# Air Pollution Forecasting with LSTM

Xiaoyang Chen, Julia Jin

George Washington University

DATS 6203: Machine Learning II

Professor: Amir Jafari

December 12, 2022

# 1 Introduction

This project predicts air pollution based on the Air Quality dataset from Kaggle. This is a dataset that reports on the weather and the level of pollution each hour for five years at the US embassy in Beijing, China. The data includes the date-time, the pollution called PM2.5 concentration, and the weather information including dew point, temperature, pressure, wind direction, wind speed and the cumulative number of hours of snow and rain. In the subsection 1.1, we provide background information about air pollution. In the subsection 1.2, the report outline is presented.

## 1.1 Background

Air pollution is one of the most concerns for urban areas. There are countries around the world have built a variety of sensing devices for monitoring PM2.5 concentrations. There were also many studies have been constructed to predict and forecast various air pollution [1]. Due to its proven track record of success with time-series data, a Long Short-Term Memory (LSTM) Recurrent Neural Network (RNN) model was chosen to perform the task of air quality forecasting. The data provides hourly average concentrations of various air pollutants around the U.S. embassy in Beijing over five years. We chose the Long Short-Term Memory (LSTM) Recurrent Neural Network (RNN) model for the air quality prediction task due to its successful track record with time-series data. This article compares the difference between the multivariate input model and the single variable input model in the LSTM model by establishing a multivariate input model and a single variable input model.

## 1.2 LSTM

According to the Wikipedia's definition, long short-term memory (LSTM) is an artificial neural network used in the fields of artificial intelligence and deep learning [2]. LSTM units are a building unit for layers of a (RNN). A RNN composed of LSTM units is often called an LSTM network. The difference between LSTM and traditional RNN neural networks is that each neuron in LSTM is a memory cell. The LSTM links the previous data information to the current neurons. Each neuron contains three gates as shown in Figure 1: input gate, forget gate, and output gate. Using the internal gate, the LSTM can solve the problem of long-term dependence of the data [1].

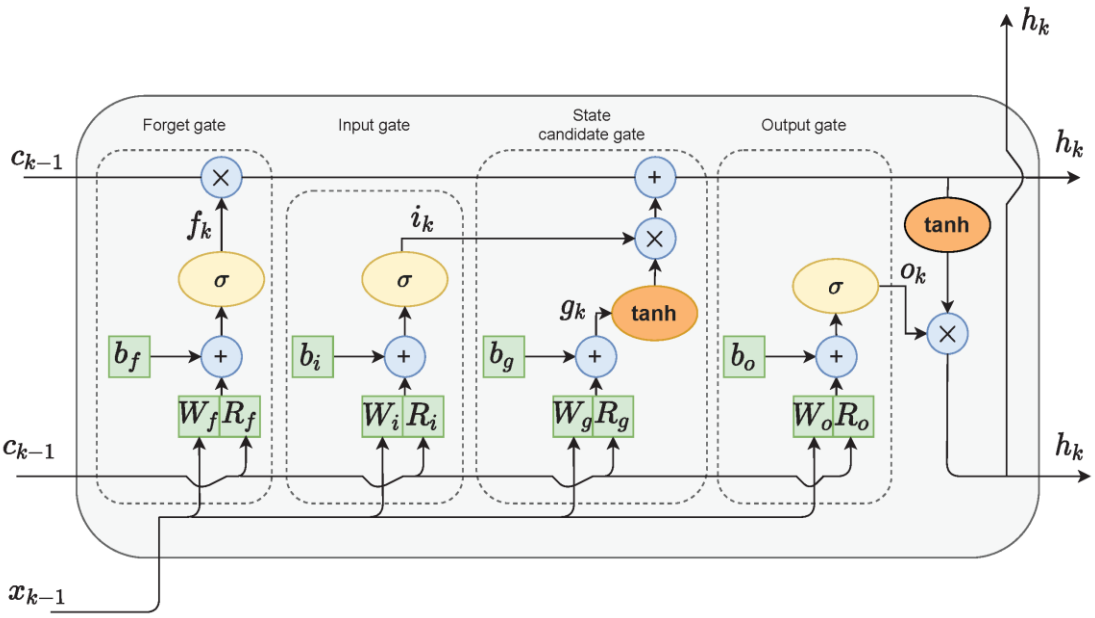


Figure 1. LSTM Architecture

1) Forget gate

One of the main properties of the LSTM is to memorize and recognize the information coming inside the network and also to discard the information which is not required to the network to learn the data and predictions. This gate is responsible for this feature of the LSTM.

