HW3 Time Series Regression

1. 資料前處理

1.1取出 10.11.12 月資料

```
import pandas as pd

data_csv = pd.read_csv('/content/drive/MyDrive/碩一上課堂/1131_dataMining/dataMining/HW3/_2021.cs
data_csv.columns = data_csv.columns.str.strip()
data_csv['日期'] = pd.to_datetime(data_csv['日期'], errors='coerce')
data_oct_nov_dec = data_csv[data_csv['日期'].dt.month.isin([10, 11, 12])].copy()
data_oct_nov_dec
```

	測站	日期	測項	0	1	2	3	4	5	6	...	14	15	16	17	18	19	20	21	22	23
4915	新竹	2021-10-01	AMB_TEMP	28.3	28.3	27.8	27.8	27.6	27.6	27.7	...	31.6	31.4	30.9	30.5	30.2	29.8	29.4	29.1	28.7	28.2
4916	新竹	2021-10-01	CH4	2.04	2.02	2.12	2.18	2.19	2.24	2.21	...	1.96	1.97	2.01	2.06	2.07	2.05	2.04	2.03	2.08	2.08
4917	新竹	2021-10-01	CO	0.34	0.3	0.3	0.29	0.3	0.33	0.44	...	0.25	0.27	0.32	0.43	0.45	0.45	0.43	0.42	0.43	0.39
4918	新竹	2021-10-01	NMHC	0.17	0.13	0.12	0.14	0.17	0.16	0.18	...	0.04	0.06	0.05	0.17	0.24	0.22	0.16	0.14	0.16	0.14
4919	新竹	2021-10-01	NO	0.9	0.2	0.5	0.4	0.2	0.6	2.2	...	0.5	0.5	0.5	0.3	0.3	0.3	0.3	0.3	0.4	0.6
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
6566	新竹	2021-12-31	THC	2.24	2.22	2.19	2.17	2.15	2.1	2.1	...	2.07	2.05	2.09	2.1	2.05	2.1	2.15	2.13	2.09	2.05
6567	新竹	2021-12-31	WD_HR	38	51	50	47	53	53	46	...	51	54	48	53	54	53	47	37	42	48
6568	新竹	2021-12-31	WIND_DIREC	37	59	37	50	62	42	41	...	66	45	40	59	57	55	41	36	53	39
6569	新竹	2021-12-31	WIND_SPEED	2.6	2.6	2.3	2.4	3.4	3.2	3.1	...	4.8	3.2	2.8	3.2	2.5	2.2	1.7	2.5	2.3	1.9
6570	新竹	2021-12-31	WS_HR	2.5	2	2	2	2.5	2.6	2.6	...	3.8	3.2	2.9	2.8	2.4	2	1.6	2	2.1	1.7

1656 rows × 27 columns

1.2缺失值以及無效值以前後 1 小時平均值取代 (如果前 1 小時仍有空值，再取更前 1 小時)

```
import numpy as np

# 將每小時的數據轉換為數值類型，將無效值轉換為NaN
for hour in range(24):
    data_oct_nov_dec[str(hour)] = pd.to_numeric(data_oct_nov_dec[str(hour)], errors='coerce')

# 針對每個小時欄位，逐行處理缺失值或無效值
for index, row in data_oct_nov_dec.iterrows():
    for hour in range(24):
        if pd.isnull(row[str(hour)]):
            # 定義前後一小時的初始值
            prev_hour = hour - 1
            next_hour = hour + 1
            prev_value = None
            next_value = None

            # 找到前一個有效值（如果有）
            while prev_hour >= 0:
                prev_value = row[str(prev_hour)]
                if not pd.isnull(prev_value):
                    break
```

```
            prev_hour -= 1

        # 找到後一個有效值（如果有）
        while next_hour <= 23:
            next_value = row[str(next_hour)]
            if not pd.isnull(next_value):
                break
            next_hour += 1

        # 計算平均值以替換
        valid_values = []
        if prev_value is not None and not pd.isnull(prev_value):
            valid_values.append(prev_value)
        if next_value is not None and not pd.isnull(next_value):
            valid_values.append(next_value)
        if valid_values:
            data_oct_nov_dec.at[index, str(hour)] = np.mean(valid_values)
```

```
# 1. 檢查是否還有缺失值
missing_values_count = data_oct_nov_dec.isnull().sum().sum()
print(f"資料集中缺失值的數量：{missing_values_count}")
```

