its negative impact on health and quality of living. The accurate forecasting of air pollution can help minimize the harm it causes. One of the more dangerous air pollutants is particulate matter less than 2.5 microns in diameter ($PM_{2.5}$). In this report, a SVR, NARX-NN, and LSTM, which are three different machine learning models, were used to predict hourly concentrations of $PM_{2.5}$ across six major Canadian cities. Datasets that included hourly temperature, dew point, air pressure, wind speed, ozone concentration, nitrogen oxide concentration, and $PM_{2.5}$ concentration were formed for each city to train these models. The NARX-NN was found to consistently perform the worst across all cities. The LSTM performed best in four of the six cities. These models were compared to other published models including the current Environment Canada model. The SVR and LSTM were found to have lower RMSE than the published models across all cities. This comparison is not entirely fair though considering that the datasets used were not identical to those used in the published models. The results justify further comparison of the LSTM with the published models and show that out of the three developed models the LSTM is the best choice for pollutant forecasting on larger scales.

Table of Contents

1 Introduction

1.1 Motivation 1

1.2 Goals 1

2 Background Research

2.1 Current Environment Canada Model 2

2.2 UBC Graduate Thesis 2

2.3 Additional Papers/Studies 3

3 Data

3.1 Data Sources 4

3.2 Data Collection 5

3.3 Data Correlation 7

3.4 Preprocessing for Time Series Forecasting in Machine Learning 8

4 Models

4.1 Support Vector Regression (SVR) 9

Theory 9

Implementation and Design 9

4.2 Non-Linear Auto-Regressive Exogenous Neural Network (NARX-NN) 10

Theory 10

Implementation and Design 13

4.3 Long Short-Term Memory (LSTM) 14

Theory 14

Implementation and Design 15

5 Results and Discussion

Datasets 16

Models 19

Future Work 24

6 Appendices

6.1 Appendix A: Word Count and Previous Work 25

6.2 Appendix B: References 28

6.3 Appendix C: Additional Tables and Figures 29

6.4 Appendix D: Figures and Tables for Remaining Cities 26

6.5 Appendix E: Example Code 46

List of Figures

Figure 1: Displays the 250 stations used by UMOS-AQ [3]. 2
Figure 2: Displays the process of using a kernel and hyperplane to make regions within data [9]. 9
Figure 3: Displays a very simple feed-forward neural network..... 11
Figure 4: Displays the sigmoid and tanh functions that are typically used as activation functions. 11
Figure 5: Displays the way an RNN passes information from previous time steps [7]. 12
Figure 6: Displays the activation function that is contained in one module of an RNN [7]. 12
Figure 7: Displays the impact of different learning rates on minimizing the cost function 14
Figure 8: Displays the cell state as well as the forget, input, and output gates of an LSTM [7]. 14
Figure 9: Displays, from left to right, the forget gate, input gate, and output gate of the LSTM [7]. 14
Figure 10: Plots of the seven variables with time for Toronto dataset. 16
Figure 11: Displays the correlations between PM2.5 and each of the other variables for Toronto. 17
Figure 12: A histogram of the PM2.5 data for Toronto. 18
Figure 13: The predicted values of the SVR Model compared to the actual test set data for Toronto. 19
Figure 14: The RMSE over multiple epochs for the Toronto NARX-NN. 20
Figure 15: Displays the predicted values of the NARX-NN Model and the test set data for Toronto. 20
Figure 16: RMSE for the validation and training data over multiple epochs for the Toronto LSTM. 21
Figure 17: Displays the predicted values of the LSMT Model compared to the test data for Toronto. 21
Figure 18: The RMSE for each city and each model from this thesis and the UBC Graduate Thesis. 22
Figure 19: Plots of each of the seven variables with time for Edmonton dataset. 29
Figure 20: The correlations between PM2.5 and each of the other variables for Edmonton. 29
Figure 21: A histogram of the PM2.5 data for Edmonton. 30
Figure 22: The predicted values of the SVR Model compared to the actual test set data for Edmonton. 30
Figure 23: The RMSE over multiple epochs for the Edmonton NARX-NN. 30
Figure 24: Displays predicted values of the NARX-NN Model compared to the test data for Edmonton.. 31
Figure 25: RMSE for the validation and training data over multiple epochs for the Edmonton LSTM. 31
Figure 26: Displays the predicted values of the LSMT Model compared to the test data for Edmonton.. 31
Figure 27: Plots the seven variables with time for Halifax dataset. 32
Figure 28: The correlations between PM2.5 and each of the other variables for Halifax. 32
Figure 29: A histogram of the PM2.5 data for Halifax. 33
Figure 30: The predicted values of the SVR Model compared to the actual test set data for Halifax. 33

Figure 31: The RMSE over multiple epochs for the Halifax NARX-NN. 33
Figure 32: Displays the predicted values of the NARX-NN Model compared to the test set for Halifax. 34
Figure 33: RMSE for the validation and training data over epochs for the Halifax LSTM. 34
Figure 34: Displays the predicted values of the LSMT Model compared to the test data for Halifax. 34
Figure 35: Plots the six variables with time for Montreal dataset. 35
Figure 36: The correlations between PM2.5 and each of the other variables for Montreal. 35
Figure 37: A histogram of the PM2.5 data for Montreal. 36
Figure 38: The predicted values of the SVR Model compared to the actual test set data for Montreal. 36
Figure 39: The RMSE over multiple epochs for the Montreal NARX-NN. 36
Figure 40: Displays predicted values of the NARX-NN Model compared to the test data for Montreal. 37
Figure 41: RMSE for the validation and training data over epochs for the Montreal LSTM. 37
Figure 42: Displays the predicted values of the LSMT Model compared to the actual test set data for Montreal. 37
Figure 43: Plots of the seven variables with time for Vancouver Dataset. 38
Figure 44: Correlations between PM2.5 and each of the other variables for Vancouver. 38
Figure 45: A histogram of the PM2.5 data for Vancouver. 39
Figure 46: The predicted values of the SVR Model compared to the actual test set data for Vancouver. 39
Figure 47: The RMSE over multiple epochs for the Vancouver NARX-NN. 39
Figure 48: Displays predicted values of the NARX-NN Model compared to the test data for Vancouver. 40
Figure 49: RMSE for the validation and training data over epochs for the Vancouver LSTM. 40
Figure 50: Displays the predicted values of the LSTM Model compared to the test set for Vancouver. 40
Figure 51: Plots each of the seven variables with time for Winnipeg dataset. 41
Figure 52: Correlations between PM2.5 and each of the other variables for Winnipeg. 41
Figure 53: A histogram of the PM2.5 data for Winnipeg. 42
Figure 54: The predicted values of the SVR Model compared to the actual test set data for Winnipeg. 42
Figure 55: The RMSE over multiple epochs for the Winnipeg NARX-NN. 42
Figure 56: The predicted values of the NARX-NN Model compared to the test data for Winnipeg. 43
Figure 57: RMSE for the validation and training data over epochs for the Winnipeg LSTM. 43
Figure 58: The predicted values of the LSMT Model compared to the actual test set data for Winnipeg. 43
Figure 59: Displays how each of the seven variables change with time for Beijing. 44
Figure 60: The correlations between PM2.5 and each of the other variables for Beijing. 44
Figure 61: A histogram of the PM2.5 data for Beijing. 45

List of Tables

Table 1: Displays the number of missing data points corrected for each city's dataset. 7
Table 2: Mean, median, mode, and standard deviation for each city's $PM_{2.5}$ data. 18
Table 3: The tuned parameters that achieved the best RMSE for each model and each city. 19
Table 4: The best RMSE achieved for each city and model. 22
Table 5: Names and locations for each weather and pollution data station used for each city. 28
Table 6: Displays processed data to be used in the models for Montreal. 28

1 Introduction

1.1 Motivation

The rapid industrialization of developing countries and the increasing frequency of climate change related disasters, such as wildfires, have raised concerns about air pollution around the world. For example, in China, 78.4% of 338 monitored cities were below healthy air quality standards [1]. Air pollution is defined as harmful substances emitted into the atmosphere. This generally includes sulphur dioxide (SO_2), nitrogen oxides (NO_x), ozone (O_3), particulate matter (PM), carbon monoxide (CO), and volatile organic compounds (VOCs) [2]. Every year, these components cause an estimated 4.3 million people to die prematurely [2]. $PM_{2.5}$ is particulate matter with a diameter less than 2.5 μm and is one of the more dangerous components of air pollution. The ability of the small particles to penetrate into the lungs can cause serious respiratory problems. This is, in part, why $PM_{2.5}$ will be predicted in this report over the other pollutants. The accurate forecasting of air pollution can reduce the negative impact it has on the health of a population by providing time for those most at risk to prepare accordingly.

1.2 Goals

The goal of this thesis is to implement and test multiple machine learning (ML) models for the prediction of hourly concentrations of $PM_{2.5}$ and compare them to the OSELM and UMOS-AQ models developed by a UBC student and Environment Canada, respectively. Specifically, three separate ML models, a SVR, NARX-NN, and LSTM, will be used to make hourly predictions for six cities including Vancouver, Toronto, Halifax, Edmonton, Winnipeg, and Montreal. These models will use some of the same input parameters as OSELM and UMOS-AQ, which includes historical hourly O_3, $PM_{2.5}$, and NO_X concentrations, as well as a variety of meteorological data. Datasets to train these models will be created using free, publicly available data. They will be implemented using various ML libraries in Python in Google Colabs. The developed models will be compared with each other as well as OSELM and UMOS-AQ for the six cities.

2 Background Research

2.1 Current Environment Canada Model

Environment Canada (EC) currently uses the GEM-MACH15 (Global Environmental Multiscale Model - Modelling Air Quality and Chemistry) for air quality (AQ) forecasting [3]. To supplement this model, EC does statistical post-processing, which improves a model's accuracy by using statistical methods to relate other data to the model outputs. EC uses a multivariate linear regression (MLR) model to do this called UMOS-AQ (Updatable Model Output Statistics – Air Quality) that predicts hourly concentrations of ozone, particulate matter, and nitrogen dioxide [3]. This model uses observations from 250 AQ stations across Canada.

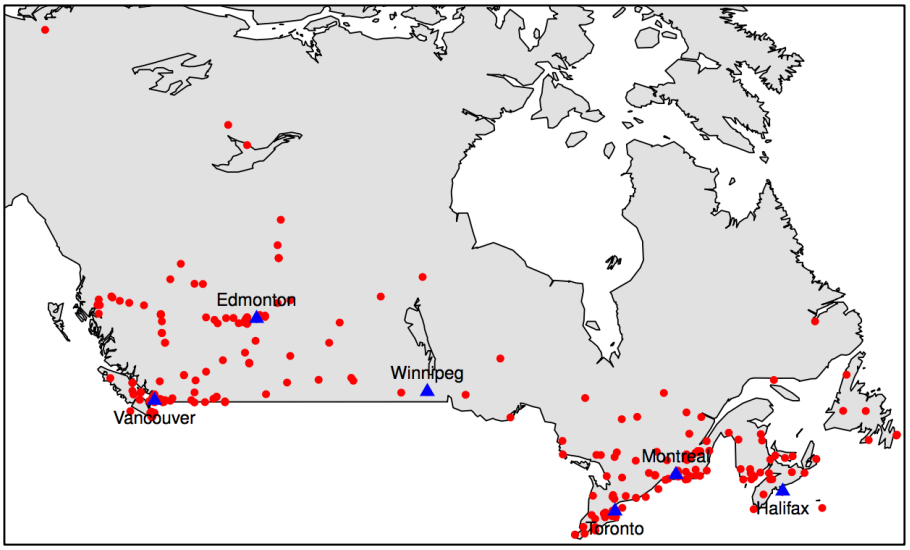

Figure 1: Displays the 250 stations used by UMOS-AQ as red dots and the 6 cities used in this thesis as blue triangles [3].

2.2 UBC Graduate Thesis

The EC model, although useful, fails to capture non-linear relationships between air pollutants and meteorological data. As a result, a UBC master's candidate built a non-linear model to outperform UMOS-AQ for his thesis [3]. The student's model, an online sequential extreme learning machine (OSELM), outperformed UMOS-AQ in all areas except for $PM_{2.5}$ concentration predictions [3]. This is another reason for focusing on the prediction of $PM_{2.5}$ in this report over other pollutants.

2.3 Additional Papers/Studies

Research is being done across the world on this topic due to the increased popularity and accessibility of machine learning (ML) techniques. A MIT undergraduate student successfully applied a Non-Linear Auto-Regressive Exogenous Neural Network (NARX-NN) to determine $PM_{2.5}$ concentrations in New York City. Many ML models have been implemented by researchers in Tehran, Iran to study air quality including long-short term memory (LSTM), NARX, and Support Vector Machines (SVMs). A majority of the research on ML models for air quality prediction has been published in the past few years, making it an exciting time to make a contribution.

Based on the background research conducted three ML models were selected. The SVR and NARX-NN models were selected because of being successfully implemented for similar problems and being of reasonable difficulty for an undergraduate to implement [4][6]. The LSTM was selected because, in a paper published in 2019, it was shown to be the best model implemented in Tehran, Iran for air quality prediction [5]. As a result, it will provide more of a challenge and more accurate results.

3 Data

The hourly meteorological and pollutant data used for the models spans from 01/01/2010 to 12/31/2015 resulting in 52,584 data points for each of the six cities. This is approximately same time period that was used for the models in the UBC paper. The creation of usable datasets was initially thought to be straight forward but turned out to be one of the more difficult and time-consuming parts of this thesis. The following section outlines the sources of the collected data and describes how the data was processed for use in the ML models.

3.1 Data Sources

Data of the hourly temperature, dew point, wind speed, and air pressure were collected from an Environment Canada database. The pollution data was obtained from another government database called the National Air Pollution Surveillance Network (NAPS), which is the same database used by the government to determine the air quality index. This data includes the hourly concentrations of ozone, nitrogen oxides, and particulate matter of 2.5 microns. These databases were the best/only choices for the data required.

Using information from the two different databases is somewhat problematic in creating accurate datasets. This is because the station used to record meteorological data for a particular city is not usually in the exact same location as the station used to record the pollution data. For example, a station measuring air pressure and wind speed closer to Lake Ontario in Toronto may have significantly different measurements than a station that is located in the inner city where the pollution data station is located. As a result of this, time was spent identifying the meteorological and pollution data stations that are closest to each other in each city. Fortunately, most cities have four or five NAPS stations to select from. The names and locations of the pollution and meteorological data station selected for each of the six cities are listed in Table 5 in Appendix C.

3.2 Dataset Creation

Initially, the code written for compiling the datasets was made using functions for each task and was intended to be robust, efficient, and reusable. Dictionaries were used to store the datasets for each city throughout the entire process so that the data for the cities could be kept neatly organized throughout.

The hourly ozone, nitrogen oxides, and particulate matter data from the NAPS database came in separate CSV files for each pollutant and for each year. Each of these files contained data from all of the 250 stations in Canada. The file paths for each pollutant and each year were iterated through, importing each year's data, and concatenating them. Then the data labelled with the NAPS IDs of the stations to be used were identified and separated for each of the six cities.

The hourly meteorological data was provided from the database in CSV files for each month but, unlike the pollution data, each file only contained the data for a particular station. So for each city's selected station, the file paths were iterated through, importing each months data and concatenating them. From there, the columns containing the data for temperature, dew point, wind speed, and air pressure were identified, taken, and concatenated with the pollution data.

This process should have been fairly easy and straightforward, however, many problems occurred when the time came to execute this code and fix errors in the resulting datasets. These problems mainly stemmed from the lack of consistency and poor quality of the NAPS datasets. There were three things about the NAPS data that made it difficult to use. First, a station that measures particulate matter does not necessarily measure ozone and nitrogen oxides too. So in some cases 3 different stations needed to be used to obtain all the pollution data wanted for a single city making the above process more complicated. In the case of Montreal, there were no stations that even measured the nitrogen oxide concentrations. Second, some stations were not in operation for the entire six year period that data was being

collected for. As a result, switching between stations part way through the period to get a full set of data for one pollutant was the only option for some cities. Third, many stations would have multiple months' worth of missing data, which is too long a period to accurately fix through linearly interpolation.

The first and second problems mentioned above made it very difficult to write robust, simplified code that could be reused for each city. This is because it became necessary to decide which stations were best to use for each of the 3 pollutants and for each of the 6 years in each of the 6 cities. As a result of the third problem mentioned, it became necessary go through all the possible stations for each city and identify the stations that were missing months' worth of data so that they would not be used.

It's possible to write code to select the best station to be used for each pollutant in each year in each city based on the above problems. However, at a certain point it was taking more time to write this code than it would to write rougher, less professional code and just go through and manually select the best stations to be used. This is what was done since the focus was supposed to be on the models and not the code used to compile the datasets. A couple functions of the initially written, more professional code that didn't end up getting used can be seen in Appendix E as an example.

Once all the data was imported and combined, linear interpolation was used to handle any smaller pieces of missing data for each city. In the meteorological data, missing data points were left blank while in the pollutant data the value -999 is used. The code written for this would cycle through all points and replace these values when identified. Outliers in each city's dataset were handled by inspecting the data, identifying the typical range of values, and filtering out data that is above or is below that range. The amount of missing data for each city's dataset is shown in the table below. Once this was complete, the datasets for each city were saved to be used in the models. An example of one of these final datasets can be viewed in Table 6 in Appendix C.

Table 1: Displays the number of missing data points that were corrected for each city's dataset.

Dataset	Number of Missing or Corrected Data Points	Percentage of Missing Data (Total Data per City: 368088)
Edmonton	9307	2.5%
Halifax	6424	1.7%
Montreal	2478	0.65%
Toronto	2651	0.72%
Vancouver	8727	2.4%
Winnipeg	9669	2.6%

3.3 Data Correlation

Pearson's correlation coefficient and the maximal information coefficient (MIC) were used to evaluate trends in the compiled data. Pearson's correlation coefficient is a measure of the linear relationship between the two variables and ranges from -1, meaning a perfect negative linear relationship, and 1, meaning a perfect positive linear relationship.

The MIC is a method developed in the last decade for detecting non-linear dependencies between variables. It yields a value between 0, meaning statistical independence, and 1, meaning a perfect relationship. It is useful because of its ability to express the strength of non-linear relationships such as exponential and sinusoidal relationships. Unfortunately, it only provides information about strength and does not express the type of the relationship or its direction.

The correlations for each dataset are displayed in plots presented later in the report. They were implemented using functions in a Python package called Maximal Information-based Nonparametric Exploration or minepy for short.

3.4 Preprocessing for Time Series Forecasting in Machine Learning

Time series data such as X(t)=0,1,2,3,4… can't be put directly into a machine learning model. It needs to be reformatted as (X(t)=0,1,2,3,4… , X(t-1)=1,2,3,4,5…) so that the model is receiving X(t-1), the current value, and learns to predict X(t), the future value. This is fairly simple with only one variable but becomes more difficult with multiple. The data was also normalized, which is necessary in machine learning when the features have different ranges. A simplified explanation for why normalization is needed is that without scaling all the features to the same range some features will have a larger impact than others during training. The resulting models trained on data without normalization and reformatting are useless.

The data for each city was then split into training, validation, and testing sets. The validation set (also called development set) is used to give an estimate of how the model will perform while the hyperparameters of the model are being tuned during training. It is created since using the testing set to check the model's performance while still tuning can lead to the model being tuned specifically for that testing dataset and performing better on it. This would lead to a biased, inflated representation of how the models would perform in the real world on brand new data.

The datasets for each city were split so that the data from 2010-2014 was used for training sets, 2014-2015 for validation, and 2015-2016 for testing. In the case of the SVR model, K-Fold Cross Validation was used during training. This method, rather than using a single validation set, takes the combined training and validation data and segments it into bins. It then takes turns training the model using one of those bins as the validation set and the other remaining bins as training data. This leads to the development of a stronger model.

4 Models

4.1 Support Vector Regression (SVR)

Theory

A Support Vector Machine (SVM) can be applied to both classification and regression problems. When applied to a regression, it is called Support Vector Regression (SVR). When training the model, hyperplanes are used to segment the data into different regions. The hyperplanes are placed and adjusted by minimizing the cost function. When new inputs are received, the output of the model corresponds to the region in which the input data is located. A part of this process is using kernels to transform data into higher dimensions [10]. By altering the constraints on the hyperplanes, the SVM can be used for regression problems and time series predictions.