2) Input gate

Input gate helps in deciding the importance of the information by updating the cell state. where the forget gate helps in the elimination of the information from the network input gate decides the measure of the importance of the information and helps the forget function in elimination of the not important information and other layers to learn the information which is important for making predictions.

3) Output gate

It is the last gate of the circuit that helps in deciding the next hidden state of the network in which information goes through the sigmoid function. Updated cell from the cell state goes to the tanh function then it gets multiplied by the sigmoid function of the output state. Which helps the hidden state to carry the information [3].

In a long short-term memory (LSTM) network, the activation function used for the gates (input, forget, and output gates) is the sigmoid function, while the activation function used for the cell state and output is the hyperbolic tangent (tanh) function.

The sigmoid function is used for the gates because it has a range of 0 to 1, which makes it well-suited for binary classification tasks. It also has a smooth derivative, which makes it easy to compute the gradient during training.

The tanh function is used for the cell state and output because it has a range of -1 to 1, which makes it well-suited for representing numerical values. It also has a non-linear shape, which allows the LSTM network to capture complex patterns in the data.

In summary, the choice of activation functions in an LSTM network is a trade-off between range, smoothness, and non-linearity, and the sigmoid and tanh functions are well-suited for the tasks performed by the gates and cell state/output, respectively.

## 1.3 Outline

In this report, we discuss models that use LSTMs to predict air quality. In section 2, the source of the data is described first, and then how we preprocess the data is discussed. And in section 2, we visualized the data, looked at the graph of different characteristic data changing according to time and the relationship diagram between each characteristic data.

In section 3, we discussed the multivariate model and univariate model established in the project. In section 4, we discuss the data results from multivariate model and univariate model, and compare and summarize in section 5.

# 2 Data Mining

## 2.1 Dataset Description

The dataset we used in this project is the air quality dataset which was reported on the weather and the level of pollution each hour for five years at the US embassy in Beijing, China.

## 2.2 Dataset Preprocessing

In Figure 2, it is the information about different columns in dataset.

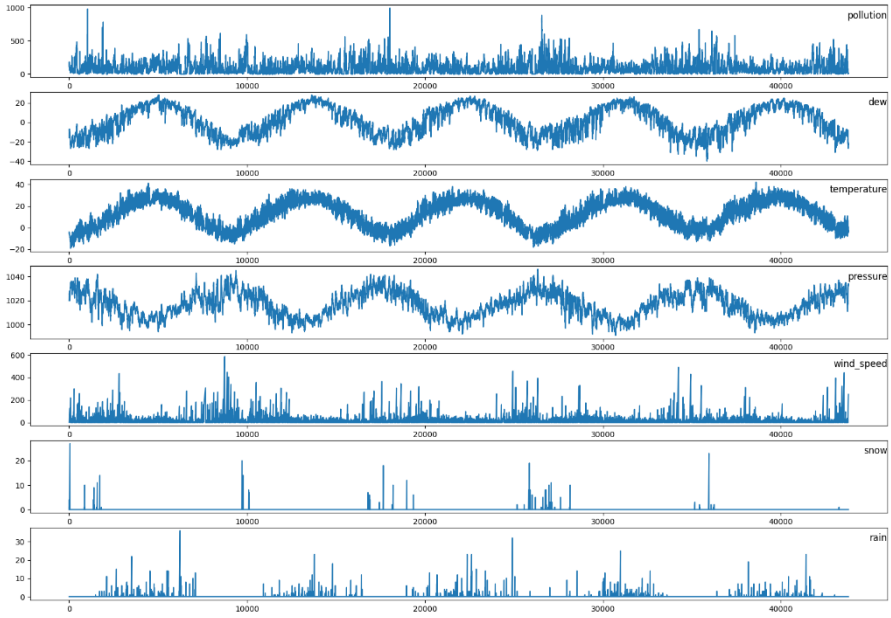


Figure 2: Columns Information

However, as shown in the Figure 2, there are 8 columns in our dataset, but only 7 are shown. The data about “wind_direction” is a “string” type. So we use the function “LabelEncoder” from “sklearn.preprocessing” to change its type into “float” [4]. Then as shown in Figure 3, we used function “matshow” from “matplotlib.pyplot” and “corr” function from “pandas” build an correlation matrix.