1.3NR 表示無降雨，以 0 取代

```
rainfall_rows = data_oct_nov_dec['測項'] == 'RAINFALL'
data_oct_nov_dec.loc[rainfall_rows, [str(hour) for hour in range(24)]] = data_oct_nov_dec.loc[ra
```

1.4將資料切割成訓練集 (10.11月) 以及測試集 (12月)

```
# 將數據切割成訓練集和測試集
train_data = data_oct_nov_dec[data_oct_nov_dec['日期'].dt.month.isin([10, 11])]
test_data = data_oct_nov_dec[data_oct_nov_dec['日期'].dt.month == 12]
```

1.5製作時序資料: 將資料形式轉換為行 (row) 代表18種屬性，欄 (column) 代表逐時數據資料

```
# 選取訓練集的數據
train_data_filtered = train_data.copy()

# 取得屬性列表
attributes = train_data_filtered['測項'].unique()

# 初始化一個空的 DataFrame 以儲存轉換後的資料
time_series_data = pd.DataFrame()

# 將每種屬性分別提取，並將其逐小時數據合併成單一列
for attribute in attributes:
    # 選取該屬性的資料，並將其逐小時數據展開成單一列
    attribute_data = train_data_filtered[train_data_filtered['測項'] == attribute]
    hourly_data = attribute_data.iloc[:, 3:].values.flatten()  # 提取逐小時數據並展開
    time_series_data[attribute] = hourly_data  # 添加為新的 DataFrame 列

# 轉置 DataFrame 使每行代表一個屬性
time_series_data = time_series_data.T
time_series_data.columns = range(61 * 24)  # 將欄名設置為逐時數據的索引
```

```
# 檢查 DataFrame 的形狀是否為 (18, 1464)
print("資料形狀:", time_series_data.shape)

# 顯示前幾行確認是否成功轉換
time_series_data
```

資料形狀: (18, 1464)

