Forecasting the 24 Hour PM2.5 Particle Concentration in Milan

Employing various Auto regression and Deep learning models for forecasting AQI

An assignment submitted in fulfillment of the requirements of IBM Introduction to Machine Learning Specialization

By

Minhaj Ahmed Ansari

1. Introduction

1.1. The main objective of this project is to forecast the value of the time series for future 24 hours using historical data.

1.2. To meet this objective, we will during the project,

- Clean the data, check for any missing data point.
- Perform EDA to determine whether our dataset is stationary or not.
- Select Model Parameters using the graphical methods.
- Perform Deep Learning using suitable methodology.
- Derive Conclusions and select the best model.
- Summary and Future Actions

2. Brief description of the data set and a summary of its attributes

2.1. The data that I am using here is publicly available in Kaggle and you can access it using the link here - https://www.kaggle.com/wiseair/air-quality-in-milan-summer-2020/tasks?taskId=2285

2.2. The data are the time series of the concentration of Particulate Matter 2.5 in the air of Milan.

2.3. The data have been sampled using a laser scattering PM2.5 sensor.

2.4. The sampling frequency varies from 1/15min to 1/1h. The dataset contains hourly means.

2.5. Data have been sampled in Milan, at 45.4582861 North 9.1675597 East.

2.6. Data have been sampled from 2020-07-24 to 2020-09-20.

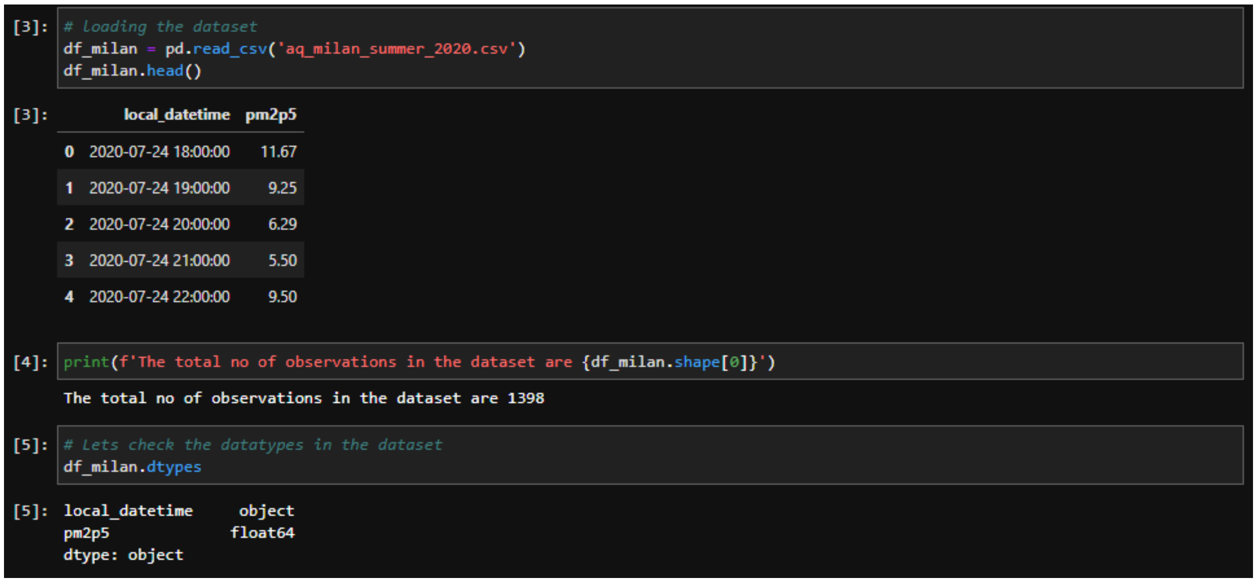
3. Data Cleaning and Wrangling

3.1. **Plan**

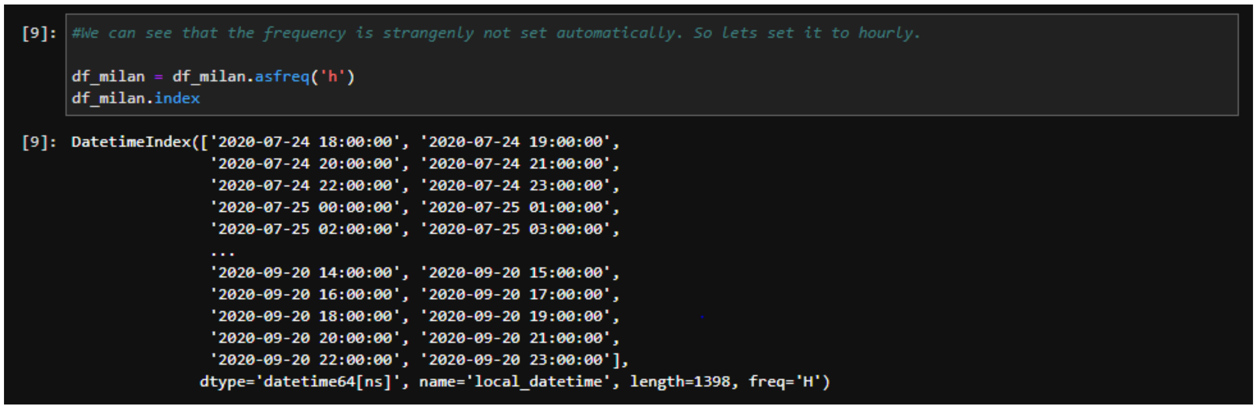
- Importing the dataset
- Check the variables for proper datatype and perform datatype transformation as required.
- Change the dataframe index to datetime index and set frequency to hours.

