Week 6 Deep learning

Introduction: This hourly data set contains the PM2.5 data in Beijing, Shanghai, Guangzhou, Chengdu and Shenyang. Meanwhile, meteorological data for each city are also included.
https://archive.ics.uci.edu/ml/datasets/PM2.5+Data+of+Five+Chinese+Cities
(https://archive.ics.uci.edu/ml/datasets/PM2.5+Data+of+Five+Chinese+Cities)

Objective

I used recurrent neural networks and LTSM model to predict the PM2.5 level in Beijing, Shanghai, Guangzhou, Chengdu and Shenyang using time series data.

Imports

In [1]:

```
import sys, os
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.simplefilter(action='ignore')
import seaborn as sns
import pandas as pd
from datetime import datetime
```

In [2]:

```
import tensorflow
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, SimpleRNN, LSTM, Activation, Dropout
```

In [3]:

```
import math
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
```

Data description

In [4]:

```
os.getcwd()
```

Out[4]:

```
'/Users/yanzhenlei'
```

In [5]:

```
df_Beijing = pd.read_csv('/Users/yanzhenlei/Beijing.csv')
df_Beijing = df_Beijing[df_Beijing.year >= 2015]
df_Beijing.head(10)
```

Out[5]:

	No	year	month	day	hour	season	PM_Dongsi	PM_Dongsihuan	PM_Nongzhangua
43824	43825	2015	1	1	0	4	5.0	32.0	8.
43825	43826	2015	1	1	1	4	4.0	12.0	7.
43826	43827	2015	1	1	2	4	3.0	19.0	7.
43827	43828	2015	1	1	3	4	4.0	9.0	11.
43828	43829	2015	1	1	4	4	3.0	11.0	5.
43829	43830	2015	1	1	5	4	3.0	18.0	3.
43830	43831	2015	1	1	6	4	3.0	20.0	6.
43831	43832	2015	1	1	7	4	3.0	22.0	7.
43832	43833	2015	1	1	8	4	NaN	NaN	Nal
43833	43834	2015	1	1	9	4	5.0	37.0	11.

In [67]:

```
df_Beijing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 8760 entries, 2015-01-01 00:00:00 to 2015-12-31 23:00:0
0
Data columns (total 19 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   No               8760 non-null   int64  
 1   year             8760 non-null   int64  
 2   month            8760 non-null   int64  
 3   day              8760 non-null   int64  
 4   hour             8760 non-null   int64  
 5   season           8760 non-null   int64  
 6   PM_Dongsi        8760 non-null   float64
 7   PM_Dongsihuan    5465 non-null   float64
 8   PM_Nongzhanguan  8760 non-null   float64
 9   PM_US Post       8631 non-null   float64
 10  DEWP             8755 non-null   float64
 11  HUMI             8421 non-null   float64
 12  PRES             8421 non-null   float64
```

There are 8760 units for analysis. We need to transform a variable into a datetime type.

In [6]:

```
df_Beijing.isnull().sum()
```

Out[6]:

```
No                   0
year                 0
month                0
day                  0
hour                 0
season               0
PM_Dongsi          164
PM_Dongsihuan     3295
PM_Nongzhanguan    287
PM_US Post         129
DEWP                 5
HUMI               339
PRES               339
TEMP                 5
cbwd                 5
Iws                  5
precipitation      459
Iprec              459
dtype: int64
```

PM_Dongsi had the the least missing values for PM values so I used PM_Dongsi as example.

In [66]:

```
df_Beijing['PM_Dongsi'].describe()
```

Out[66]:

```
count    8760.000000
mean       87.145776
std        91.899063
min         3.000000
25%        22.000000
50%        58.000000
75%       117.000000
max       685.000000
Name: PM_Dongsi, dtype: float64
```

In [8]:

```
df_Beijing['PM_Dongsi'] = df_Beijing['PM_Dongsi'].interpolate()
```

In [9]:

```
def make_date(row):
    return datetime(year = row['year'], month = row['month'], day = row['day'],
df_Beijing['date'] = df_Beijing.apply(make_date,axis=1)
```

In [10]:

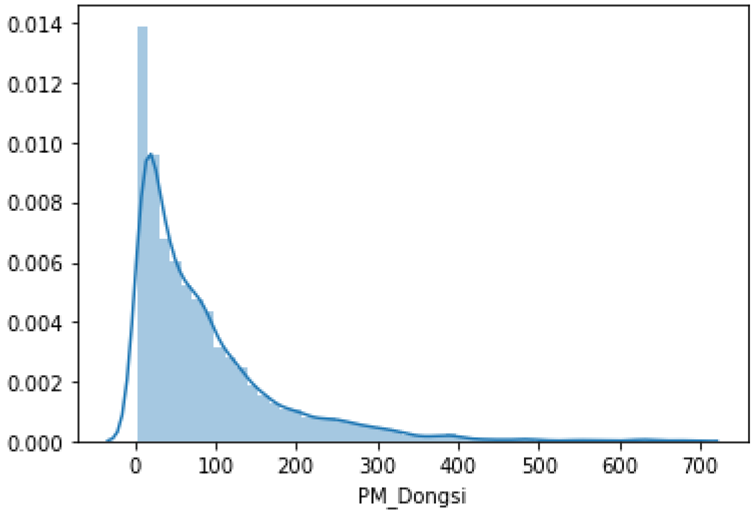
```
df_Beijing.set_index(df_Beijing.date,inplace=True)
```