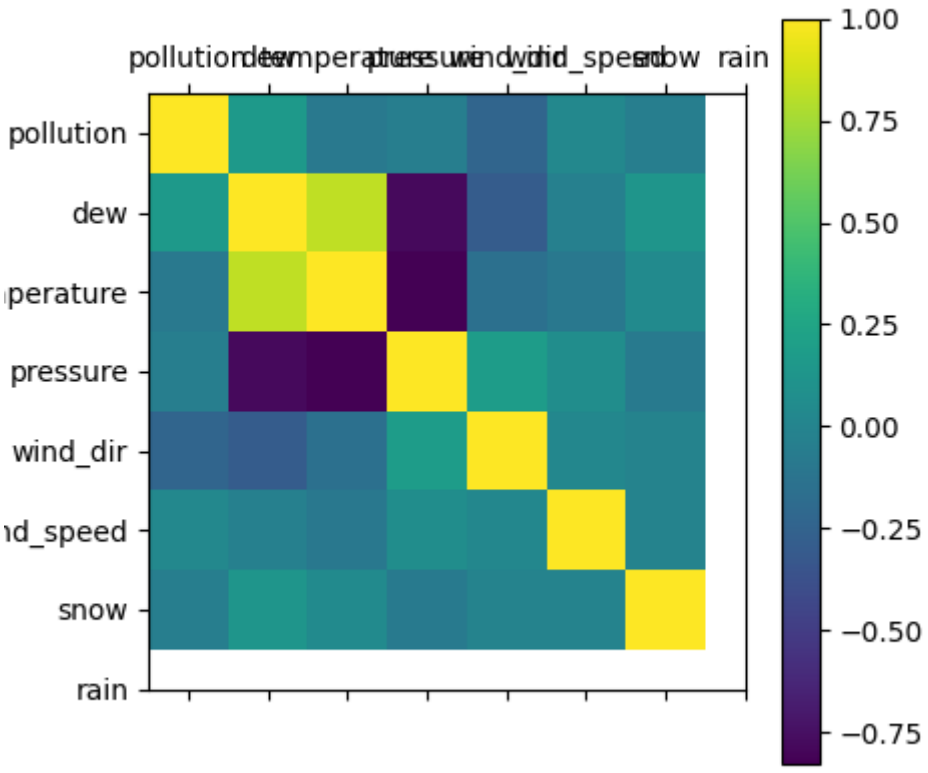


Figure 3: Correlation Matrix

In addition to doing data visualization, we also used “MinMaxScaler” to normalize the data. The benefit of normalizing the data is that it can increase the convergence speed of the iterative solution and improve the accuracy of the iterative solution.

Another situation is we are using LSTM, but we only have data from 2010 to 2014. Before machine learning can be used, time series forecasting problems must be re-framed as supervised learning problems. From a sequence to pairs of input and output sequences. Then we used the function “series_to_supervised” as shown in Figure 4.

```python
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 = pd.concat(cols, axis=1)
    agg.columns = names
    # drop rows with NaN values
    if dropnan:
        agg.dropna(inplace=True)

    return agg
```

Figure 4: Convert Time Series to Supervised Problem

The function is defined with default parameters so that it will construct a dataframe with t - 1 as X and t as y [5].