3.2. **Actions**

- Importing necessary modules and dataset.
- Checking the dataset for missing values and datatypes. During this step it was identified that the datetime values were stored as object datatype.
- Changed the datetime data to datetime datatypes.
- Changed the dataframe index to datetime index and set frequency to hour.



```
[3]: # Loading the dataset
     df_milan = pd.read_csv('aq_milan_summer_2020.csv')
     df_milan.head()

[3]:      local_datetime  pm2p5
     0  2020-07-24 18:00:00  11.67
     1  2020-07-24 19:00:00   9.25
     2  2020-07-24 20:00:00   6.29
     3  2020-07-24 21:00:00   5.50
     4  2020-07-24 22:00:00   9.50

[4]: print(f'The total no of observations in the dataset are {df_milan.shape[0]}')

     The total no of observations in the dataset are 1398

[5]: # Lets check the datatypes in the dataset
     df_milan.dtypes

[5]: local_datetime     object
     pm2p5             float64
     dtype: object
```



```
[9]: #We can see that the frequency is strangenly not set automatically. So lets set it to hourly.

     df_milan = df_milan.asfreq('h')
     df_milan.index

[9]: DatetimeIndex(['2020-07-24 18:00:00', '2020-07-24 19:00:00',
                    '2020-07-24 20:00:00', '2020-07-24 21:00:00',
                    '2020-07-24 22:00:00', '2020-07-24 23:00:00',
                    '2020-07-25 00:00:00', '2020-07-25 01:00:00',
                    '2020-07-25 02:00:00', '2020-07-25 03:00:00',
                    ...
                    '2020-09-20 14:00:00', '2020-09-20 15:00:00',
                    '2020-09-20 16:00:00', '2020-09-20 17:00:00',
                    '2020-09-20 18:00:00', '2020-09-20 19:00:00',
                    '2020-09-20 20:00:00', '2020-09-20 21:00:00',
                    '2020-09-20 22:00:00', '2020-09-20 23:00:00'],
                   dtype='datetime64[ns]', name='local_datetime', length=1398, freq='H')
```

- Our dataset contains 1398 observations and 1 feature – 'pm2p5' of datatype float64.

3.3. **Summary**

- Imported the data and transformed the date into datetime variable.
- Changed the dataframe index to datetime index and set frequency to hour.

4. Exploratory Data Analysis

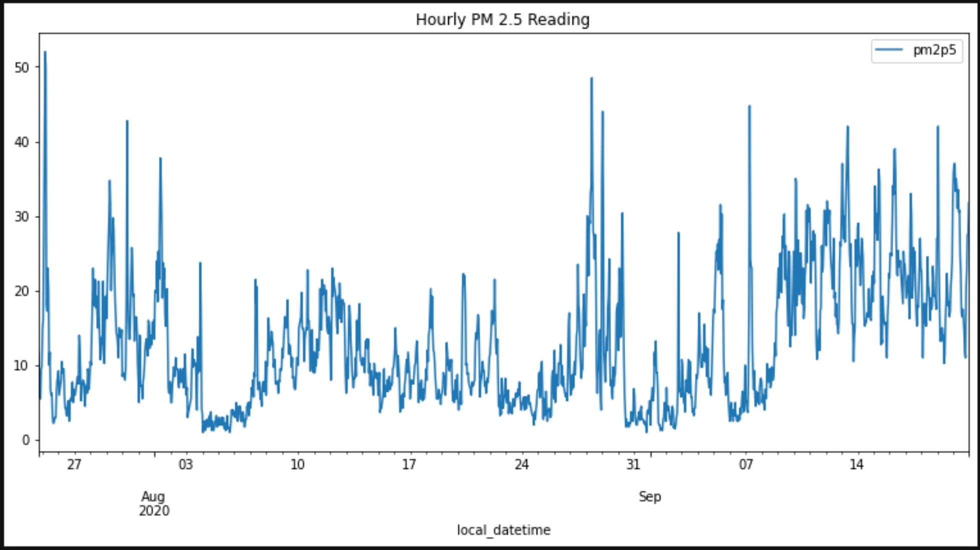
4.1. **Plan**

- Check the run sequence plot of the data and check the data for stationarity.
- Plot histograms, ACF and PACF plots to determine the further actions required to achieve stationarity.
- Determine the hyper-parameters such as p, d, q and P, D, Q, s.

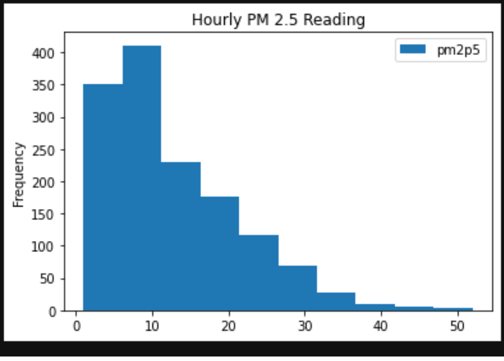
4.2. **Actions**

- Visual analysis of the data by plotting run sequence, ACF, PACF and Histogram plots.

