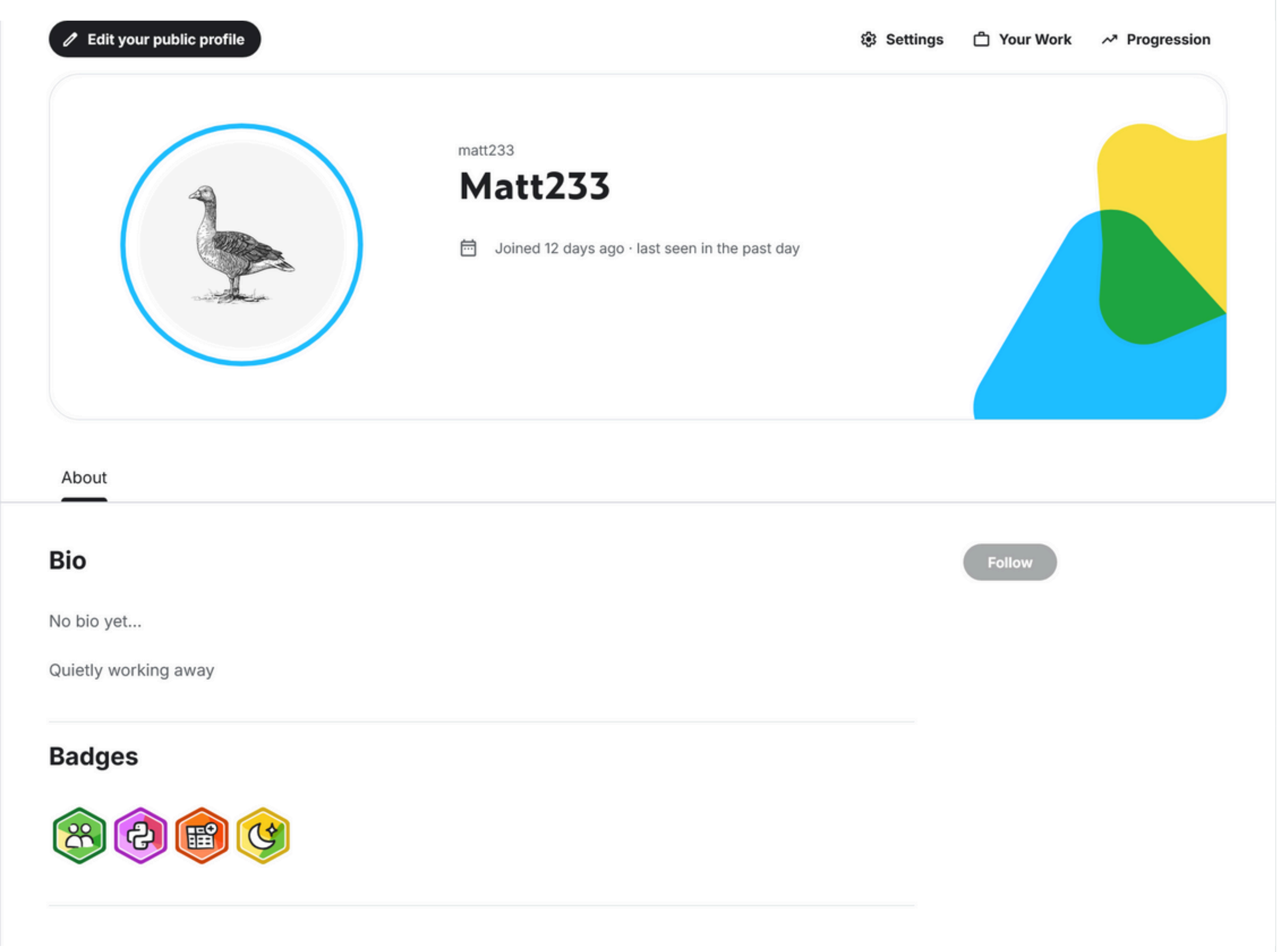
