```
#importing the necessary libraries and dependencies(pandas,numpy,seaborn,sklearn,tensorflow,keras,sklearn,matplotlib)
import pandas as pd
import numpy as np
import seaborn as sns;
from sklearn import preprocessing
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from numpy import array
from keras.models import Sequential
from keras.layers import LSTM, Dense, Dropout
from keras.callbacks import ReduceLROnPlateau
from keras.optimizers import Adam
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.metrics import r2_score #metrices
from statsmodels.graphics.tsaplots import plot_acf #autocorrelation


from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

New Section

```
df = pd.read_csv("/content/drive/MyDrive/modified_data_delhi.csv")
print(df)
```

```
            No  year  month  day  hour   PM2.5    PM10    SO2    NO2     CO  \
0            1  2015      1    1     5  454.58  976.99   6.17  37.41  26.19
1            2  2015      1    1     6  454.58  862.23   7.33  32.06  11.04
2            3  2015      1    1     7  454.58  731.83   8.00  30.97   8.39
3            4  2015      1    1     8  454.58  725.74   9.00  26.83   7.89
4            5  2015      1    1     9  454.58  656.93   6.67  26.22   4.25
...        ...   ...    ...  ...   ...     ...     ...    ...    ...    ...
48183    48184  2020      6   30    20   43.19  123.26  10.64  22.28   0.92
48184    48185  2020      6   30    21   45.30  125.22  10.59  22.91   1.04
48185    48186  2020      6   30    22   49.04  129.39  10.20  20.95   1.00
48186    48187  2020      6   30    23   50.51  128.86  10.98  20.40   0.94
48187    48188  2020      7    1     0   54.01  128.66  11.15  21.05   0.97

             O3   TEMP   PRES   DEWP  RAIN      wd  WSPM
0         16.00   8.41  99.26  -0.89   0.0  341.24  4.76
1         12.33   8.07  99.31  -1.13   0.0  338.96  4.61
2         58.67   9.61  99.38  -0.08   0.0  336.68  4.32
3         13.42  13.13  99.47  -1.18   0.0  334.49  3.23
4         55.30  15.64  99.52  -0.95   0.0  333.63  2.04
...         ...    ...    ...    ...   ...     ...   ...
48183     44.13  34.61  97.60  22.33   0.0  118.47  5.57
48184     39.62  34.06  97.65  22.37   0.0  121.46  6.06
48185     38.95  33.46  97.66  22.39   0.0  125.41  6.43
48186     34.53  32.87  97.66  22.40   0.0  128.50  6.54
48187     29.73  32.27  97.66  22.41   0.0  129.76  6.11

[48188 rows x 17 columns]
```

```
#number of rows and columns in the dataset
print(df.shape)
```

```
(48188, 17)
```

```
#checking how many null values are in each column and performing summation
df.isnull().sum()
```

```
No          0
year        0
month       0
day         0
hour        0
PM2.5     375
PM10     2418
SO2      2852
NO2       330
CO        364
O3       2201
TEMP        0
PRES        0
```

```
DEWP        0
RAIN        0
wd          0
WSPM        0
dtype: int64
```

Just doing `df.dropna()` drops all the NaN values only for the current execution of the cell. If you do the above `df.isnull().sum()` now, you can see that null values still persists. You can solve this by assigning the obtained output of `df.dopna()` to the variable `df` which stores our data (dataframe)

```
# dropping all the rows with NaN values(having no number)
df = df.dropna()
print(df.shape)#(48188, 17)initial number of rows and colmuns
```

```
(44587, 17)
```

```
#defining training and testing data
x_train = df[:31210] #70% data for x_train
y_train = x_train['PM2.5']
x_test = df[31210:44587] #30% data x_test
y_test = x_test['PM2.5']
#print(y_test)
```

```
print(x_train.head())#returns first 5 rows of data
print(x_train.shape)
print(y_train.head())
print(y_train)
```

```
   No  year  month  day  hour   PM2.5    PM10   SO2    NO2     CO     O3  \
0   1  2015      1    1     5  454.58  976.99  6.17  37.41  26.19  16.00
1   2  2015      1    1     6  454.58  862.23  7.33  32.06  11.04  12.33
2   3  2015      1    1     7  454.58  731.83  8.00  30.97   8.39  58.67
3   4  2015      1    1     8  454.58  725.74  9.00  26.83   7.89  13.42
4   5  2015      1    1     9  454.58  656.93  6.67  26.22   4.25  55.30

    TEMP   PRES  DEWP  RAIN      wd  WSPM
0   8.41  99.26 -0.89   0.0  341.24  4.76
1   8.07  99.31 -1.13   0.0  338.96  4.61
2   9.61  99.38 -0.08   0.0  336.68  4.32
3  13.13  99.47 -1.18   0.0  334.49  3.23
4  15.64  99.52 -0.95   0.0  333.63  2.04
(31210, 17)
0    454.58
1    454.58
2    454.58
3    454.58
4    454.58
Name: PM2.5, dtype: float64
0        454.58
1        454.58
2        454.58
3        454.58
4        454.58
          ...
34806    276.26
34807    225.68
34808    184.17
34809    159.84
34810    147.54
Name: PM2.5, Length: 31210, dtype: float64
```

There are many pollutants. Let's first try to predict PM2.5 concentration values. Let the years 2015 and 2020 be the testing set. As you can see below, these 5 years account for 20% of the data (test set) Here we r going

```
df.loc[31210:44587].count() / df.shape[0] * 100 #30% of data x test(bottom)
```

```
No       30.004261
year     30.004261
month    30.004261
day      30.004261
hour     30.004261
PM2.5    30.004261
PM10     30.004261
SO2      30.004261
NO2      30.004261
CO       30.004261
O3       30.004261
TEMP     30.004261
PRES     30.004261
DEWP     30.004261
```

```
RAIN     30.004261
wd       30.004261
WSPM     30.004261
dtype: float64
```

Double-click (or enter) to edit

```
#Normalizing training data
train_norm = x_train['PM2.5']

#converted into array as all the methods available are for arrays and not lists
train_norm_arr = np.asarray(train_norm)
train_norm = np.reshape(train_norm_arr, (-1, 1))

#Scaling all values between 0 and 1 so that large values don't just dominate
scaler = MinMaxScaler(feature_range=(0, 1))
train_norm = scaler.fit_transform(train_norm)
for i in range(5):
    print(train_norm[i])
```

```
[0.51141949]
[0.51141949]
[0.51141949]
[0.51141949]
[0.51141949]
```

Even after normalization and scaing, null values are possible (many people disregard this). Let's check if any null values are present.

