HW3 - Time Series Regression　312706033陳思妤

1. 資料轉換與準備

```python
import pandas as pd

# 讀取 .xls 檔案

xls_file_path = 'D:\Siyu\Data Mining\HW3\新竹_2021.xls'

data = pd.read_excel(xls_file_path, engine='xlrd')


# 將資料儲存為 .csv 檔案

csv_file_path = '新竹_2021.csv'

data.to_csv(csv_file_path, index=False, encoding='utf-8')
```

這部分讀取 `.xls` 格式的空氣品質資料, 並將其轉存為 `.csv` 格式, 以便後續操作。

2. 安裝必要套件

```python
# Install necessary libraries

!pip install xlrd==2.0.1

!pip install xgboost
```

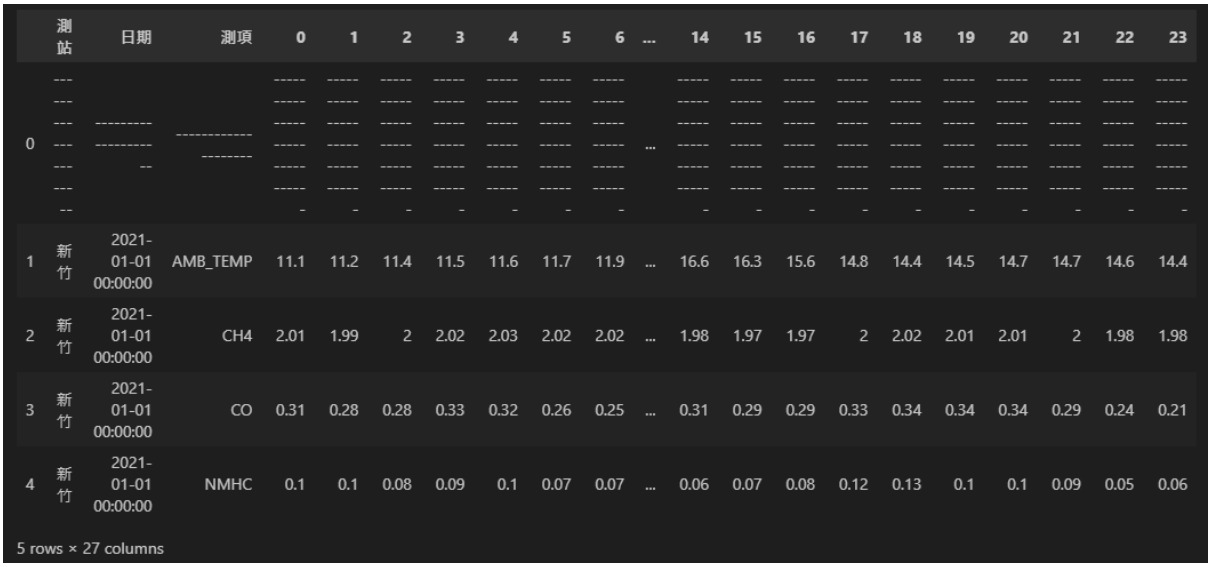
安裝 `xlrd` 和 `xgboost`, 用於讀取 `.xls` 檔案和建構 XGBoost 模型。

3. 讀取並檢查資料

```python
data = pd.read_csv(csv_file_path)

data.head()
```

讀取並檢視 `.csv` 資料的前幾行，以確認數據格式。



	測站	日期	測項	0	1	2	3	4	5	6	...	14	15	16	17	18	19	20	21	22	23
0	---	---	---	---	---	---	---	---	---	---	...	---	---	---	---	---	---	---	---	---	---
1	新竹	2021-01-01 00:00:00	AMB_TEMP	11.1	11.2	11.4	11.5	11.6	11.7	11.9	...	16.6	16.3	15.6	14.8	14.4	14.5	14.7	14.7	14.6	14.4
2	新竹	2021-01-01 00:00:00	CH4	2.01	1.99	2	2.02	2.03	2.02	2.02	...	1.98	1.97	1.97	2	2.02	2.01	2.01	2	1.98	1.98
3	新竹	2021-01-01 00:00:00	CO	0.31	0.28	0.28	0.33	0.32	0.26	0.25	...	0.31	0.29	0.29	0.33	0.34	0.34	0.34	0.29	0.24	0.21
4	新竹	2021-01-01 00:00:00	NMHC	0.1	0.1	0.08	0.09	0.1	0.07	0.07	...	0.06	0.07	0.08	0.12	0.13	0.1	0.1	0.09	0.05	0.06