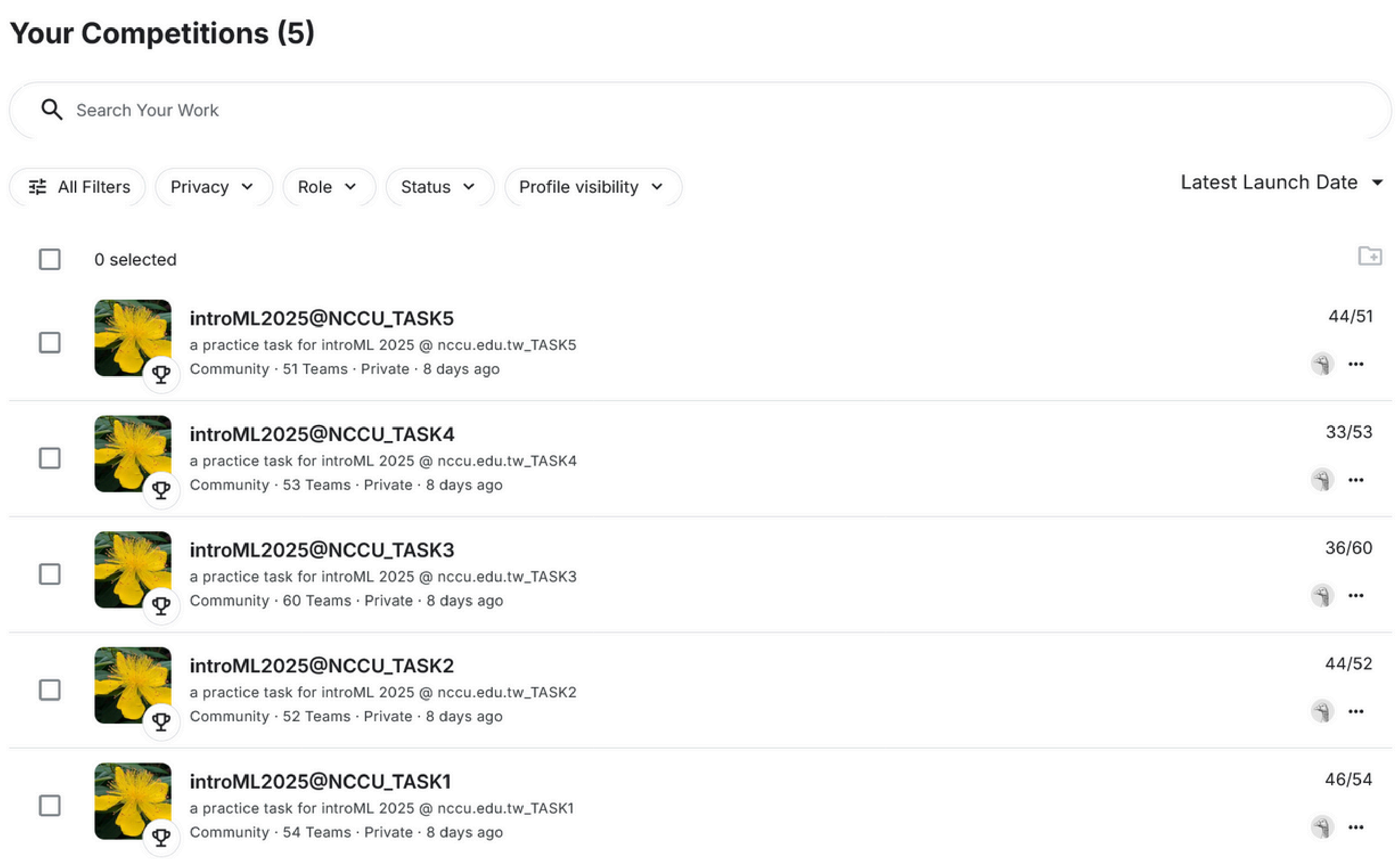
機器學習程式Project