```
count = 0
for i in range(len(train_norm)):
    if train_norm[i] == 0:
        count = count +1
print('Number of null values in train_norm = ', count)
```

```
Number of null values in train_norm =  1
```

```
#removing null values
train_norm = train_norm[train_norm!=0]
```

```
#Normalizing testing data and repeating the same process as done for training data
test_norm = x_test['PM2.5']
test_norm_arr = np.asarray(test_norm)
test_norm = np.reshape(test_norm_arr, (-1, 1))
scaler = MinMaxScaler(feature_range=(0, 1))
test_norm = scaler.fit_transform(test_norm)
for i in range(5):
    print(test_norm[i])
```

```
[0.15754158]
[0.15711328]
[0.17049754]
[0.21908535]
[0.30325743]
```

```
count = 0
for i in range(len(test_norm)):
    if test_norm[i] == 0:
        count = count + 1
print('Number of null values in test_norm = ', count)
```

```
Number of null values in test_norm =  1
```

```
#removing null values
test_norm = test_norm[test_norm != 0]
```

```
print(train_norm.shape)#return number of rows and columns from given data
print(test_norm.shape)
```

```
(31209,)
(13376,)
```

Since this is a time series data, we should be predicting the values after looking at a set of values rather than just a single value like we usually do. This takes into account the correlation between the data points and the timestamps. Because the neighbours should be considered for how the values change over time. Let's define a function to do this.

The below function called split_sequence splits the sequence into sets of n values. This n is given as n_steps (step_size). For example, if n=3, we split the sequence in groups of 3. We create 2 empty lists and append the split sequences.

```
def split_sequence(sequence, n_steps):
    X, y = list(), list()
    for i in range(len(sequence)):
        # find the end of this pattern
        end_ix = i + n_steps
        # check if we are beyond the sequence
        if end_ix > len(sequence)-1:
            break
        # gather input and output parts of the pattern
        seq_x, seq_y = sequence[i:end_ix], sequence[end_ix]
        X.append(seq_x)
        y.append(seq_y)
    return array(X),array(y)
```

Here the number of features = 1 as we will be predicting a single value. Let's reshape the split sequences into the format of number of rows, number of columns. (shape[0], shape[1]). In the output, we can see that groups of 3 since n_steps = 3 have been obtained.

```
n_steps = 3
X_split_train, y_split_train = split_sequence(train_norm, n_steps)
#for i in range(len(X_split_train)):
    #print(X_split_train[i], y_split_train[i])
n_features = 1
X_split_train = X_split_train.reshape((X_split_train.shape[0], X_split_train.shape[1], n_features))
for i in range(5):
    print(X_split_train)
```

```
 [[[0.51141949]
   [0.51141949]
   [0.51141949]]

  [[0.51141949]
   [0.51141949]
   [0.51141949]]

  [[0.51141949]
   [0.51141949]
   [0.51141949]]

  ...

  [[0.32219199]
   [0.30758327]
   [0.24976567]]

  [[0.30758327]
   [0.24976567]
   [0.2023159 ]]

  [[0.24976567]
   [0.2023159 ]
   [0.17450447]]]
 [[[0.51141949]
   [0.51141949]
   [0.51141949]]

  [[0.51141949]
   [0.51141949]
   [0.51141949]]

  [[0.51141949]
   [0.51141949]
   [0.51141949]]

  ...

  [[0.32219199]
   [0.30758327]
   [0.24976567]]

  [[0.30758327]
   [0.24976567]
   [0.2023159 ]]

  [[0.24976567]
   [0.2023159 ]
   [0.17450447]]]
 [[[0.51141949]
   [0.51141949]
   [0.51141949]]
```

```
[[0.51141949]
 [0.51141949]
 [0.51141949]]
```

You can see below that, we predict the value for the first 3 values, then consider that output as one of the 3 values in the next set. For example, we preedict 0.1 first, then we take that 0.1 as input in the second set and so on.

```
X_split_test, y_split_test = split_sequence(test_norm, n_steps)
for i in range(5):
    print(X_split_test[i], y_split_test[i])
n_features = 1
X_split_test = X_split_test.reshape((X_split_test.shape[0], X_split_test.shape[1], n_features))
```

```
[0.15754158 0.15711328 0.17049754] 0.21908534989411568
[0.15711328 0.17049754 0.21908535] 0.3032574297475432
[0.17049754 0.21908535 0.30325743] 0.41904014086182684
[0.21908535 0.30325743 0.41904014] 0.44580864682228083
[0.30325743 0.41904014 0.44580865] 0.46179836771599203
```

Let's define our neural network using LSTM (Long Short-Term Memory). We will add 50 nodes in our first layer with a ReLU (Rectified Linear Unit) activation. The shape of the nodes will be the step size and the number of features. We will then add a dense layer with one node for the output.

To optimize our neural network, we can try out different optimizers such as Stochastic Gradient Descent (SGD), Adam, AdaBoost, RMSProp, among others. We can set the learning rate (lr), decay (by how much to decay the learning rate), momentum (how much should the gradient descent be accelerated to dampen oscillations), and nesterov (whether to use nesterov momentum, which has stronger convergence for convex functions).

Finally, we will compile our model using MSE (Mean Squared Error) as our loss function.

```
# Define model
model = Sequential()
model.add(LSTM(128, activation='relu', input_shape=(n_steps, n_features), return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(64, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1))

# Define optimizer
adam = Adam(lr=0.001, decay=1e-6)

# Compile model
model.compile(optimizer=adam, loss='mse', metrics=['mae', 'accuracy'])

# Add learning rate reduction callback
lr_reducer = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3, verbose=1)
```

```
/usr/local/lib/python3.9/dist-packages/keras/optimizers/legacy/adam.py:117: UserWarning: The `lr` argument is deprecated, use `lea
  super().__init__(name, **kwargs)
```

```
# Fit model
hist = model.fit(X_split_train, y_split_train, validation_data=(X_split_test, y_split_test), epochs=50, batch_size=64, callbacks=[lr_re
```

```
Epoch 1/50
488/488 [==============================] - 15s 22ms/step - loss: 0.0035 - mae: 0.0371 - accuracy: 3.2045e-05 - val_loss: 8.2157
Epoch 2/50
488/488 [==============================] - 9s 19ms/step - loss: 0.0018 - mae: 0.0257 - accuracy: 3.2045e-05 - val_loss: 4.6382e
Epoch 3/50
488/488 [==============================] - 9s 19ms/step - loss: 0.0016 - mae: 0.0237 - accuracy: 3.2045e-05 - val_loss: 4.1880e
Epoch 4/50
488/488 [==============================] - 10s 21ms/step - loss: 0.0016 - mae: 0.0231 - accuracy: 3.2045e-05 - val_loss: 3.8176
Epoch 5/50
488/488 [==============================] - ETA: 0s - loss: 0.0016 - mae: 0.0229 - accuracy: 3.2045e-05
Epoch 5: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
488/488 [==============================] - 8s 16ms/step - loss: 0.0016 - mae: 0.0229 - accuracy: 3.2045e-05 - val_loss: 3.6438e
Epoch 6/50
488/488 [==============================] - 10s 21ms/step - loss: 0.0016 - mae: 0.0224 - accuracy: 3.2045e-05 - val_loss: 4.1041
Epoch 7/50
488/488 [==============================] - 12s 24ms/step - loss: 0.0015 - mae: 0.0225 - accuracy: 3.2045e-05 - val_loss: 3.9186
Epoch 8/50
488/488 [==============================] - 10s 21ms/step - loss: 0.0015 - mae: 0.0224 - accuracy: 3.2045e-05 - val_loss: 3.0869
Epoch 9/50
488/488 [==============================] - 9s 18ms/step - loss: 0.0015 - mae: 0.0224 - accuracy: 3.2045e-05 - val_loss: 4.6810e
Epoch 10/50
488/488 [==============================] - 10s 21ms/step - loss: 0.0015 - mae: 0.0223 - accuracy: 3.2045e-05 - val_loss: 3.4190
Epoch 11/50
487/488 [============================>.] - ETA: 0s - loss: 0.0015 - mae: 0.0222 - accuracy: 3.2084e-05
Epoch 11: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
```