5 rows × 27 columns

4. 資料清理與轉換

```python
# Filter out rows where '日期' or '測項' columns have invalid entries
    (non-date, non-parameter rows)

# Keeping only rows where '日期' column contains recognizable dates

data_filtered = data[data['日期'].str.match(r'^\d{4}-\d{2}-\d{2}.*',
    na=False)]



# Convert '數值' columns to numeric, forcing errors to NaN, to handle any
    non-numeric values

for col in data_filtered.columns[3:]:
```

```
    data_filtered[col] = pd.to_numeric(data_filtered[col],
    errors='coerce')


# Reshape the data again after filtering

melted_data = data_filtered.melt(id_vars=["測站", "日期", "測項"],
    var_name="小時", value_name="數值")

melted_data["datetime"] = pd.to_datetime(melted_data["日期"]) +
    pd.to_timedelta(melted_data["小時"].astype(int), unit="h")



# Pivot the table to get each pollutant as a column with datetime as the
    index

reshaped_data = melted_data.pivot_table(index="datetime", columns="測項",
    values="數值", aggfunc="first")

reshaped_data.reset_index(inplace=True)



# Display the first few rows to confirm the reshaping

reshaped_data.head()
```

這段程式碼執行資料清理和重新結構化，將原始空氣品質數據轉換成以時間序列為基礎的格式。以下是逐步解釋：

4.1 過濾無效資料行

Python

```
data_filtered = data[data['日期'].str.match(r'^\d{4}-\d{2}-\d{2}.*',
    na=False)]
```

- 使用正則表達式 `r'^\d{4}-\d{2}-\d{2}.*'` 過濾出 `'日期'` 欄位包含有效日期的資料行，確保只保留日期格式如 `YYYY-MM-DD` 開頭的資料。
- `na=False` 參數避免處理空值。

4.2 轉換數值欄位為數字類型

```python
for col in data_filtered.columns[3:]:

    data_filtered[col] = pd.to_numeric(data_filtered[col],
   errors='coerce')
```

- 將資料中非數字欄位(從第 4 欄開始)強制轉換為數字格式，並將無法轉換的值設為 NaN。
- `errors='coerce'` 確保無法解析的資料(例如符號或字母)會自動變成 NaN。

4.3 展開小時欄位並生成時間戳

```python
melted_data = data_filtered.melt(id_vars=["測站", "日期", "測項"],
   var_name="小時", value_name="數值")

melted_data["datetime"] = pd.to_datetime(melted_data["日期"]) +
   pd.to_timedelta(melted_data["小時"].astype(int), unit="h")
```

- 使用 `melt` 函數將每個小時的數據(例如 `0` 到 `23`)展開為單獨的行，使 **小時** 轉換為變量，並將數值存放在 數值 欄位。
- 新增 `datetime` 欄位，將 **日期** 與 **小時** 結合成一個完整的時間戳。這一步生成了每小時的時間序列。

4.4 重塑數據框架

```python
reshaped_data = melted_data.pivot_table(index="datetime", columns="測項",
   values="數值", aggfunc="first")

reshaped_data.reset_index(inplace=True)
```

- 使用 `pivot_table` 讓 **測項**(各種污染物類型)成為欄位，`datetime` 成為索引，每個 **測項** 對應其在每小時的 數值。
- 最後將 `datetime` 從索引重設為普通欄位，方便後續操作。