# 3 Model Description

## 3.1 Multivariate Model

For building this model, we divided our dataset into two parts. The dataset from 2010 to 2013 is used as the training part, and the dataset from 2014 is used as the validation part. For our model, we used batch normalization which is used to improve the performance and stability of neural

networks. This normalization helps to reduce the internal covariate shift, which is the change in the distribution of the inputs to a layer caused by the change in the parameters of the previous layer. It can improve the convergence rate and overall performance of the network. As shown in Figure 5, we also used dropout layer. It is used to prevent overfitting in our networks. It works by randomly dropping out, or setting to 0, a certain number of outputs from a layer during training. It has the effect of reducing the number of connections between the layers, which in turn reduces the complexity of the model and helps to prevent overfitting. In “tensorflow.keras” library, there is a function called “LSTM” which can be easily applied [6]. The activation function is already set as “tanh” and the recurrent activation is set as “sigmoid” as we talked in previous section.

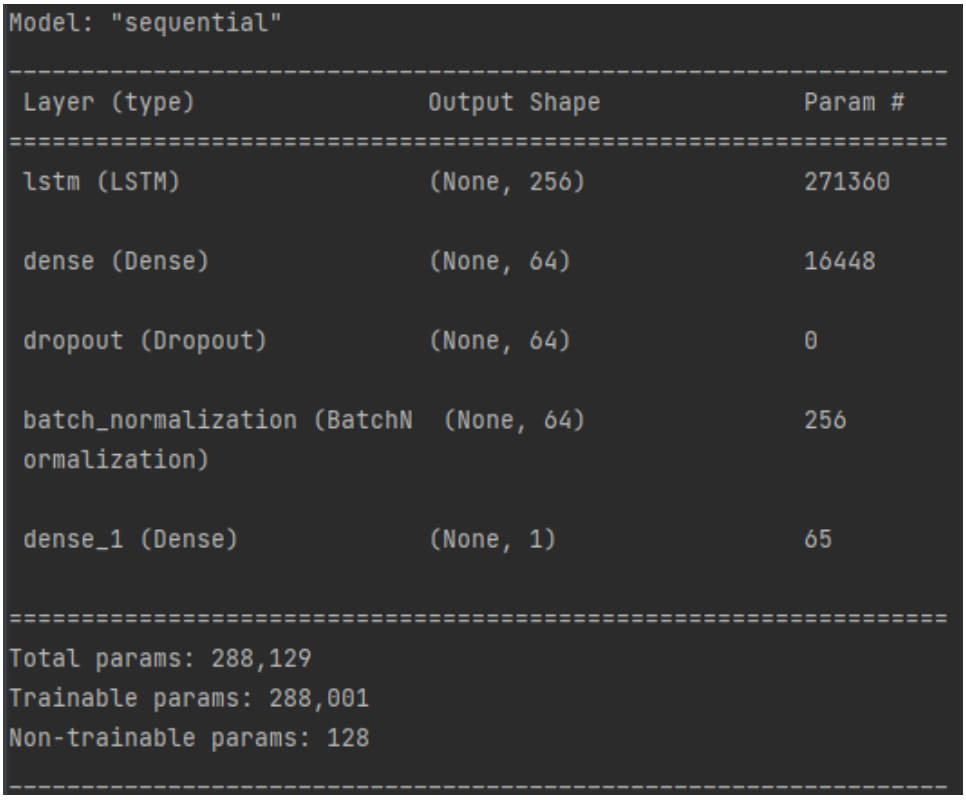


Figure 5: Multivariate Model Summary

## 3.2 Univariate Model

For building this model, it differs from the multivariate model in reading all the features. In the data preparation of the univariate model, we only read the data of the pollution feature and preprocess it. We also divided dataset into two parts. As shown in the Figure 6, the parameters in LSTM layer is not the same as multivariate model's.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 256)               264192

 dense (Dense)               (None, 64)                16448

 dropout (Dropout)           (None, 64)                0

 batch_normalization (BatchN  (None, 64)               256
 ormalization)