```
488/488 [==============================] - 8s 17ms/step - loss: 0.0015 - mae: 0.0222 - accuracy: 3.2045e-05 - val_loss: 3.5225e
Epoch 12/50
488/488 [==============================] - 9s 19ms/step - loss: 0.0015 - mae: 0.0221 - accuracy: 3.2045e-05 - val_loss: 3.1660e
Epoch 13/50
488/488 [==============================] - 10s 21ms/step - loss: 0.0015 - mae: 0.0219 - accuracy: 3.2045e-05 - val_loss: 3.3078
Epoch 14/50
488/488 [==============================] - ETA: 0s - loss: 0.0015 - mae: 0.0219 - accuracy: 3.2045e-05
Epoch 14: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.
488/488 [==============================] - 8s 16ms/step - loss: 0.0015 - mae: 0.0219 - accuracy: 3.2045e-05 - val_loss: 3.8172e
Epoch 15/50
488/488 [==============================] - 11s 22ms/step - loss: 0.0015 - mae: 0.0218 - accuracy: 3.2045e-05 - val_loss: 3.0909
Epoch 16/50
488/488 [==============================] - 10s 21ms/step - loss: 0.0015 - mae: 0.0217 - accuracy: 3.2045e-05 - val_loss: 3.4178
Epoch 17/50
486/488 [============================>.] - ETA: 0s - loss: 0.0015 - mae: 0.0217 - accuracy: 3.2150e-05
Epoch 17: ReduceLROnPlateau reducing learning rate to 6.25000029685907e-05.
488/488 [==============================] - 8s 17ms/step - loss: 0.0015 - mae: 0.0217 - accuracy: 3.2045e-05 - val_loss: 3.0924e
Epoch 18/50
488/488 [==============================] - 10s 20ms/step - loss: 0.0015 - mae: 0.0217 - accuracy: 3.2045e-05 - val_loss: 3.0193
Epoch 19/50
488/488 [==============================] - 9s 18ms/step - loss: 0.0015 - mae: 0.0217 - accuracy: 3.2045e-05 - val_loss: 3.0061e
Epoch 20/50
487/488 [============================>.] - ETA: 0s - loss: 0.0015 - mae: 0.0215 - accuracy: 3.2084e-05
Epoch 20: ReduceLROnPlateau reducing learning rate to 3.125000148429535e-05.
488/488 [==============================] - 9s 18ms/step - loss: 0.0015 - mae: 0.0215 - accuracy: 3.2045e-05 - val_loss: 3.0169e
Epoch 21/50
488/488 [==============================] - 10s 20ms/step - loss: 0.0015 - mae: 0.0216 - accuracy: 3.2045e-05 - val_loss: 3.0167
Epoch 22/50
488/488 [==============================] - 8s 17ms/step - loss: 0.0015 - mae: 0.0216 - accuracy: 3.2045e-05 - val_loss: 3.0939e
Epoch 23/50
485/488 [============================>.] - ETA: 0s - loss: 0.0015 - mae: 0.0216 - accuracy: 3.2216e-05
Epoch 23: ReduceLROnPlateau reducing learning rate to 1.5625000742147677e-05
```

```
# fit model
#hist = model.fit(X_split_train, y_split_train, validation_data=(X_split_test, y_split_test), epochs=10, verbose = 1)
```

```
print(hist.history.keys())
```

```
dict_keys(['loss', 'mae', 'accuracy', 'val_loss', 'val_mae', 'val_accuracy', 'lr'])
```

```
import xgboost as xgb

# Get predictions from the LSTM model on the test data
yhat = model.predict(X_split_test)

# Reshape yhat to match the shape of X_split_test
yhat_reshaped = np.reshape(yhat, (yhat.shape[0], 1, yhat.shape[1]))
yhat_reshaped = np.repeat(yhat_reshaped, n_steps, axis=1)

# Reshape X_split_test to match the shape of yhat
X_test_reshaped = np.reshape(X_split_test, (X_split_test.shape[0], X_split_test.shape[1], 1))

# Combine the LSTM predictions with other features in your test data
X_test_combined = np.concatenate([X_test_reshaped, yhat_reshaped], axis=2)

# Train an XGBoosting tree model on the combined data
xgb_model = xgb.XGBRegressor()
xgb_model.fit(np.reshape(X_test_combined, (X_test_combined.shape[0], -1)), np.reshape(y_split_test, (y_split_test.shape[0],)))

# Make final predictions using the Integrated dual model
y_pred_combined = np.concatenate([X_test_reshaped, yhat_reshaped], axis=2)
y_pred_final = xgb_model.predict(np.reshape(y_pred_combined, (y_pred_combined.shape[0], -1)))
```

```
418/418 [==============================] - 2s 4ms/step
```

```
#y_split_test, yhat
# calculate MSE and RMSE
mse = mean_squared_error(y_split_test, y_pred_final)
rmse = np.sqrt(mse)
print('MSE: %.5f' % mse)
print('RMSE: %.5f' % rmse)

# calculate MAE
mae = mean_absolute_error(y_split_test, y_pred_final)
print('MAE: %.5f' % mae)

# calculate MAPE
mape = np.mean(np.abs((y_split_test - y_pred_final) / y_split_test)) * 100
print('MAPE: %.5f%%' % mape)

# calculate R-squared
```

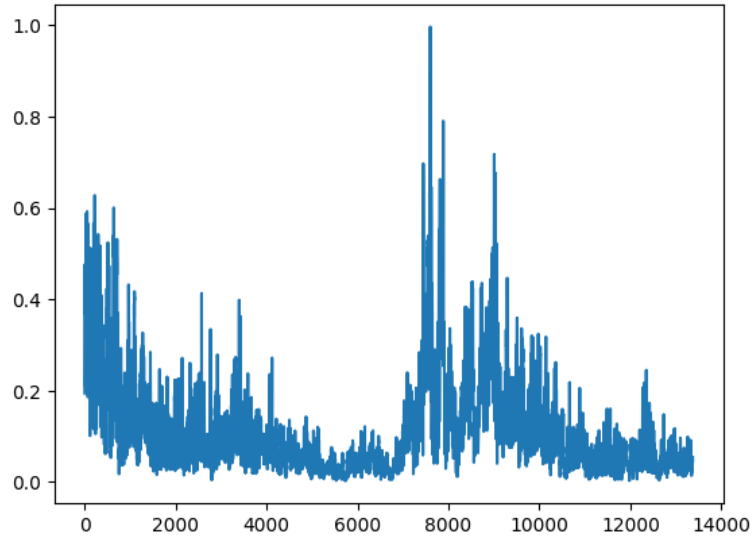
```
r2 = r2_score(y_split_test, y_pred_final)
print('R-squared: %.5f' % r2)

# calculate IA
ia = 1 - np.sum(np.abs(np.array(y_pred_final) - np.array(y_split_test))) / np.sum(y_split_test)
print('IA: %.5f' % ia)
```