In [68]:

```
sns.distplot(df_Beijing['PM_Dongsi'])
```

Out[68]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f98be25ae80>



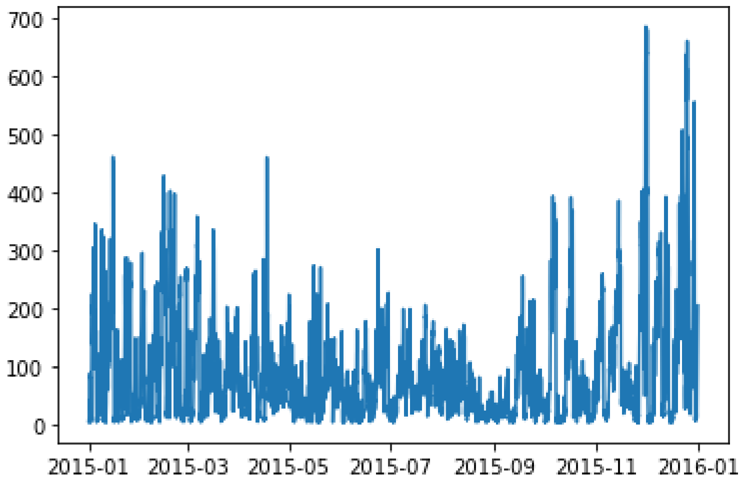
Most of the PM_Dongsi level concentrated in the zone less than 100 but the extreme large level is also abudant

In [11]:

```
plt.plot(df_Beijing['PM_Dongsi'])
```

Out[11]:

[<matplotlib.lines.Line2D at 0x7f9908674438>]



Data engineering

We use the last 56 days of the PM_Dongsi series, and will train a model that takes in 12 time steps in order to predict the next time step. We use the last day of data for visually testing the model.

In [12]:

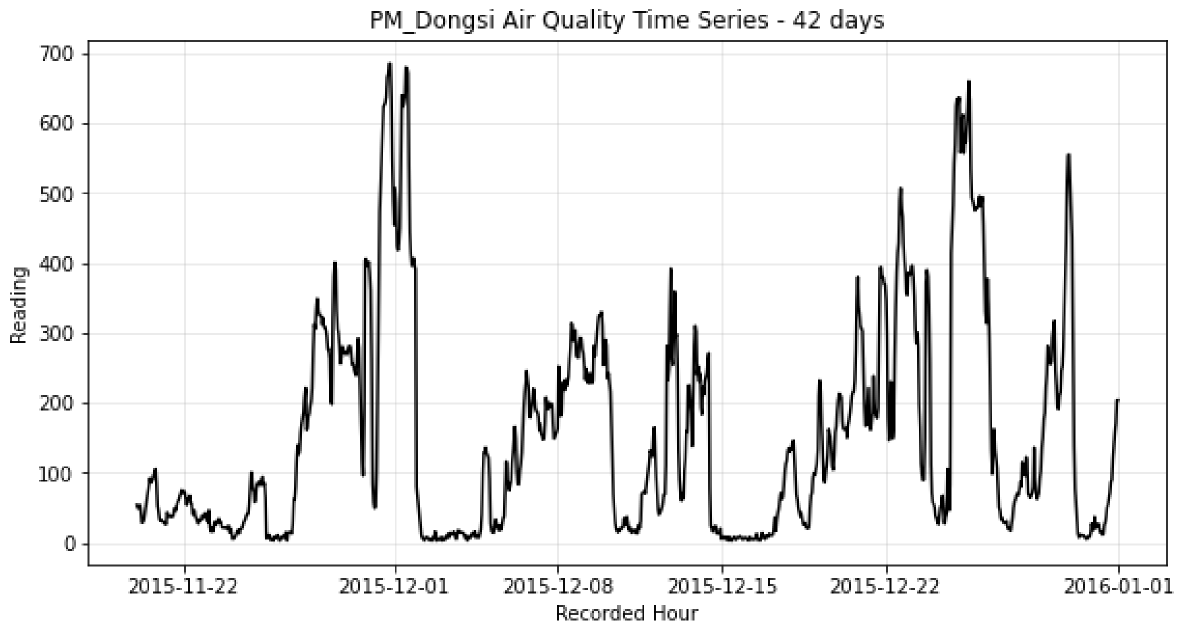
```
def get_n_last_days(df,series_name,n_days):
    return df[series_name][-(24*n_days):]
```