	0	1	2	3	4	5	6	7	8	9	...	1454	1455	1456	1457	1458	1459	1460	1461	1462	1463
AMB_TEMP	28.30	28.30	27.80	27.80	27.60	27.60	27.70	28.40	30.00	30.90	...	23.60	23.20	21.70	20.80	20.50	20.30	19.90	19.40	19.00	18.40
CH4	2.04	2.02	2.12	2.18	2.19	2.24	2.21	2.17	2.13	2.07	...	1.98	1.98	2.01	2.08	2.09	2.10	2.10	2.09	2.07	2.05
CO	0.34	0.30	0.30	0.29	0.30	0.33	0.44	0.62	0.60	0.45	...	0.21	0.22	0.31	0.44	0.46	0.46	0.45	0.42	0.37	0.33
NMHC	0.17	0.13	0.12	0.14	0.17	0.16	0.18	0.23	0.20	0.13	...	0.08	0.10	0.11	0.13	0.10	0.11	0.10	0.09	0.08	0.07
NO	0.90	0.20	0.50	0.40	0.20	0.60	2.20	3.60	2.70	1.30	...	1.80	1.60	1.50	1.10	0.80	1.10	1.00	1.10	0.90	0.80
NO2	18.80	11.90	15.10	12.80	14.90	17.50	20.00	22.10	18.10	13.20	...	7.60	9.30	12.80	13.40	12.70	12.50	10.50	8.80	8.20	7.10
NOx	19.80	12.20	15.60	13.20	15.10	18.20	22.30	25.70	20.80	14.60	...	9.50	10.90	14.40	14.50	13.60	13.70	11.50	10.00	9.20	8.00
O3	16.00	21.50	16.90	16.40	12.60	11.30	13.00	15.20	32.70	52.60	...	30.10	29.10	30.40	37.40	36.30	34.90	33.50	33.80	32.60	34.70
PM10	28.00	24.00	29.00	32.00	31.00	32.00	46.00	48.00	58.00	51.00	...	21.00	20.00	33.00	63.00	56.00	58.00	46.00	52.00	54.00	55.00
PM2.5	28.00	22.00	26.00	24.00	28.00	20.00	31.00	37.00	38.00	28.00	...	9.00	11.00	17.00	34.00	34.00	36.00	34.00	32.00	32.00	25.00
RAINFALL	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	...	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
RH	71.00	66.00	76.00	79.00	81.00	78.00	81.00	79.00	74.00	67.00	...	55.00	57.00	67.00	67.00	65.00	65.00	62.00	58.00	56.00	52.00
SO2	1.80	1.00	1.00	1.30	1.50	1.40	1.70	2.50	4.20	3.60	...	2.20	2.00	2.10	2.30	2.30	2.20	2.20	2.10	2.10	2.10
THC	2.21	2.15	2.24	2.32	2.36	2.40	2.39	2.40	2.33	2.20	...	2.06	2.08	2.12	2.21	2.19	2.21	2.20	2.18	2.15	2.12
WD_HR	62.00	121.00	164.00	156.00	110.00	117.00	175.00	199.00	248.00	253.00	...	47.00	45.00	59.00	58.00	53.00	58.00	56.00	47.00	45.00	51.00
WIND_DIREC	33.00	183.00	160.00	151.00	90.00	151.00	212.00	239.00	258.00	246.00	...	55.00	55.00	60.00	59.00	52.00	57.00	54.00	35.00	56.00	52.00
WIND_SPEED	0.80	1.20	1.10	0.60	0.90	0.50	0.80	1.50	1.60	2.10	...	2.70	3.60	3.20	3.60	2.90	3.80	3.80	4.00	4.40	4.60
WS_HR	0.70	0.60	0.60	0.40	0.50	0.40	0.40	0.40	1.00	1.50	...	2.70	2.80	2.90	3.10	2.90	3.40	3.50	3.60	3.80	3.90

18 rows × 1464 columns

2.1 預測目標準備

Y_1hr：預測未來第 1 小時 (Y[0] 為第 6 小時的 PM2.5)

```
# 提取 PM2.5 的逐小時數據（已經轉換為 (18, 1464) 格式後的 time_series_data）
pm25_series = time_series_data.loc['PM2.5'].values

# 構建未來第 1 小時的特徵 (X) 和目標 (Y)
X_1hr = []
Y_1hr = []

# 切割資料，X 為 0~5 小時的資料，Y 為第 6 小時的 PM2.5 值
for i in range(len(pm25_series) - 6):
    X_1hr.append(pm25_series[i:i+6])  # 取 6 小時的數據作為特徵
    Y_1hr.append(pm25_series[i+6])    # 第 6 小時的數據作為目標

# 轉換成 numpy array，方便後續處理
X_1hr = np.array(X_1hr)
Y_1hr = np.array(Y_1hr)
```

Y_6hr：預測未來第 6 小時 (Y[0] 為第 11 小時的 PM2.5)

```
# 構建未來第 6 小時的特徵 (X) 和目標 (Y)
X_6hr = []
Y_6hr = []

# 切割資料，X 為 0~5 小時的資料，Y 為第 11 小時的 PM2.5 值
for i in range(len(pm25_series) - 11):
    X_6hr.append(pm25_series[i:i+6])   # 取 6 小時的數據作為特徵
    Y_6hr.append(pm25_series[i+11])    # 第 11 小時的數據作為目標

# 轉換成 numpy array
```

```
X_6hr = np.array(X_6hr)
Y_6hr = np.array(Y_6hr)
```

X_pm25：只有 PM2.5 (e.g. X[0] 會有 6 個特徵，即第 0 ~ 5 小時的 PM2.5 數值)

```
# 提取 PM2.5 的逐小時數據（此時應該已經清理了多餘空格）
pm25_series = time_series_data.loc['PM2.5'].values

# 構建特徵矩陣 X (只有 PM2.5)
X_pm25 = []

# 切割資料，X 為 0~5 小時的資料
for i in range(len(pm25_series) - 6):
    X_pm25.append(pm25_series[i:i+6])  # 取 6 小時的 PM2.5 數據作為特徵

# 轉換成 numpy array
X_pm25 = np.array(X_pm25)
```