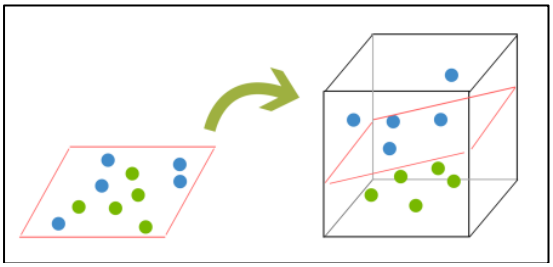

Figure 2: Displays the process of using a kernel and hyperplane to make regions within data [9].

Implementation and Design

The SVR was implemented using Scikit-Learn, which is a free ML library for Python. This library allows the choice of the following kernels: linear, polynomial, radial basis function (RBF), sigmoid, or a custom kernel. If the polynomial kernel is chosen, then its degree needs to be specified. The hyperparameter C, named the regularization parameter, acts to control the model's fit like a penalty term. A higher value of C results in the model being penalized more strongly for not splitting training points correctly. As a result, the hyperplane is placed so that more of the training points are correctly separated. Selecting a value of C that is too high can lead to overfitting and poor performance on the test set. In this case, the model would fit the training data extremely well but would not generalize to work well on other data.

Gamma, the kernel coefficient, is another parameter that can be selected. It defines how far the influence of a single training example reaches, with low values resulting in a high reach and high values resulting in a close reach. For example, a high gamma would mean that the hyperplane in the above figure would only be influenced by the training points closest to it while a low gamma would mean the points farther away from the plane would have influence on its position too. High values of gamma can also result in overfitting.

Initially, a function was being created to cycle through all the possible kernels, gamma parameters, and kernel coefficients to select the best performing combination. However, a function was found called GridSearchCV in the Scikit-Learn library that does this. As a result, this function was used to tune these parameters for the SVR models of each city.

4.2 Non-Linear Auto-Regressive Exogenous Neural Network (NARX-NN)

Theory

Neural networks (NN) are composed of nodes, symbolized by circles, and the connections that link them together. Each connection has a weight associated with it, which is applied to the input to that connection. At each node, every connection's weight is applied to its input and they are all added together, like a weighted average. The result is then put into an activation function, which is typically a sigmoid or tanh function, to compress output values into a certain range such as 0 to 1. The resulting value is the output for that particular node. Networks are composed of layers of these nodes. The first layer is associated with the model's inputs, the final layer with the outputs, and the layers in between are referred to as hidden layers. Types of networks differ in how they are connected. Equation (1) and (2) show the weighted average that takes place at each node and the commonly used sigmoid activation function. These equations are labelled to correspond with figure below of a feedforward neural network, which is a very simple type of NN.

$$Z_i = \sum_{j=1}^{M} W_{ij} * X_i \qquad (1)$$

$$A = \frac{1}{(1-e^{(-z)})} \qquad (2)$$

Where M is the number of connections to that node and W the weight of each connection.

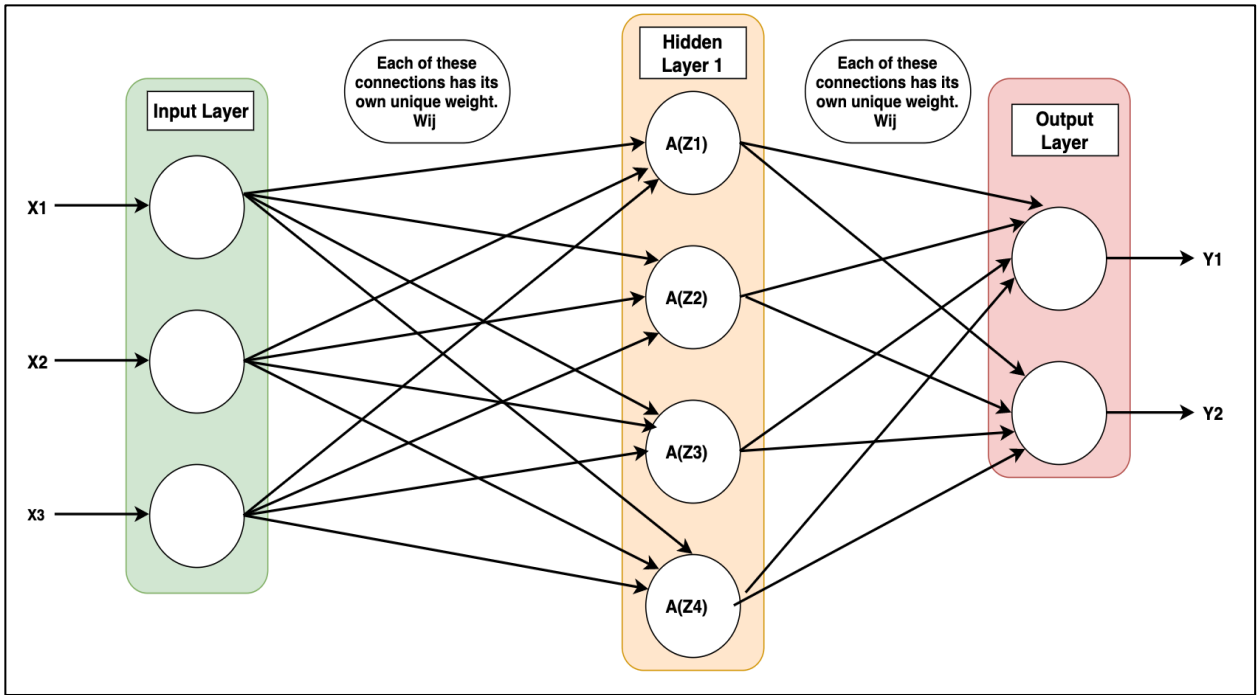

Figure 3: Displays a very simple feed-forward neural network with each of its layers, connections, and nodes labelled.

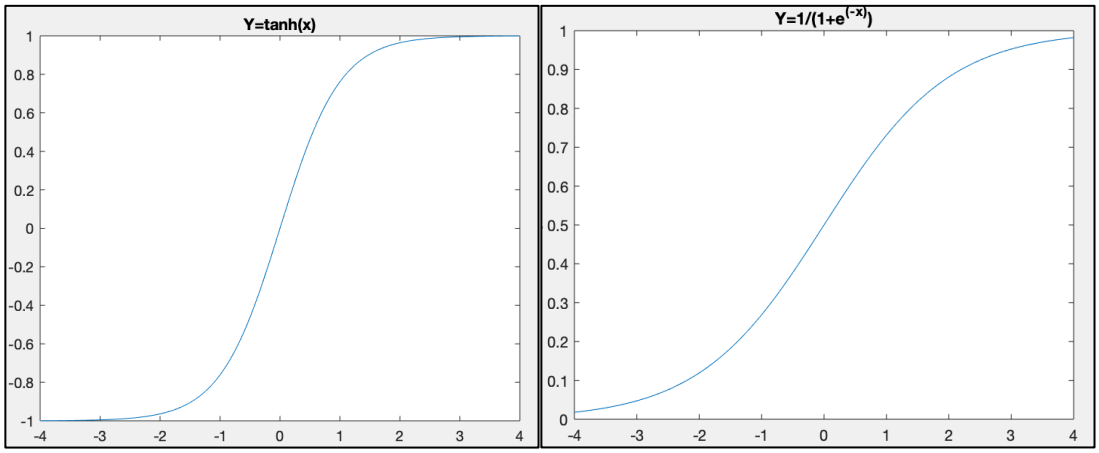

Figure 4: Displays the sigmoid and tanh functions that are typically used as activation functions.

The network learns by changing the weights of each connection until it produces the desired results for the given inputs. This is done through the minimization of a cost function. The cost function measures how far off the predicted outputs are from those desired. Minimization is usually accomplished through gradient descent, which changes each connection's weight based on the derivative of the cost function with respect to that weight, the gradient.

The backpropagation algorithm determines the change in weights during training. Its design relies on the chain rule. The mean squared error (MSE), shown in Equation (3), is usually used as the cost function.

$$C = MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - Y'_i)^2 \qquad (3)$$

Where n is the number of data points.

Recurrent Neural Networks (RNN) and Non-Linear Auto-Regressive Exogenous Neural Network (NARX-NN)

The basic neural network shown above is not capable of using the model's output from an earlier time-step as an input to the current time step. Thus, they are not very useful for time-series predictions. RNNs allow the passing of information from previous time steps back into the model.

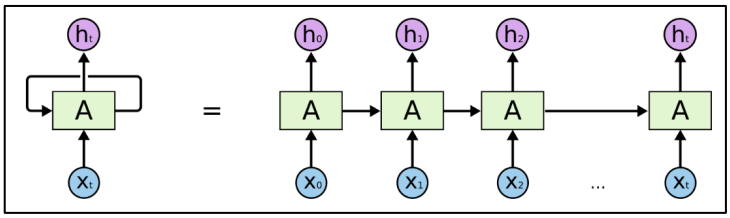

Figure 5: Displays the way an RNN passes information from previous time steps where x and h are the inputs and outputs at each time steps and A is an activation function [7].

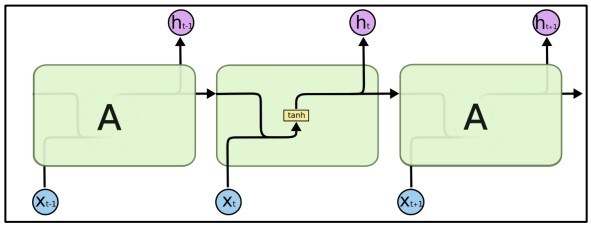

Figure 6: Displays the activation function that is contained in one module of an RNN [7].

RNNs work well when the information that is being passed back into the model is from very recent time steps. However, they fail in long-term dependencies when more information is used from further in the past [7]. The NARX-NN is a straightforward modification to RNNs with more elaborate feedback connections spanning several layers that can somewhat fix this problem [8]. LSTMs are a variation of an RNN that solves this issue even more effectively.

Implementation and Design

The NARX-NN was implemented using the ML Python library named PyNeurGen. The decisions that need to be made in the design of this model are: the best activation function to use, the amount of information from previous time steps being fed into the current time step, and the number of nodes in the input, hidden, and output layers.

The ReLU activation function, which is the preferred activation function in a majority of problems, was not available in this library so the tanh activation function was used. It is preferred over the sigmoid since optimization is easier [11]. Increasing the number of variables that are being input into the model from previous time steps will increase the likelihood of the vanishing gradient problem. In short, this problem arises in the backpropagation algorithm and decreases the accuracy of the model and increases the time it takes to train. As a result, data should not be used that's farther than 1 to 5 time steps in the past.

The number of nodes in the input layer is simply the number of inputs to the model. This is the six meteorological and pollutant variables from the datasets as well as the number of variables passed from earlier time steps. The output layer only contains one node because only the $PM_{2.5}$ concentration is being predicted by the model. One hidden layer is sufficient for most ML problems and the number of nodes in it should fall between the number of nodes in the input and output layers.

The parameters to be tuned in this model are the learning rate and the number of epochs. As mentioned earlier, training the model relies on gradient descent to minimize a cost function. The learning rate impacts the size of the change to the NN weights at each step of this minimization. If this rate/step size is too big then minimization may never be achieved but if it is too small then it may take an extremely long time to train. A for loop was used to cycle through possible learning rates to decide which rate to use. The epoch is the number of times the model runs through the entire training dataset while learning. This is analogous to a student learning math by repeatedly doing a homework question. At a certain point, the student is no longer

learning and is only memorizing so will be less successful in solving brand-new problems. This parameter was tuned by plotting the RMSE at each epoch to find where it was minimized.

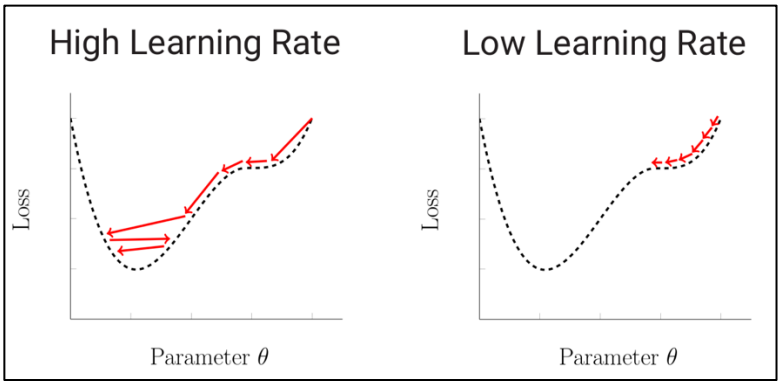

Figure 7: Displays the impact of different learning rates on minimizing the cost (loss) function where theta is the weight [12].

4.3 Long Short-Term Memory (LSTM)

Theory

Both RNNs and LSTMs can be thought of as a chain of modules, but they differ in the content of these modules. The RNN module is one layer while the LSTM module has 4 interacting layers [7].

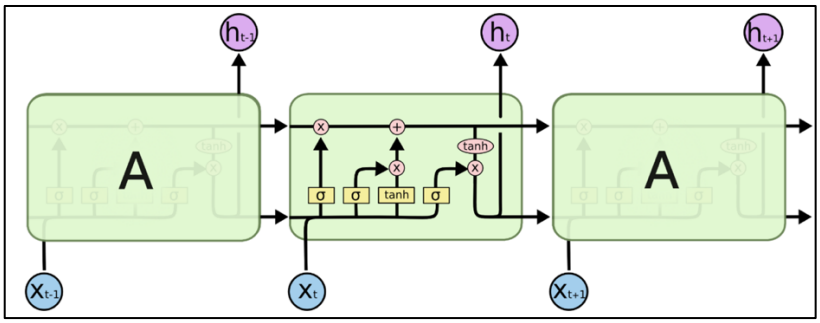

Figure 8: Displays the cell state as well as the forget, input, and output gates of an LSTM [7].

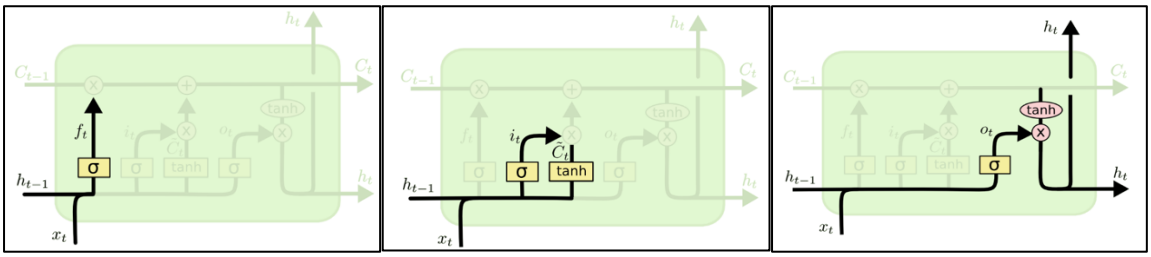

Figure 9: Displays, from left to right, the forget gate, input gate, and output gate of the LSTM [7].

The top pathway of the module is called the cell state, which is the path where the retained information flows through the module. Three gates using four NN layers are used to modify and manipulate the information in the pathway/cell state. These are the forget gate, which determines whether past information should still be kept, the input gate, which controls whether new information is retained, and the output gate, which will produce the output for that time step. Through the connection of many nodes and modules the network is able to selectively remember and use past information to make predictions.

Implementation and Design

The LSTM was implemented using Keras, which is another free library for neural networks written in Python. The earlier models used basic gradient descent as the optimization algorithm. For this model, the Adam optimizer will be used since this ML library provides a variety of optimizers to choose from. The Adam optimizer was selected because it allows for faster gradient descent and is more accurate. This library also allows for the use of dropout, which is a computationally cheap method of reducing overfitting in a NN. It involves dropping or temporarily removing nodes from the neural network. It is recommended that a dropout rate of 50% be used for the best results [13]. ReLU activation functions were an option in this library so were used instead of sigmoid and tanh because they help to avoid the vanishing gradient problem [11]. The number of epochs used by the LSTM must be optimized. Similar to the NARX-NN, the RMSE of the LSTM was plotted over multiple epochs to identify and display where it is minimized and select the corresponding number of epochs.

5 Results and Discussion

Please note that all the figures that are shown in the following section were completed for all 6 cities. For the sake of report readability, only the figures for Toronto will be shown in this section. The figures for the other five cities can be found in Appendix D.

Datasets

This subsection establishes an understanding of the data. Based on the following plot, all the meteorological and pollution variables vary somewhat cyclically.

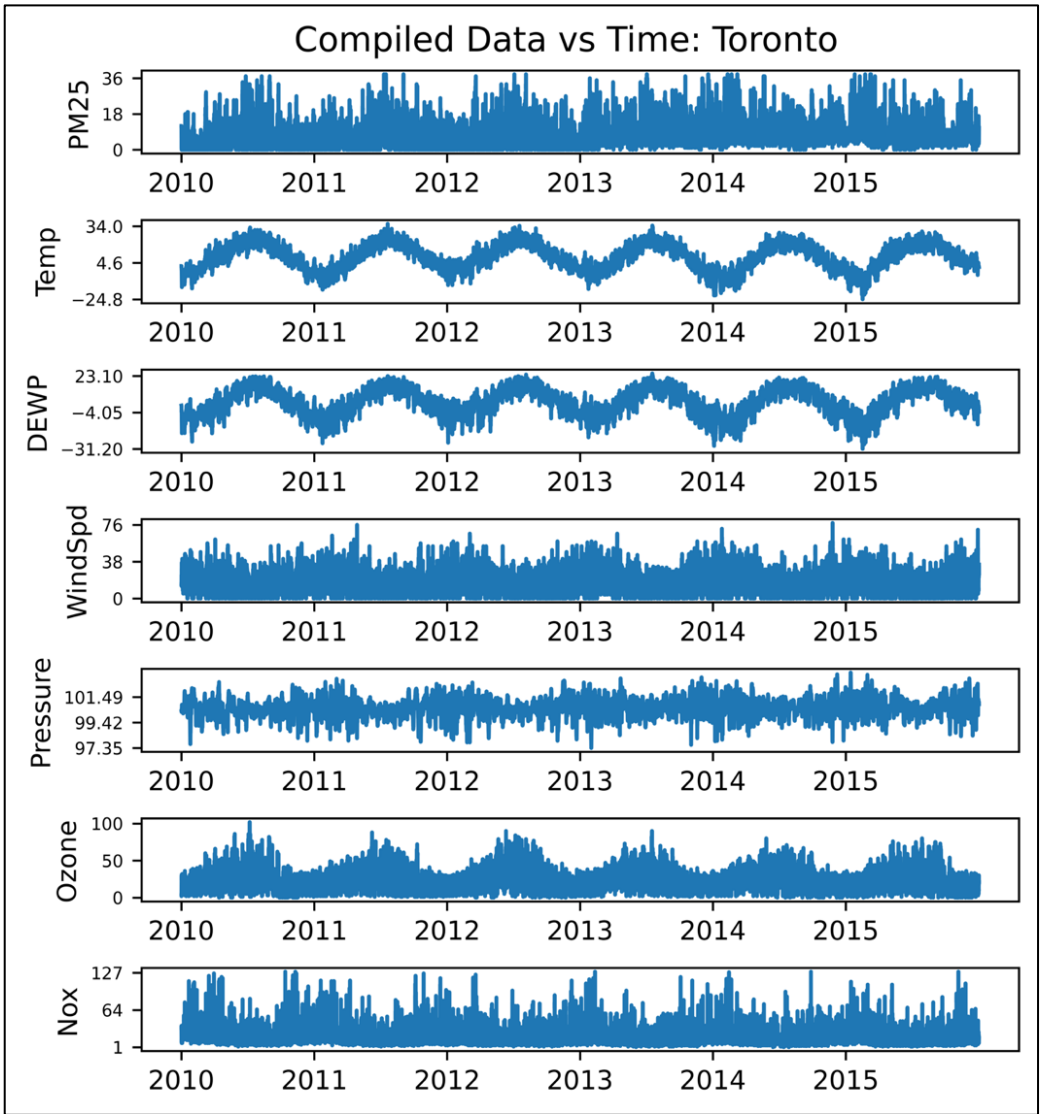

Figure 10: Plots of the seven variables with time for Toronto dataset.