In [13]:

```
def plot_n_last_days(df,series_name,n_days):
    plt.figure(figsize=(10,5))
    plt.plot(get_n_last_days(df,series_name,n_days),'k-')
    plt.title('{0} Air Quality Time Series - {1} days'
              .format(series_name, n_days))
    plt.xlabel('Recorded Hour')
    plt.ylabel('Reading')
    plt.grid(alpha=0.3)
```

In [14]:

```
plot_n_last_days(df_Beijing,'PM_Dongsi',42)
```



In [43]:

```
def get_keras_format_series(series):
    series = np.array(series)
    return series.reshape(series.shape[0], series.shape[1], 1)
```

In [44]:

```
def get_train_test_data(df, series_name, series_days, input_hours,
                        test_hours, sample_gap=3):

    forecast_series = get_n_last_days(df, series_name, series_days).values

    train = forecast_series[:-test_hours]
    test = forecast_series[-test_hours:]

    train_X, train_y = [], []

    for i in range(0, train.shape[0]-input_hours, sample_gap):
        train_X.append(train[i:i+input_hours])
        train_y.append(train[i+input_hours])

    train_X = get_keras_format_series(train_X)
    train_y = np.array(train_y)
    test_X_init = test[:input_hours]
    test_y = test[input_hours:]

    return train_X, test_X_init, train_y, test_y
```

In [45]:

```
series_days = 56
input_hours = 12
test_hours = 24

train_X, test_X_init, train_y, test_y = \
    (get_train_test_data(df_Beijing, 'PM_Dongsi', series_days,
                         input_hours, test_hours))
```

In [46]:

```
train_X.shape
```

Out[46]:

```
(436, 12, 1)
```

In [47]:

```
train_y.shape
```

Out[47]:

```
(436,)
```

In [48]:

```
test_X_init.shape
```

Out[48]:

```
(12,)
```

In [49]:

```
test_y.shape
```

Out[49]:

```
(12,)
```

Data Modeling

In [50]:

```
def fit_SimpleRNN(train_X, train_y, cell_units, epochs):
    model = Sequential()
    model.add(SimpleRNN(cell_units, input_shape=(train_X.shape[1],1)))
    model.add(Dense(1))
    model.compile(loss='mean_squared_error', optimizer='adam')
    model.fit(train_X, train_y, epochs=epochs, batch_size=64, verbose=0)
    return model
```

In [51]:

```
model = fit_SimpleRNN(train_X, train_y, cell_units=10, epochs=10)
```

In [52]:

```
def predict(X_init, n_steps, model):

    X_init = X_init.copy().reshape(1,-1,1)
    preds = []

    for _ in range(n_steps):
        pred = model.predict(X_init)
        preds.append(pred)
        X_init[:,:-1,:] = X_init[:,1:,:]
        X_init[:,-1,:] = pred

    preds = np.array(preds).reshape(-1,1)

    return preds
```

In [53]:

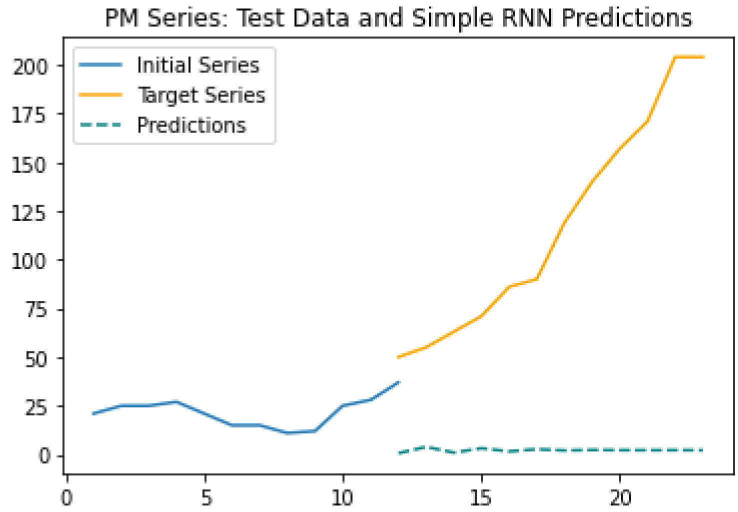
```
def predict_and_plot(X_init, y, model, title):

    y_preds = predict(test_X_init, n_steps=len(y), model=model)
    start_range = range(1, test_X_init.shape[0]+1)
    predict_range = range(test_X_init.shape[0], test_hours)
    plt.plot(start_range, test_X_init)
    plt.plot(predict_range, test_y, 'orange')
    plt.plot(predict_range, y_preds, 'teal', linestyle='--')

    plt.title(title)
    plt.legend(['Initial Series','Target Series','Predictions'])
```