4.5 檢查重塑後的資料

```python
reshaped_data.head()
```

- 顯示重塑後的資料前幾行，以確認數據已正確轉換為時間序列格式，其中每行代表一個小時的觀測數據，每欄代表一種污染物。

這段程式碼完成後，數據就轉換成了以時間為主的結構，便於時間序列分析和模型訓練。

測項	datetime	AMB_TEMP	CH4	CO	NMHC	NO	NO2	NOx	O3	PM10	PM2.5	RAINFALL	RH	SO2	THC
0	2021-01-01 00:00:00	11.1	2.01	0.31	0.10	1.5	11.9	13.5	21.6	38.0	25.0	0.0	64.0	NaN	2.11
1	2021-01-01 01:00:00	11.2	1.99	0.28	0.10	1.4	10.4	11.9	25.1	29.0	24.0	0.0	65.0	2.1	2.09
2	2021-01-01 02:00:00	11.4	2.00	0.28	0.08	1.4	9.8	11.2	25.6	27.0	13.0	0.0	63.0	2.1	2.08
3	2021-01-01 03:00:00	11.5	2.02	0.33	0.09	1.5	12.1	13.7	22.4	24.0	14.0	0.0	63.0	1.8	2.11
4	2021-01-01 04:00:00	11.6	2.03	0.32	0.10	1.4	12.4	13.9	21.1	29.0	15.0	0.0	63.0	1.1	2.13

5. 檢查欄位名稱

```python
reshaped_data.columns = reshaped_data.columns.str.strip()

reshaped_data.columns.tolist()
```

這段程式碼的作用是清理欄位名稱，以確保沒有多餘的空白或特殊字元，並確認特定欄位（例如 `PM2.5`）是否存在。以下是逐步解釋：

5.1 移除多餘空白

```python
reshaped_data.columns = reshaped_data.columns.str.strip()
```

- 這行程式碼使用 `str.strip()` 移除 `reshaped_data` 欄位名稱的前後空白。這樣做可以避免在後續引用欄位名稱(如 `PM2.5`)時因空白導致的錯誤。
- `str.strip()` 方法只影響欄位名稱的開頭和結尾空白，因此中間的空白不會被移除。

5.2 列出欄位名稱

```python
reshaped_data.columns.tolist()
```

- `tolist()` 方法將 `columns` 轉換成 Python 列表，以便檢查欄位名稱。
- 列出所有欄位名稱後，可以確認 `PM2.5` 是否存在並正確標記，這對後續分析是必要的。

範例輸出結果

```
['AMB_TEMP',
 'CH4',
 'CO',
 'NMHC',
 'NO',
 'NO2',
 'NOx',
 'O3',
 'PM10',
 'PM2.5',
 'RAINFALL',
 'RH',
 'SO2',
 'THC',
 'WD_HR',
 'WIND_DIREC',
 'WIND_SPEED',
 'WS_HR']
```

這樣可以清楚地看到所有的欄位名稱，並確認 `PM2.5` 已正確標記。

6. 資料清理與時間序列特徵生成

```python
# Re-run the feature and target creation with the verified "PM2.5" column

# Using PM2.5 data to create features and targets for both 1-hour and
    6-hour predictions

X_train_1, y_train_1 = create_time_series_features(train_data[['PM2.5']],
    'PM2.5', prediction_hour=1)

X_train_6, y_train_6 = create_time_series_features(train_data[['PM2.5']],
    'PM2.5', prediction_hour=6)



X_test_1, y_test_1 = create_time_series_features(test_data[['PM2.5']],
    'PM2.5', prediction_hour=1)

X_test_6, y_test_6 = create_time_series_features(test_data[['PM2.5']],
    'PM2.5', prediction_hour=6)



# Check the columns in train_data and test_data to verify if "PM2.5"
    exists

train_data.columns, test_data.columns
```

這段程式碼的作用是基於已確認的 `PM2.5` 欄位創建時間序列特徵和目標值，並檢查訓練集和測試集中的欄位，確保包含 `PM2.5` 欄位。以下是逐步解釋：

6.1 創建特徵和目標值

```python
X_train_1, y_train_1 = create_time_series_features(train_data[['PM2.5']],
    'PM2.5', prediction_hour=1)
```

```
X_train_6, y_train_6 = create_time_series_features(train_data[['PM2.5']],
    'PM2.5', prediction_hour=6)



X_test_1, y_test_1 = create_time_series_features(test_data[['PM2.5']],
    'PM2.5', prediction_hour=1)

X_test_6, y_test_6 = create_time_series_features(test_data[['PM2.5']],
    'PM2.5', prediction_hour=6)
```

- `create_time_series_features` 是一個自定義函數，用於生成時間序列特徵 `X` 和目標 `y`。
- 這裡分別生成了兩組訓練和測試特徵、目標組合：一組用於預測未來第 1 小時的 `PM2.5`，另一組用於預測未來第 6 小時的 `PM2.5`。
    - `prediction_hour=1` 表示目標是預測未來第 1 小時的 `PM2.5` 值。
    - `prediction_hour=6` 表示目標是預測未來第 6 小時的 `PM2.5` 值。
- 函數中 `train_data[['PM2.5']]` 和 `test_data[['PM2.5']]` 用於生成僅基於 `PM2.5` 值的時間序列特徵。