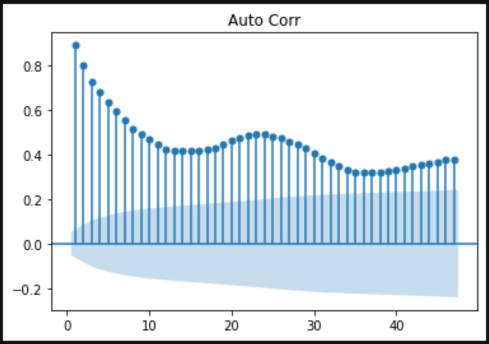
```
[79]: # lets plot the run sequence plot
      df_milan.plot(figsize=plotsize, title='Hourly PM 2.5 Reading');
```

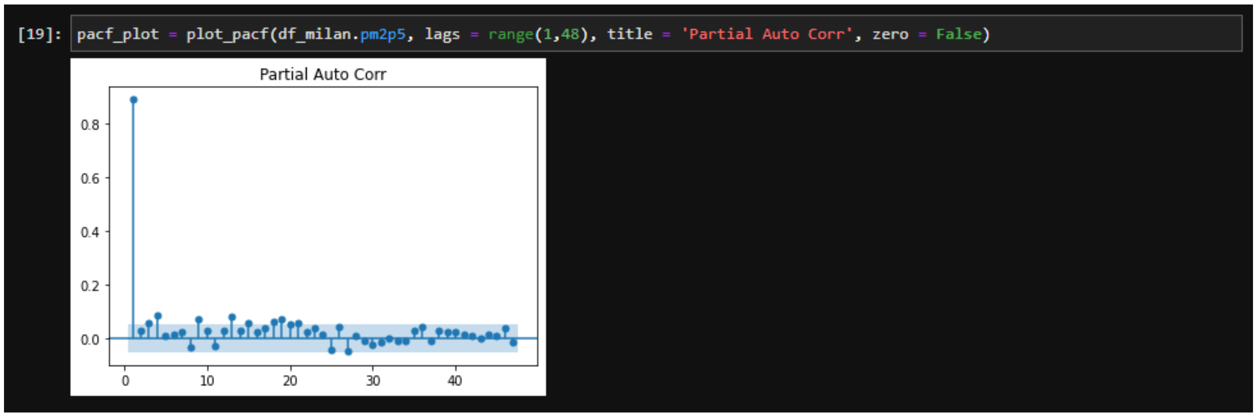


```
[16]: # Lets plot the histogram for our latter reference

      df_milan.plot.hist();
      plt.title('Hourly PM 2.5 Reading');
```



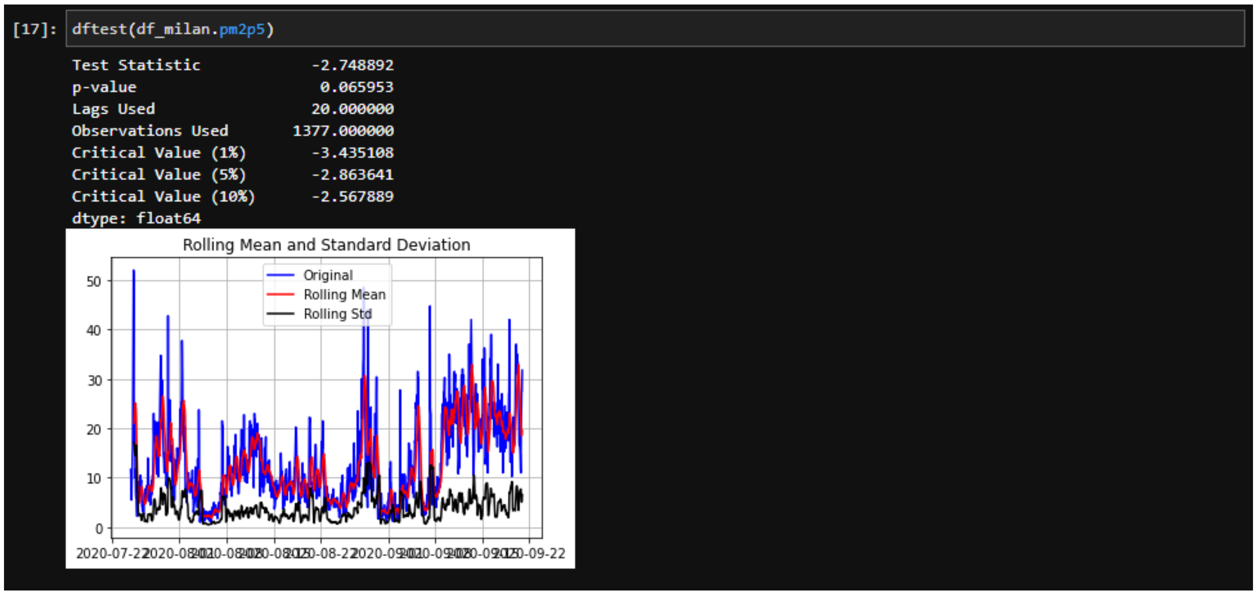
```
[18]: acf_plot = plot_acf(df_milan.pm2p5, lags = range(1,48), title = 'Auto Corr', zero = False)
```





```
[19]: pacf_plot = plot_pacf(df_milan.pm2p5, lags = range(1,48), title = 'Partial Auto Corr', zero = False)
```

- We see there is some kind of Autocorrelation in the data and probably its mostly because of the lag-1 Partial Autocorrelation. Also, It seems that our data certainly doesn't contain any type of trend but looks like we don't have constant mean and variance throughout our data.
- We verified above assumptions using statistical method called - Augmented Dickey–Fuller test.
- In statistics and econometrics, an augmented Dickey–Fuller test (ADF) tests the null hypothesis that a unit root is present in a time series sample. The alternative hypothesis is different depending on which version of the test is used but is usually stationarity or trend-stationarity. It is an augmented version of the Dickey–Fuller test for a larger and more complicated set of time series models.