```
MSE: 0.00005
RMSE: 0.00728
MAE: 0.00518
MAPE: 8.67098%
R-squared: 0.99564
IA: 0.95444
```

Below, I have plotted the actual true values (first plot) and preedicted values (second plot). One can visually see that the distribution is almost the same. This says that our predictions are very accurate.

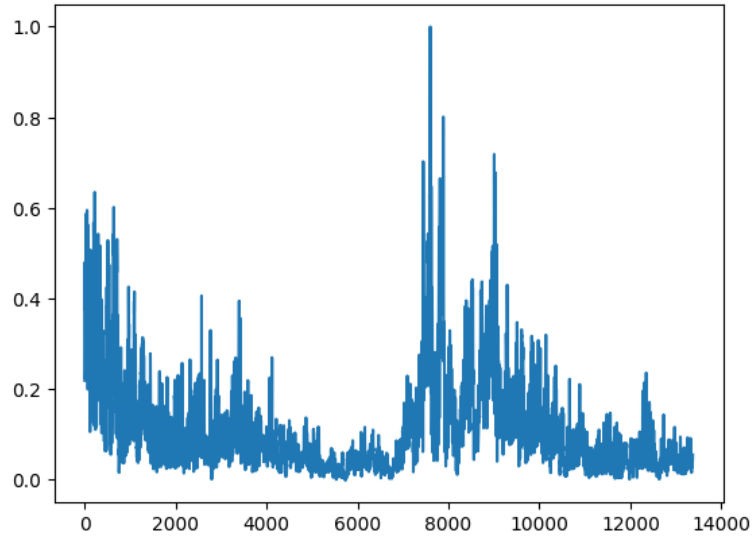
```
plt.plot(y_pred_final)
```

```
[<matplotlib.lines.Line2D at 0x7fdf9f4f9b20>]
```




```
plt.plot(y_split_test)
```

```
[<matplotlib.lines.Line2D at 0x7fdf9f45ca30>]
```

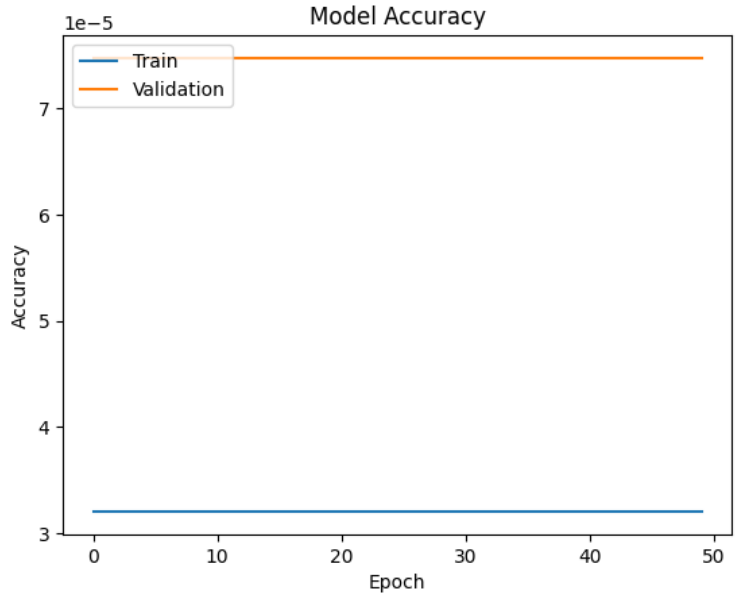



```
train_acc = model.evaluate(X_split_train, y_split_train, verbose=0)[1]
test_acc = model.evaluate(X_split_test, y_split_test, verbose=0)[1]
print('Train: %.5f, Test: %.5f' % (train_acc, test_acc))
```

```
Train: 0.01873, Test: 0.01101
```

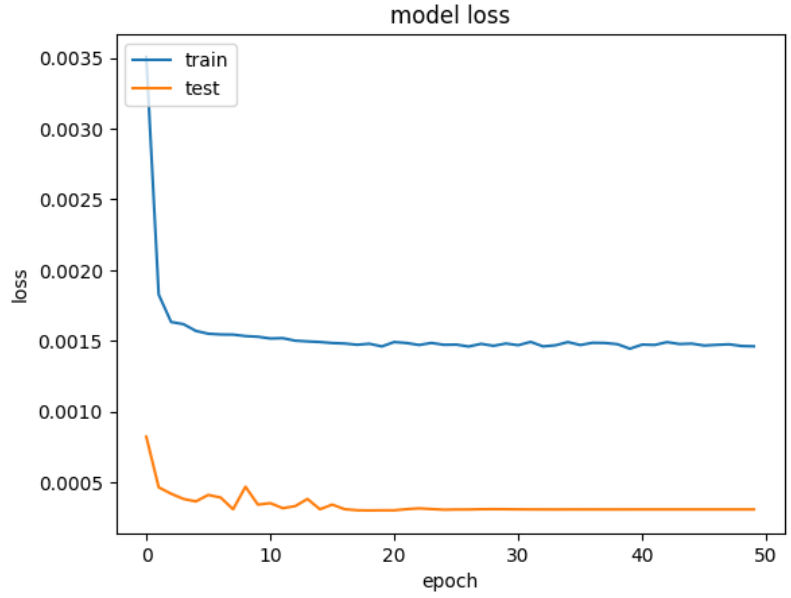
```
# Plot training and validation accuracy
plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.title('Model Accuracy')
```

```
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```



Above, accuracy increase a lot in the last few epochs. Below, the loss gradually decrease. These are positive signs that our model is doing very good.

```
# summarize history for loss
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



Until now, we just ran our model for prediction of a single pollutant. We have 6 pollutants in our dataset and can make predictions for all of them. So, I have made a function which can be used to predict the other pollutants rather than having to write the code again and again. I have commented the function calls. You can fork this kernel to uncomment and predit the other pollutants (Coz it would take up a lot of space and time).

```
def compute(var):
    train_norm = x_train[var]
    train_norm_arr = np.asarray(train_norm)
    train_norm = np.reshape(train_norm_arr, (-1, 1))
    scaler = MinMaxScaler(feature_range=(0, 1))
```

```
        train_norm = scaler.fit_transform(train_norm)
        train_norm = train_norm[train_norm != 0]

        test_norm = x_test[var]
        test_norm_arr = np.asarray(test_norm)
        test_norm = np.reshape(test_norm_arr, (-1, 1))
        scaler = MinMaxScaler(feature_range=(0, 1))
        test_norm = scaler.fit_transform(test_norm)
        test_norm = test_norm[test_norm != 0]

        X_split_train, y_split_train = split_sequence(train_norm, n_steps)
        X_split_train = X_split_train.reshape((X_split_train.shape[0], X_split_train.shape[1], n_features))

        X_split_test, y_split_test = split_sequence(test_norm, n_steps)
        X_split_test = X_split_test.reshape((X_split_test.shape[0], X_split_test.shape[1], n_features))

        # Fit model
        hist = model.fit(X_split_train, y_split_train, validation_data=(X_split_test, y_split_test), epochs=50, batch_size=64, callbacks=[l

        yhat = model.predict(X_split_test)