組別名稱：Matt233　　組員：陳俊諺

一、Kaggle參賽介面和五個任務的參與證明





二、程式運作原理及方法

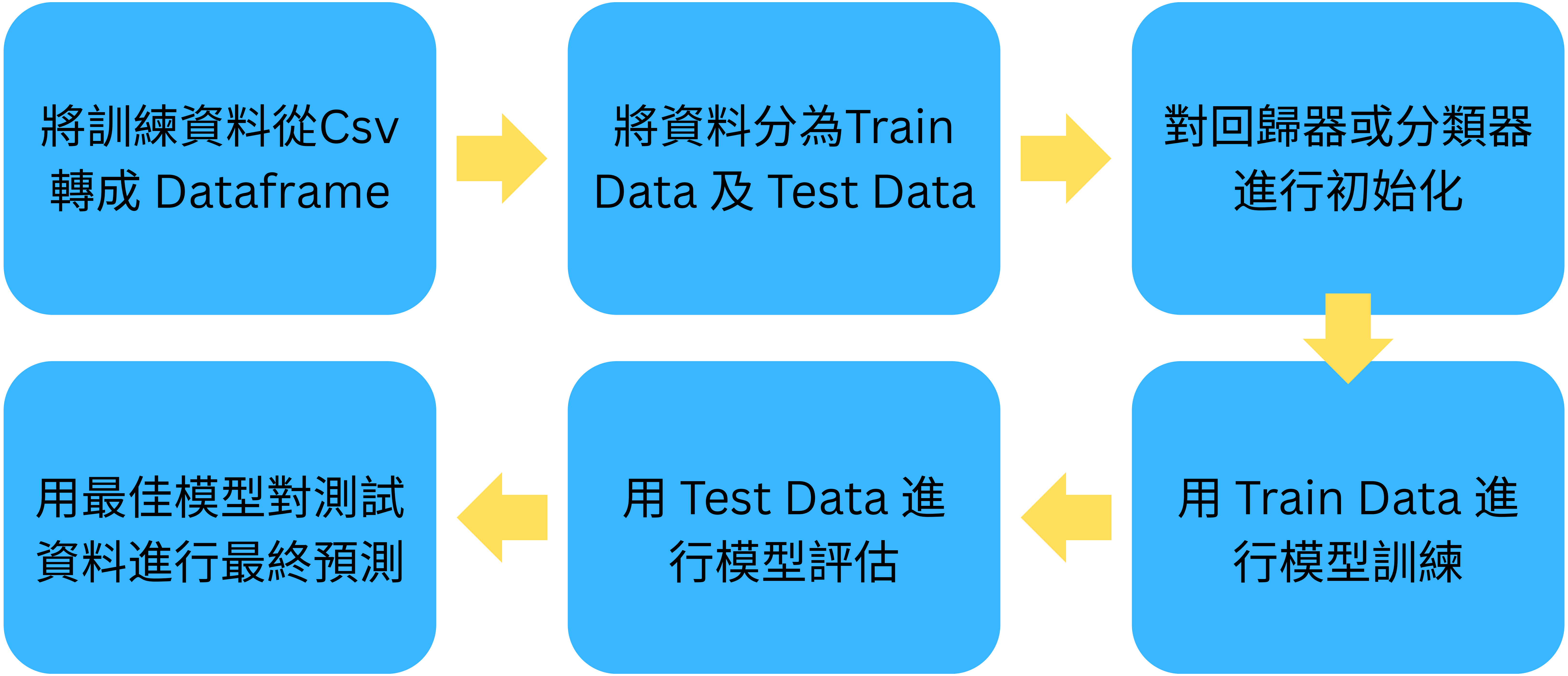
1. GitHub 網址： https://github.com/7m4tt/ml_finalproject

2. 使用到的 Python 套件：
   a. Numpy + Pandas: 用在 Csv 檔和 Dataframe 間的轉換
   b. Scikit-learn + Xgboost: 提供各種回歸器和分類器
   c. Joblib: 用多線程加速運算(不一定要)

3. 將 5 個 Task 分成兩類：
   a. Task 1-3: 使用 11 種不同回歸器進行回歸任務
   b. Task 4-5: 使用 7 種不同的分類器進行分類任務

二、程式運作原理及方法

整體流程圖:



二、程式運作原理及方法

步驟1: 將 Csv 轉成 Dataframe

根據 task 變數的值來讀取不同 Csv 檔， 不同的 Task 的 features 也不同， 需要分別處理。

結束後 X_df 會儲存訓練資料的 features，Y_df 會存 values 。

```
# choose variables
task = "task1"
train_percentage = 0.8
n_jobs = -1

# load dataframe form csv
df = pd.read_csv(f"./data/{task}_train.csv")
if task == "task1":
    X_df = df[[f"x_{i}" for i in range(1, 11)]]
elif task == "task2":
    X_df = df[["x"]]
elif task == "task3":
    X_df = df[[f"x{i}" for i in range(1, 10)]]
y_df = df[["value"]].values.ravel()
```

二、程式運作原理及方法

步驟2: 將 Dataframe 分成 Train Data 和 Test Data

Train_percentage 可以依照變數調整， 目前都是用 80% 訓練資料。
Shuffle 可以讓訓練資料不固定， 讓每次訓練出的模型有些許差異。

結束後 X_train, y_train 會存測試資料的 feature 和 values;
X_test, y_test 會分別存驗證資料，用於後面不同模型的優劣評估。

```
# split dataframe into train and test data
X_train, X_test, y_train, y_test = train_test_split(X_df, y_df, train_size = train_percentage, shuffle = True)
```