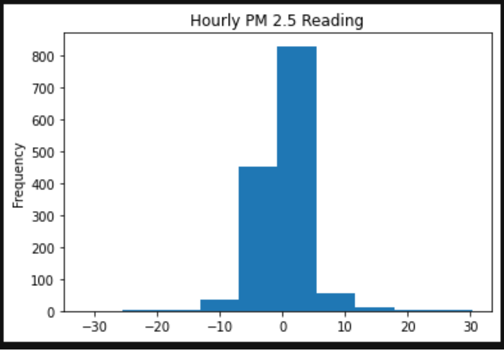
```
[17]: dftest(df_milan.pm2p5)

      Test Statistic                -2.748892
      p-value                        0.065953
      Lags Used                     20.000000
      Observations Used           1377.000000
      Critical Value (1%)           -3.435108
      Critical Value (5%)           -2.863641
      Critical Value (10%)          -2.567889
      dtype: float64
```

- Clearly, the time series is non-stationary as the p-value is 0.065 higher that alpha (confidence interval - 0.05).

- Now, to achieve stationarity we can adopt various strategies but the one which I am going to adopt is the simplest - Differencing.

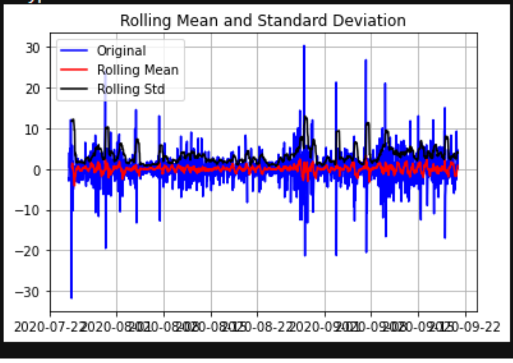
```
[21]: df_milan['difference'] = df_milan.diff()
```

```
[22]: # Lets check out dataset for normality

      df_milan['difference'].plot.hist();
      plt.title('Hourly PM 2.5 Reading');
```



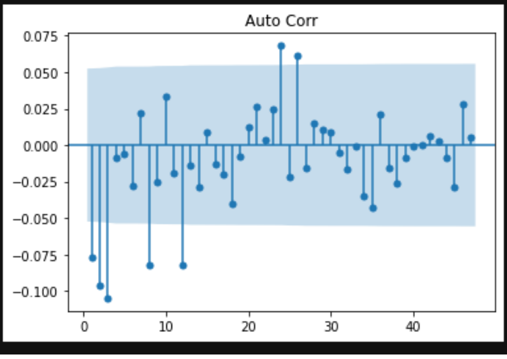
- We performed ADF test once again after performing the differencing.

```
[23]: # testing for rejecting null hypothesis
      dftest(df_milan.difference.dropna())

      Test Statistic         -1.346356e+01
      p-value                 3.491807e-25
      Lags Used               2.100000e+01
      Observations Used       1.375000e+03
      Critical Value (1%)    -3.435115e+00
      Critical Value (5%)    -2.863644e+00
      Critical Value (10%)   -2.567890e+00
      dtype: float64
```

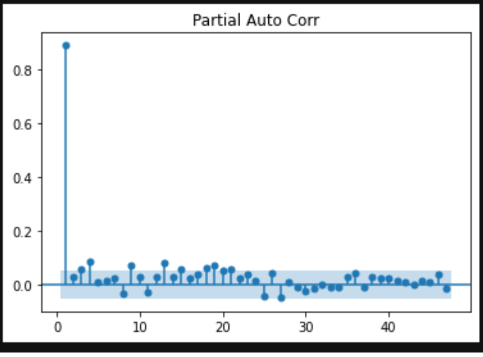


- So, clearly, we have achieved stationarity as well as quite reasonably constant mean and variance. Further time to check PACF and ACF plots to finalize our hyper-parameters for next step.

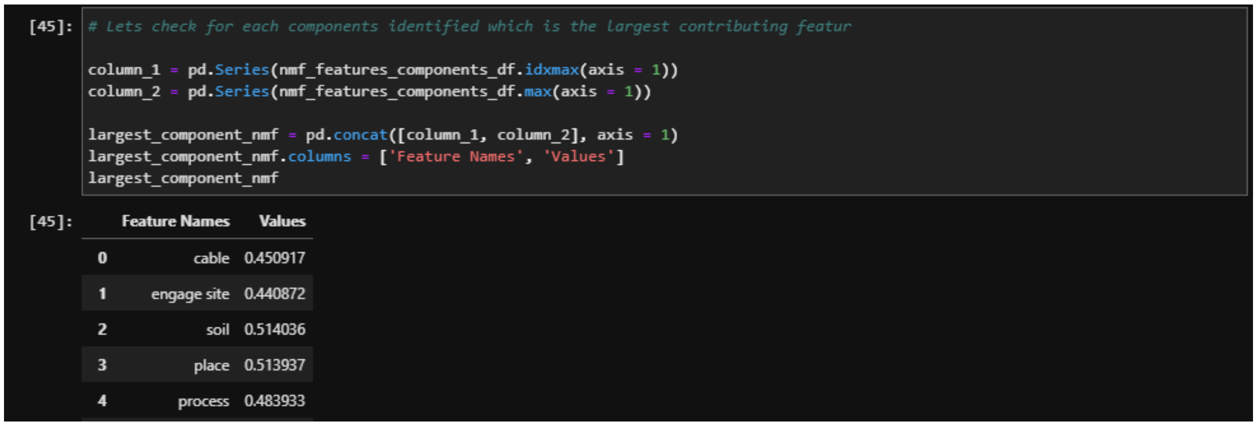
```
[24]: acf_plot = plot_acf(df_milan.difference.dropna(), lags = range(1,48), title = 'Auto Corr', zero = False)
```



```
[25]: pacf_plot = plot_pacf(df_milan.pm2p5, lags = range(1,48), title = 'Partial Auto Corr', zero = False)
```