        #y_split_test, yhat
        # calculate MSE and RMSE
        mse = mean_squared_error(y_split_test, yhat)
        rmse = np.sqrt(mse)
        print('MSE: %.5f' % mse)
        print('RMSE: %.5f' % rmse)

        # calculate MAE
        mae = mean_absolute_error(y_split_test, yhat)
        print('MAE: %.5f' % mae)

        # calculate MAPE
        mape = np.mean(np.abs((y_split_test - yhat) / y_split_test)) * 100
        print('MAPE: %.5f%%' % mape)

        # calculate R-squared
        r2 = r2_score(y_split_test, yhat)
        print('R-squared: %.5f' % r2)

        # calculate IA
        ia = 1 - np.sum(np.abs(np.array(yhat) - np.array(y_split_test))) / np.sum(y_split_test)
        print('IA: %.5f' % ia)

        train_acc = model.evaluate(X_split_train, y_split_train, verbose=0)[1]
        test_acc = model.evaluate(X_split_test, y_split_test, verbose=0)[1]
        print('Train: %.5f, Test: %.5f' % (train_acc, test_acc))

        plt.plot(hist.history['accuracy'])
        plt.plot(hist.history['val_accuracy'])
        plt.title('model accuracy')
        plt.ylabel('accuracy')
        plt.xlabel('epoch')
        plt.legend(['train', 'test'], loc='upper left')
        plt.show()

        plt.plot(hist.history['loss'])
        plt.plot(hist.history['val_loss'])
        plt.title('model loss')
        plt.ylabel('loss')
        plt.xlabel('epoch')
        plt.legend(['train', 'test'], loc='upper left')
        plt.show()


    compute('PM10')
```

```
Epoch 1/50
488/488 [==============================] - 11s 23ms/step - loss: 0.0031 - mae: 0.0365 - accuracy: 3.
Epoch 2/50
488/488 [==============================] - 8s 17ms/step - loss: 0.0031 - mae: 0.0366 - accuracy: 3.2
Epoch 3/50
488/488 [==============================] - 11s 22ms/step - loss: 0.0031 - mae: 0.0364 - accuracy: 3.
Epoch 4/50
487/488 [============================>.] - ETA: 0s - loss: 0.0031 - mae: 0.0366 - accuracy: 3.2084e-
Epoch 4: ReduceLROnPlateau reducing learning rate to 1.525878978725359e-08.
488/488 [==============================] - 11s 22ms/step - loss: 0.0031 - mae: 0.0366 - accuracy: 3.
Epoch 5/50
488/488 [==============================] - 8s 17ms/step - loss: 0.0031 - mae: 0.0364 - accuracy: 3.2
Epoch 6/50
488/488 [==============================] - 10s 22ms/step - loss: 0.0031 - mae: 0.0366 - accuracy: 3.
Epoch 7/50
486/488 [============================>.] - ETA: 0s - loss: 0.0031 - mae: 0.0365 - accuracy: 3.2150e-
Epoch 7: ReduceLROnPlateau reducing learning rate to 7.629394893626795e-09.
488/488 [==============================] - 11s 23ms/step - loss: 0.0031 - mae: 0.0365 - accuracy: 3.
Epoch 8/50
488/488 [==============================] - 9s 18ms/step - loss: 0.0032 - mae: 0.0366 - accuracy: 3.2
Epoch 9/50
488/488 [==============================] - 12s 25ms/step - loss: 0.0031 - mae: 0.0366 - accuracy: 3.
Epoch 10/50
486/488 [============================>.] - ETA: 0s - loss: 0.0032 - mae: 0.0366 - accuracy: 3.2150e-
Epoch 10: ReduceLROnPlateau reducing learning rate to 3.814697446813398e-09.
488/488 [==============================] - 11s 22ms/step - loss: 0.0032 - mae: 0.0366 - accuracy: 3.
Epoch 11/50
488/488 [==============================] - 10s 20ms/step - loss: 0.0031 - mae: 0.0364 - accuracy: 3.
Epoch 12/50
488/488 [==============================] - 9s 19ms/step - loss: 0.0032 - mae: 0.0366 - accuracy: 3.2
Epoch 13/50
487/488 [============================>.] - ETA: 0s - loss: 0.0032 - mae: 0.0366 - accuracy: 3.2084e-
Epoch 13: ReduceLROnPlateau reducing learning rate to 1.907348723406699e-09.
488/488 [==============================] - 10s 22ms/step - loss: 0.0032 - mae: 0.0366 - accuracy: 3.
Epoch 14/50
488/488 [==============================] - 10s 20ms/step - loss: 0.0031 - mae: 0.0364 - accuracy: 3.
Epoch 15/50
488/488 [==============================] - 9s 18ms/step - loss: 0.0031 - mae: 0.0364 - accuracy: 3.2
Epoch 16/50
486/488 [============================>.] - ETA: 0s - loss: 0.0031 - mae: 0.0364 - accuracy: 3.2150e-
Epoch 16: ReduceLROnPlateau reducing learning rate to 9.536743617033494e-10.
488/488 [==============================] - 11s 22ms/step - loss: 0.0031 - mae: 0.0364 - accuracy: 3.
Epoch 17/50
488/488 [==============================] - 10s 22ms/step - loss: 0.0031 - mae: 0.0364 - accuracy: 3.
Epoch 18/50
488/488 [==============================] - 8s 17ms/step - loss: 0.0031 - mae: 0.0367 - accuracy: 3.2
Epoch 19/50
485/488 [============================>.] - ETA: 0s - loss: 0.0031 - mae: 0.0363 - accuracy: 3.2216e-
Epoch 19: ReduceLROnPlateau reducing learning rate to 4.768371808516747e-10.
488/488 [==============================] - 11s 22ms/step - loss: 0.0031 - mae: 0.0363 - accuracy: 3.
Epoch 20/50
488/488 [==============================] - 10s 21ms/step - loss: 0.0031 - mae: 0.0364 - accuracy: 3.
Epoch 21/50
488/488 [==============================] - 9s 17ms/step - loss: 0.0031 - mae: 0.0365 - accuracy: 3.2
Epoch 22/50
485/488 [============================>.] - ETA: 0s - loss: 0.0031 - mae: 0.0365 - accuracy: 3.2216e-
Epoch 22: ReduceLROnPlateau reducing learning rate to 2.3841859042583735e-10.
488/488 [==============================] - 11s 22ms/step - loss: 0.0031 - mae: 0.0365 - accuracy: 3.
Epoch 23/50
488/488 [==============================] - 11s 22ms/step - loss: 0.0031 - mae: 0.0366 - accuracy: 3.
Epoch 24/50
488/488 [==============================] - 8s 17ms/step - loss: 0.0031 - mae: 0.0366 - accuracy: 3.2
Epoch 25/50
485/488 [============================>.] - ETA: 0s - loss: 0.0031 - mae: 0.0366 - accuracy: 3.2216e-
Epoch 25: ReduceLROnPlateau reducing learning rate to 1.1920929521291868e-10.
488/488 [==============================] - 11s 22ms/step - loss: 0.0031 - mae: 0.0366 - accuracy: 3.
Epoch 26/50
488/488 [==============================] - 11s 23ms/step - loss: 0.0031 - mae: 0.0365 - accuracy: 3.
Epoch 27/50
488/488 [==============================] - 8s 17ms/step - loss: 0.0031 - mae: 0.0365 - accuracy: 3.2
```

