HW2 - Time Series Regression

一、 程式碼截圖與講解：

1. 資料前處理：

a、 讀檔，並取出 10.11.12 月資料轉存：

使用 pandas 將檔案讀入，轉存為僅有 10-12 月份資料的新檔案

```
# 1-a Read file and select data from 2018/10/01-12/31
#data = pd.read_excel('E:/DB/HW2/106年新竹站_20180309.xls')
#data = data[data["日期"].between("2017/10/01","2017/12/31")].drop("測站",axis=1)
#data = data.reset_index().drop('index',axis=1)
#data.to_excel("E:/DB/HW2/preprocess_data.xls")

# open the preprocess_data
data = pd.read_excel('E:/DB/HW2/preprocess_data.xls')
```

b、 缺失值及空值以前後一小時平均填補

i. **空值**：全部填入#

ii. **缺失值**：先將 data 依照測項分群，再把各群 flatten 後，將型態為 str 之項，以前後一小時的值填補，若前一小時仍有空值，再取更前一小時)，全數處理後再塞回分群內

c、 以 replace 將 NR 補零

```
# 1-b,1-c
# 1-c replace NR with 0
data = data.replace('NR',0)
data = data.fillna('#')
data = data.groupby(['測項'])
data = [data.get_group(x) for x in data.groups]
data2 = data.copy()

# 1-b complete the loss data
for i in range(18):
    flat,df = [],[]
    flat = data[i].drop(columns = ['日期','測項'])
    df = data[i].loc[:, data[i].columns.intersection(['日期','測項'])].reset_index().drop('index',axis=1)

    flat = flat.values.flatten()
    for j in range(2208):
        if (isinstance(flat[j], str)):
            start = flat[j-1]
            startidx = j
            j = j +1
            while (isinstance(flat[j], str)):
                j = j+1
            end = flat[j]
            endidx = j-1
            for k in (startidx,endidx):
                flat[k] = (start+end)/2
    flat = pd.DataFrame(np.asarray([flat]).reshape(92,24))
    data2[i] = pd.concat([df, flat],axis=1)
```

d、 將資料切成訓練集(10-11 月)與訓練集(12 月)

```
# 1-d split data into training and testing set
data = data2.copy()
data2 = pd.concat([data2[i] for i in range(0,18)],axis=0)

data2['日期'] = pd.to_datetime(data2['日期'])

train = data2[(data2['日期'] >= '2017/10/01 00:00:00') & (data2['日期'] <= '2017/11/30 00:00:00')]
test = data2[(data2['日期'] >= '2017/12/01 00:00:00') & (data2['日期'] <= '2017/12/31 00:00:00')]
```

e、 製作時序資料，row 設成 18 個測項，column 為時序資料

將剛剛的分群資料逐條 concat 回總體的 data

```
# 1-e make data into time series
A = list(data2.測項.unique())
train = train.groupby(['測項'])
train = [train.get_group(x) for x in train.groups]
test = test.groupby(['測項'])
test = [test.get_group(x) for x in test.groups]

for i in range(18):
    flat_train,flat_test = [],[]
    train[i] = list(pd.DataFrame(train[i]).drop(columns = ['日期','測項']).values.flatten())
    test[i] = list(pd.DataFrame(test[i]).drop(columns = ['日期','測項']).values.flatten())

train = pd.concat([pd.DataFrame(train[i] for i in range(18))],axis=0)
test = pd.concat([pd.DataFrame(test[i] for i in range(18))],axis=0)
test.index = A
train.index = A
```

2. 時間序列：

a、 設定 test_y 和 train_y

```
# 2-a setting train_y & test_y
test_y = test.iloc[9][6:1464]
train_y =  train.iloc[9][6:1464]
```

b、 設定 train_x 和 test_x，也設定好只有 pm2.5 版本的 x

```
# 2-b.1 train_x_pm & test_x_pm
train_x_pm = np.array([train.iloc[9][j:j+window].tolist() for j in range(len(train.columns) - window)]).reshape(-1,6)
test_x_pm = np.array([test.iloc[9][j:j+window].tolist() for j in range(len(test.columns) - window)]).reshape(-1,6)


# 2-b.2 train_x & test_x
train_x,test_x = [],[]
for i in range(len(train.columns) - window):
    train_x.append(train.values[:,i:i+window])
train_x = np.asarray(train_x)

for i in range(len(test.columns) - window):
    test_x.append(test.values[:,i:i+window])
test_x = np.asarray(test_x)
```

c、 分別使用 Linear Regression 和 Forest Regression 來建模

```
# 2-c.1 Linear regression
# Create the regression object
regr = linear_model.LinearRegression()
regr_pm = linear_model.LinearRegression()
# Train the model using train_x
regr.fit(train_x.reshape((-1,6*18)), train_y)
train_x_pm = list(train_x_pm)

regr_pm.fit(train_x_pm, train_y)

# Make predictions using the testing set
pred_y = regr.predict(test_x.reshape((-1,6*18)))
pred_y_pm = regr_pm.predict(test_x_pm)
```

```
# 2-c.2 Random Forest Regression
from sklearn.ensemble import RandomForestRegressor

regr = RandomForestRegressor(max_depth=15, random_state=0,n_estimators=100)
regr.fit(train_x.reshape(-1,6*18), train_y)

regr_pm = RandomForestRegressor(max_depth=15, random_state=0,n_estimators=100)
regr_pm.fit(train_x_pm, train_y)

pred_y = regr.predict(test_x.reshape(-1,6*18))
pred_y_pm = regr_pm.predict(test_x_pm)
```

d、 計算這四種組合的 MAE

```
# 2-d.1 Linear regression :The mean absolute error
print("Linear - Mean absolute error: %.2f"
      % mean_absolute_error(test_y, pred_y))
print("Linear_pm - Mean absolute error: %.2f"
      % mean_absolute_error(test_y, pred_y_pm))
```

```
# 2-d.2 Random Forest Regression :The mean absolute error
print("Random Forest - Mean absolute error: %.2f"
      % mean_absolute_error(test_y, pred_y))
print("Random Forest_pm - Mean absolute error: %.2f"
      % mean_absolute_error(test_y, pred_y_pm))
```

二、 程式實作結果：四種 MAE 的計算結果

```
In [1]: runfile('E:/DB/HW2/DMHW2_0753407_劉宥守.py', wdir='E:/DB/HW2')
Linear - Mean absolute error: 2.46
Linear_pm - Mean absolute error: 2.57
Random Forest - Mean absolute error: 3.22
Random Forest_pm - Mean absolute error: 3.07
```

三、 作業心得：

這次作業遇到的三個主要障礙是：

1. 在資料型態上的轉換：array、dataframe、list
2. 從 dataframe 取出 dataframe 子集
3. 資料 rolling 時候的 index 計算

謝謝助教把作業引導寫得這麼仔細，在做的時候是很大的 hint。希望下次做作業的時候可以更順手一點，不要再卡在一些奇怪的問題太久了 QQ

關於實驗結果，不知道為什麼實驗結果裡 Random forest 的結果都比 linear 的結果差，另外也發現，在 Linear 當中有其他數值的幫忙，有效減低 mae，但是在 Random forest tree 當中卻讓 mae 提高了，不知道是不是因為要多考量其他因素而導致樹變得比只有 pm2.5 做為 training data 時複雜。