二、程式運作原理及方法

步驟3: 初始化使用到的工具

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_10
1.820092671	0.159476041	0.582088152	0.373024294	0.4429733	0.997336...	0.425162...	0.862518...	0.013017...	0.732293...
-2.935780...	-0.334643...	1.923014708	-0.98002979	-0.364581...	0.6311673	0.944345...	0.545469...	0.0[illegible]7425...	0.407760...
-0.544894...	-0.895405...	0.051421313	0.212953514	-0.390282...	0.164976...	0.926398...	0.562035...	0.163946...	0.639655...
-1.11909594	0.715454224	1.362240412	1.133906354	-1.120645...	0.529612...	0.083461...	0.459197...	0.448113...	0.802316...
1.171124068	0.053572605	0.70380436	0.95991199	-0.967464...	0.239100...	0.487664...	0.690517...	0.877919...	0.513217...

1. Pipeline 和 Scaler 目的:
Scaler 確保每個特徵的貢獻度相同；Pipeline則確保每次 Fold 都會初始化一個新的 Scaler，避免不同資料洩漏 (前一次 Fold 的數據影響後面的數據)。

2. GridSearchCV 和 param_grid目的:
有些回歸器沒有提供 CV 模組，可以用這兩個工具進行不同超參數的 Cross Validation，以找到最合適的超參數。

二、程式運作原理及方法

步驟3(續): 使用到的回歸器

1. 線性回歸器: LinearRegression, RidgeCV, LassoCV, ElasticNetCV, LarsCV, BayesianRidge, HuberRegressor

2. 非線性回歸器: RandomForestRegressor, KernelRidge, XGBRegressor, GPRegressor

```
# regressor candidates
candidates = []
if Enable_LinearRegression == True:
    candidates.append(("LinearRegression", LinearRegression(n_jobs = n_jobs)))
if Enable_Ridge == True:
    candidates.append(("Ridge", Pipeline([('scaler', StandardScaler()), ('ridge', RidgeCV(alphas = np.logspace(-6, 6, 13)))])))
if Enable_Lasso == True:
    candidates.append(("Lasso", Pipeline([('scaler', StandardScaler()), ('lasso', LassoCV(alphas = np.logspace(-6, 6, 13), n_jobs = n_jobs))])))
if Enable_ElasticNet == True:
    candidates.append(("ElasticNet", Pipeline([('scaler', StandardScaler()), ('elasticnet', ElasticNetCV(alphas = np.logspace(-6, 6, 13), l1_ratio = [0.1, 0.5, 0.7, 0.9, 0.95, 1.0], n_jobs = n_jobs))])))
if Enable_Lars == True:
    candidates.append(("Lars", LarsCV(cv = 5, n_jobs = n_jobs)))
if Enable_BayesianRidge == True:
    candidates.append(("BayesianRidge", BayesianRidge()))
if Enable_HuberRegressor == True:
    HuberRegressorCV = GridSearchCV(
        estimator = Pipeline([('scaler', StandardScaler()), ('huber', HuberRegressor())]),
        param_grid = {
            "huber__epsilon": [1.1, 1.35, 1.5, 2.0]
        },
        scoring = "neg_mean_squared_error",
        cv = 5,
        n_jobs = n_jobs
    )
    candidates.append(("HuberRegressor", HuberRegressorCV))
if Enable_RandomForestRegressor == True:
    RandomForestRegressorCV = GridSearchCV(
        estimator = RandomForestRegressor(),
        param_grid = {
            "n_estimators": [50, 100, 200],
            "max_depth": [5, 10, None],
            "min_samples_split": [2, 5, 10]
        },
        scoring = "neg_mean_squared_error",
        cv = 5,
        n_jobs = n_jobs
    )
    candidates.append(("RandomForestRegressor", RandomForestRegressorCV))
if Enable_KernelRidge == True:
    KernelRidgeCV = GridSearchCV(
        estimator = KernelRidge(),
        param_grid = {
            "alpha": [0.1, 1.0, 10.0],
            "kernel": ['rbf'],
            "gamma": [0.001, 0.01, 0.1]
        },
        scoring = "neg_mean_squared_error",
        cv = 5,
        n_jobs = n_jobs
    )
    candidates.append(("KernelRidge", KernelRidgeCV))
if Enable_XGBRegressor == True:
    XGBRegressorCV = GridSearchCV(
        estimator = xgb.XGBRegressor(objective = "reg:squarederror"),
        param_grid = {
            'n_estimators': [50, 100, 200],
            'max_depth': [3, 6, None],
            'learning_rate': [0.05, 0.1],
        },
        scoring = "neg_mean_squared_error",
        cv = 5,
        n_jobs = n_jobs
    )
    candidates.append(("XGBRegressor", XGBRegressorCV))
if Enable_GPRegressor == True:
    candidates.append(("GPRegressor", Pipeline([("scaler", StandardScaler()), ('gpregressor', GaussianProcessRegressor(kernel = C(1.0) * RBF(1.0, ((1e-7, 1e5))), n_restarts_optimizer = 10))])))
```