- By looking at PACF plot it looks like we have on our hand an AR-1 Model (p = 1). Also, ACF plot has all negative lags now, which means our differencing operation has treated our dataset for autocorrelation.
- Also, it suggests we chose MA-3 model with q = 3 and d = 1.



```
[45]: # Lets check for each components identified which is the largest contributing featur

      column_1 = pd.Series(nmf_features_components_df.idxmax(axis = 1))
      column_2 = pd.Series(nmf_features_components_df.max(axis = 1))

      largest_component_nmf = pd.concat([column_1, column_2], axis = 1)
      largest_component_nmf.columns = ['Feature Names', 'Values']
      largest_component_nmf
```

[45]:

	Feature Names	Values
0	cable	0.450917
1	engage site	0.440872
2	soil	0.514036
3	place	0.513937
4	process	0.483933

4.3. **Summary**

- We identified that our data is not stationary - i.e., it didn't have constant mean and variance. Also, it contained auto-correlation

- There is no seasonality as such, but we will keep our options open during the model selection.
- We performed hypothesis testing to confirm our assumptions.
- We performed first order differencing operating and achieved significant stationarity which was also confirmed by the ADF hypothesis test results.
- We identified our hyper parameters as below
    - p = 1
    - d = 1
    - q = 3

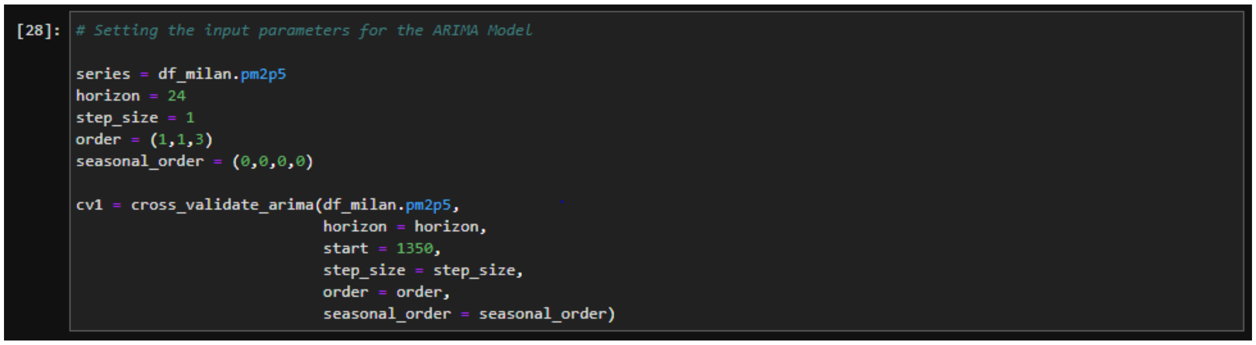
5. **Model Selection and Evaluation**

5.1. **Plan**

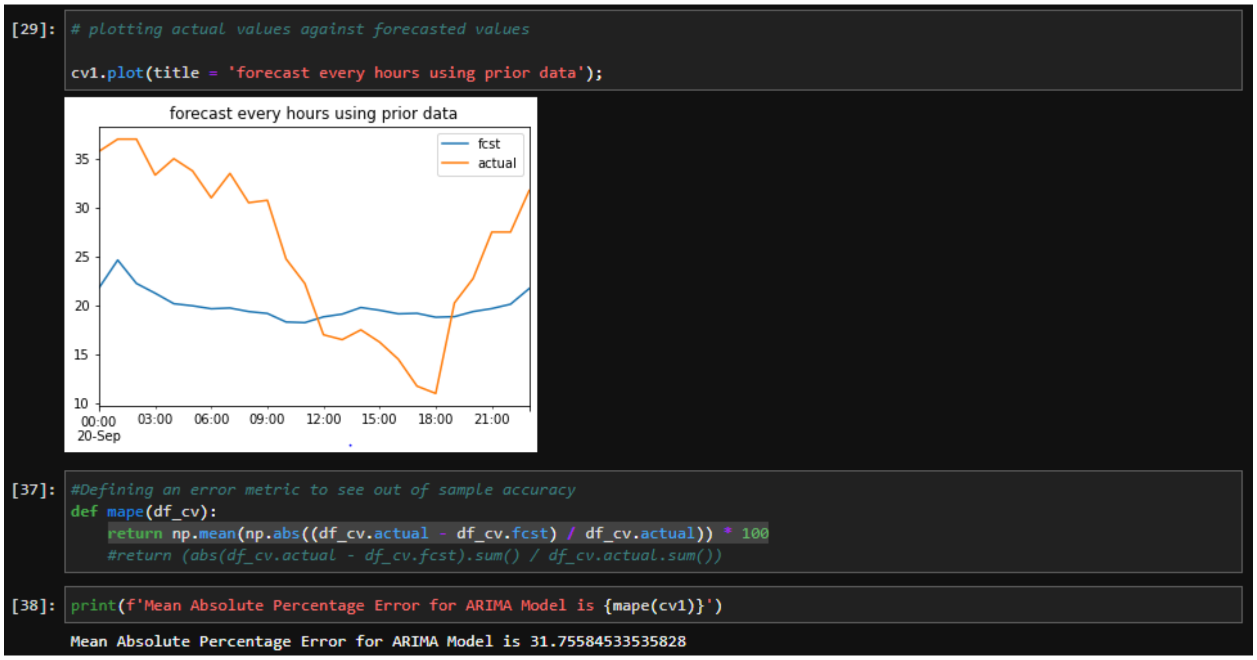
- We will try three models here
    - ARIMA Model
    - SARIMAX Model
    - Simple RNN Model
- Select the best model using Mean absolute percentage error metrics