In [41]:

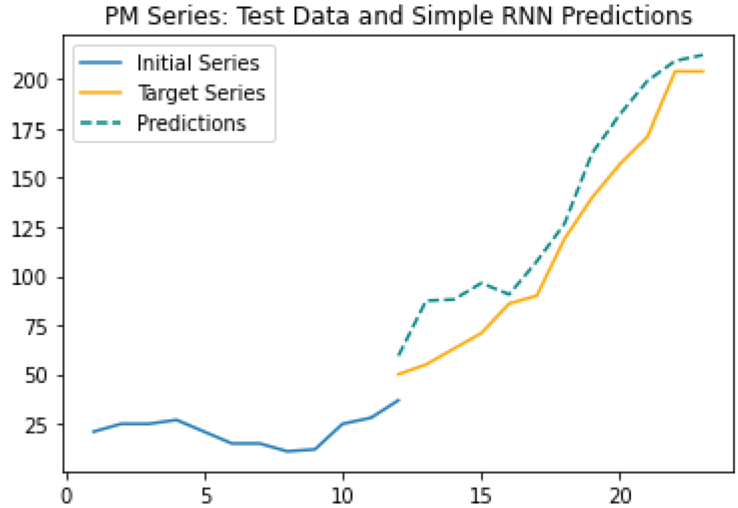
```
predict_and_plot(test_X_init,test_y,model,'PM Series: Test Data and Simple RNN P
```



The predicts is bad. We can also pass over the training data many more times, increasing epochs, giving the model more opportunity to learn the patterns in the data.

In [54]:

```
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 P
```



In [55]:

```
model.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn_6 (SimpleRNN)     (None, 30)                960
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 31
=================================================================
Total params: 991
Trainable params: 991
Non-trainable params: 0
_________________________________________________________________
```

LSTM Modeling

In [60]:

```
def fit_LSTM(train_X, train_y, cell_units, epochs):

    model = Sequential()

    model.add(LSTM(cell_units, input_shape=(train_X.shape[1],1)))

    model.add(Dense(1))

    model.compile(loss='mean_squared_error', optimizer='adam')
    model.fit(train_X, train_y, epochs=epochs, batch_size=64, verbose=0)

    return model
```

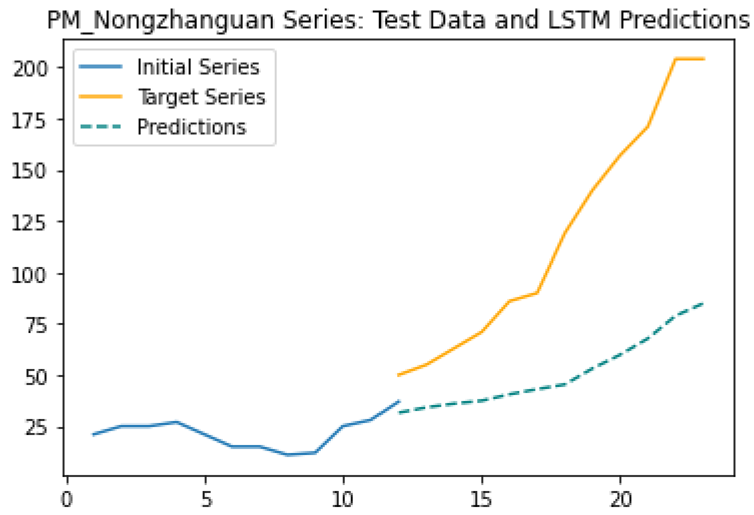
In [65]:

```
series_days = 56
input_hours = 12
test_hours = 24

train_X, test_X_init, train_y, test_y = \
    (get_train_test_data(df_Beijing, 'PM_Dongsi', series_days,
                         input_hours, test_hours))

model = fit_LSTM(train_X, train_y, cell_units=30, epochs=1200)

predict_and_plot(test_X_init, test_y, model,
                 'PM_Nongzhanguan Series: Test Data and LSTM Predictions')
```



LTSM seems not a good way to predict the PM2.5

Insights and key findings

I used two deep learning models(LTSM and RNN) to predict PM2.5 in time series data. The RNNs model perform better than LTSM model to predict data in my attemp. The parameters(more cell units and more epochs) should be tuned for more attemps. Better result than before and it fits the curve more accurately. There are 991 parameters to run so the computation would be slow.

Summary

Flaws and next step

The attemps seems too simple since the time-limited attemps. I should try more parameters in series_days,input_hours,test_hours to figure out the optimal models. Maybe write a loop function to run as much as parameters to find the optimal parameters combination. I can increase the cell_Units and try more training epochs.