二、程式運作原理及方法

步驟3(續): 使用到的分類器

1. 線性分類器: LogisticRegression

2. 非線性回歸器:
KNeighborsClassifier, SVC,
DecisionTreeClassifier,
RandomForestClassifier,
XGBClassifier,
GradientBoostingClassifier

```python
# regressor candidates
candidates = []
if Enable_LogisticRegression == True:
    LR_CV = GridSearchCV(
        estimator = Pipeline([("scaler", StandardScaler()), ("lr", LogisticRegression(solver = "lbfgs"))]),
        param_grid = {
            "lr__C": np.logspace(-3, 3, 7),
        },
        scoring = "accuracy",
        cv = 5,
        n_jobs = n_jobs
    )
    candidates.append(("LogisticRegression", LR_CV))

if Enable_KNeighborsClassifier == True:
    KNN_CV = GridSearchCV(
        estimator = Pipeline([("scaler", StandardScaler()), ("knn", KNeighborsClassifier())]),
        param_grid = {
            "knn__n_neighbors": [3, 5, 7, 9],
        },
        scoring = "accuracy",
        cv = 5,
        n_jobs = n_jobs
    )
    candidates.append(("KNeighborsClassifier", KNN_CV))

if Enable_SVC == True:
    SVC_CV = GridSearchCV(
        estimator = Pipeline([("scaler", StandardScaler()), ("svc", SVC())]),
        param_grid = {
            "svc__C": [0.1, 1, 10],
            "svc__kernel": ["rbf", "linear"],
        },
        scoring = "accuracy",
        cv = 5,
        n_jobs = n_jobs
    )
    candidates.append(("SVC", SVC_CV))

if Enable_DecisionTreeClassifier == True:
    candidates.append(("DecisionTreeClassifier", DecisionTreeClassifier()))

if Enable_RandomForestClassifier == True:
    RandomForestClassifierCV = GridSearchCV(
        estimator = RandomForestClassifier(),
        param_grid = {
            "n_estimators": [50, 100, 200],
            "max_depth": [5, 10, None],
        },
        scoring = "accuracy",
        cv = 5,
        n_jobs = n_jobs
    )
    candidates.append(("RandomForestClassifier", RandomForestClassifierCV))

if Enable_XGBClassifier == True:
    XGBClassifierCV = GridSearchCV(
        estimator = xgb.XGBClassifier(objective = "multi:softprob", eval_metric = "mlogloss"),
        param_grid = {
            "n_estimators": [50, 100, 200],
            "max_depth": [3, 6, None],
            "learning_rate": [0.05, 0.1],
        },
        scoring = "accuracy",
        cv = 5,
        n_jobs = n_jobs
    )
    candidates.append(("XGBClassifier", XGBClassifierCV))

if Enable_GradientBoostingClassifier == True:
    GradientBoostingClassifierCV = GridSearchCV(
        estimator = GradientBoostingClassifier(),
        param_grid = {
            "n_estimators": [50, 100],
            "max_depth": [3, 5],
        },
        scoring = "accuracy",
        cv = 5,
        n_jobs = n_jobs
    )
    candidates.append(("GradientBoostingClassifier", GradientBoostingClassifierCV))
```