 dense_1 (Dense)             (None, 1)                 65

=================================================================
Total params: 280,961
Trainable params: 280,833
Non-trainable params: 128
_________________________________________________________________
```

Figure 6: Univariate Model Summary

# 4 Numerical Results

## 4.1 Multivariate Model

Here, we start with the results of the multivariate model. The training loss v.s. the test loss plot for the multivariate model is shown in Figure 7.

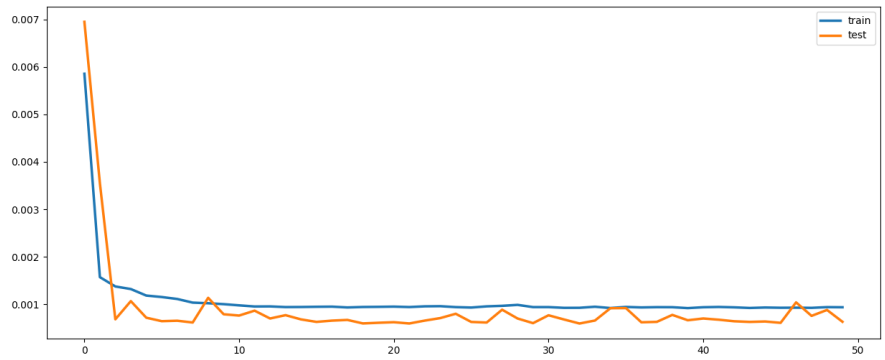


Figure 7: Training loss vs test loss for Multivariate Model

It shows that the model is learning and after about 7 epochs, the test loss is not stable. The reason may be that the dataset is not large enough.

As shown in Figure 8, it is the test result of multivariate model.

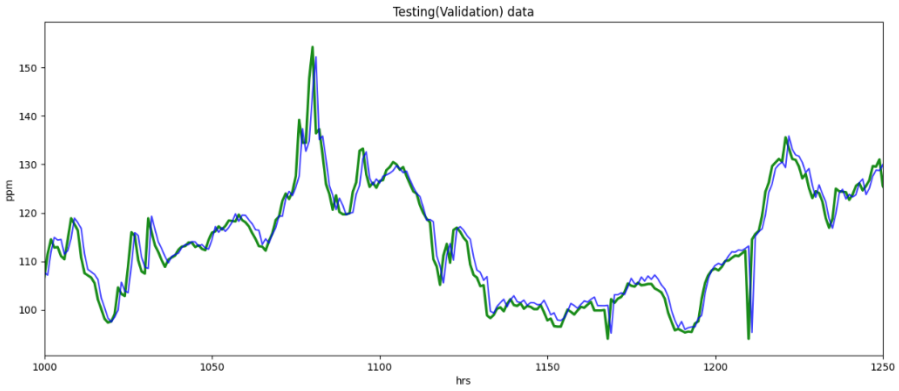


Figure 8: Test Result for Multivariate Model

It shows that the multivariate model predicts the air quality well. Most of the time, the predicted data is not much different from the actual data. By calculating the Root Mean Squared Error (RMSE) is 2.32079.

## 4.2 Univariate Model

The training loss v.s. the test loss plot for the univariate model is shown in Figure 9.

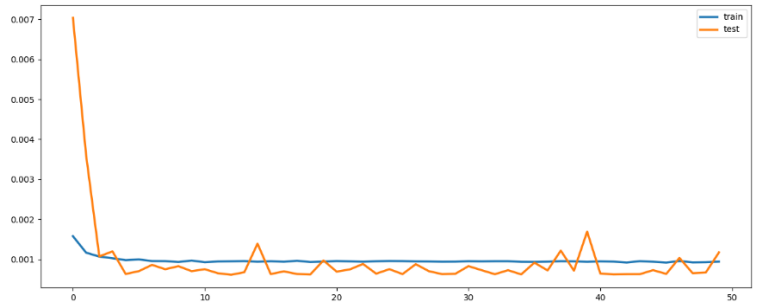


Figure 9: Training loss vs test loss for Univariate Model

In univariate model, it shows that the model is learning and after about 5 epochs, the test loss is not stable. The reason we think is same as the multivariate model. The dataset is not large enough.

As shown in Figure 10, it is the test result of univariate model.

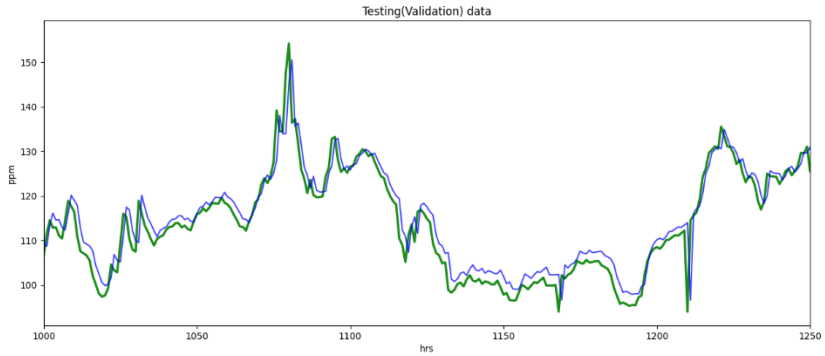


Figure 10: Test result for Univariate Model

It shows that the univariate model predicts the air quality well but not as well as multivariate model. Although the trend of the predicted data is basically consistent with the trend of the actual data, the difference in the middle is still relatively large compared with the multivariate model. Through calculation, the RMSE value can be obtained as 3.15849.