5.2. **ARIMA Model**

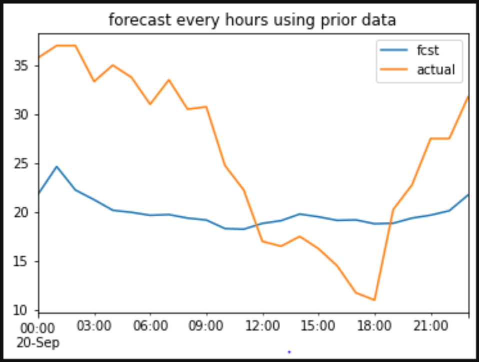
- We defined model using selected hyper parameters in previous step.
- The modelling was performed using cross validation function, with a time horizon of 24 hours and step size of 1 hour



```
[28]: # Setting the input parameters for the ARIMA Model

      series = df_milan.pm2p5
      horizon = 24
      step_size = 1
      order = (1,1,3)
      seasonal_order = (0,0,0,0)

      cv1 = cross_validate_arima(df_milan.pm2p5,
                                 horizon = horizon,
                                 start = 1350,
                                 step_size = step_size,
                                 order = order,
                                 seasonal_order = seasonal_order)
```



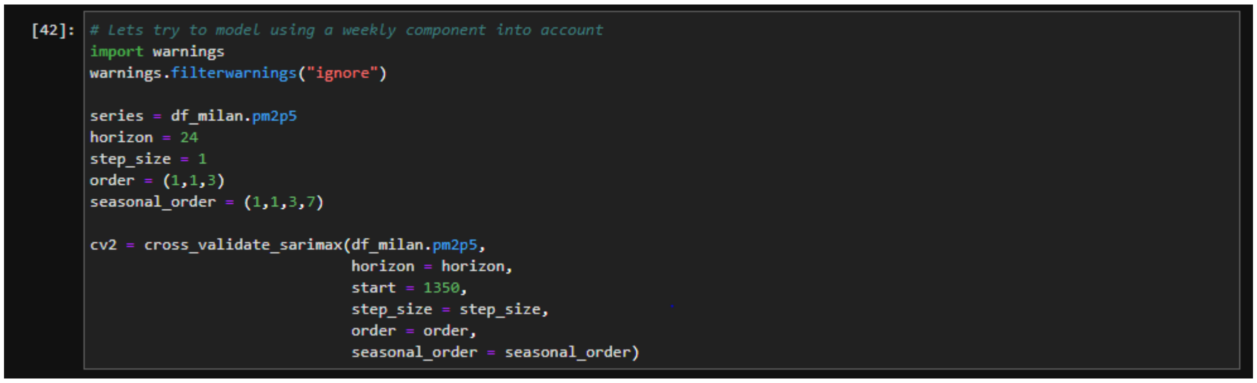
```
[29]: # plotting actual values against forecasted values

      cv1.plot(title = 'forecast every hours using prior data');
```