```
compute('SO2')
```

```
Epoch 1/50
488/488 [==============================] - 11s 22ms/step - loss: 0.0010 - mae: 0.0181 - accuracy: 3.
Epoch 2/50
488/488 [==============================] - 11s 22ms/step - loss: 0.0010 - mae: 0.0181 - accuracy: 3.
Epoch 3/50
488/488 [==============================] - 8s 17ms/step - loss: 0.0010 - mae: 0.0180 - accuracy: 3.2
Epoch 4/50
485/488 [============================>.] - ETA: 0s - loss: 0.0010 - mae: 0.0181 - accuracy: 0.0000e+
Epoch 4: ReduceLROnPlateau reducing learning rate to 2.328306547127318e-13.
488/488 [==============================] - 11s 22ms/step - loss: 0.0010 - mae: 0.0181 - accuracy: 0.
Epoch 5/50
488/488 [==============================] - 10s 21ms/step - loss: 0.0010 - mae: 0.0181 - accuracy: 3.
Epoch 6/50
488/488 [==============================] - 8s 17ms/step - loss: 0.0010 - mae: 0.0180 - accuracy: 0.0
Epoch 7/50
488/488 [==============================] - ETA: 0s - loss: 0.0010 - mae: 0.0182 - accuracy: 3.2045e-
Epoch 7: ReduceLROnPlateau reducing learning rate to 1.164153273563659e-13.
488/488 [==============================] - 10s 21ms/step - loss: 0.0010 - mae: 0.0182 - accuracy: 3.
Epoch 8/50
488/488 [==============================] - 10s 21ms/step - loss: 0.0010 - mae: 0.0182 - accuracy: 3.
Epoch 9/50
488/488 [==============================] - 8s 17ms/step - loss: 0.0010 - mae: 0.0181 - accuracy: 0.0
Epoch 10/50
487/488 [============================>.] - ETA: 0s - loss: 0.0010 - mae: 0.0181 - accuracy: 3.2084e-
Epoch 10: ReduceLROnPlateau reducing learning rate to 5.820766367818295e-14.
488/488 [==============================] - 11s 22ms/step - loss: 0.0010 - mae: 0.0181 - accuracy: 3.
Epoch 11/50
488/488 [==============================] - 10s 21ms/step - loss: 0.0010 - mae: 0.0181 - accuracy: 0.
Epoch 12/50
488/488 [==============================] - 8s 17ms/step - loss: 0.0010 - mae: 0.0181 - accuracy: 3.2
Epoch 13/50
485/488 [============================>.] - ETA: 0s - loss: 0.0010 - mae: 0.0181 - accuracy: 0.0000e+
Epoch 13: ReduceLROnPlateau reducing learning rate to 2.9103831839091474e-14.
488/488 [==============================] - 11s 22ms/step - loss: 0.0010 - mae: 0.0181 - accuracy: 0.
Epoch 14/50
488/488 [==============================] - 11s 23ms/step - loss: 0.0010 - mae: 0.0181 - accuracy: 3.
Epoch 15/50
488/488 [==============================] - 9s 17ms/step - loss: 0.0010 - mae: 0.0182 - accuracy: 3.2
Epoch 16/50
488/488 [==============================] - ETA: 0s - loss: 0.0010 - mae: 0.0181 - accuracy: 0.0000e+
Epoch 16: ReduceLROnPlateau reducing learning rate to 1.4551915919545737e-14.
488/488 [==============================] - 10s 21ms/step - loss: 0.0010 - mae: 0.0181 - accuracy: 0.
Epoch 17/50
488/488 [==============================] - 10s 21ms/step - loss: 0.0010 - mae: 0.0181 - accuracy: 3.
Epoch 18/50
488/488 [==============================] - 8s 17ms/step - loss: 0.0010 - mae: 0.0182 - accuracy: 3.2
Epoch 19/50
485/488 [============================>.] - ETA: 0s - loss: 0.0010 - mae: 0.0181 - accuracy: 3.2216e-
Epoch 19: ReduceLROnPlateau reducing learning rate to 7.275957959772868e-15.
488/488 [==============================] - 11s 22ms/step - loss: 0.0010 - mae: 0.0181 - accuracy: 3.
Epoch 20/50
488/488 [==============================] - 10s 21ms/step - loss: 0.0010 - mae: 0.0180 - accuracy: 0.
Epoch 21/50
488/488 [==============================] - 8s 17ms/step - loss: 0.0010 - mae: 0.0182 - accuracy: 3.2
Epoch 22/50
486/488 [============================>.] - ETA: 0s - loss: 0.0010 - mae: 0.0181 - accuracy: 0.0000e+
Epoch 22: ReduceLROnPlateau reducing learning rate to 3.637978979886434e-15.
488/488 [==============================] - 11s 22ms/step - loss: 0.0010 - mae: 0.0182 - accuracy: 0.
Epoch 23/50
488/488 [==============================] - 10s 20ms/step - loss: 0.0010 - mae: 0.0182 - accuracy: 0.
Epoch 24/50
488/488 [==============================] - 9s 18ms/step - loss: 0.0010 - mae: 0.0181 - accuracy: 3.2
Epoch 25/50
488/488 [==============================] - ETA: 0s - loss: 0.0010 - mae: 0.0181 - accuracy: 3.2045e-
Epoch 25: ReduceLROnPlateau reducing learning rate to 1.818989489943217e-15.
488/488 [==============================] - 11s 22ms/step - loss: 0.0010 - mae: 0.0181 - accuracy: 3.
Epoch 26/50
488/488 [==============================] - 10s 20ms/step - loss: 0.0010 - mae: 0.0182 - accuracy: 3.
Epoch 27/50
488/488 [==============================] - 9s 17ms/step - loss: 0.0010 - mae: 0.0182 - accuracy: 3.2
```