The temperature, dew point, and ozone concentration all peak mid-summer and are at their minimum in January with the opposite being true for wind speed and NO_x concentration. Across all of the city datasets, the ozone and NO_x vary similarly with time. $PM_{2.5}$ seems to be less consistent across all cities but does seem to vary cyclically with time in each case. The following plots are useful for looking for any correlations between $PM_{2.5}$ and the other variables.

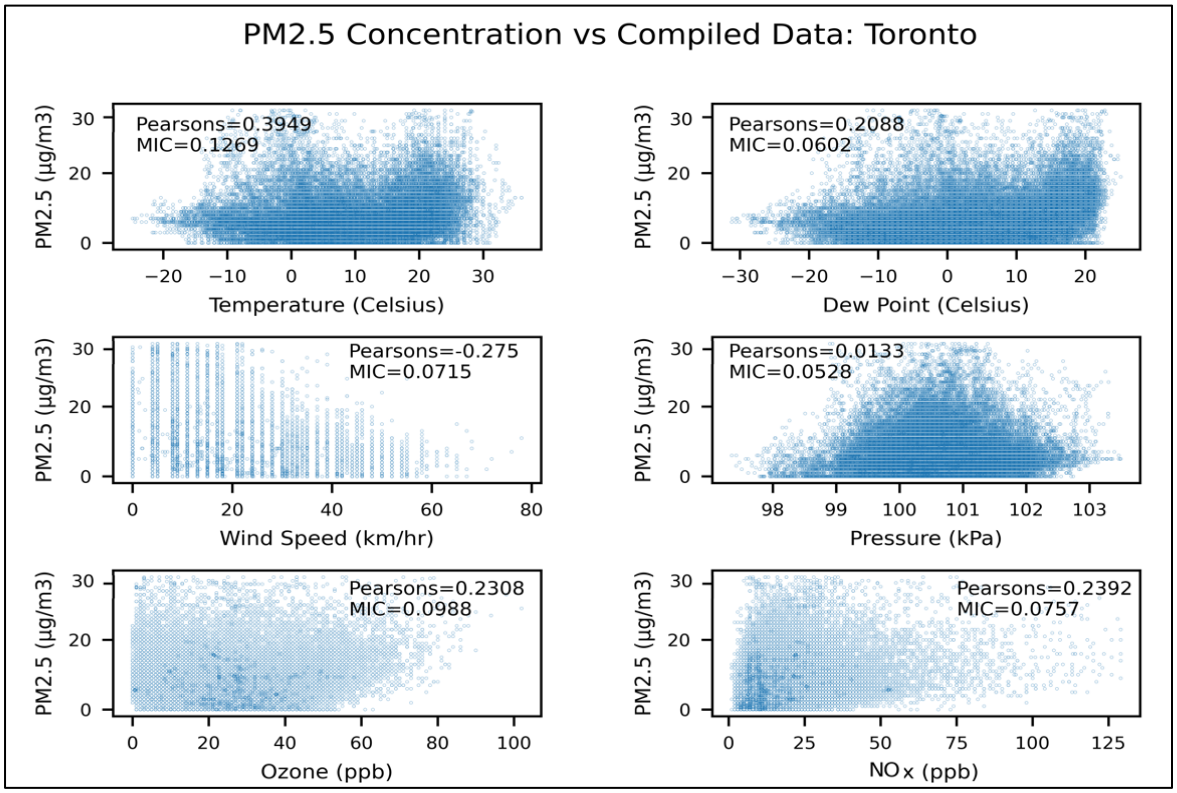

Figure 11: Displays the correlations between $PM_{2.5}$ and each of the other variables for Toronto.

Across all cities, the MIC and Pearson correlation coefficient indicate a weak to moderate correlation with the other variables. Although correlation can help in understanding the data it does not imply causation and does not mean ML techniques will be ineffective as a result of the weak correlations. These correlations don't consider that this is time-series data. They don't provide information on the more complicated relationships between a $PM_{2.5}$ value and the data from previous time-steps. The following histogram provides an understanding of the distribution of the $PM_{2.5}$ concentrations.

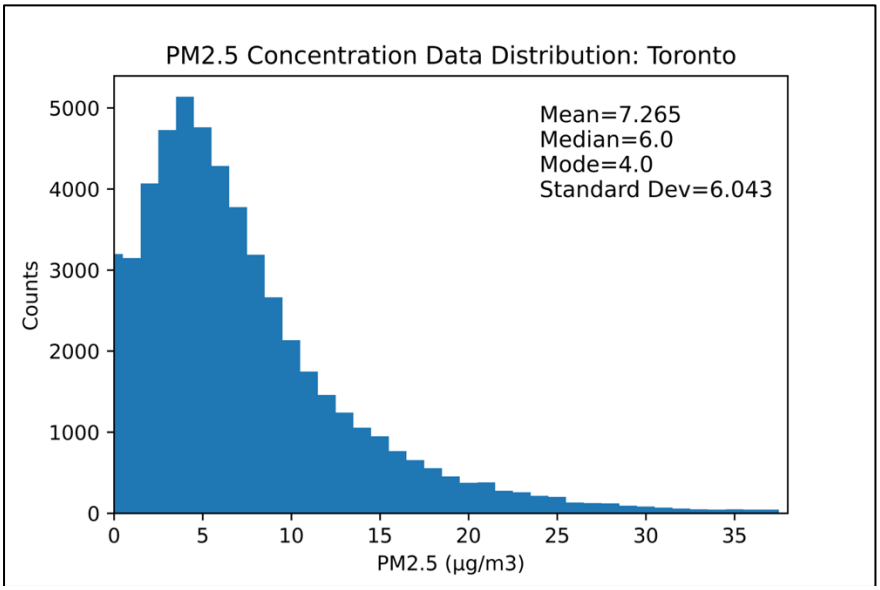

Figure 12: A histogram of the $PM_{2.5}$ data and the mean, median, mode, and standard deviation for Toronto.

The histograms for all of the cities are right-skewed like this one. Edmonton, Montreal, and Toronto have the highest mean concentrations of $PM_{2.5}$, respectively. The mean for Montreal and Edmonton is almost double that of Vancouver which is the city with the lowest mean concentration. The data for Edmonton is, on average, spread furthest from the mean while the Vancouver data is most closely spread around its mean.

Table 2: The mean, median, mode, and standard deviation for each city's $PM_{2.5}$ concentration data.

City	Mean ($\mu g/m^3$)	Median ($\mu g/m^3$)	Mode ($\mu g/m^3$)	Standard Deviation ($\mu g/m^3$)
Edmonton	9.714	7	5	8.309
Halifax	5.669	5	4	4.228
Montreal	9.678	7	4	6.791
Toronto	7.265	6	4	6.043
Vancouver	5.211	4	3	4.015
Winnipeg	6.143	5	3	4.841

A $PM_{2.5}$ dataset for Beijing from Peking University was also plotted to provide comparison with a popular dataset from a polluted city. These plots are shown in Figures 59-61 in Appendix D. The data looks very similar in terms of shape. It is the difference in magnitude that makes this comparison valuable. The mean concentration of $PM_{2.5}$ in Beijing is 98 $\mu g/m^3$, which is 10 to 20 times higher than major Canadian cities, highlighting the stark difference in air pollution between the countries.

Models

The hyperparameters of each city's model were tuned using the methods discussed above. The following table shows the tuned hyperparameter that minimize the RMSE.

Table 3: The tuned parameters that achieved the best RMSE for each model and each city.

City	SVR			NARX-NN		LSTM
	Kernel	Kernel Coefficient	Gamma	Learning Rate	Epochs	Epochs
EDM	RBF	1.5	0.007	0.1	3	22
HAL	RBF	20	0.05	0.1	1	7
MON	RBF	5	0.005	0.07	4	6
TOR	RBF	0.01	10	0.01	5	10
VAN	RBF	2	0.005	0.2	1	12
WIN	RBF	3	0.01	0.01	2	18

After tuning, Toronto's SVR model had a RMSE of 2.201 $\mu g/m^3$. A comparison between the model's predictions and test data can be seen in the following figure.

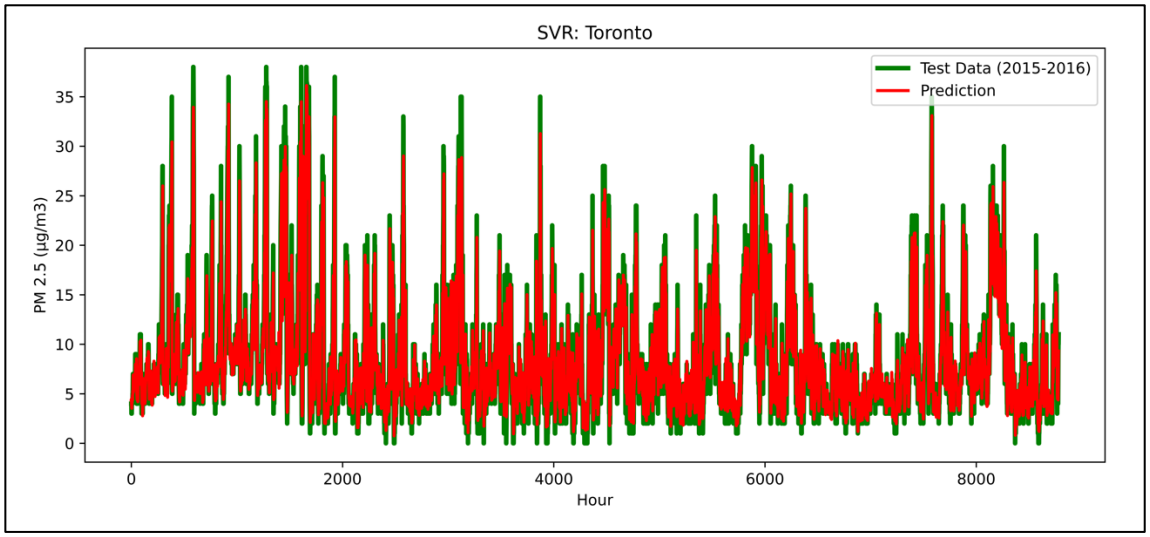

Figure 13: The predicted values of the SVR Model compared to the actual test set data for Toronto.

The lowest RMSE achieved after tuning the NARX-NN for Toronto was 6.37 $\mu g/m^3$. The following plot was used to select the optimal number of epochs.

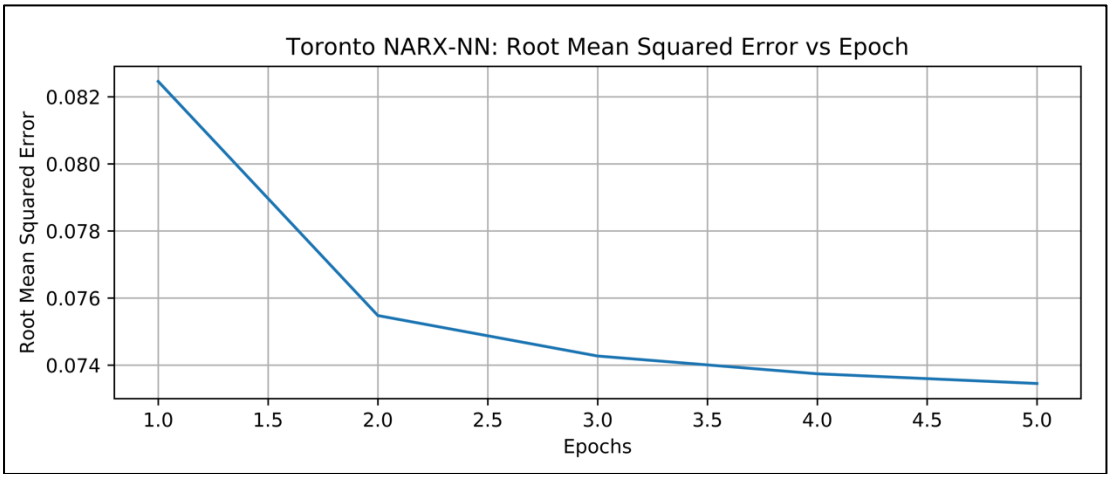

Figure 14: The RMSE over multiple epochs for the Toronto NARX-NN.

The RMSE in the above plot is using the normalized predictions before they have been rescaled and is on the validation set. As can be seen from the above figure, the decrease in RMSE per epoch of training becomes significantly less from 4 to 5. The slope continues to flatten past 5 as well, which is why 5 epochs was the best choice for this parameter. The following figure displays how the NARX-NN predictions compare with the true test data.

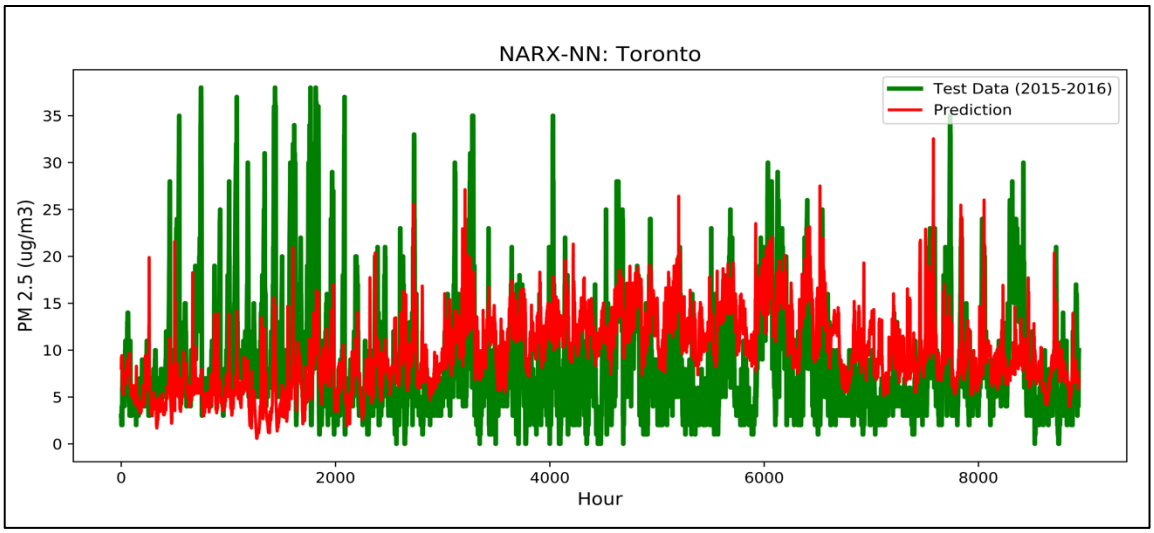

Figure 15: Displays the predicted values of the NARX-NN Model compared to the actual test set data for Toronto.

Again, please note that all of this has been completed across all 6 cities, figures for which are available in Appendix D. The lowest RMSE obtained by the LSTM for Toronto was 1.89 $\mu g/m^3$. The following plot displays the change in the RMSE for the training and validation sets over 50 epochs.

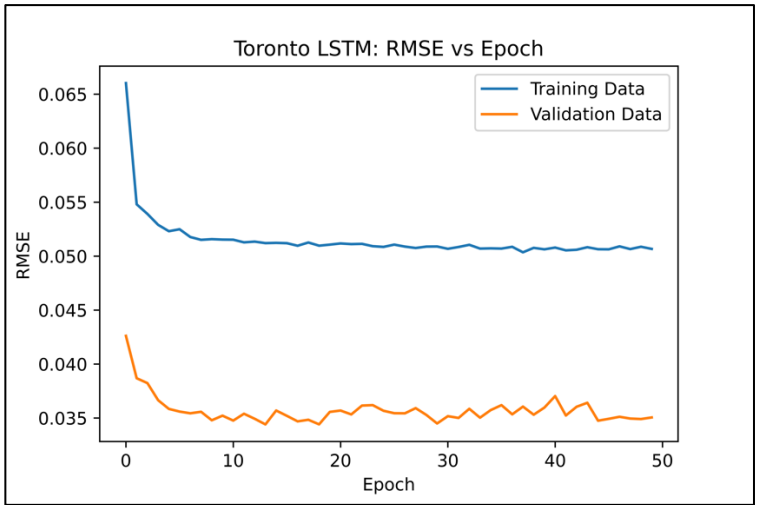

Figure 16: Displays how the RMSE for the validation and training data changes over epochs for the Toronto LSTM.

Based on the above figure, it is best to use around 10 epochs for training the Toronto LSTM since the reduction in the RMSE becomes minimal for the training and validation data after that point. The following figure displays how the LSTM predictions compare with the test data.

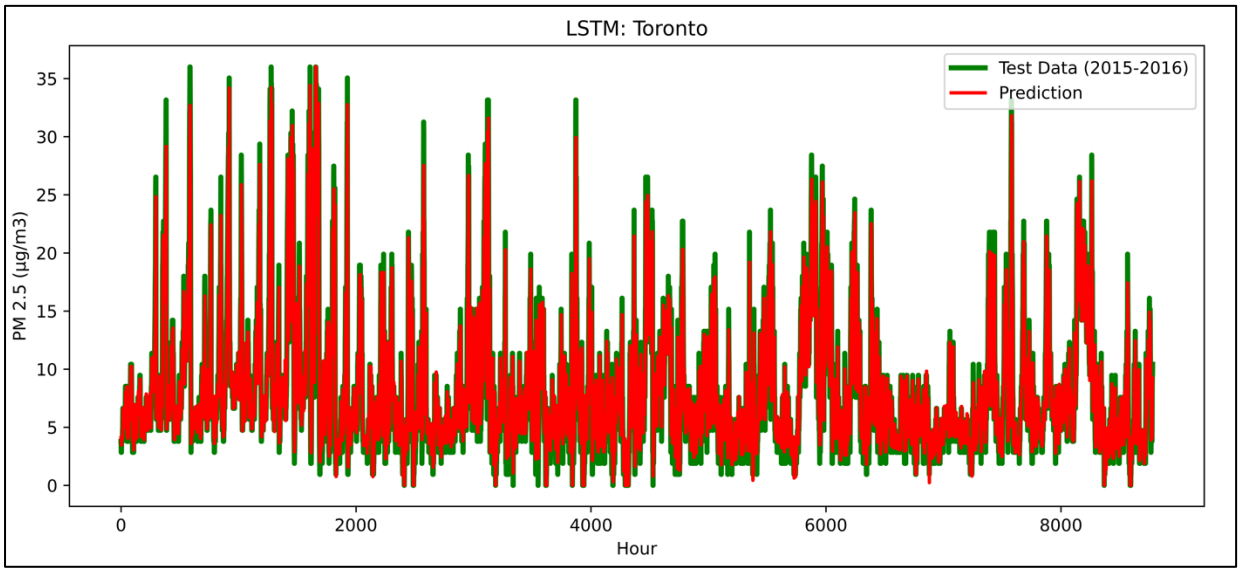

Figure 17: Displays the predicted values of the LSMT Model compared to the actual test set data for Toronto.

Table 4: The best RMSE achieved for each city and model.

City	SVR (μg/m³)	NARX-NN (μg/m³)	LSTM (μg/m³)
Edmonton	2.96	6.36	2.11
Halifax	2.29	3.84	3.29
Montreal	2.32	7.10	1.83
Toronto	2.20	6.37	1.89
Vancouver	1.63	4.27	2.27
Winnipeg	1.69	4.35	1.67

The NARX-NN consistently performed the worst across all cities in the prediction of $PM_{2.5}$. The NARX-NN appears to be following similar patterns as the true datasets in each city, however, the magnitudes of the $PM_{2.5}$ concentrations are off. This is a big problem because forecasting $PM_{2.5}$ is only valuable if the more extreme concentrations can be predicted. It is at the extremes where $PM_{2.5}$ concentrations will be most harmful to the health of a city's population. The LSTM performed better in all cities except for Halifax and Vancouver. This is what was expected from the background research since LSTMs are known to be the more complex and accurate models. The following figure compares the models in this report with those discussed in the UBC Graduate Thesis for the prediction of $PM_{2.5}$ concentration.