# 5 Conclusion

In this project, we used LSTM model to predict the air quality situation. We mainly compared the difference between multivariate model and univariate model. In addition, transform existing time series data into a supervised learning problem. By comparing the RMSE of the multivariate model and the RMSE of the univariate model, we can conclude that the performance of the multivariate model is better than that of the univariate model. A multivariate model is able to capture more information about the data and make more accurate predictions.

Future work could be targeted at using a deeper and wider LSTM network, using a learning rate schedule, and using a different optimization algorithm. Moreover, we also can use GRU to compare the performance between GRU and LSTM.

# Reference

[1] Tsai, Zeng, Y.-R., & Chang, Y.-S. (2018). Air Pollution Forecasting Using RNN with LSTM. 2018 16TH IEEE INT CONF ON DEPENDABLE, AUTONOM AND SECURE COMP, 16TH IEEE INT CONF ON PERVAS INTELLIGENCE AND COMP, 4TH IEEE INT CONF ON BIG DATA INTELLIGENCE AND COMP, 3RD IEEE CYBER SCI AND TECHNOL CONGRESS (DASC/PICOM/DATACOM/CYBERSCITECH), 1074–1079. https://doi.org/10.1109/DASC/PiCom/DataCom/CyberSciTec.2018.00178

[2] Wikipedia, “Long short-term memory,” 2022. [Online] Available at: https://en.wikipedia.org/wiki/Long_short-term_memory, Accessed: 2022.

[3] “A Complete Guide to LSTM Architecture and its Use in Text Classification,” 2022. [Online] Available at: https://analyticsindiamag.com/a-complete-guide-to-lstm-architecture-and-its-use-in-text-classification/, Accessed: 2022.

[4] “sklearn.preprocessing.LabelEncoder.” [Online] Available at: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html, Accessed: 2022.

[5] “How to Convert a Time Series to a Supervised Learning Problem in Python,” 2022. [Online] Available at: https://machinelearningmastery.com/convert-time-series-supervised-learning-problem-python/, Accessed: 2022.

[6] “tf.keras.layers.LSTM” [Online] Available at: https://www.tensorflow.org/api_docs/python/tf/keras/layers/LSTM, Accessed: 2022.

# Appendix

## Code for Multivariate Model