二、程式運作原理及方法

步驟4: 訓練模型

所有被初始化完成的回歸器/分類器會被放進 candidates，其中 candidates[i][0] 存回歸器/分類器的名稱， candidate[i][1]存回歸器/分類器本身，並用 X_train 和 y_train 進行模型訓練。

```python
# train candidates with train data
with parallel_backend("threading", prefer = "threads"):
    for candidate in candidates:
        print(f"Training with {candidate[0]}...")
        candidate[1].fit(X = X_train, y = y_train)
```

parallel_backend 只是為了避免出現 No child process 的 Error，並不引響結果。

二、程式運作原理及方法

步驟5: 用 X_test, y_test 對回歸器進行評估並選出最佳模型

利用已訓練好的模型對 X_test 進行預測，並將該預測和 y_test 做比對計算 MSE，結束後會印出各個模型的 MSE 並選出最佳模型。

```python
# calculate MSE with remaining test data
min_mse = float("inf")
president = None
print(f"{"=" * 50}\nModel Name{" " * 37}MSE\n{"=" * 50}")
for candidate in candidates:
    mse = mean_squared_error(y_true = y_test, y_pred = candidate[1].predict(X_test))
    print(f"{candidate[0]:<33}{mse:.15f}")
    if mse < min_mse:
        min_mse = mse
        president = candidate
print(f"{"=" * 50}\nBest Model: {president[0]}\nMSE = {min_mse}\n{"=" * 50}")
```

```
==================================================
Model Name                                     MSE
==================================================
LinearRegression                 4.406663224953653
Ridge                            4.396128924440905
Lasso                            4.398972134429613
ElasticNet                       4.398972134429613
Lars                             4.394764783574372
BayesianRidge                    4.395168437903259
HuberRegressor                   4.446833164437898
RandomForestRegressor            5.183182550013515
KernelRidge                      4.501712572319430
XGBRegressor                     4.996421576862728
GPRegressor                      15.130199538842215
==================================================
Best Model: Lars
MSE = 4.394764783574372
==================================================
```

二、程式運作原理及方法

步驟5(續): 用 X_test, y_test 對分類器進行評估並選出最佳模型

利用已訓練好的模型對 X_test 進行預測，並將該預測和 y_test 做比對計算 Accuracy，結束後會印出各個模型的 Accuracy 並選出最佳模型。

```python
# calculate accuracy with remaining test data
max_acc = 0.0
president = None
print(f"{"=" * 39}\nModel Name{" " * 21}Accuracy\n{"=" * 39}")
for candidate in candidates:
    acc = accuracy_score(y_true = y_test, y_pred = candidate[1].predict(X_test))
    print(f"{candidate[0]:<33}{acc:.4f}")
    if acc > max_acc:
        max_acc = acc
        president = candidate
print(f"{"=" * 39}\nBest Model: {president[0]}\nAccuracy = {max_acc:.4f}\n{"=" * 39}")
```

```
=======================================
Model Name                     Accuracy
=======================================
LogisticRegression               1.0000
KNeighborsClassifier             1.0000
SVC                              1.0000
DecisionTreeClassifier           0.9988
RandomForestClassifier           0.9994
XGBClassifier                    0.9988
GradientBoostingClassifier       0.9988
=======================================
Best Model: LogisticRegression
Accuracy = 1.0000
=======================================
```

二、程式運作原理及方法

步驟6: 對測試資料進行預測並輸出成 Csv 檔

透過跟步驟1一樣的方式將測試資料載入並轉成 X_df，並用最佳模型對 X_df 進行預測，最後將預測的 y_df 轉成 Csv 檔輸出。

```
# use president to pridect final output
df = pd.read_csv(f"./data/{task}_test.csv")
if task == "task1":
    X_df = df[[f"x_{i}" for i in range(1, 11)]]
elif task == "task2":
    X_df = df[["x"]]
else:
    X_df = df[[f"x{i}" for i in range(1, 10)]]
y_df = president[1].predict(X_df)
output_df = pd.DataFrame({"id": df["id"], "value": y_df})
output_df.to_csv(f"./output/{task}_{president[0]}_{min_mse:.6f}.csv", index = False)
print("finished output.")
```

```
# use president to pridect final output
df = pd.read_csv(f"./data/{task}_test.csv")
if task == "task4":
    X_df = df[[f"x_{i}" for i in range(1, 11)]]
elif task == "task5":
    X_df = df[[f"x_{i}" for i in range(1, 21)]]
y_df = le.inverse_transform(president[1].predict(X_df))
output_df = pd.DataFrame({"id": df["id"], "value": y_df})
output_df.to_csv(f"./output/{task}_{president[0]}_{max_acc:.4f}.csv", index = False)
print("finished output.")
```

報告結束，謝謝大家