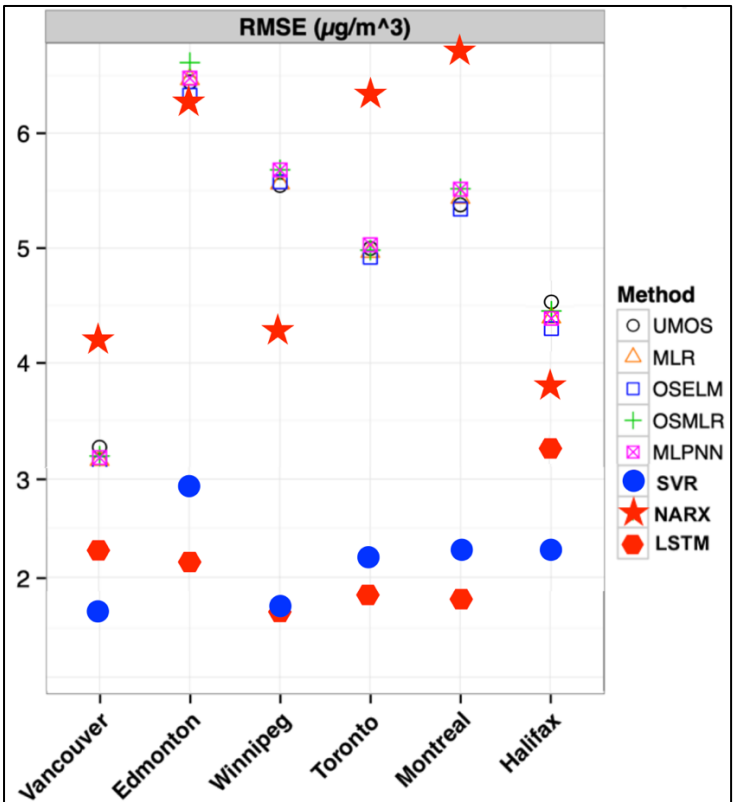

Figure 18: The RMSE for each city and each model from this thesis and the UBC Graduate Thesis.

This figure is a modified version of one found in the UBC paper. UMOS, the government model, and OSELM, the UBC student's best model, are particularly of interest. Across all, cities the SVR and LSTM were found to perform better than them. It must be noted that this comparison is not entirely fair and is not a perfect "apples to apples" comparison of models. This is because the exact same datasets were not used. The best attempt possible was made to create a similar dataset since the exact same dataset was unobtainable. Similar variables and sources such as the NAPS database were used over approximately the same time frame. As a result of this, the most that can be fairly concluded about the comparison between the models in this paper and the UBC paper is that the LSTM and SVR are potentially better models and further comparison is justified. They should now be compared on identical datasets.

On average, the NARX took the longest amount of time to train while the LSTM took the least. It was expected that the NARX would be slowest since the package that was used to implement it is not as polished as those used for the other two models. Additionally, the use of the Adam optimizer in the LSTM makes gradient descent faster, which contributed to making the LSTM training time fastest. A much more legitimate way of comparing the time complexity of the models is through Big O Notation, which describes how the time or memory requirements of an algorithm grow. The SVR is $O(n^2)$ when C, the kernel coefficient, is small and $O(n^3)$ when C is large [14]. The n is the number of training points. It is primarily due to the kernel in the model that makes it so computationally expensive [14]. The time complexity of the LSTM is unaffected by the number of training points due to its structure. The LSTM complexity instead depends on the number of weights (w) in the neural network and is equal to O(w) per time step [15]. This is an important distinction because if the SVR was applied to larger datasets, which would be the case in full AQ forecasting, then it would become significantly more computationally expensive. The weights and thus the computational expense of the LSTM would likely increase if it were applied to full AQ forecasting as well. However, the LSTM time complexity would not increase nearly as much as in the SVR since the change in the number of training points would be much greater than the change in weights. Thus, out of the two best performing models, the LSTM is a better selection for use in larger scale AQ forecasting.

In time-series forecasting it is desirable to have the model continually being updated as new data is being generated in the world. For example, if a model is trained on 2018-2019 data then it is useful to be able to update the already trained model with the new information that is created with the 2019-2020 year. The SVR, LSTM, and NARX, are trained through batch learning which means every time they are updated, they must be retrained on the entire dataset including both old and newly obtained data. This increasingly becomes a disadvantage as datasets increase in size. The alternative is online sequential learning models which are only updated using the new data. This is important for problems where quickly adapting to new patterns is necessary. Some of the models in the UBC paper were online sequential learning. This method is much better for air quality forecasting that is applied over many years of data and across many cities. The SVR and LSTM can be adapted to be online sequential models but that is beyond the scope of this thesis and a good option for future work.

All the tools and data necessary to make and develop these models were free. Running ML models on local GPU can be difficult but Google Colab allows free access to NVIDIA Tesla K80 GPU providing 1.8TFlops and has a 12GB RAM. This was more than sufficient for this thesis. If these models are further developed to become more complex and the datasets are expanded, then it may be worth paying money to improve performance. This would be the case if a more elaborate model was developed to predict all of the pollutants discussed and not only $PM_{2.5}$. Google Colab's TPU, which delivers 180TFlops and provides a 64GB RAM would be a good option. The cheapest version offered for access to the cloud TPU is $1.35 USD/ hour, which is very reasonable compared to the alternative of actually buying hardware.

Future Work

It would be valuable to build online sequential versions of these models and implement them on the same datasets since the comparison with the online sequential models from the UBC papers are not entirely fair. Additionally, trying to determine which features are more valuable in the prediction of $PM_{2.5}$ would be useful. There are many other pollutant concentrations, meteorological measurements, and other variables that could be used to make an even more elaborate dataset too. This project could be expanded and applied to the prediction of the other main pollutants so that an air quality index could be designed and compared to the one offered by the Canadian government.

6 Appendices

6.1 Appendix A: Word Count and Previous Work

Word Count = 5700

Significantly more time was spent doing this thesis than the recommended 5 hours per week. This is due to the goals of the thesis, the poor quality of the NAPS dataset, and since programming in Python and working with ML was relatively new to me. No work was completed prior to September 2019. The introduction, model theory, and parts of the dataset creation sections, were completed in the fall term of 2019 while everything else was completed during the winter term in 2020.

6.2 Appendix B: References

[1] Li, Xiang, et al. “Deep Learning Architecture for Air Quality Predictions.” *Environmental Science and Pollution Research*, Vol. 23, No. 22, Published: 19 Oct. 2016.

[2] Ritchie, Hannah, and Max Roser. “Air Pollution.” *Our World in Data*, 17 Apr. 2017, https://ourworldindata.org/air-pollution.

[3] Peng, Huiping. “Air Quality Prediction by Machine Learning Methods”, Graduate Thesis, Dept. of Atmospheric Science, *University of British Columbia,* 2015.

[4] Delavar, Mahmoud, et al. “A Novel Method for Improving Air Pollution Prediction Based on Machine Learning Approaches: A Case Study Applied to the Capital City of Tehran.” *International Journal of Geo-Information*, Vol. 8, No. 2, Pg. 99, 23 Feb. 2019.

[5] Karimian, Hamed, et al. “Evaluation of Different Machine Learning Approaches to Forecasting PM2.5 Mass Concentrations.” *Aerosol and Air Quality Research*, Vol. 19, No. 6, June 2019

.

[6] Keeler, Rachel. “A machine learning model of Manhattan air pollution at high spatial resolution”, Undergraduate Thesis, Department of Atmospheric Science, *Massachusetts Institute of Technology*, June 2014. Available: https://dspace.mit.edu/handle/1721.1/90659

[7] Olah, Christopher. “Understanding LSTM Networks.” *Colah’s Blog,* 27 Aug. 2015. Available: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[8] Boussaada, Zina, et al. “A Nonlinear Autoregressive Exogenous (NARX) Neural Network Model for the Prediction of the Daily Direct Solar Radiation.” *Energies-MPDI*, Vol. 11, No. 3, 10 Mar. 2018, pg. 620., doi:10.3390/en11030620.

[9] Bambrick, Noel, “Support Vector Machines: A Simple Explanation.” *KDnuggets*, July 2016. Available: https://www.kdnuggets.com/2016/07/support-vector-machines-simple-explanation.html

[10] Sanz, Hector, et al. “SVM-RFE: Selection and Visualization of the Most Relevant Features through Non-Linear Kernels.” *BMC Bioinformatics*, vol. 19, no. 1, Nov. 2018, doi:10.1186/s12859-018-2451-4.

[11] Jain, Pawan, “Complete Guide of Activation Functions”, Towards Data Science, 12 July 2019.Available: https://towardsdatascience.com/complete-guide-of-activation-functions-34076e95d044

[12] Winovich, Nick, “Deep Learning”, Purdue University Mathematics Department, 2018. Available: https://www.math.purdue.edu/~nwinovic/deep_learning_optimization.html

[13] Srivastava, Nitish, et al. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”, *Journal of Machine Learning Research, vol. 15, pg. 1929-1958, 2014. [Online]. Available:* https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf

[14] Bottou, Léon, “Support Vector Machine Solver”, *Large Scale Kernel Machines, MIT Press pg. 1-18,* 2007. Available: https://leon.bottou.org/publications/pdf/lin-2006.pdf

[15] Tsironi, Eleni, et al. “An analysis of Convolutional Long Short-Term Memory Recurrent Neural Networks for gesture recognition”, *Neurocomputing,* vol. 268, no. 13, pg. 76-86, 13 December 2017.

6.3 Appendix C: Additional Tables and Figures

Table 5: Displays the names and locations for each meteorological and the PM2.5 pollution data station used for each city.

City	Weather Station (Name and Climate ID)	Weather Station Coordinates (Latitude, Longitude)	PM2.5 Pollution Station (NAPS Station ID)	PM2.5 Pollution Station Coordinates (Latitude, Longitude)
Edmonton	Edmonton International Airport (3012216)	(53.31, -113.58)	90130	(53.54445, -113.49884)
Halifax	Halifax International Airport (8202251)	(44.88, -63.5)	30120	(44.719799, -63.480754)
Montreal	Montreal International Airport (7025251)	(45.47, -73.74)	50103	(45.641026, -73.499682)
Toronto	Toronto City Center (6158359)	(43.63, -79.4)	31103	(43.662972, -79.388111)
Vancouver	Vancouver International Airport (1108447)	(49.2, -123.18)	100118	(49.26167, -123.163472)
Winnipeg	Winnipeg International Airport (5023227)	(49.91, -97.24)	70119	(49.89799, -97.14653)

Table 6: Displays processed data to be used in the models for Montreal for half a day.

Index	Date	Year	Month	Day	Hour	PM25	Temp	DEWP	WindSpd	Pressure	Ozone
1	2010-01-01 0:00	2010	1	1	1	27	-3.8	-5	11	101.11	13
2	2010-01-01 1:00	2010	1	1	2	26	-4.7	-5.4	15	101.05	15
3	2010-01-01 2:00	2010	1	1	3	24	-4.6	-5.3	15	101.05	17
4	2010-01-01 3:00	2010	1	1	4	22	-4.6	-5.4	15	101.02	15
5	2010-01-01 4:00	2010	1	1	5	19	-4.9	-5.7	15	100.99	20
6	2010-01-01 5:00	2010	1	1	6	19	-4.9	-5.7	15	100.95	21
7	2010-01-01 6:00	2010	1	1	7	18	-5.1	-6	15	100.95	23
8	2010-01-01 7:00	2010	1	1	8	16	-5.2	-6	15	100.93	24
9	2010-01-01 8:00	2010	1	1	9	16	-5.1	-6.1	13	100.93	25
10	2010-01-01 9:00	2010	1	1	10	14	-4.9	-5.9	13	100.92	25
11	2010-01-01 10:00	2010	1	1	11	13	-4.3	-5.5	17	100.9	25
12	2010-01-01 11:00	2010	1	1	12	11	-4	-5.1	13	100.82	27

6.4 Appendix D: Figures and Tables for Remaining Cities

Edmonton

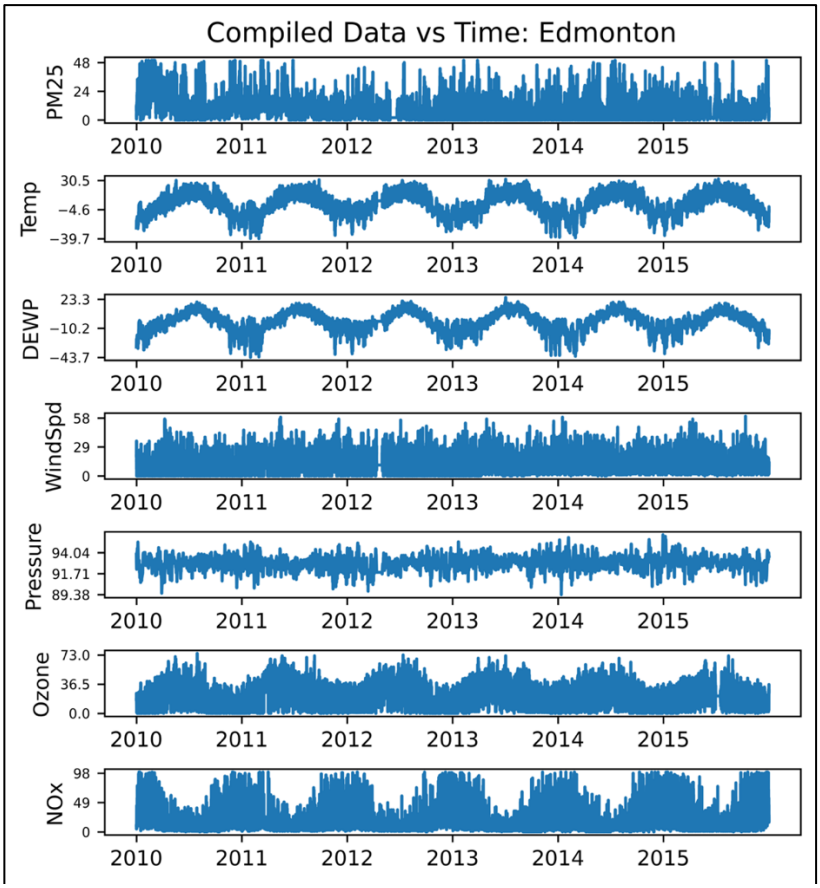

Figure 19: Plots of each of the seven variables with time for Edmonton dataset.

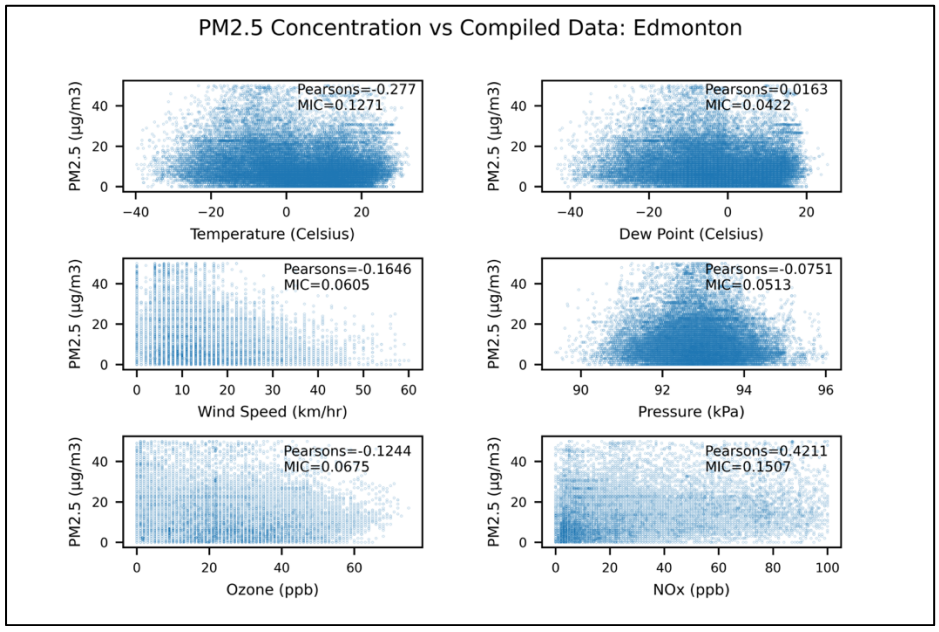

Figure 20: The correlations between $PM_{2.5}$ and each of the other variables for Edmonton.

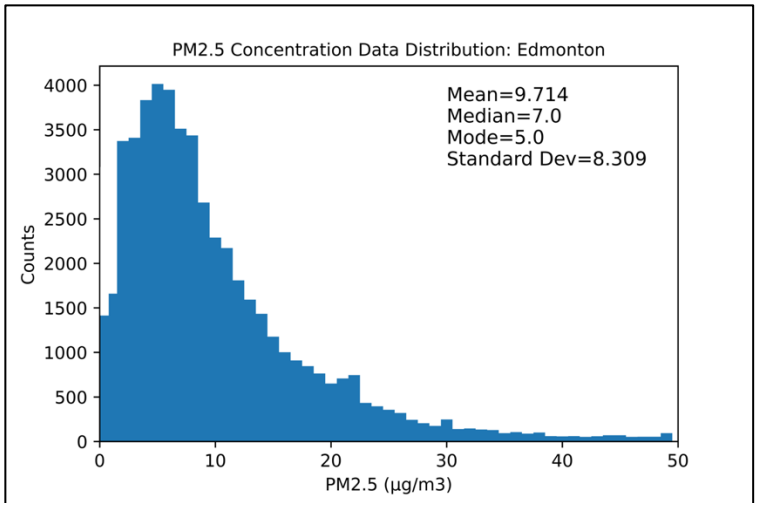

Figure 21: A histogram of the $PM_{2.5}$ data and the mean, median, mode, and standard deviation for Edmonton.

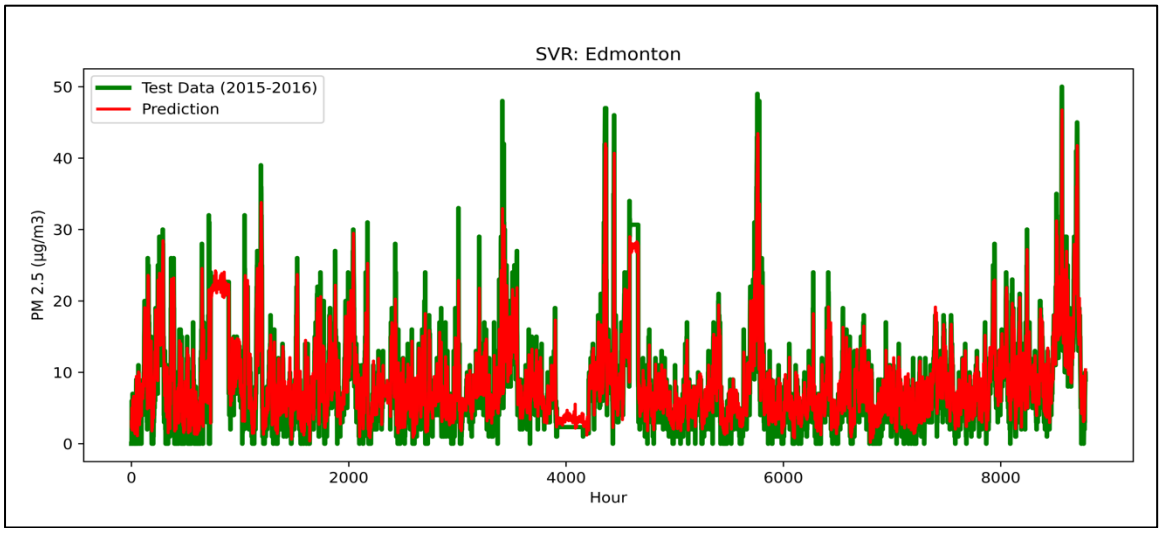

Figure 22: The predicted values of the SVR Model compared to the actual test set data for Edmonton.

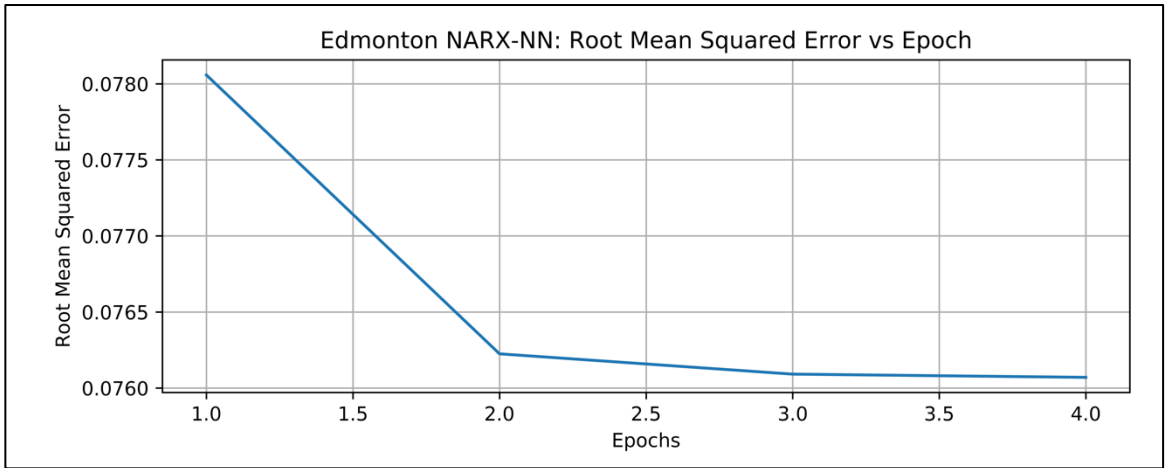

Figure 23: The RMSE over multiple epochs for the Edmonton NARX-NN.