```
compute('NO2')
```

```
compute('CO')
```

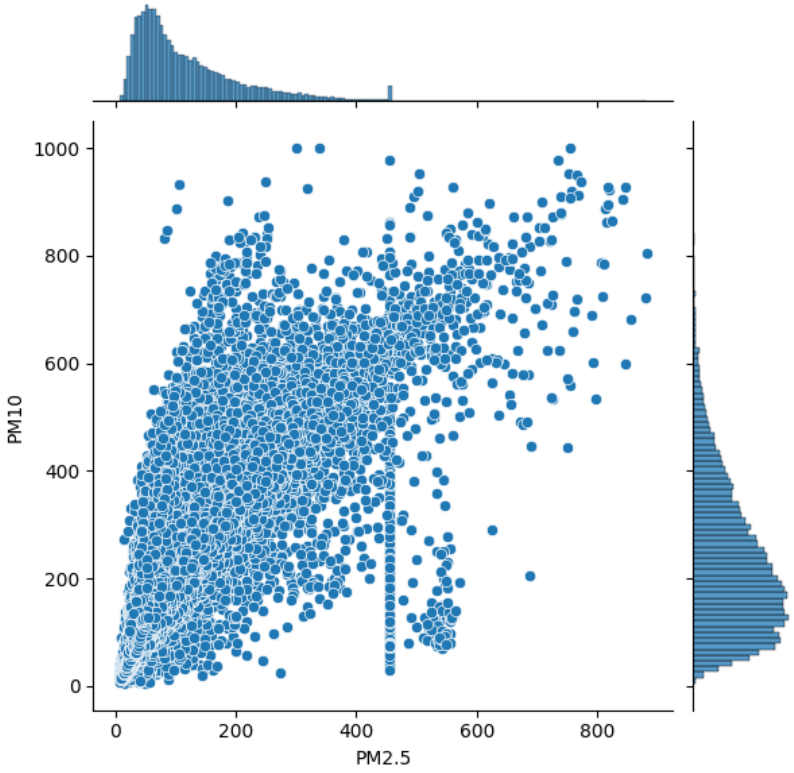
```
Epoch 1/50
485/485 [==============================] - 8s 17ms/step - loss: 0.0013 - mae: 0.0197 - accuracy: 3.2
Epoch 2/50
485/485 [==============================] - 10s 21ms/step - loss: 0.0013 - mae: 0.0196 - accuracy: 3.
Epoch 3/50
485/485 [==============================] - 12s 24ms/step - loss: 0.0013 - mae: 0.0197 - accuracy: 3.
Epoch 4/50
485/485 [==============================] - ETA: 0s - loss: 0.0013 - mae: 0.0197 - accuracy: 3.2246e-
Epoch 4: ReduceLROnPlateau reducing learning rate to 5.421011119911722e-23.
485/485 [==============================] - 8s 16ms/step - loss: 0.0013 - mae: 0.0197 - accuracy: 3.2
Epoch 5/50
485/485 [==============================] - 10s 22ms/step - loss: 0.0013 - mae: 0.0197 - accuracy: 3.
Epoch 6/50
485/485 [==============================] - 10s 22ms/step - loss: 0.0013 - mae: 0.0197 - accuracy: 3.
Epoch 7/50
485/485 [==============================] - ETA: 0s - loss: 0.0013 - mae: 0.0197 - accuracy: 3.2246e-
Epoch 7: ReduceLROnPlateau reducing learning rate to 2.710505559955861e-23.
485/485 [==============================] - 8s 17ms/step - loss: 0.0013 - mae: 0.0197 - accuracy: 3.2
Epoch 8/50
485/485 [==============================] - 11s 22ms/step - loss: 0.0013 - mae: 0.0198 - accuracy: 3.
Epoch 9/50
485/485 [==============================] - 10s 22ms/step - loss: 0.0013 - mae: 0.0197 - accuracy: 3.
Epoch 10/50
485/485 [==============================] - ETA: 0s - loss: 0.0013 - mae: 0.0198 - accuracy: 3.2246e-
Epoch 10: ReduceLROnPlateau reducing learning rate to 1.3552527799779304e-23.
485/485 [==============================] - 8s 17ms/step - loss: 0.0013 - mae: 0.0198 - accuracy: 3.2
Epoch 11/50
485/485 [==============================] - 11s 22ms/step - loss: 0.0012 - mae: 0.0196 - accuracy: 3.
Epoch 12/50
485/485 [==============================] - 10s 21ms/step - loss: 0.0013 - mae: 0.0197 - accuracy: 3.
Epoch 13/50
482/485 [============================>.] - ETA: 0s - loss: 0.0013 - mae: 0.0198 - accuracy: 3.2417e-
Epoch 13: ReduceLROnPlateau reducing learning rate to 6.776263899889652e-24.
488/488 [==============================] - 8s 17ms/step - loss: 0.0013 - mae: 0.0196 - accuracy: 3.1
```

```
compute('O3')
```

```
Epoch 1/50
488/488 [==============================] - 9s 18ms/step - loss: 0.0010 - mae: 0.0189 - accuracy: 3.2
Epoch 2/50
488/488 [==============================] - 11s 22ms/step - loss: 0.0010 - mae: 0.0189 - accuracy: 3.
Epoch 3/50
```

```
sns.jointplot(x=df['PM2.5'], y=df['PM10'], data = df)
```

```
<seaborn.axisgrid.JointGrid at 0x7fdf9f261040>
```




```
488/488 [==============================] - 8s 17ms/step - loss: 9.9428e-04 - mae: 0.0188 - accuracy:
```

The above plot gives us the idea that these two conentrations are positively correlated with very few outliers.

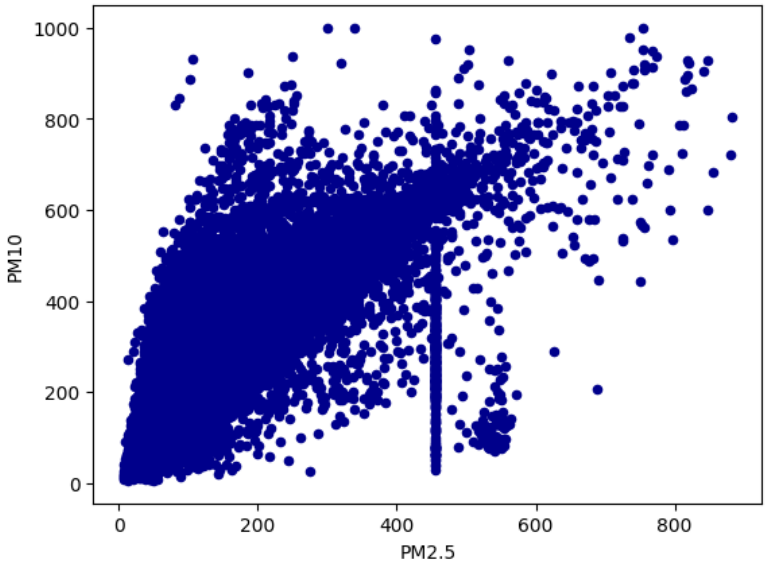
```
Epoch 19/50
```

```
#finding correlation
corrmat = df.corr()
fig, ax = plt.subplots(figsize=(11,11))
sns.heatmap(corrmat)
```

```
<Axes: >
```

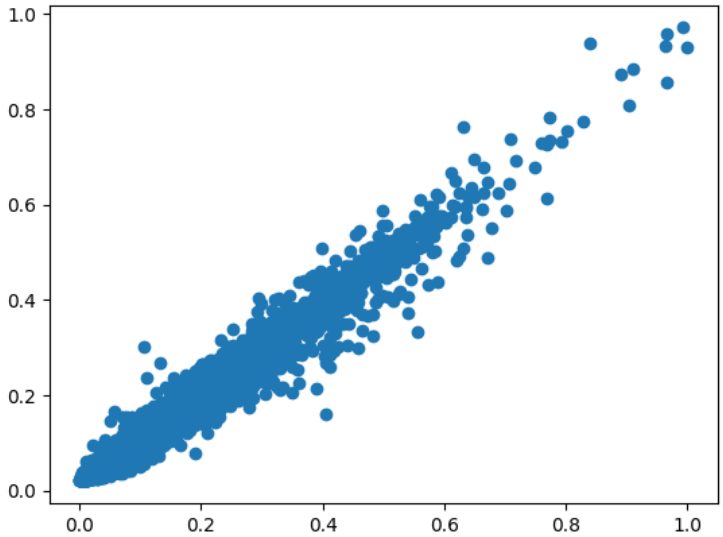

```
df.plot.scatter(x='PM2.5', y='PM10', c='DarkBlue')
```

```
/usr/local/lib/python3.9/dist-packages/pandas/plotting/_matplotlib/core.py:1114: UserWarning: No data f
  scatter = ax.scatter(
<Axes: xlabel='PM2.5', ylabel='PM10'>
```