```
import warnings
warnings.filterwarnings('ignore')

import tensorflow as tf
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import *
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
from sklearn.metrics import mean_squared_error as mse
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.losses import MeanSquaredError
from tensorflow.keras.metrics import RootMeanSquaredError
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.layers import LSTM, Dense, Dropout,
BatchNormalization
from tensorflow.keras.models import Sequential

print('Imports Complete')
# ---------------------------- read data --------------------------
--
df = pd.read_csv('LSTM-Multivariate_pollution.csv')
# print(df.head())
# print(df.shape)
# print(df.info())
# print(df.describe())
# ---------------------------- data pre-processing -----------------
------------
col_names = ['pollution', 'dew', 'temperature', 'pressure',
'wind_dir', 'wind_speed', 'snow', 'rain']
df.drop_duplicates(inplace=True)
df.dropna(inplace=True)
# print(df.shape)

df.index = pd.to_datetime(df['date'], format='%Y.%m.%d %H:%M:%S')
# print(df.head())
df.drop('date', axis=1, inplace=True)
df.columns = col_names
features = df.values
# print(df.head())

columns = [0, 1, 2, 3, 5, 6, 7]
plt.figure(figsize=(20,14))
for i, c in enumerate(columns, 1):
    plt.subplot(len(columns), 1, i)
    plt.plot(features[:, c])
    plt.title(df.columns[c], y=0.75, loc="right")
# plt.show()

plt.matshow(df.corr())
plt.xticks(range(len(col_names)), col_names)
plt.yticks(range(len(col_names)), col_names)
```

```
plt.colorbar()
# plt.show()

# print(df["wind_dir"].unique())
wind_dir_encoder = LabelEncoder()
df["wind_dir"] = wind_dir_encoder.fit_transform(df["wind_dir"])
df["wind_dir"] = df["wind_dir"].astype(float)
# print(df.head())

values = df.values
target = df['pollution']
# plt.plot(target)
# plt.show()

# How to Convert a Time Series to a Supervised Learning Problem in
Python
# https://machinelearningmastery.com/convert-time-series-supervised-
learning-problem-python/
# The function is defined with default parameters so that if you call
it with just your data, it will construct a DataFrame with t-1
# as X and t as y
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 = pd.concat(cols, axis=1)
    agg.columns = names
    # drop rows with NaN values
    if dropnan:
        agg.dropna(inplace=True)

    return agg


scaler = MinMaxScaler(feature_range=(0, 1))
scaled_dataset = scaler.fit_transform(values)
# print(scaled_dataset.shape[1])
reframed = series_to_supervised(scaled_dataset, 1, 1)
# print(reframed.head())
# print(reframed.shape)

reframed.drop(reframed.columns[[9, 10, 11, 12, 13, 14, 15]], axis=1,
inplace=True)
# print(reframed.head())
```

```
# print(reframed.shape)

values = reframed.values
# First 4 years data
n_train_hours = 365 * 24 * 4

train = values[:n_train_hours, :]
test = values[n_train_hours:, :]

# split into input and outputs
train_X, train_y = train[:, :-1], train[:, -1]
test_X, test_y = test[:, :-1], test[:, -1]

# reshape input to be 3D :- (no.of samples, no.of timesteps, no.of
features)
train_X = train_X.reshape((train_X.shape[0], 1, train_X.shape[1]))
test_X = test_X.reshape((test_X.shape[0], 1, test_X.shape[1]))
# print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)


#----------------------------- Design Model -------------------------
---
model = Sequential()
model.add(LSTM(256, input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dense(64))
model.add(Dropout(0.25))
model.add(BatchNormalization())
model.add(Dense(1))

model.summary()

model.compile(loss='mse', optimizer='adam')

history = model.fit(train_X, train_y, epochs=50, batch_size=128,
validation_data=(test_X, test_y))

plt.figure(figsize=(15,6))
plt.plot(history.history['loss'], label='train', linewidth = 2.5)
plt.plot(history.history['val_loss'], label='test',  linewidth = 2.5)
plt.legend()
plt.show()

# ---------------------------- Pridict ------------------------------
--
prediction = model.predict(test_X)
prediction = prediction.ravel()

ture_test = test[:, 8]

poll = np.array(df["pollution"])

meanop = poll.mean()
stdop = poll.std()

ture_test = ture_test * stdop + meanop
prediction = prediction * stdop + meanop

plt.figure(figsize=(15, 6))
plt.xlim([1000, 1250])
plt.ylabel("ppm")
plt.xlabel("hrs")
```

```
plt.plot(ture_test, c="g", alpha=0.90, linewidth=2.5)
plt.plot(prediction, c="b", alpha=0.75)
plt.title("Testing(Validation) data")
plt.show()

rmse = np.sqrt(mse(ture_test, prediction))
print("Test(Validation) RMSE =", rmse)
```

## Code for Univariate Model