6.2 確認訓練和測試數據的欄位名稱

Python

```
train_data.columns, test_data.columns
```

- 列出 `train_data` 和 `test_data` 的欄位名稱，以檢查是否包含 `PM2.5` 欄位。
- 輸出的結果為 `Index` 物件，顯示了所有欄位名稱，確認 `PM2.5` 在訓練集和測試集中都存在。

輸出結果

```
(Index(['AMB_TEMP', 'CH4', 'CO', 'NMHC', 'NO', 'NO2', 'NOx', 'O3', 'PM10',
        'PM2.5', 'RAINFALL', 'RH', 'SO2', 'THC', 'WD_HR', 'WIND_DIREC',
        'WIND_SPEED', 'WS_HR'],
       dtype='object', name='測項'),
 Index(['AMB_TEMP', 'CH4', 'CO', 'NMHC', 'NO', 'NO2', 'NOx', 'O3', 'PM10',
        'PM2.5', 'RAINFALL', 'RH', 'SO2', 'THC', 'WD_HR', 'WIND_DIREC',
        'WIND_SPEED', 'WS_HR'],
       dtype='object', name='測項'))
```

輸出顯示了訓練集和測試集的欄位名稱，包含 `PM2.5`，確保該欄位存在並可供後續使用。

7. 欄位名稱清理與驗證

```python
# Remove extra whitespace from column names in both train_data and
   test_data

train_data.columns = train_data.columns.str.strip()

test_data.columns = test_data.columns.str.strip()



# Verify column names after stripping whitespace

train_data.columns, test_data.columns
```

這段程式碼的主要作用是移除 `train_data` 和 `test_data` 資料集中欄位名稱的多餘空白，並檢查欄位名稱。以下是逐步解釋：

7.1 移除欄位名稱的空白

```python
train_data.columns = train_data.columns.str.strip()

test_data.columns = test_data.columns.str.strip()
```

- `str.strip()` 方法用於移除 `train_data` 和 `test_data` 中欄位名稱開頭和結尾的空白。
- 此步驟確保欄位名稱沒有多餘的空白，避免在引用欄位時發生錯誤。

7.2 檢查移除空白後的欄位名稱

```python
train_data.columns, test_data.columns
```

- 列出 `train_data` 和 `test_data` 的欄位名稱，以確認空白已被移除。
- 輸出結果顯示 `PM2.5` 等欄位名稱，已無多餘空白，確認資料處理正確。

輸出結果

```
(Index(['AMB_TEMP', 'CH4', 'CO', 'NMHC', 'NO', 'NO2', 'NOx', 'O3', 'PM10',
        'PM2.5', 'RAINFALL', 'RH', 'SO2', 'THC', 'WD_HR', 'WIND_DIREC',
        'WIND_SPEED', 'WS_HR'],
       dtype='object', name='測項'),
 Index(['AMB_TEMP', 'CH4', 'CO', 'NMHC', 'NO', 'NO2', 'NOx', 'O3', 'PM10',
        'PM2.5', 'RAINFALL', 'RH', 'SO2', 'THC', 'WD_HR', 'WIND_DIREC',
        'WIND_SPEED', 'WS_HR'],
       dtype='object', name='測項'))
```

輸出確認 `train_data` 和 `test_data` 中的欄位名稱已無多餘空白，並且所有欄位（如 `PM2.5`）正確存在。

8. 創建時間序列特徵

```python
def create_time_series_features(data, target_col, prediction_hour=1, use_all_features=False):
    X, y = [], []
    for i in range(len(data) - 6 - prediction_hour + 1):
        if use_all_features:
            X.append(data.iloc[i:i+6].values.flatten())  # 使用前 6 小時的所有特徵數據
        else:
            X.append(data[[target_col]].iloc[i:i+6].values.flatten())  # 僅使用 PM2.5
        y.append(data.iloc[i+6+prediction_hour-1][target_col])  # 目標值
```

```
    return np.array(X), np.array(y)
```