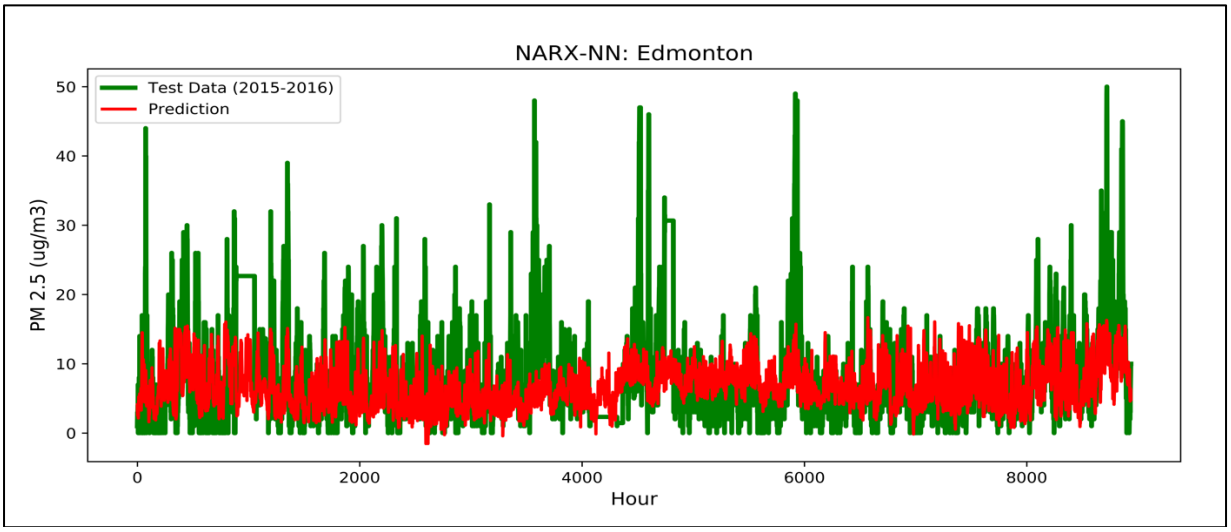

Figure 24: Displays the predicted values of the NARX-NN Model compared to the actual test set data for Edmonton.

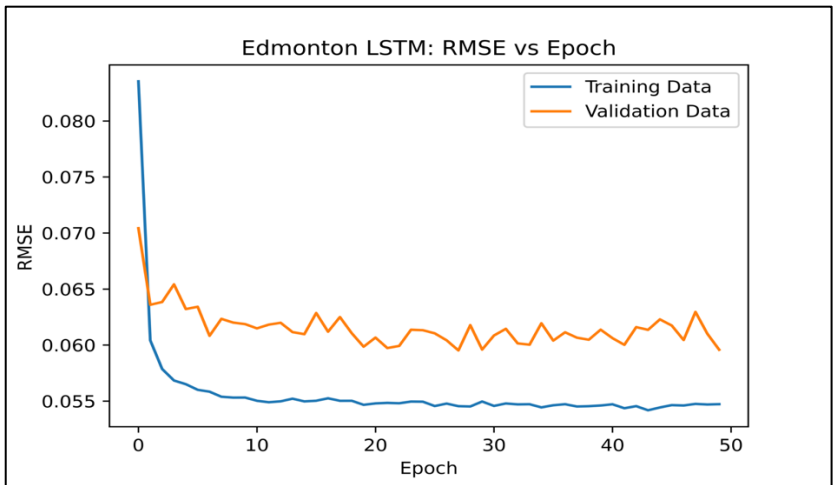

Figure 25: Displays how the RMSE for the validation and training data changes over epochs for the Edmonton LSTM.

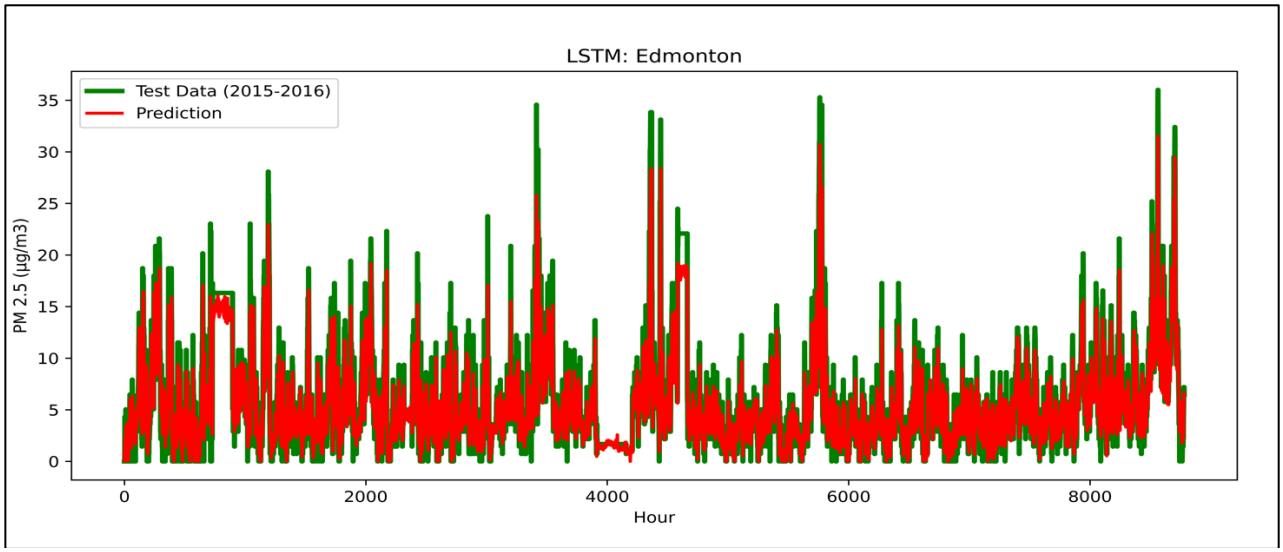

Figure 26: Displays the predicted values of the LSMT Model compared to the actual test set data for Edmonton.

Halifax

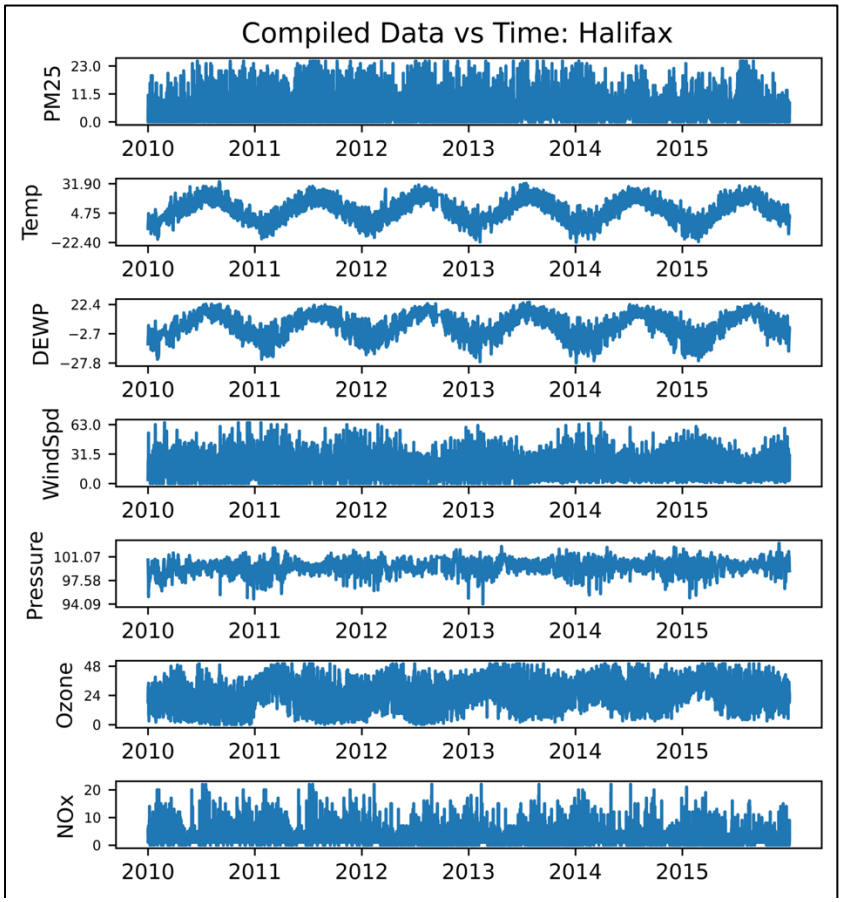

Figure 27: Plots the seven variables with time for Halifax dataset.

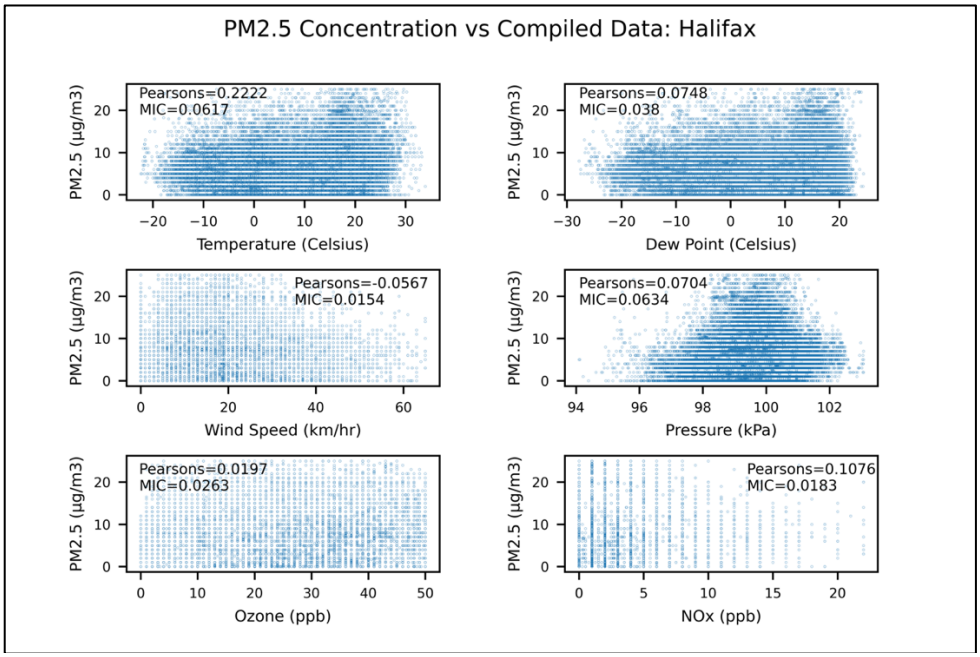

Figure 28: The correlations between $PM_{2.5}$ and each of the other variables for Halifax.

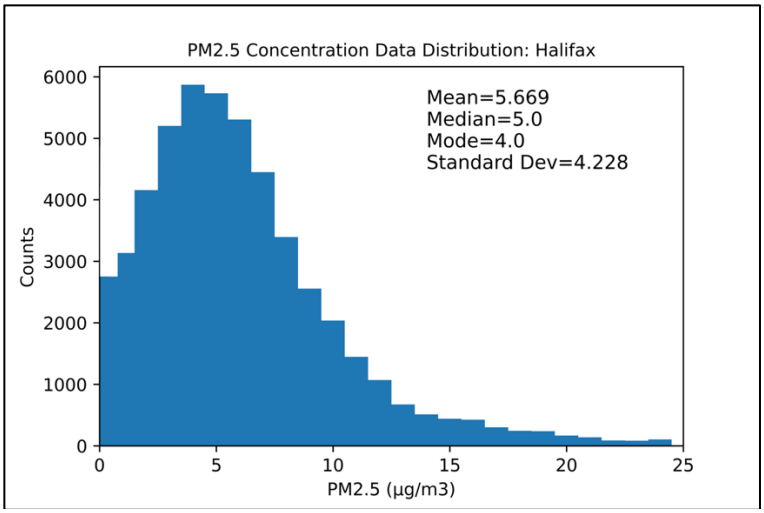

Figure 29: A histogram of the $PM_{2.5}$ data and the mean, median, mode, and standard deviation for Halifax.

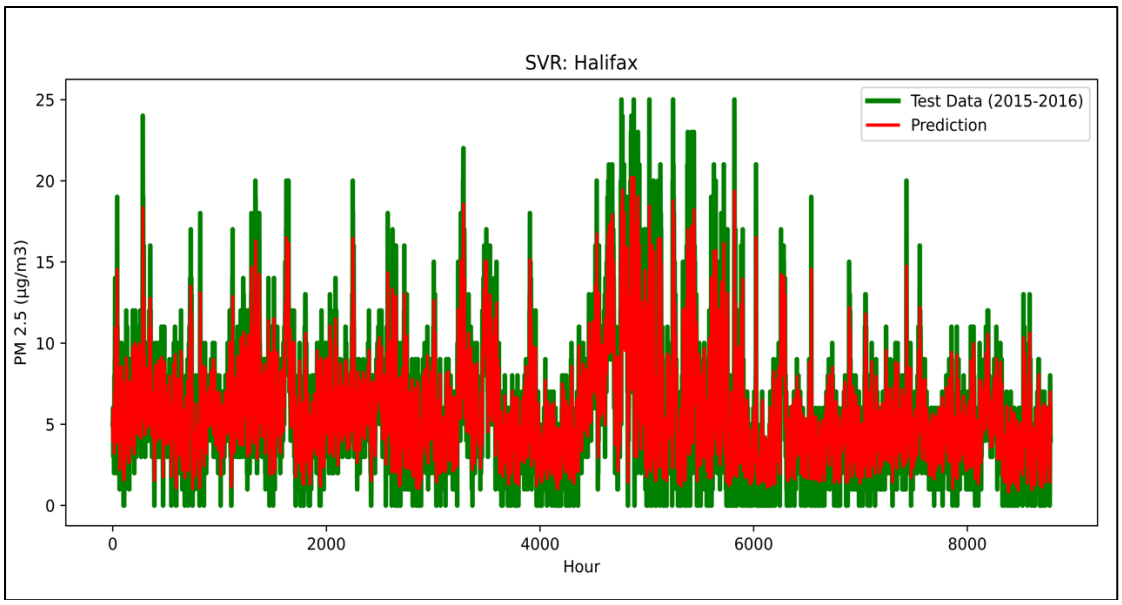

Figure 30: The predicted values of the SVR Model compared to the actual test set data for Halifax.

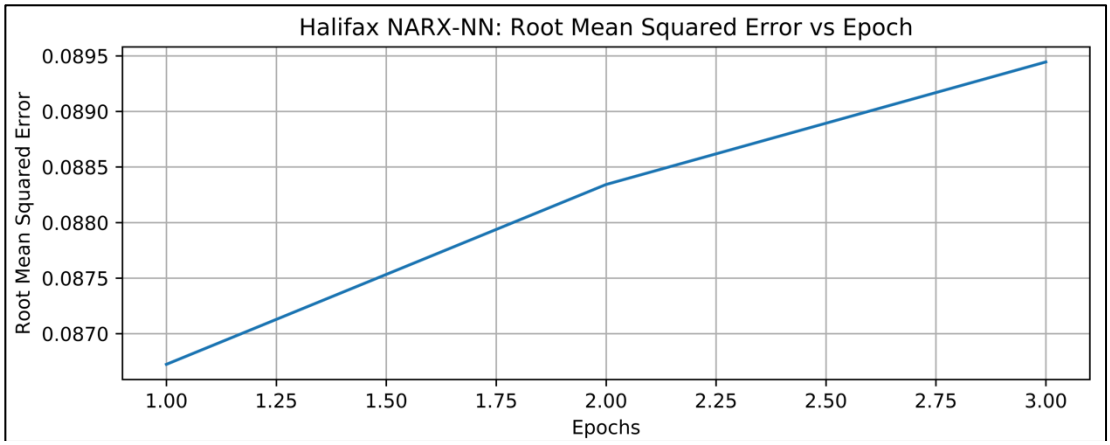

Figure 31: The RMSE over multiple epochs for the Halifax NARX-NN.

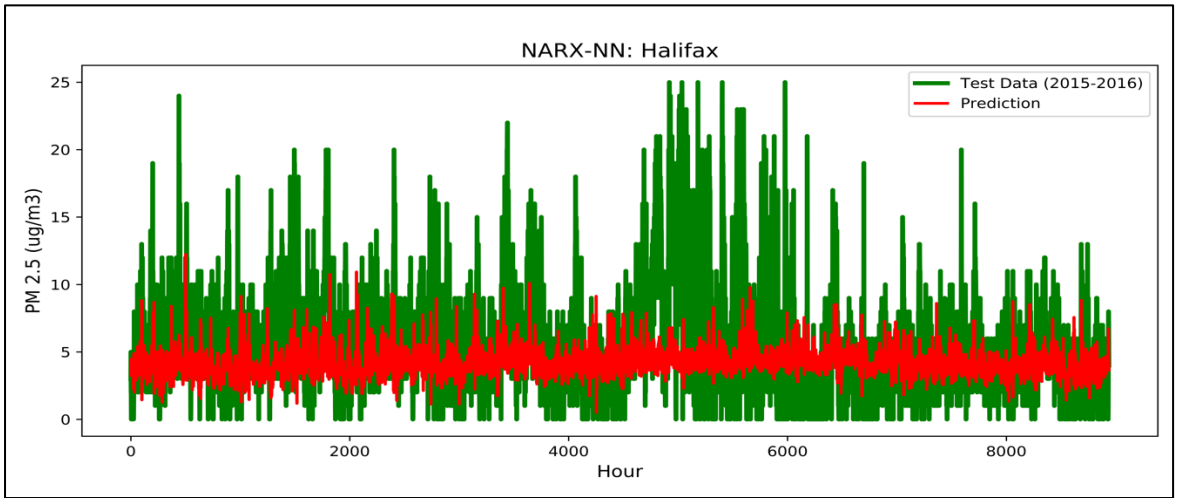

Figure 32: Displays the predicted values of the NARX-NN Model compared to the actual test dataset for Halifax.

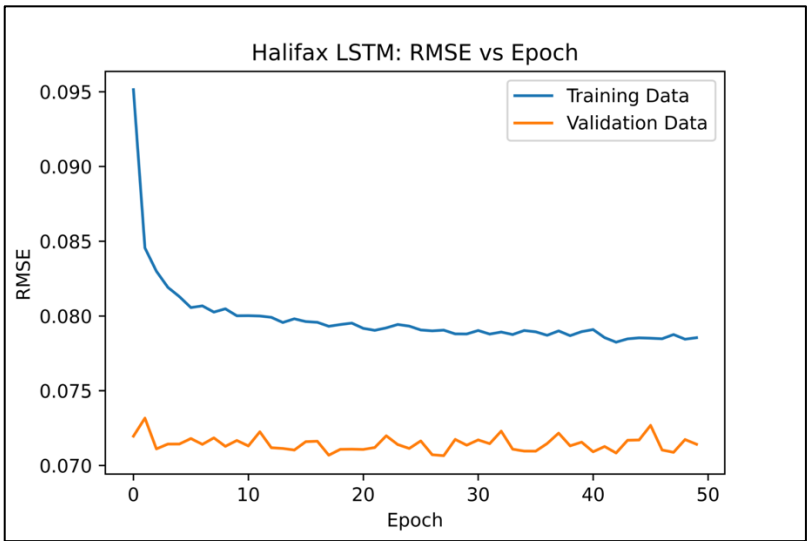

Figure 33: Displays how the RMSE for the validation and training data changes over epochs for the Halifax LSTM.