```
plt.scatter(y_split_test, yhat)
```

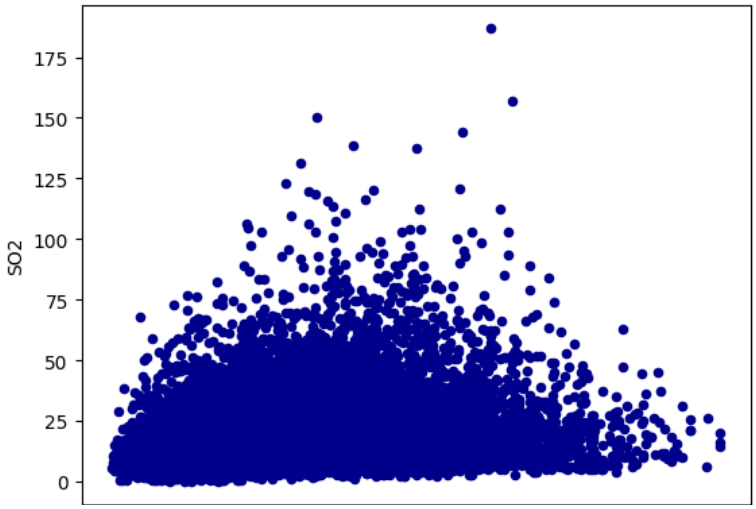
```
<matplotlib.collections.PathCollection at 0x7fdf9bd10880>
```



Double-click (or enter) to edit

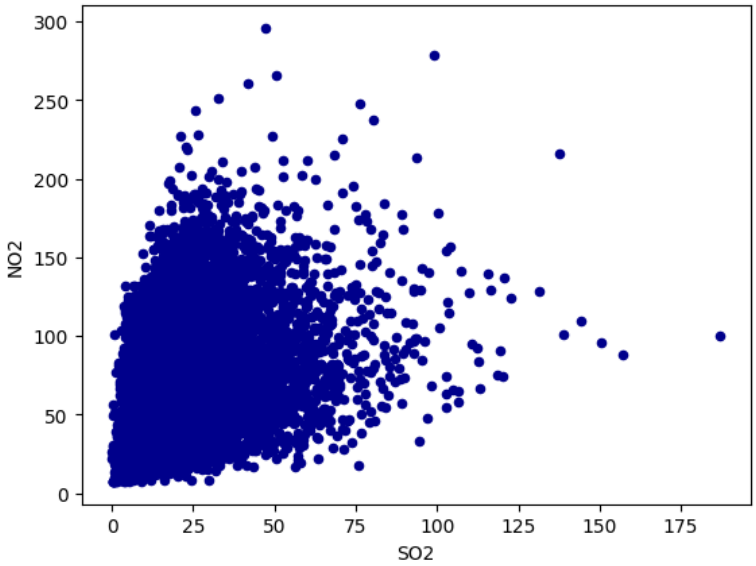
```
df.plot.scatter(x='PM10', y='SO2', c='DarkBlue')
```

```
<Axes: xlabel='PM10', ylabel='SO2'>
```



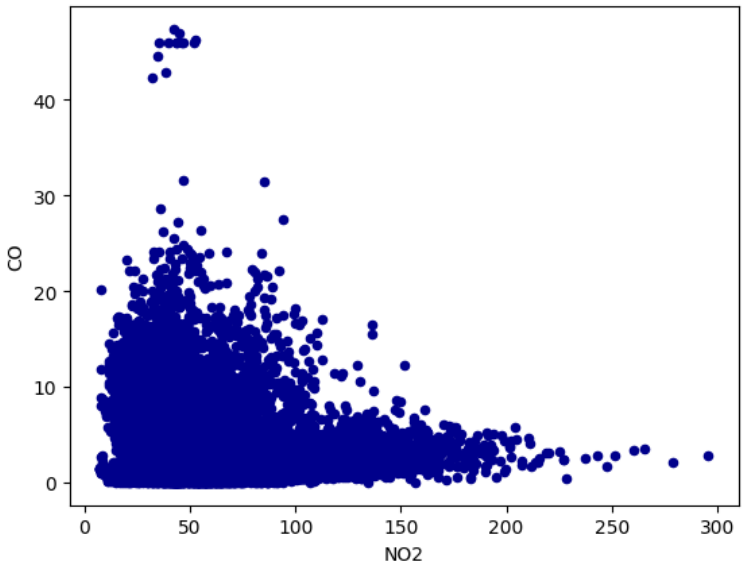

```
df.plot.scatter(x='SO2', y='NO2', c='DarkBlue')
```

```
<Axes: xlabel='SO2', ylabel='NO2'>
```



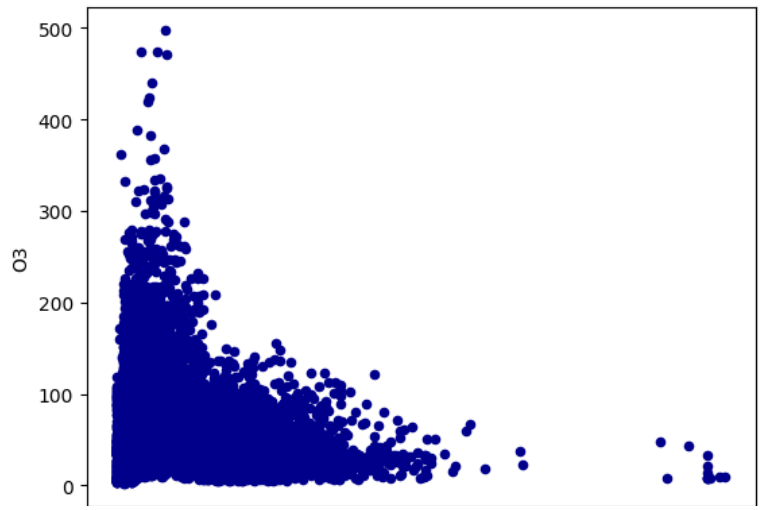

```
df.plot.scatter(x='NO2', y='CO', c='DarkBlue')
```

```
<Axes: xlabel='NO2', ylabel='CO'>
```




```
df.plot.scatter(x='CO', y='O3', c='DarkBlue')
```

```
<Axes: xlabel='CO', ylabel='O3'>
```



Heatmap is a very useful visualization tool to know how much each feature is correlated. vmax = max value of the heatmap fmt = number of decimal places upto which the value is shown square = do you want the heatmap to be square shaped linewidth = width of the lines in the heatmap annot = should the boxes be labelled with the value.

```
correlations = df.corr()
fig, ax = plt.subplots(figsize=(15,15))
sns.heatmap(correlations, vmax=1.0, center=0, fmt='.2f', square=True, linewidths=.5, annot=True, cbar_kws={"shrink": .70})
plt.show();
```