X_all_features：所有 18 種屬性 (e.g. X[0] 會有 18*6 個特徵，即第 0 ~ 5 小時的所有 18 種屬性數值)

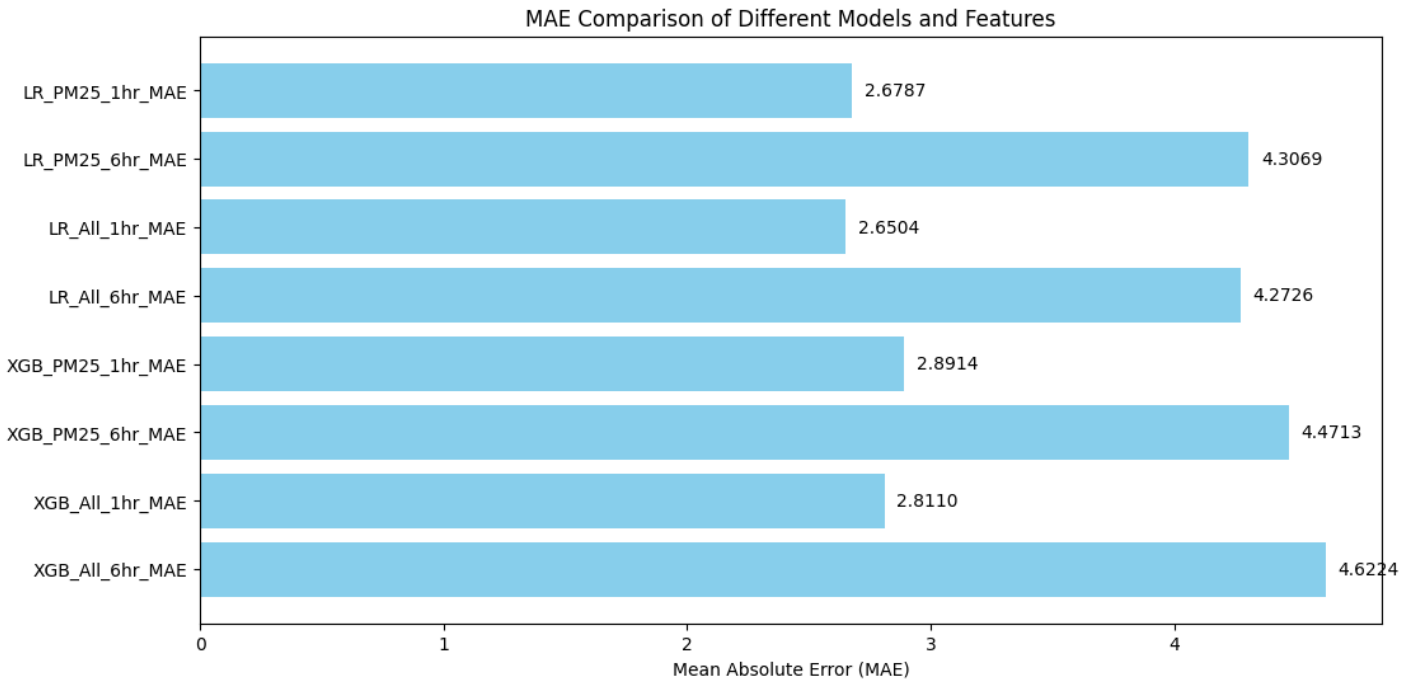
```
# 提取所有屬性的逐小時數據
X_all_features = []

# 切割資料，每次取出 6 小時的所有屬性數據 (18*6 個特徵)
for i in range(len(pm25_series) - 6):
    # 取出 0~5 小時的所有屬性數據並展平成一維
    X_all_features.append(time_series_data.iloc[:, i:i+6].values.flatten())

# 轉換成 numpy array
X_all_features = np.array(X_all_features)
```