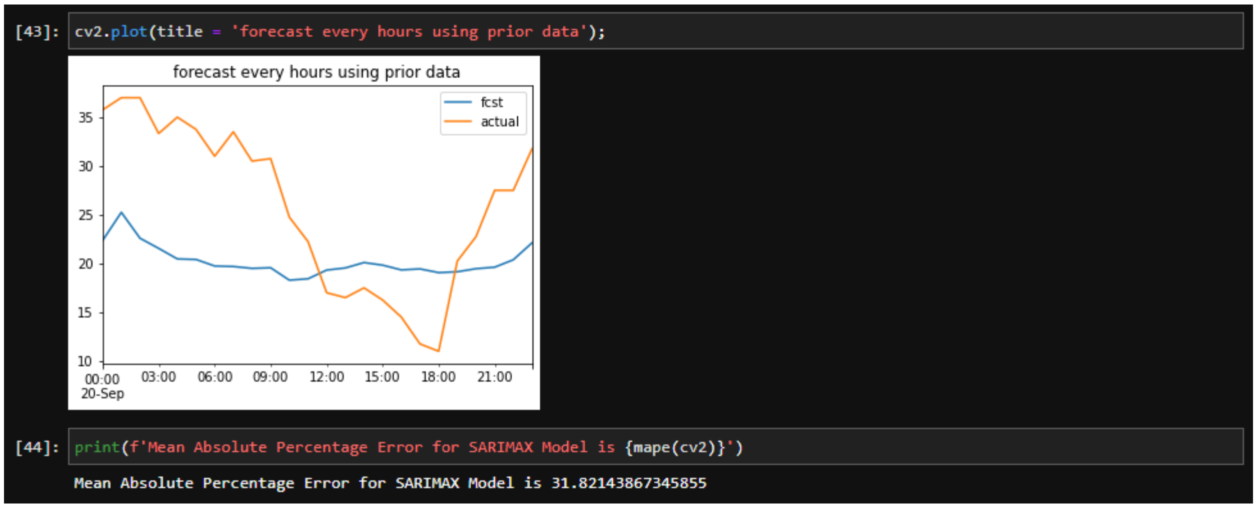
```
[37]: #Defining an error metric to see out of sample accuracy
      def mape(df_cv):
          return np.mean(np.abs((df_cv.actual - df_cv.fcst) / df_cv.actual)) * 100
          #return (abs(df_cv.actual - df_cv.fcst).sum() / df_cv.actual.sum())

[38]: print(f'Mean Absolute Percentage Error for ARIMA Model is {mape(cv1)}')

      Mean Absolute Percentage Error for ARIMA Model is 31.75584533535828
```

5.3. **SARIMAX**

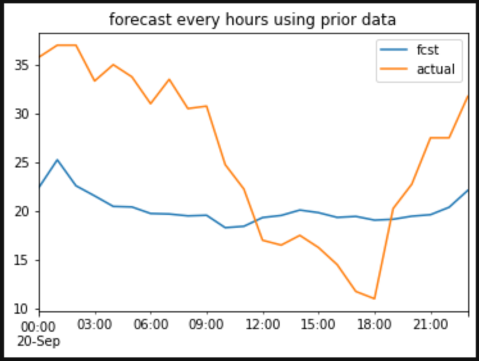
- Just to ensure we don't miss out on any seasonality, we also performed modelling of SARIMAX Model with Seasonality set to 7 (for weekly data).



```
[42]: # Lets try to model using a weekly component into account
      import warnings
      warnings.filterwarnings("ignore")

      series = df_milan.pm2p5
      horizon = 24
      step_size = 1
      order = (1,1,3)
      seasonal_order = (1,1,3,7)

      cv2 = cross_validate_sarimax(df_milan.pm2p5,
                                   horizon = horizon,
                                   start = 1350,
                                   step_size = step_size,
                                   order = order,
                                   seasonal_order = seasonal_order)
```



```
[43]: cv2.plot(title = 'forecast every hours using prior data');
```

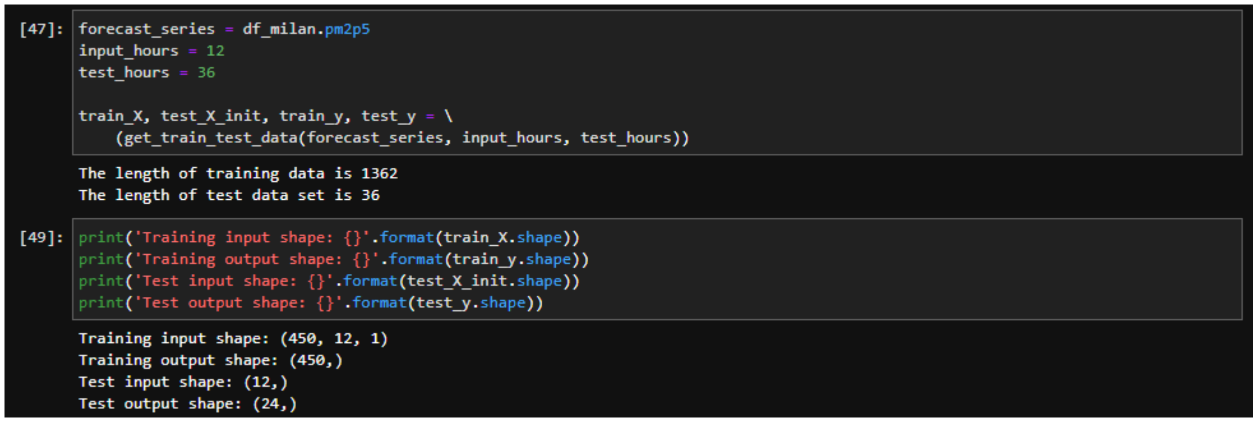


```
[44]: print(f'Mean Absolute Percentage Error for SARIMAX Model is {mape(cv2)}')

      Mean Absolute Percentage Error for SARIMAX Model is 31.82143867345855
```