```
import warnings
warnings.filterwarnings('ignore')

import tensorflow as tf
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import *
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
from sklearn.metrics import mean_squared_error as mse
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.losses import MeanSquaredError
from tensorflow.keras.metrics import RootMeanSquaredError
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.layers import LSTM, Dense, Dropout,
BatchNormalization
from tensorflow.keras.models import Sequential

print('Imports Complete')
# ---------------------------- read data ---------------------------
--
df = pd.read_csv('LSTM-Multivariate_pollution.csv')
# print(df.head())
# print(df.shape)
# print(df.info())
# print(df.describe())
# ---------------------------- data pre-processing -----------------
------------
col_names = ['pollution', 'dew', 'temperature', 'pressure',
'wind_dir', 'wind_speed', 'snow', 'rain']
df.drop_duplicates(inplace=True)
df.dropna(inplace=True)
# print(df.shape)

df.index = pd.to_datetime(df['date'], format='%Y.%m.%d %H:%M:%S')
# print(df.head())
df.drop('date', axis=1, inplace=True)
df.columns = col_names
df.drop(['dew', 'temperature', 'pressure', 'wind_dir', 'wind_speed',
'snow', 'rain'], axis=1, inplace=True)
values = df.values
features = df.values
# print(df.head())

plt.figure(figsize=(20, 14))
plt.plot(df['pollution'])
plt.title('pollution', y=0.75, loc="right")
plt.show()

col_names = df.columns.tolist()
print(col_names)

# How to Convert a Time Series to a Supervised Learning Problem in
Python
# https://machinelearningmastery.com/convert-time-series-supervised-
learning-problem-python/
```

```
# The function is defined with default parameters so that if you call
it with just your data, it will construct a DataFrame with t-1
# as X and t as y
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 = pd.concat(cols, axis=1)
    agg.columns = names
    # drop rows with NaN values
    if dropnan:
        agg.dropna(inplace=True)

    return agg


scaler = MinMaxScaler(feature_range=(0, 1))
scaled_dataset = scaler.fit_transform(values)
# print(scaled_dataset.shape[1])
reframed = series_to_supervised(scaled_dataset, 1, 1)
print(reframed.head())
print(reframed.shape)

# print(reframed.head())
# print(reframed.shape)

values = reframed.values
# First 4 years data
n_train_hours = 365 * 24 * 4

train = values[:n_train_hours, :]
test = values[n_train_hours:, :]
# print(train)
# print(test)

# split into input and outputs
train_X, train_y = train[:, :-1], train[:, -1]
test_X, test_y = test[:, :-1], test[:, -1]

# reshape input to be 3D :- (no.of samples, no.of timesteps, no.of
features)
train_X = train_X.reshape((train_X.shape[0], 1, train_X.shape[1]))
test_X = test_X.reshape((test_X.shape[0], 1, test_X.shape[1]))
# print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)
```

```
#---------------------------- Design Model ------------------------
---
model = Sequential()
model.add(LSTM(256, input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dense(64))
model.add(Dropout(0.25))
model.add(BatchNormalization())
model.add(Dense(1))

model.summary()

model.compile(loss='mse', optimizer='adam')

history = model.fit(train_X, train_y, epochs=50, batch_size=128,
validation_data=(test_X, test_y))

plt.figure(figsize=(15,6))
plt.plot(history.history['loss'], label='train', linewidth = 2.5)
plt.plot(history.history['val_loss'], label='test',  linewidth = 2.5)
plt.legend()
plt.show()

# ---------------------------- Pridict -----------------------------
--
prediction = model.predict(test_X)
prediction = prediction.ravel()

ture_test = test[:, 1]

poll = np.array(df["pollution"])

meanop = poll.mean()
stdop = poll.std()

ture_test = ture_test * stdop + meanop
prediction = prediction * stdop + meanop

plt.figure(figsize=(15, 6))
plt.xlim([1000, 1250])
plt.ylabel("ppm")
plt.xlabel("hrs")
plt.plot(ture_test, c="g", alpha=0.90, linewidth=2.5)
plt.plot(prediction, c="b", alpha=0.75)
plt.title("Testing(Validation) data")
plt.show()

rmse = np.sqrt(mse(ture_test, prediction))
print("Test(Validation) RMSE =", rmse)
```