模型訓練

```
# 初始化模型
results = {}

# 1. Linear Regression - 單一 PM2.5 特徵
lr_pm25_1hr = LinearRegression().fit(X_pm25_1hr, Y_pm25_1hr)
lr_pm25_6hr = LinearRegression().fit(X_pm25_6hr, Y_pm25_6hr)

results['LR_PM25_1hr_MAE'] = mean_absolute_error(Y_test_pm25_1hr, lr_pm25_1hr.predict(X_test_pm2
results['LR_PM25_6hr_MAE'] = mean_absolute_error(Y_test_pm25_6hr, lr_pm25_6hr.predict(X_test_pm2

# 2. Linear Regression - 全部屬性特徵
lr_all_1hr = LinearRegression().fit(X_all_features_1hr, Y_pm25_1hr)
lr_all_6hr = LinearRegression().fit(X_all_features_6hr, Y_pm25_6hr)

results['LR_All_1hr_MAE'] = mean_absolute_error(Y_test_pm25_1hr, lr_all_1hr.predict(X_test_all_f
results['LR_All_6hr_MAE'] = mean_absolute_error(Y_test_pm25_6hr, lr_all_6hr.predict(X_test_all_f

# 3. XGBoost - 單一 PM2.5 特徵
xgb_pm25_1hr = xgb.XGBRegressor(n_estimators=50, max_depth=2).fit(X_pm25_1hr, Y_pm25_1hr)
xgb_pm25_6hr = xgb.XGBRegressor(n_estimators=50, max_depth=2).fit(X_pm25_6hr, Y_pm25_6hr)

results['XGB_PM25_1hr_MAE'] = mean_absolute_error(Y_test_pm25_1hr, xgb_pm25_1hr.predict(X_test_p
results['XGB_PM25_6hr_MAE'] = mean_absolute_error(Y_test_pm25_6hr, xgb_pm25_6hr.predict(X_test_p
```

```
# 4. XGBoost - 全部屬性特徵
xgb_all_1hr = xgb.XGBRegressor(n_estimators=50, max_depth=2).fit(X_all_features_1hr, Y_pm25_1hr)
xgb_all_6hr = xgb.XGBRegressor(n_estimators=50, max_depth=2).fit(X_all_features_6hr, Y_pm25_6hr)

results['XGB_All_1hr_MAE'] = mean_absolute_error(Y_test_pm25_1hr, xgb_all_1hr.predict(X_test_all
results['XGB_All_6hr_MAE'] = mean_absolute_error(Y_test_pm25_6hr, xgb_all_6hr.predict(X_test_all

# 格式化輸出模型結果
print("模型結果:")
for key, value in results.items():
    print(f"{key}: {value}")
```

```
模型結果:
LR_PM25_1hr_MAE: 2.6786989412341002
LR_PM25_6hr_MAE: 4.306893747261182
LR_All_1hr_MAE: 2.6503923724061837
LR_All_6hr_MAE: 4.272576712123222
XGB_PM25_1hr_MAE: 2.891446951605117
XGB_PM25_6hr_MAE: 4.4712536982266995
XGB_All_1hr_MAE: 2.810995272504605
XGB_All_6hr_MAE: 4.622417306184444
```



結論

- **預測未來1小時的誤差普遍比預測未來6小時的誤差小**：
  - 所有模型在預測1小時時的MAE較小，這是因為預測時間越長，數據的變異性越大，模型越難準確預測。
- **使用所有屬性作為特徵比僅使用PM2.5作為特徵略有提升**：
  - 使用全部屬性特徵的模型（例如 `LR_All_1hr_MAE` 和 `XGB_All_1hr_MAE`）的MAE略低於僅使用PM2.5特徵的模型，表示多特徵有助於提高預測準確性，但改善有限。
- **線性回歸在此情境下的表現優於XGBoost**：
  - 線性回歸的MAE普遍較低，特別是在預測未來6小時時，表示對於這些特徵和目標，線性模型可能更適合。
- **XGBoost模型在長時間預測上表現相對較差**：
  - XGBoost模型的MAE在6小時預測中更高，可能是因為它更適合處理具有高度非線性和交互的數據。