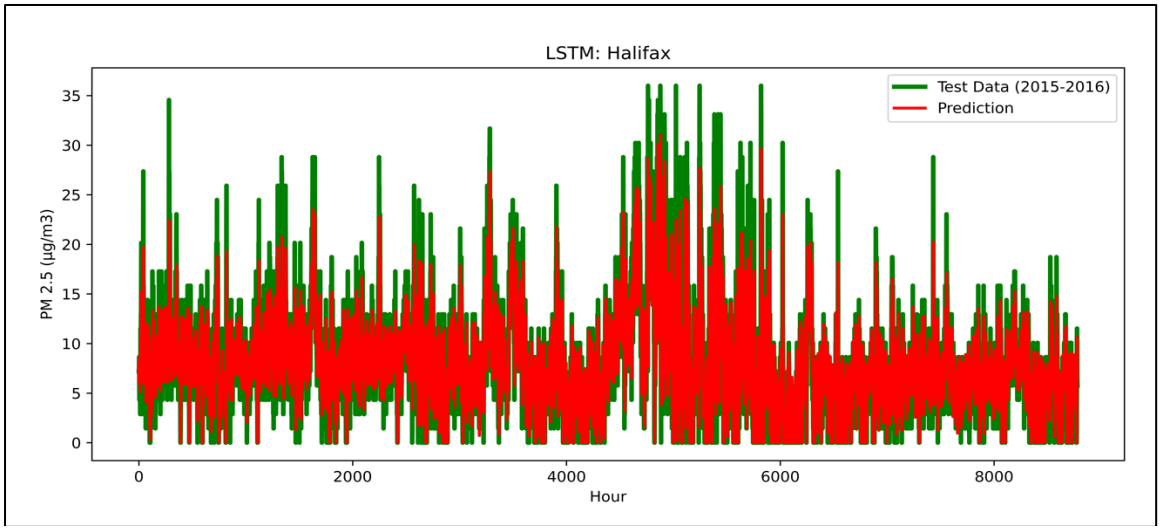

Figure 34: Displays the predicted values of the LSMT Model compared to the actual test set data for Halifax.

Montreal

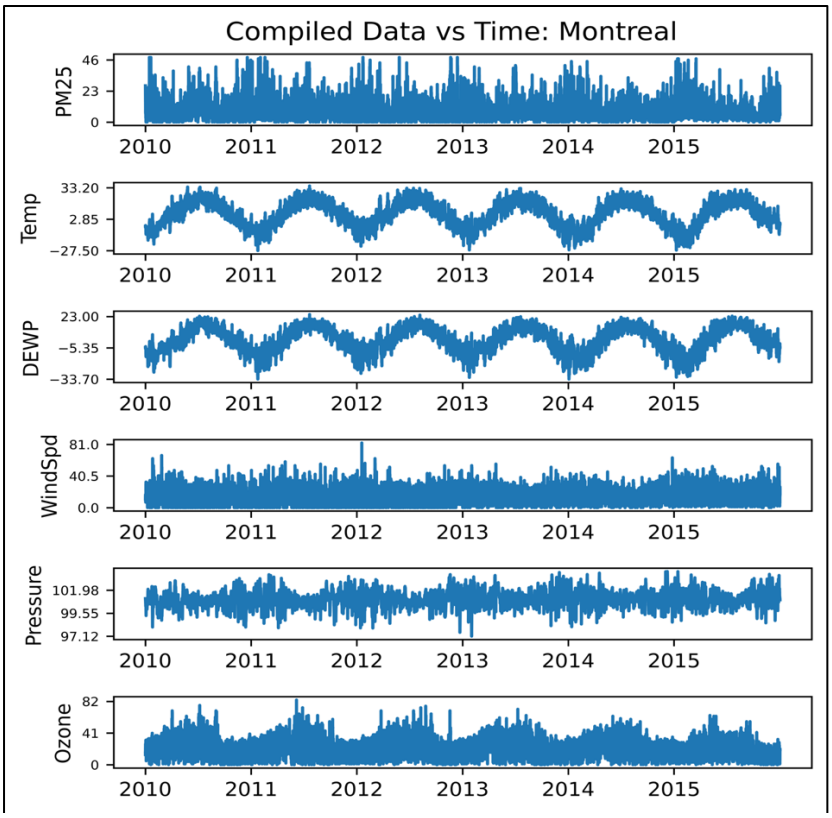

Figure 35: Plots the six variables with time for Montreal dataset.

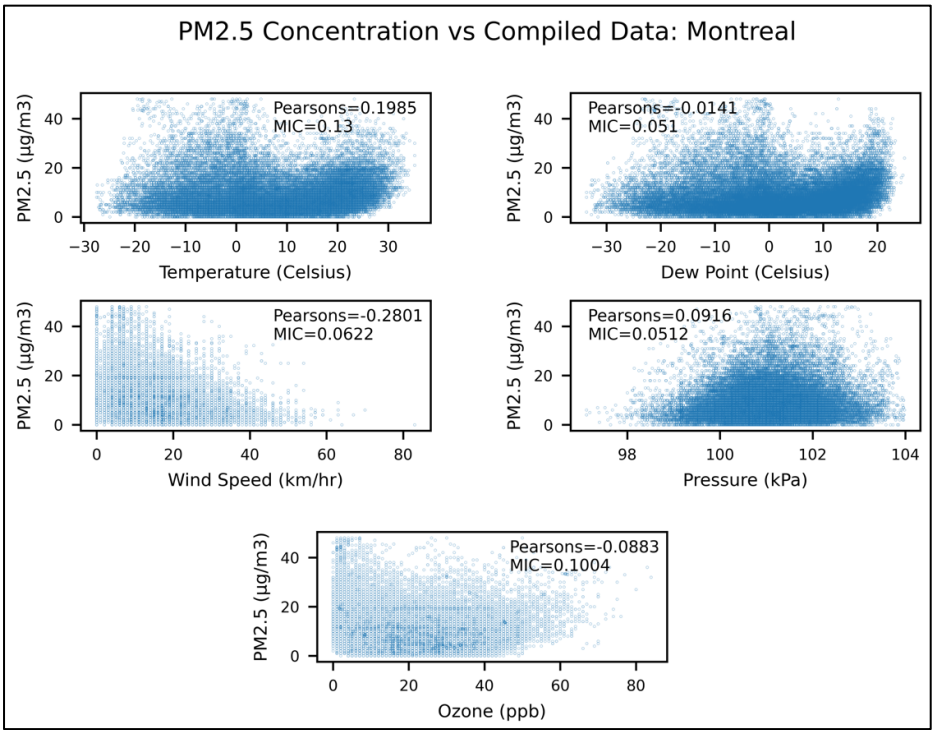

Figure 36: The correlations between $PM_{2.5}$ and each of the other variables for Montreal.

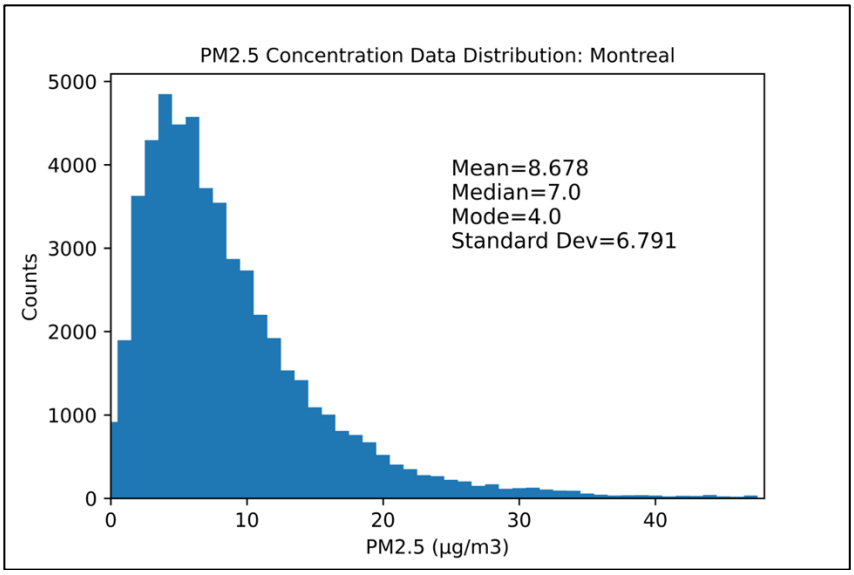

Figure 37: A histogram of the $PM_{2.5}$ data and the mean, median, mode, and standard deviation for Montreal.

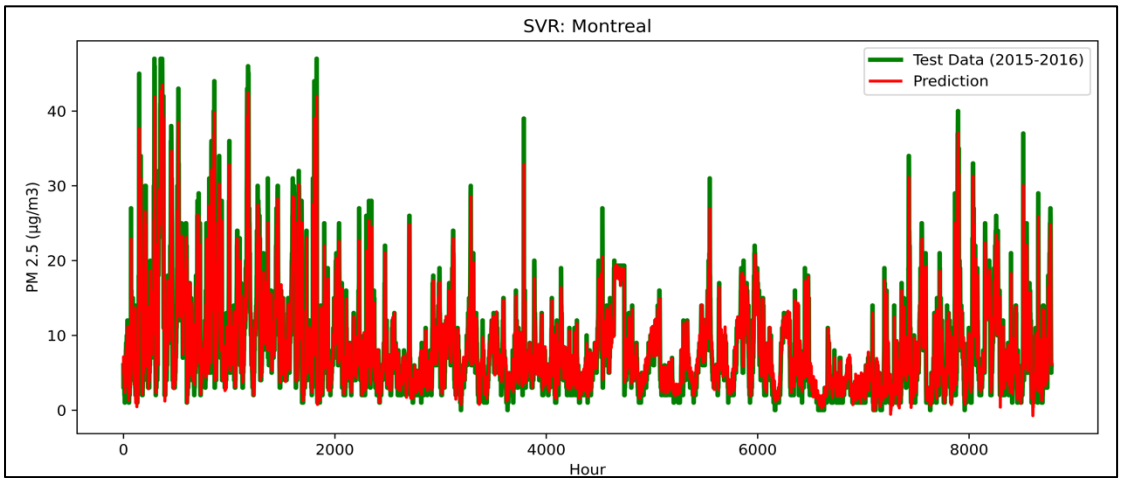

Figure 38: The predicted values of the SVR Model compared to the actual test set data for Montreal.

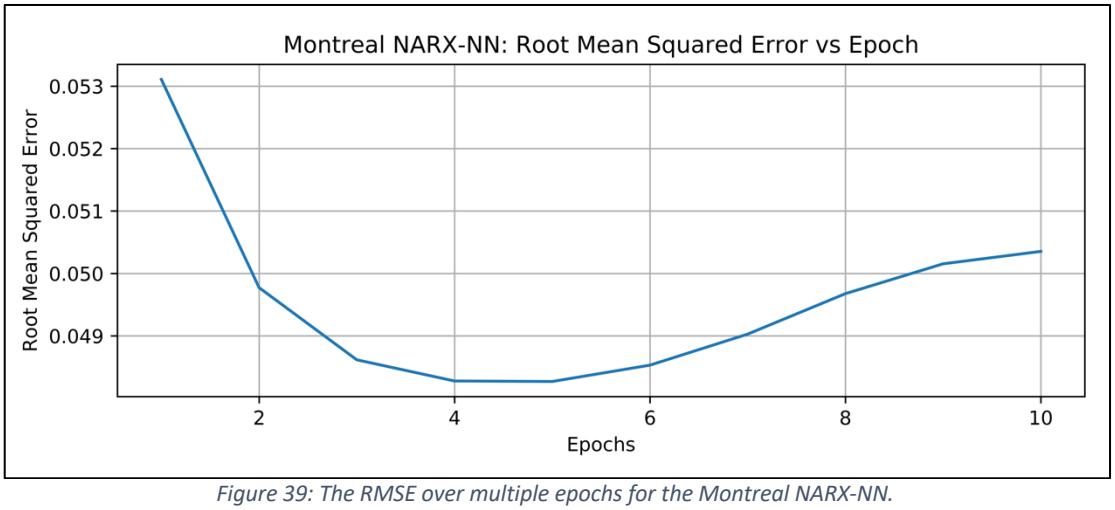

Figure 39: The RMSE over multiple epochs for the Montreal NARX-NN.

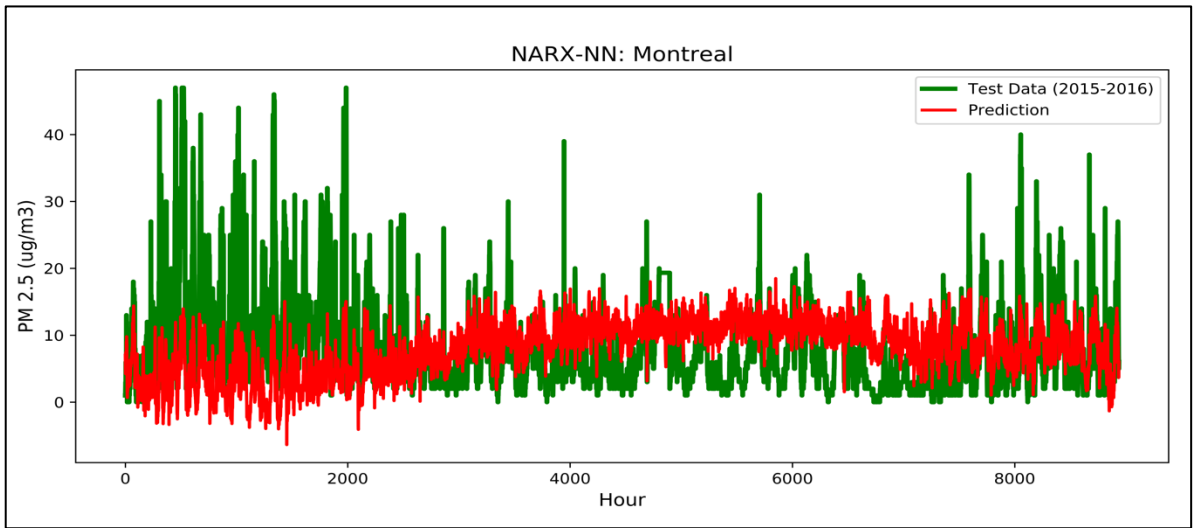

Figure 40: Displays the predicted values of the NARX-NN Model compared to the actual test dataset for Montreal.

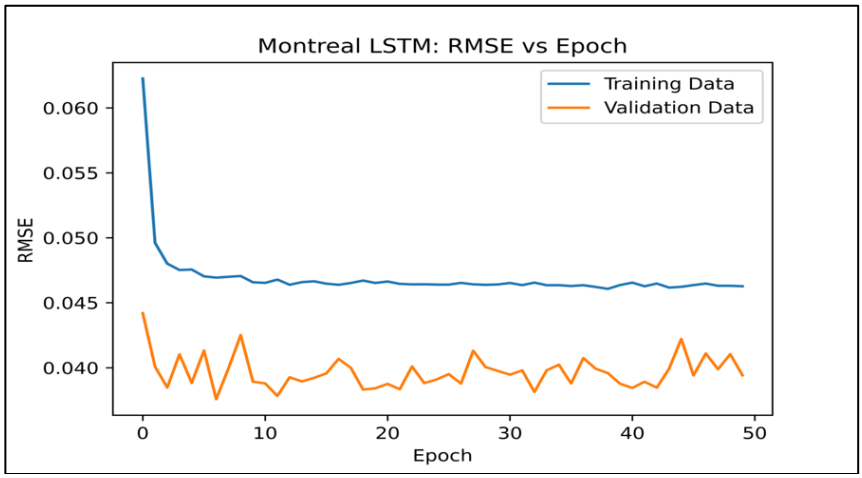

Figure 41: Displays how the RMSE for the validation and training data changes over epochs for the Montreal LSTM.

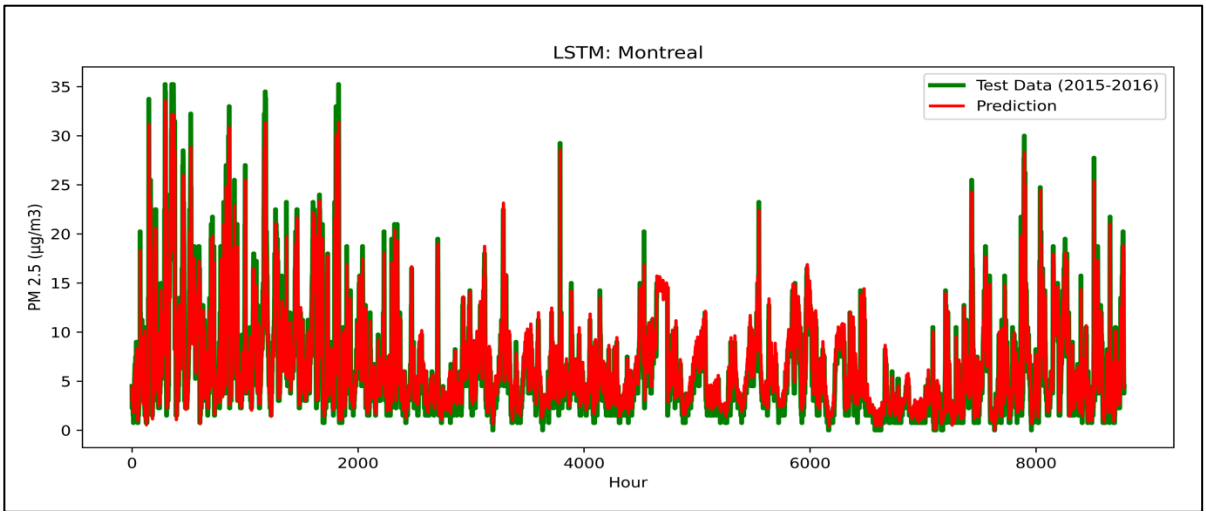

Figure 42: Displays the predicted values of the LSMT Model compared to the actual test set data for Montreal.

Vancouver

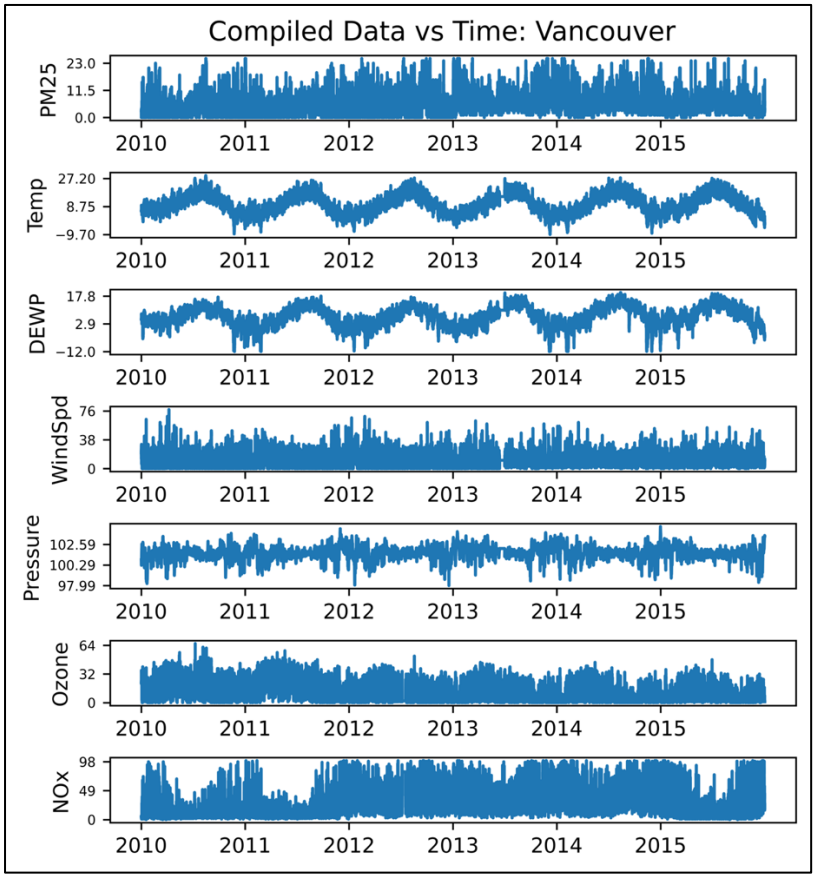

Figure 43: Plots of the seven variables with time for Vancouver Dataset.