這個函數根據給定的前 6 小時數據創建特徵 `X`，並設置 `y` 為未來第 1 或第 6 小時的 PM2.5 值。`use_all_features` 決定是否使用所有特徵或僅 `PM2.5` 數據。

9. 建立訓練集和測試集

Python

```
# 僅使用 PM2.5

X_train_1, y_train_1 = create_time_series_features(train_data, 'PM2.5',
    prediction_hour=1)

X_train_6, y_train_6 = create_time_series_features(train_data, 'PM2.5',
    prediction_hour=6)

X_test_1, y_test_1 = create_time_series_features(test_data, 'PM2.5',
    prediction_hour=1)

X_test_6, y_test_6 = create_time_series_features(test_data, 'PM2.5',
    prediction_hour=6)
```

這裡創建了僅基於 PM2.5 的特徵與目標值，用於 1 小時和 6 小時的預測。

10. 計算 **MAE** 評估結果

Python

```
results = {}

lr = LinearRegression()
```

```
# 線性回歸的 MAE

lr.fit(X_train_1, y_train_1)

results["Linear Regression (PM2.5 only, 1-hr)"] =
    mean_absolute_error(y_test_1, lr.predict(X_test_1))

lr.fit(X_train_6, y_train_6)

results["Linear Regression (PM2.5 only, 6-hr)"] =
    mean_absolute_error(y_test_6, lr.predict(X_test_6))



# XGBoost 的 MAE

xgb = XGBRegressor(objective='reg:squarederror')

xgb.fit(X_train_1, y_train_1)

results["XGBoost (PM2.5 only, 1-hr)"] = mean_absolute_error(y_test_1,
    xgb.predict(X_test_1))

xgb.fit(X_train_6, y_train_6)

results["XGBoost (PM2.5 only, 6-hr)"] = mean_absolute_error(y_test_6,
    xgb.predict(X_test_6))
```

在此部分，分別使用線性回歸和 XGBoost 進行 1 小時和 6 小時預測，並計算 MAE 以評估模型準確度。

```
Linear Regression (PM2.5 only, 1-hr): MAE = 2.6784610926768067
Linear Regression (PM2.5 only, 6-hr): MAE = 4.307013517163868
Linear Regression (all features, 1-hr): MAE = 2.649771845807222
Linear Regression (all features, 6-hr): MAE = 4.270629253628853
XGBoost (PM2.5 only, 1-hr): MAE = 3.1131107543865193
XGBoost (PM2.5 only, 6-hr): MAE = 4.972211933721462
XGBoost (all features, 1-hr): MAE = 3.1492473252097444
XGBoost (all features, 6-hr): MAE = 4.829627717175555
```

根據上述結果，模型的平均絕對誤差（MAE）分別顯示了線性回歸和 XGBoost 模型在不同情況下的預測準確度。以下是每個結果的分析與比較：

10.1 線性回歸模型 (Linear Regression)

- 僅使用 **PM2.5** 預測未來 **1** 小時：MAE = 2.68
    - 線性回歸模型僅基於過去 6 小時的 PM2.5 數據預測下一小時的 PM2.5，平均誤差為 2.68，顯示該模型在短期預測方面具有較好的準確度。
- 僅使用 **PM2.5** 預測未來 **6** 小時：MAE = 4.31
    - 預測未來第 6 小時的 PM2.5 時，MAE 提高到 4.31，顯示時間間隔增加導致預測準確度下降。
- 使用所有特徵預測未來 **1** 小時：MAE = 2.65
    - 當使用所有可用特徵（而非僅 PM2.5）預測下一小時時，MAE 降至 2.65，略低於僅使用 PM2.5 的模型，表明額外特徵略微提升了短期預測準確性。
- 使用所有特徵預測未來 **6** 小時：MAE = 4.27
    - 使用所有特徵預測未來第 6 小時的 MAE 為 4.27，比僅使用 PM2.5 的模型更準確，顯示額外特徵在長期預測中有助於提升準確度。