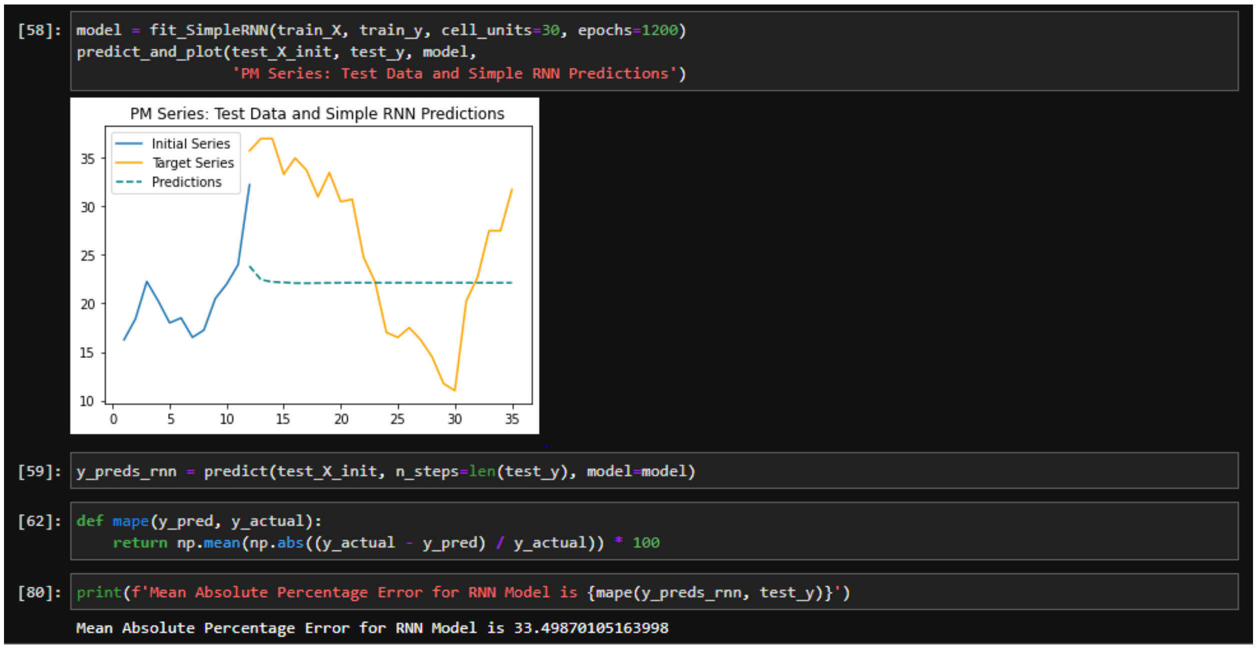
- The MAPE Error of SARIMAX model was higher than ARIMA Model (31.821 and 31.755 respectively).
- So, it can be safely assumed that our data definitely doesn't contains any significant seasonal component.

5.4. **RNN Model**

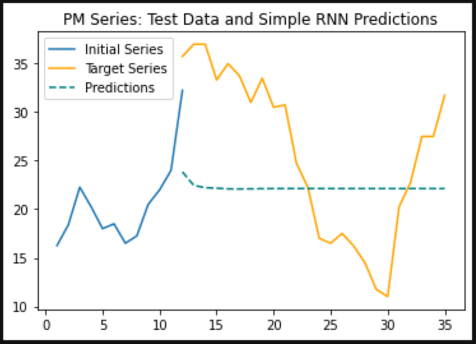
- We wrote functions to convert data series into keras friendly format and holded out last 36 hours of data for testing.



```
[47]: forecast_series = df_milan.pm2p5
      input_hours = 12
      test_hours = 36

      train_X, test_X_init, train_y, test_y = \
          (get_train_test_data(forecast_series, input_hours, test_hours))

      The length of training data is 1362
      The length of test data set is 36

[49]: print('Training input shape: {}'.format(train_X.shape))
      print('Training output shape: {}'.format(train_y.shape))
      print('Test input shape: {}'.format(test_X_init.shape))
      print('Test output shape: {}'.format(test_y.shape))

      Training input shape: (450, 12, 1)
      Training output shape: (450,)
      Test input shape: (12,)
      Test output shape: (24,)
```



```
[58]: model = fit_SimpleRNN(train_X, train_y, cell_units=30, epochs=1200)
      predict_and_plot(test_X_init, test_y, model,
                       'PM Series: Test Data and Simple RNN Predictions')
```



```
[59]: y_preds_rnn = predict(test_X_init, n_steps=len(test_y), model=model)

[62]: def mape(y_pred, y_actual):
          return np.mean(np.abs((y_actual - y_pred) / y_actual)) * 100

[80]: print(f'Mean Absolute Percentage Error for RNN Model is {mape(y_preds_rnn, test_y)}')

      Mean Absolute Percentage Error for RNN Model is 33.49870105163998
```

- We ran model for 1200 epochs and achieved a MAPE of 33.498 which was worse than both ARIMA and SARIMAX.

5.5. **Summary**

- We trained and tested three models - ARIMA, SARIMAX and a Simple RNN Model. The performance of each model was measured using Mean Absolute Percentage Error Metric.
- For each model we kept hold out set to predict 24 hours into the future.
- The ARIMA Model performed best, with MAPE value of 31.755

6. Conclusion and Future Directions

6.1. We started this project with an objective of determining the best model to forecast 24 hours into future the PM2.5 concentration value in Milan.

6.2. In conclusion we can state that, The ARIMA Model forecast with minimum MAPE and proves to be best among SARIMAX and RNN Models.

6.3. To further improve our performance we can,

- Also consider the LSTM Models which have proven better track record for Time-series data. However, due to paucity of time we were not able to perform LSTM modelling as part of our project.
- The dataset was actually proved to be too small for any such forecasting exercise; larger dataset running into probably a complete quarter, or year would be more helpful.
- We could also have used Auto ARIMA Models for automatically tune the best hyper-parameter using AIC metric. This could be tried in future for achieving better performance.
- We can also try other types of transformations on our data to achieve stationarity.