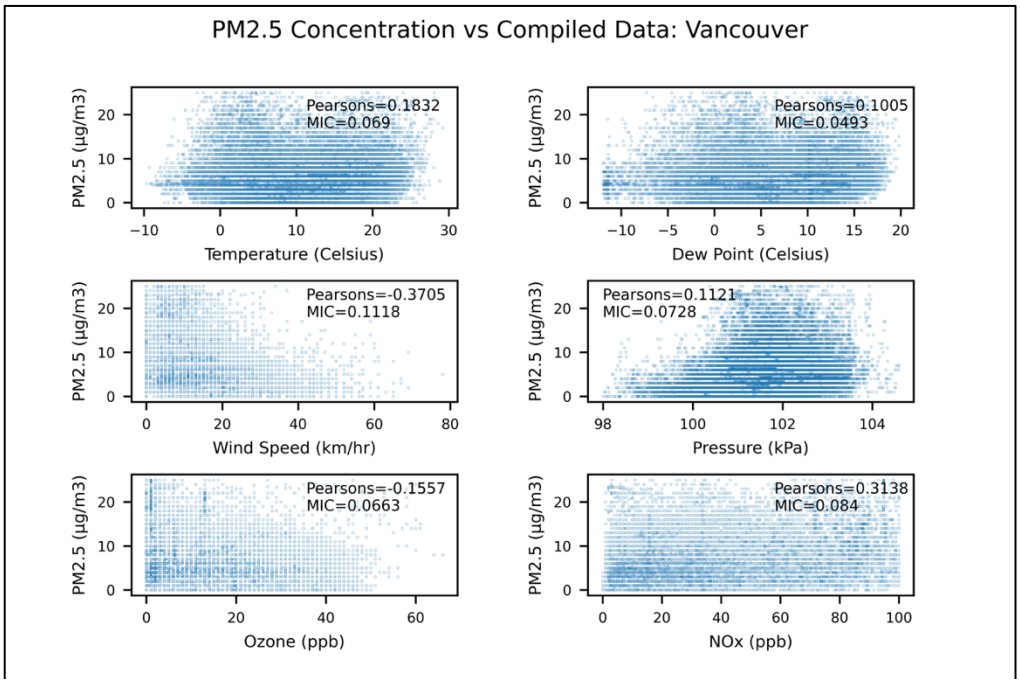

Figure 44: Correlations between $PM_{2.5}$ and each of the other variables for Vancouver.

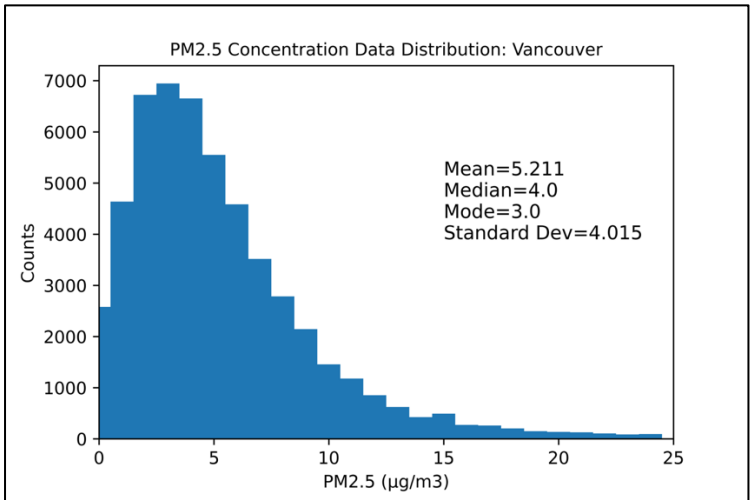

Figure 45: A histogram of the $PM_{2.5}$ data and the mean, median, mode, and standard deviation for Vancouver.

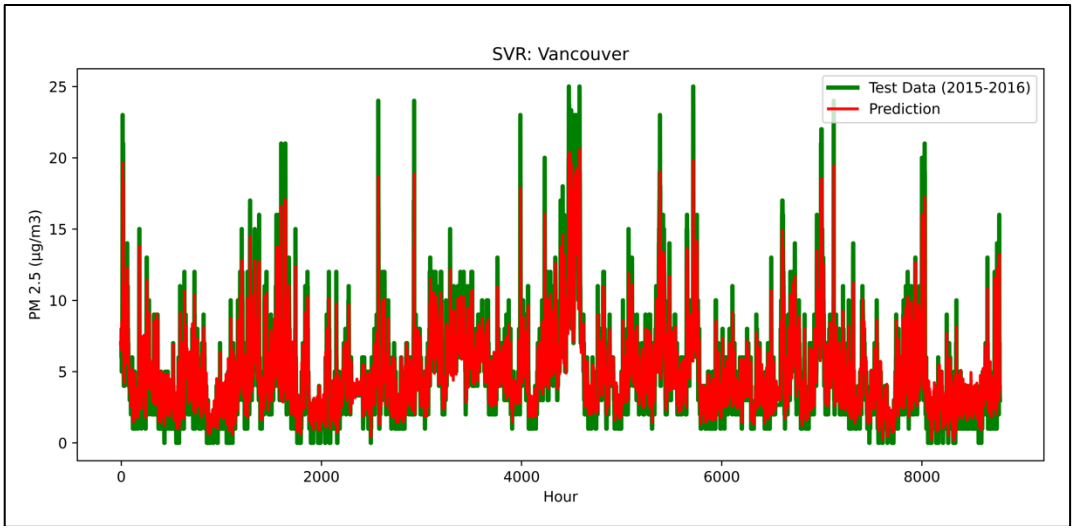

Figure 46: The predicted values of the SVR Model compared to the actual test set data for Vancouver.

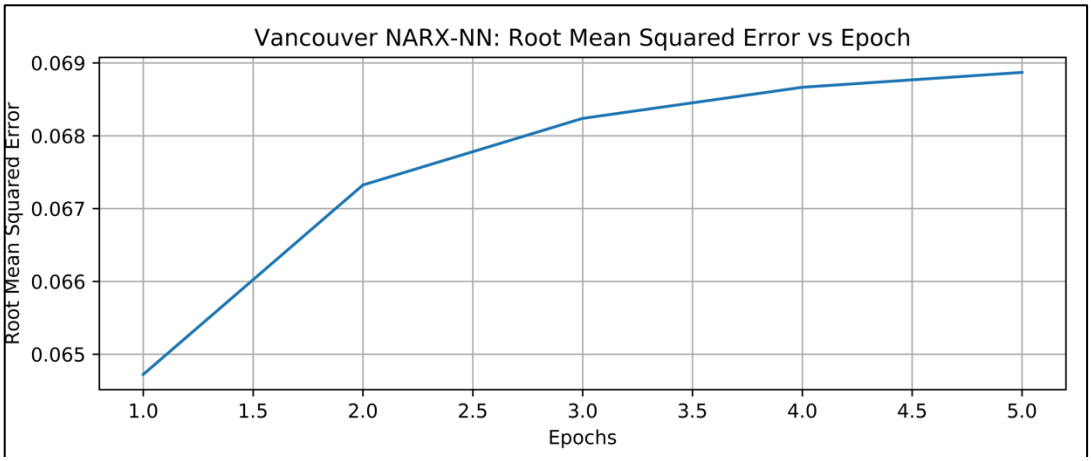

Figure 47: The RMSE over multiple epochs for the Vancouver NARX-NN.

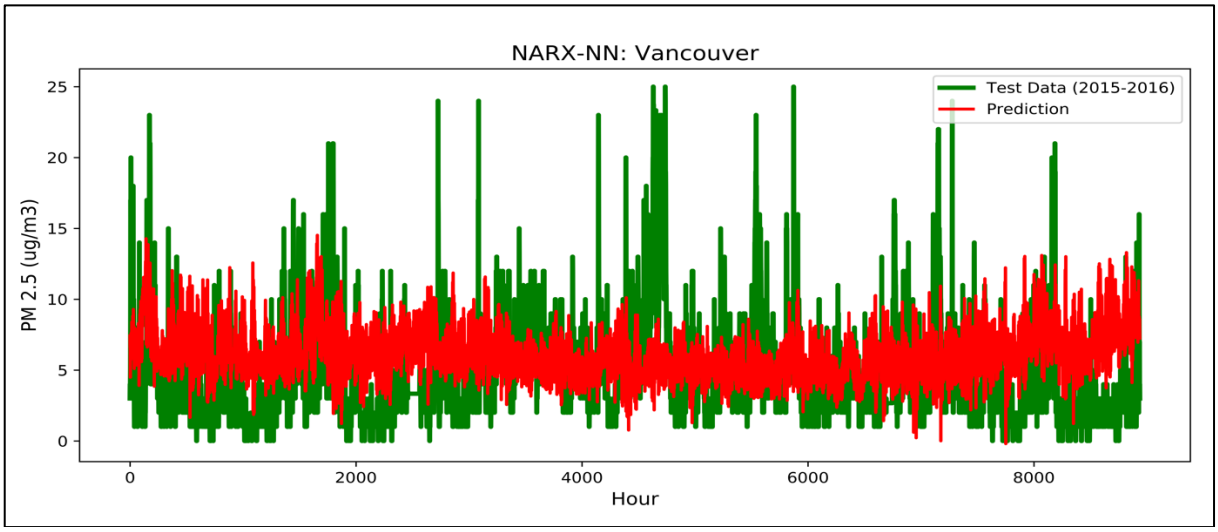

Figure 48: Displays the predicted values of the NARX-NN Model compared to the actual test dataset for Vancouver.

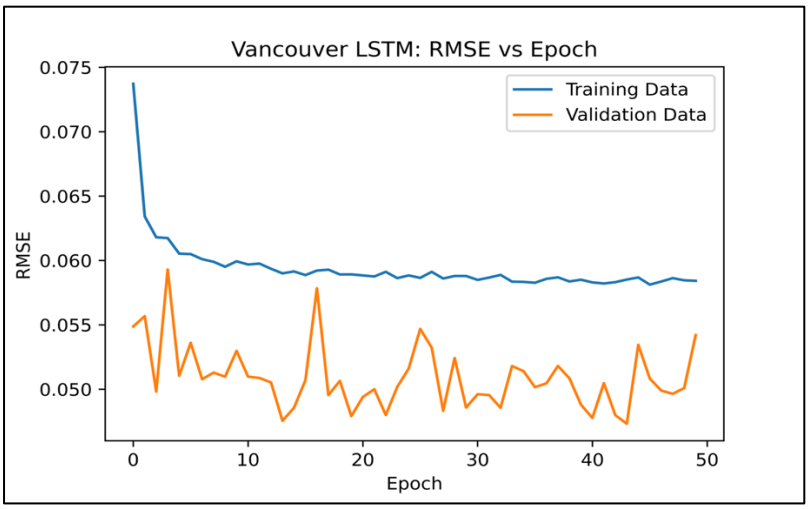

Figure 49: Displays how the RMSE for the validation and training data changes over epochs for the Vancouver LSTM.

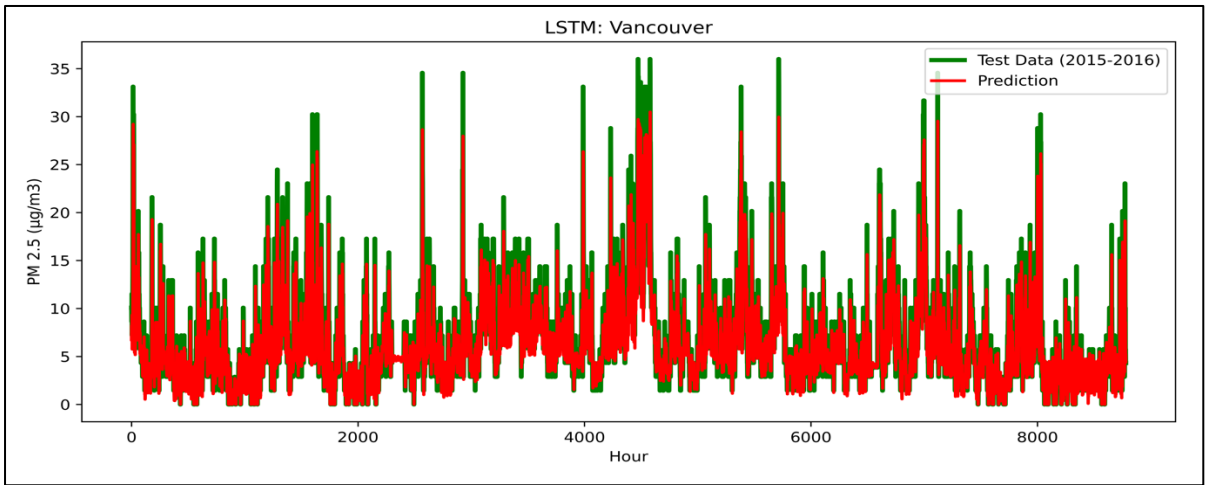

Figure 50: Displays the predicted values of the LSMT Model compared to the actual test set data for Vancouver.

Winnipeg

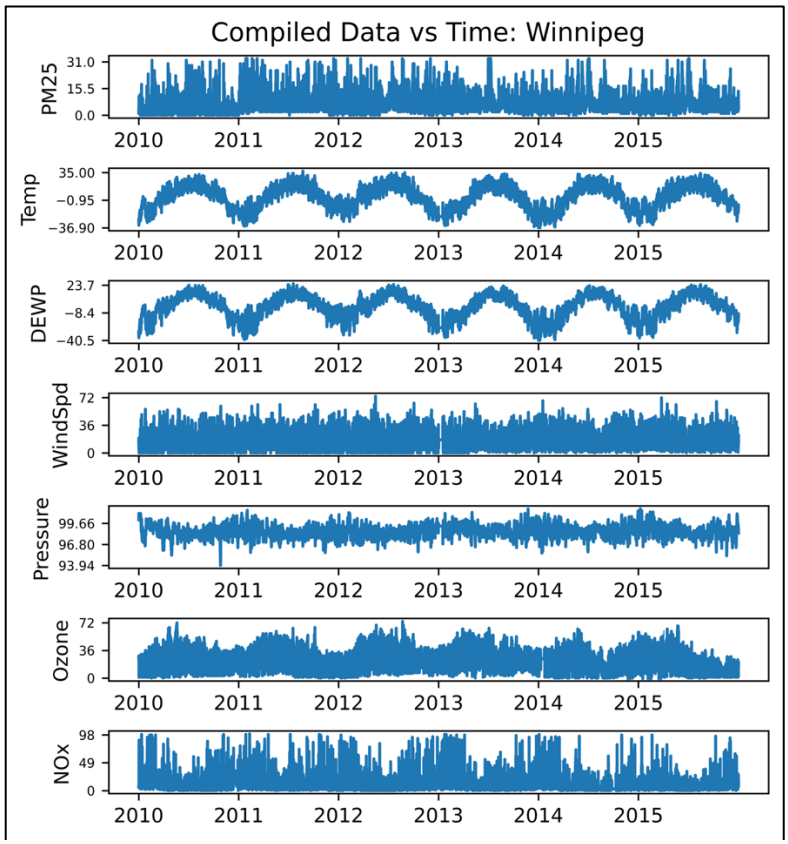

Figure 51: Plots each of the seven variables with time for Winnipeg dataset.

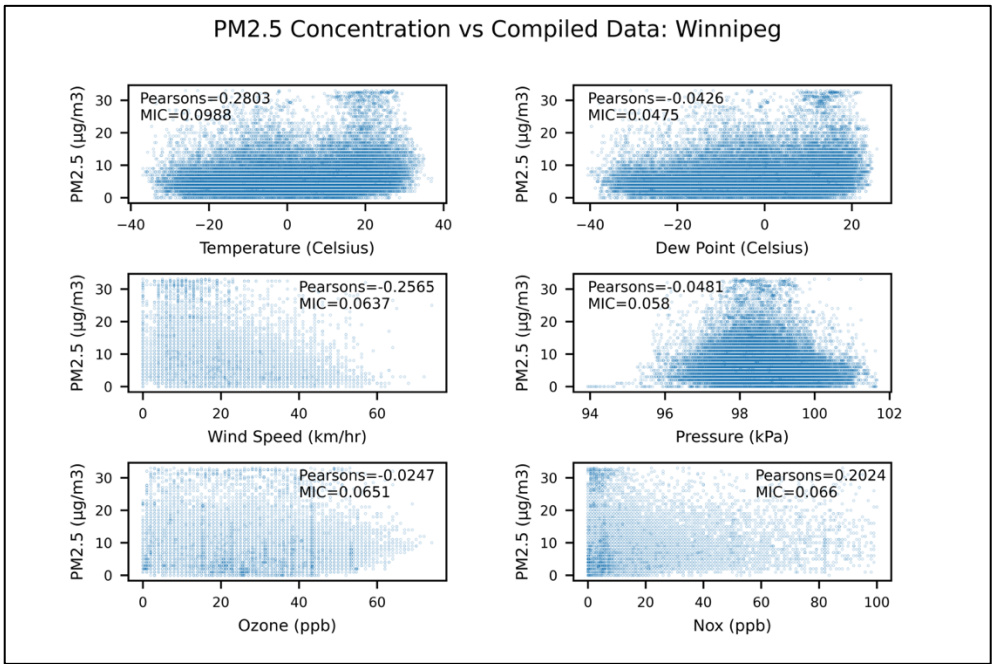

Figure 52: Correlations between $PM_{2.5}$ and each of the other variables for Winnipeg.

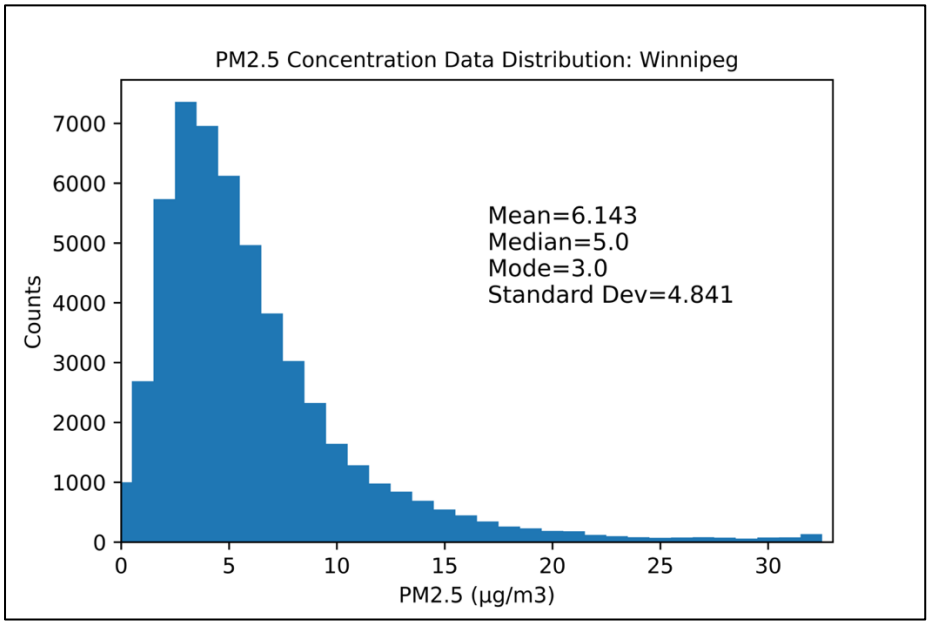

Figure 53: A histogram of the $PM_{2.5}$ data and the mean, median, mode, and standard deviation for Winnipeg.

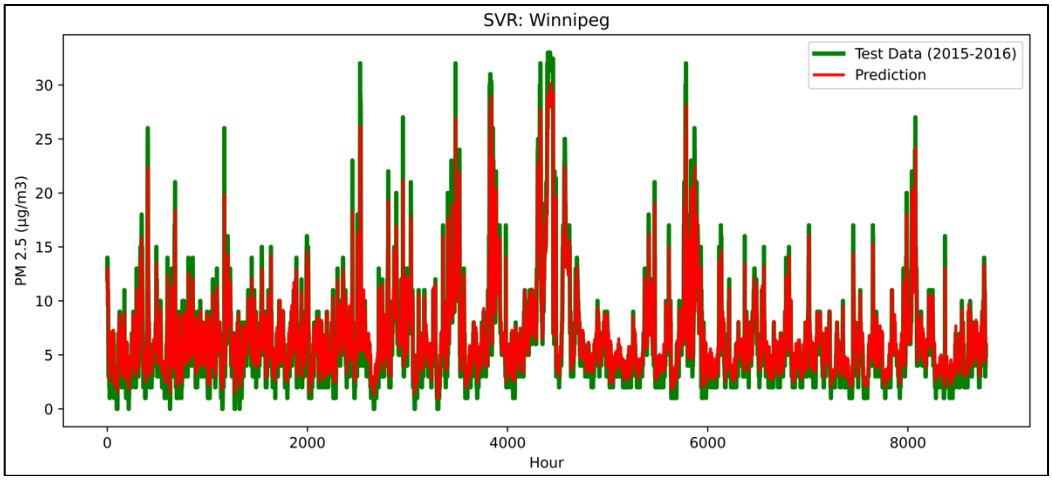

Figure 54: The predicted values of the SVR Model compared to the actual test set data for Winnipeg.