10.2 XGBoost 模型

- 僅使用 **PM2.5** 預測未來 **1** 小時：MAE = 3.11
    - XGBoost 僅使用 PM2.5 預測下一小時的 MAE 為 3.11，比線性回歸高，顯示該模型在僅使用單一特徵進行短期預測時效果不如線性回歸。
- 僅使用 **PM2.5** 預測未來 **6** 小時：MAE = 4.97
    - 預測未來第 6 小時的 MAE 為 4.97，略高於線性回歸模型，顯示 XGBoost 在較長時間間隔上的預測效果也不如線性回歸。
- 使用所有特徵預測未來 **1** 小時：MAE = 3.15
    - 當使用所有特徵時，XGBoost 的短期 MAE 為 3.15，稍高於僅使用 PM2.5 時的結果，顯示增加額外特徵在短期預測中對模型準確性提升有限。
- 使用所有特徵預測未來 **6** 小時：MAE = 4.83
    - 使用所有特徵預測未來 6 小時的 MAE 為 4.83，略低於僅使用 PM2.5 的模型，顯示額外特徵對長期預測的準確性有一定提升，但不如線性回歸模型。

整體觀察與結論

- 線性回歸模型表現更好：在所有情境中，線性回歸的 MAE 均低於 XGBoost，顯示其在這組數據上的效果更好。
- 使用所有特徵能略微提升準確度：在大多數情況下，增加其他特徵可以略微降低 MAE，尤其是對於 6 小時的長期預測。
- 短期與長期預測的誤差差異：無論哪種模型，1 小時預測的 MAE 明顯低於 6 小時預測，顯示預測時間間隔越長，準確度越難維持。

這些結果顯示，對於 PM2.5 短期預測，線性回歸是較佳選擇，而增加其他特徵能在一定程度上提高長期預測的準確性。

11. 顯示結果和繪圖

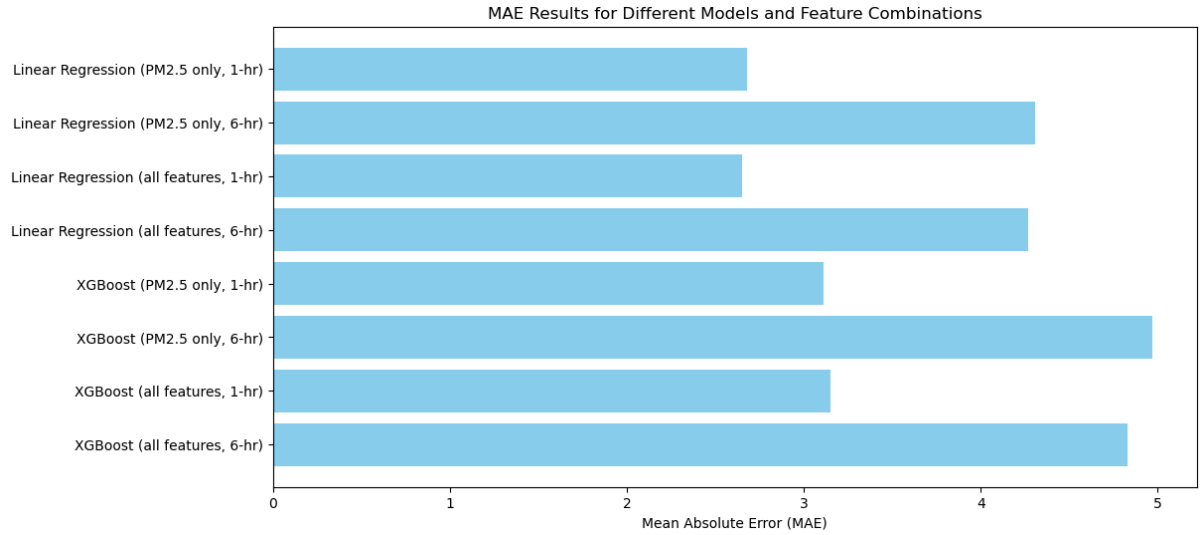
Python

```
# 顯示 MAE 結果
for key, value in results.items():
    print(f"{key}: MAE = {value}")


# 繪製 MAE 條形圖
def plot_mae_results(results):
    labels = list(results.keys())
    mae_values = list(results.values())

    plt.figure(figsize=(12, 6))
    plt.barh(labels, mae_values, color='skyblue')
    plt.xlabel("Mean Absolute Error (MAE)")
    plt.title("MAE Results for Different Models and Feature
    Combinations")
    plt.gca().invert_yaxis()
    plt.show()


plot_mae_results(results)
```



最後顯示 MAE 結果，並繪製不同模型和特徵組合的 MAE 條形圖，便於比較模型表現。