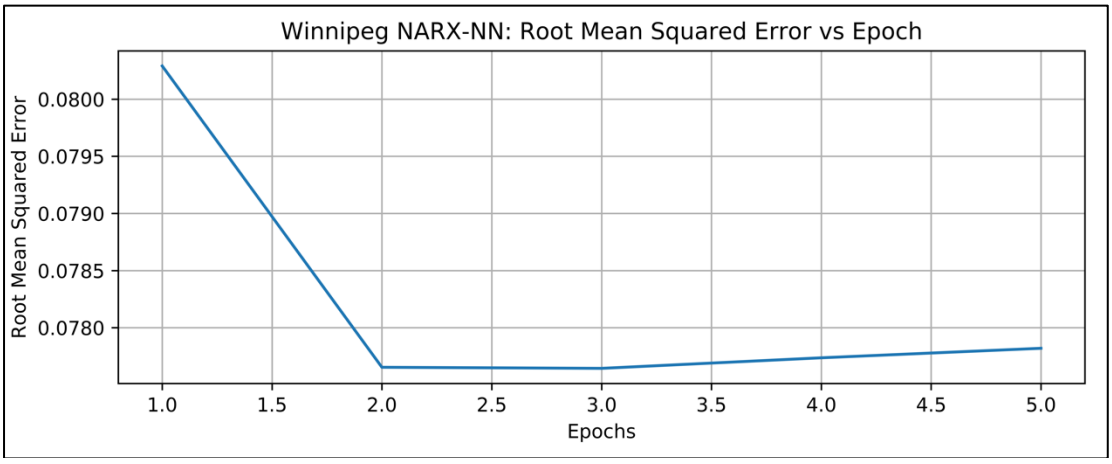

Figure 55: The RMSE over multiple epochs for the Winnipeg NARX-NN.

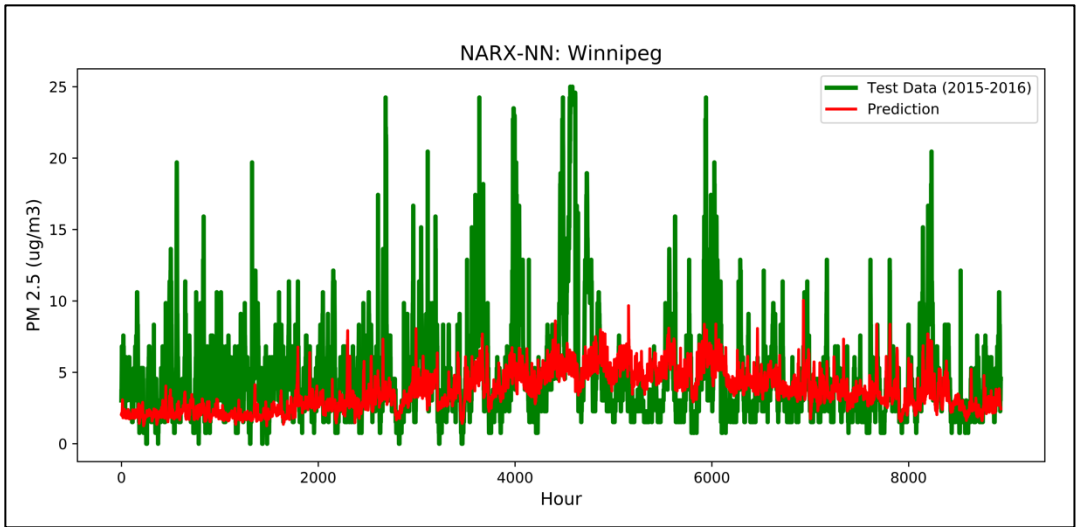

Figure 56: The predicted values of the NARX-NN Model compared to the actual test dataset for Winnipeg.

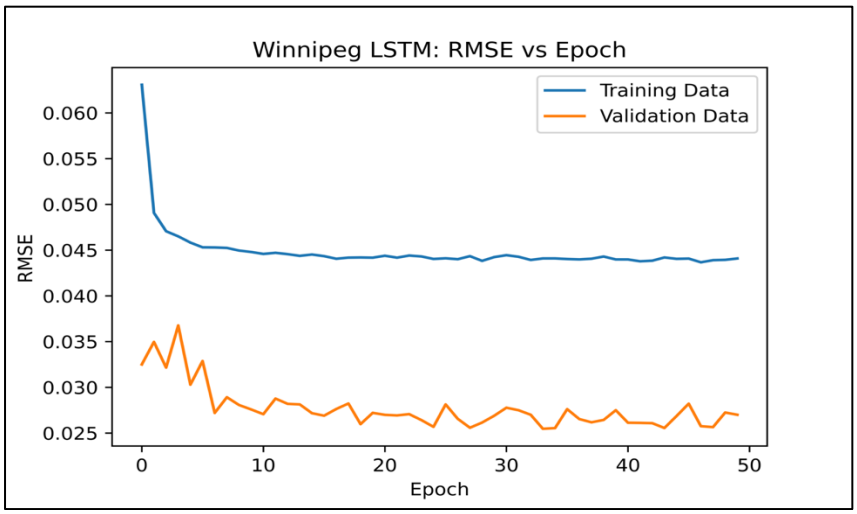

Figure 57: Displays how the RMSE for the validation and training data changes over epochs for the Winnipeg LSTM.

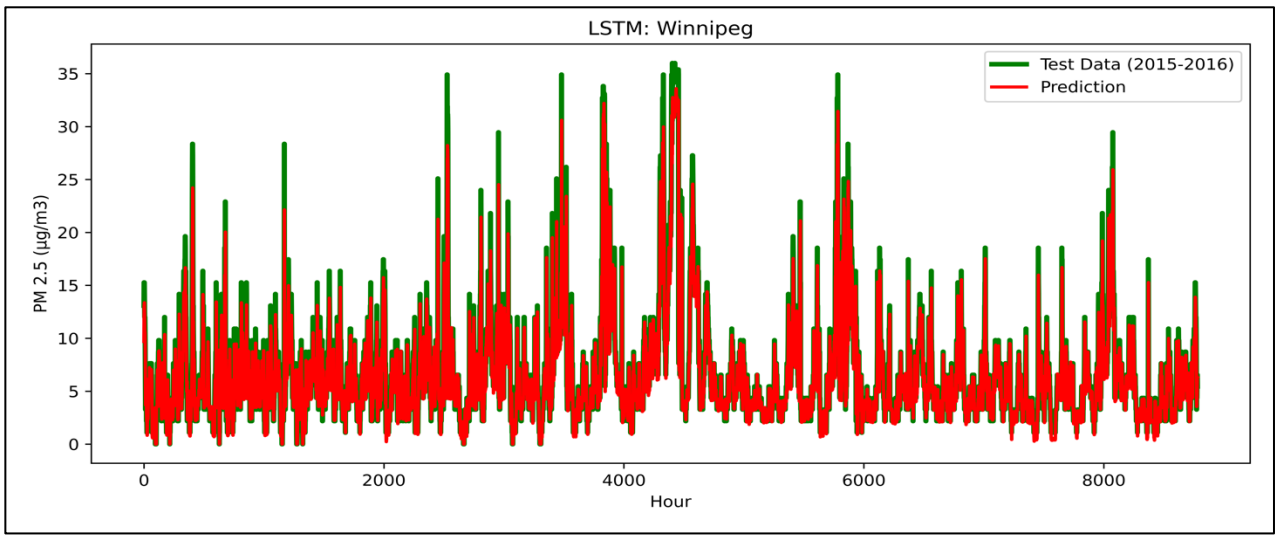

Figure 58: The predicted values of the LSMT Model compared to the actual test set data for Winnipeg.

Beijing

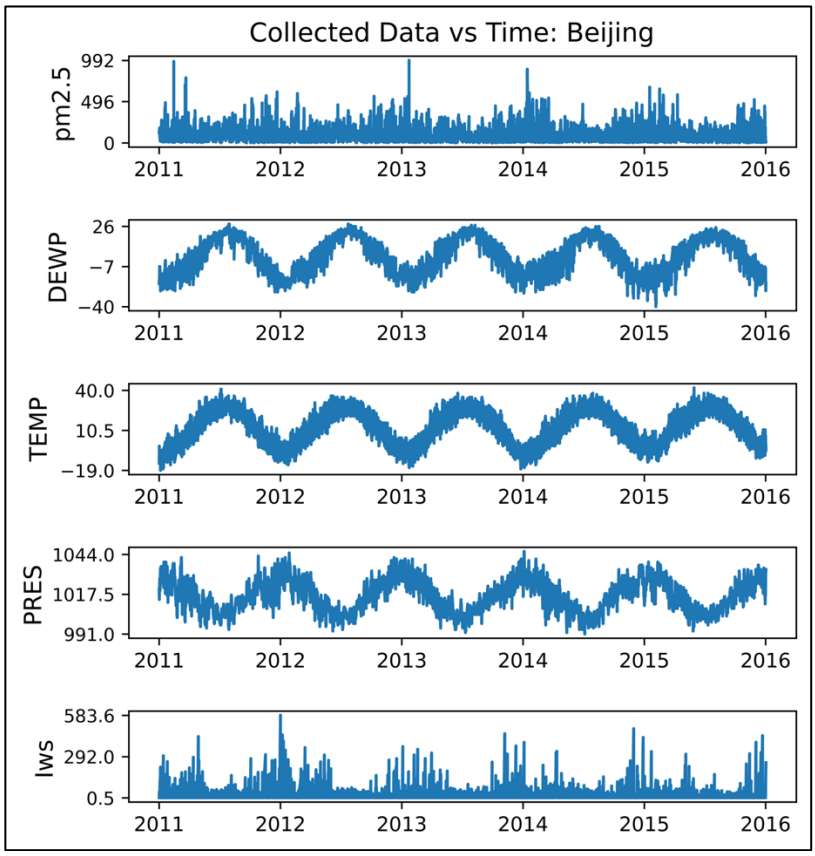

Figure 59: Displays how each of the seven variables change with time for Beijing.

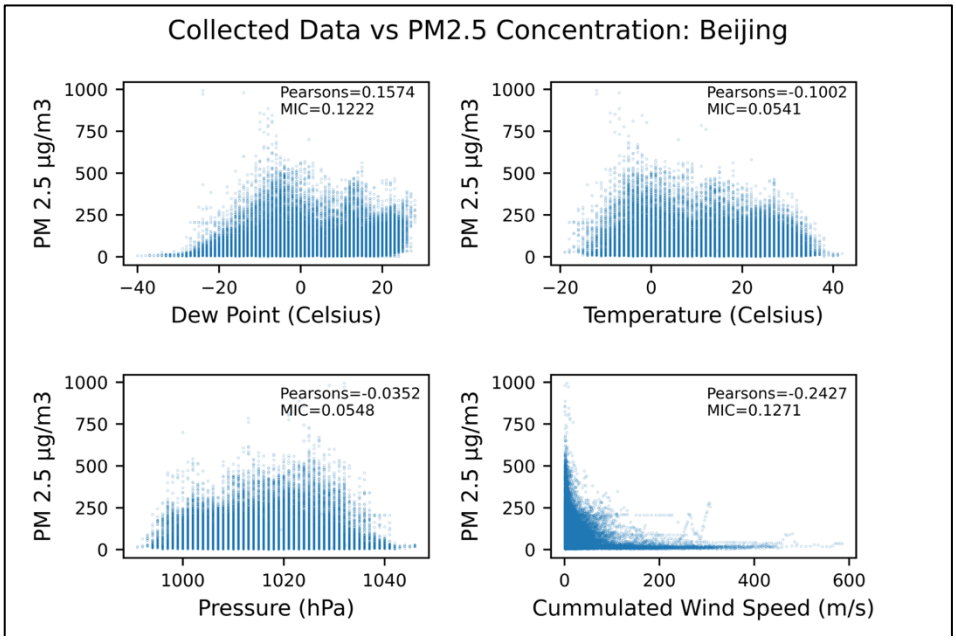

Figure 60: The correlations between $PM_{2.5}$ and each of the other variables for Beijing.

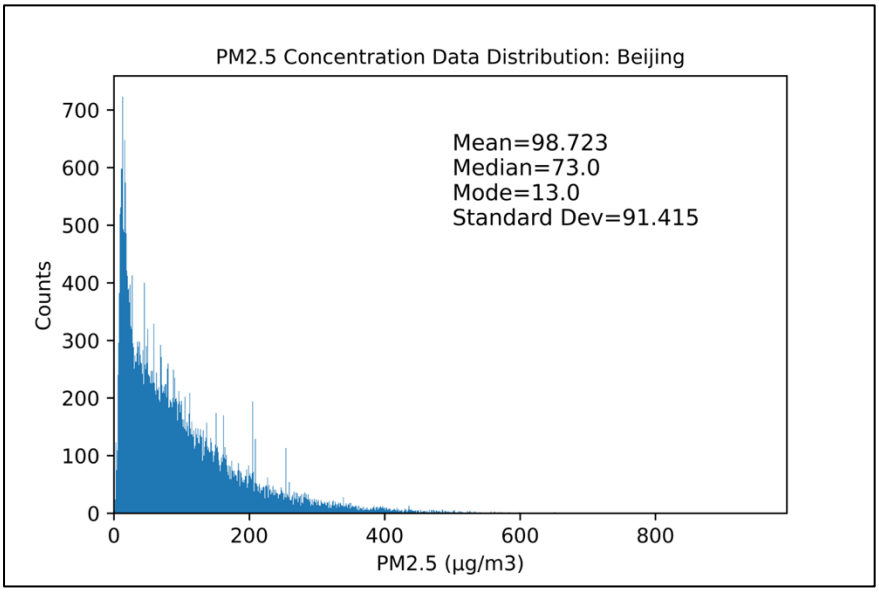

Figure 61: A histogram of the $PM_{2.5}$ data and the mean, median, mode, and standard deviation for Beijing.

6.5 Appendix E: Example Code

```
def pollutionDataFrames(start,end):
#Input: Start and End year of the data
#Outputs: dictionary containing dataframes for the pollution data from each city

  #Define NAPSIDs for the station to be used in each city
  NAPSID={'Edmonton':'90120',
  'Halifax':'30113',
  'Montreal':'50126',
  'Toronto':'60433',
  'Vancouver':'100118',
  'Winnipeg':'70118'};

  cityNames=('Edmonton','Halifax','Montreal','Toronto','Vancouver','Winnipeg');

  pollutionData={} #Initialize a dictionary

  #Locate, concatenate, and add pollution data to dictionary for each city:

  for i in range(0,len(cityNames)):

    #splitCities function imports all years data for each city and pollutant
    #then identifies and isolates the NAPSID for each city over the time period

    pollution=splitCities(start,end,NAPSID[cityNames[i]])
    pollutionData[cityNames[i]]=pollution

  return pollutionData
```

Figure 62: Pollution data import function in the higher quality code/ first attempt for creating the city datasets.

```
def importWeather():
#Output:dictionary containing dataframes for weather of each city
#Datafiles for each city have been saved starting at 1 for January 2010
#and increaing to 72 for December 2016

  #initialize path location array
  pathWeather = [k for  k in range(0, 72)]

  #Initialize dictionary for city weather data
  weatherData={}

  cityNames=('Edmonton','Halifax','Montreal','Toronto','Vancouver','Winnipeg');

  #Cycle through each city
  for k in range(0,len(cityNames)):

    basename = "/content/drive/My Drive/FinalModels/Preprocessing/" + cityNames[k] +'/'

    #Initialize arrays
    inTemp,inRelHum,inVis,inWind,Pressure,Date,Year,Month,Day,Hour= ([] for j in range(10));

    #Cycle through each month for that cities data
    for i in range(0, 72):
      pathWeather[i] = basename + str(i+1) + ".csv"

      datasetWeather = pd.read_csv(pathWeather[i])

      if i==0:

        inTemp=datasetWeather.iloc[:,9]
        inVis=datasetWeather.iloc[:,19]
        inWind=datasetWeather.iloc[:,17]
        inPressure=datasetWeather.iloc[:,21]
        Date=datasetWeather.iloc[:,4]
        Year=datasetWeather.iloc[:,5]
        Month=datasetWeather.iloc[:,6]
        Day=datasetWeather.iloc[:,7]

      else:

        inTemp=pd.concat((inTemp,datasetWeather.iloc[:,9]),axis=0)
        inVis=pd.concat((inVis,datasetWeather.iloc[:,19]),axis=0)
        inWind=pd.concat((inWind,datasetWeather.iloc[:,17]),axis=0)
        inPressure=pd.concat((inPressure,datasetWeather.iloc[:,21]),axis=0)
        Date=pd.concat((Date,datasetWeather.iloc[:,4]),axis=0)
        Year=pd.concat((Year,datasetWeather.iloc[:,5]),axis=0)
        Month=pd.concat((Month,datasetWeather.iloc[:,6]),axis=0)
        Day=pd.concat((Day,datasetWeather.iloc[:,7]),axis=0)

    weather=pd.concat((Date,Year,Month,Day,inTemp,inVis,inWind,inPressure),axis=1);

    weatherData[cityNames[k]]=weather

  return weatherData
```

Figure 63: Weather data import function in the higher quality code/ first attempt for creating the city datasets.

```
# Directly from the Colah Turorial: How to Convert a Time Series
# to a Supervised Learning Problem in Python
# This changes time series data and converts it to a formate
# usable for machine learning problems.

def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
  n_vars = 1 if type(data) is list else data.shape[1]
  df = pd.DataFrame(data)
  cols, names = list(), list()
  # input sequence (t-n, ... t-1)
  for i in range(n_in, 0, -1):
    cols.append(df.shift(i))
    names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
  # forecast sequence (t, t+1, ... t+n)
  for i in range(0, n_out):
    cols.append(df.shift(-i))
    if i == 0:
      names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
    else:
      names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
  # put it all together
  agg = concat(cols, axis=1)
  agg.columns = names
  # drop rows with NaN values
  if dropnan:
    agg.dropna(inplace=True)
  return agg

# load dataset
values = new_dataset.values
# ensure all data is float
values = values.astype('float32')

# normalize features
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(values)
```

Figure 64: Displays the code used for reformatting the time series data.

```
SVR=svm.SVR(kernel='poly',gamma=0.0001,C=1000000)
SVR.fit(X_train,Y_train.ravel())

Y_predict=SVR.predict(X_test)
```

Figure 65: Displays part of the code for implementing the SVR.

```
#Node paramaters that need to be selected

input_nodes = 17 # 6 dataset variables, 5 from previous time steps,
hidden_nodes = 10
output_nodes = 1 #PM2.5 prediction

#Left as default from PyNeurGen Example

output_order = 3
incoming_weight_from_output = .6
input_order = 2
incoming_weight_from_input = .4

net = NeuralNet()
net.init_layers(input_nodes, [hidden_nodes], output_nodes,
        NARXRecurrent(
            output_order,
            incoming_weight_from_output,
            input_order,
            incoming_weight_from_input))

net.randomize_network()
```

```
net.set_all_inputs(all_inputs.values)
net.set_all_targets(all_targets.values)
```

```
#Paramater that needs to be optimized
net.set_learnrate(.007)
```

```
#Setting Activation Function
net.layers[1].set_activation_type('tanh')
```

```
net.learn(epochs=5, show_epoch_results=False,random_testing=False)
```

Figure 66: Part of the code used for the NARX-NN Models.

```
model=Sequential()
model.add(LSTM(30, input_shape=(TRAIN_X.shape[1], TRAIN_X.shape[2])))
model.add (Dropout(0.5))
model.add(Dense(1,activation='relu'))
model.add(Dense(1,activation='relu'))
model.compile(loss='mae', optimizer='adam')
keras.callbacks.History()
history=model.fit(TRAIN_X, TRAIN_y, epochs=30, validation_data=(VAL_X, VAL_y))
```

Figure 67: Part of the code for implementing the LSTM.