Machine learning final project

生醫三 辛明峯

Report abstract

- Data introduction
- Data statistical analysis
- Data initial introduction
- Data preprocessing
- Model training
- conclusion

Data introduction

This data is from the Korea Sports promotion foundation
https://www.bigdata-culture.kr/bigdata/user/data_market/detail.do?id=ace0aea7-5eee-48b9-b616-637365d665c1

```
body_perform = pd.read_csv("C:\\Users\\User\\Desktop\\ML_final\\bodyPerformance.csv")
```

The name of each column:

1. age
2. gender
3. height_cm
4. weight_kg
5. body fat_%
6. diastolic
7. systolic
8. gripForce
9. sit and bend forward_cm
10. sit-ups counts in 2 min
11. broad jump_cm
12. class {"A","B","C","D"} A the best

	age	gender	height_cm	weight_kg	body fat_%	diastolic	systolic	gripForce	sit and bend forward_cm	sit-ups counts	broad jump_cm	class
0	27.0	M	172.3	75.24	21.3	80.0	130.0	54.9	18.4	60.0	217.0	C
1	25.0	M	165.0	55.80	15.7	77.0	126.0	36.4	16.3	53.0	229.0	A
2	31.0	M	179.6	78.00	20.1	92.0	152.0	44.8	12.0	49.0	181.0	C
3	32.0	M	174.5	71.10	18.4	76.0	147.0	41.4	15.2	53.0	219.0	B
4	28.0	M	173.8	67.70	17.1	70.0	127.0	43.5	27.1	45.0	217.0	B

Data introduction

```
body_perform.shape
✓ 0.0s

(13393, 12)
```

```
A 25.08 %
B 25.07 %
C 25.09 %
D 25.09 %
13393
```

There is no missing values
And the data seem vary good?

```
body_perform.info()
✓ 0.1s

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13393 entries, 0 to 13392
Data columns (total 12 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   age                        13393 non-null  float64
 1   gender                     13393 non-null  object
 2   height_cm                  13393 non-null  float64
 3   weight_kg                  13393 non-null  float64
 4   body fat_%                 13393 non-null  float64
 5   diastolic                  13393 non-null  float64
 6   systolic                   13393 non-null  float64
 7   gripForce                  13393 non-null  float64
 8   sit and bend forward_cm    13393 non-null  float64
 9   sit-ups counts             13393 non-null  float64
 10  broad jump_cm              13393 non-null  float64
 11  class                      13393 non-null  object
```

```
body_perform.isnull().sum()
✓ 0.0s

age                        0
gender                     0
height_cm                  0
weight_kg                  0
body fat_%                 0
diastolic                  0
systolic                   0
gripForce                  0
sit and bend forward_cm    0
sit-ups counts             0
broad jump_cm              0
class                      0
dtype: int64
```

Data introduction

But these people definitely pass away

```
body_perform.describe()
```

✓ 0.3s

	age	height_cm	weight_kg	body fat_%	diastolic	systolic	gripForce	sit and bend forward_cm	sit-ups counts	broad jump_cm
count	13393.000000	13393.000000	13393.000000	13393.000000	13393.000000	13393.000000	13393.000000	13393.000000	13393.000000	13393.000000
mean	36.775106	168.559807	67.447316	23.240165	78.796842	130.234817	36.963877	15.209268	39.771224	190.129627
std	13.625639	8.426583	11.949666	7.256844	10.742033	14.713954	10.624864	8.456677	14.276698	39.868000
min	21.000000	125.000000	26.300000	3.000000	0.000000	0.000000	0.000000	-25.000000	0.000000	0.000000
25%	25.000000	162.400000	58.200000	18.000000	71.000000	120.000000	27.500000	10.900000	30.000000	162.000000
50%	32.000000	169.200000	67.400000	22.800000	79.000000	130.000000	37.900000	16.200000	41.000000	193.000000
75%	48.000000	174.800000	75.300000	28.000000	86.000000	141.000000	45.200000	20.700000	50.000000	221.000000
max	64.000000	193.800000	138.100000	78.400000	156.200000	201.000000	70.500000	213.000000	80.000000	303.000000

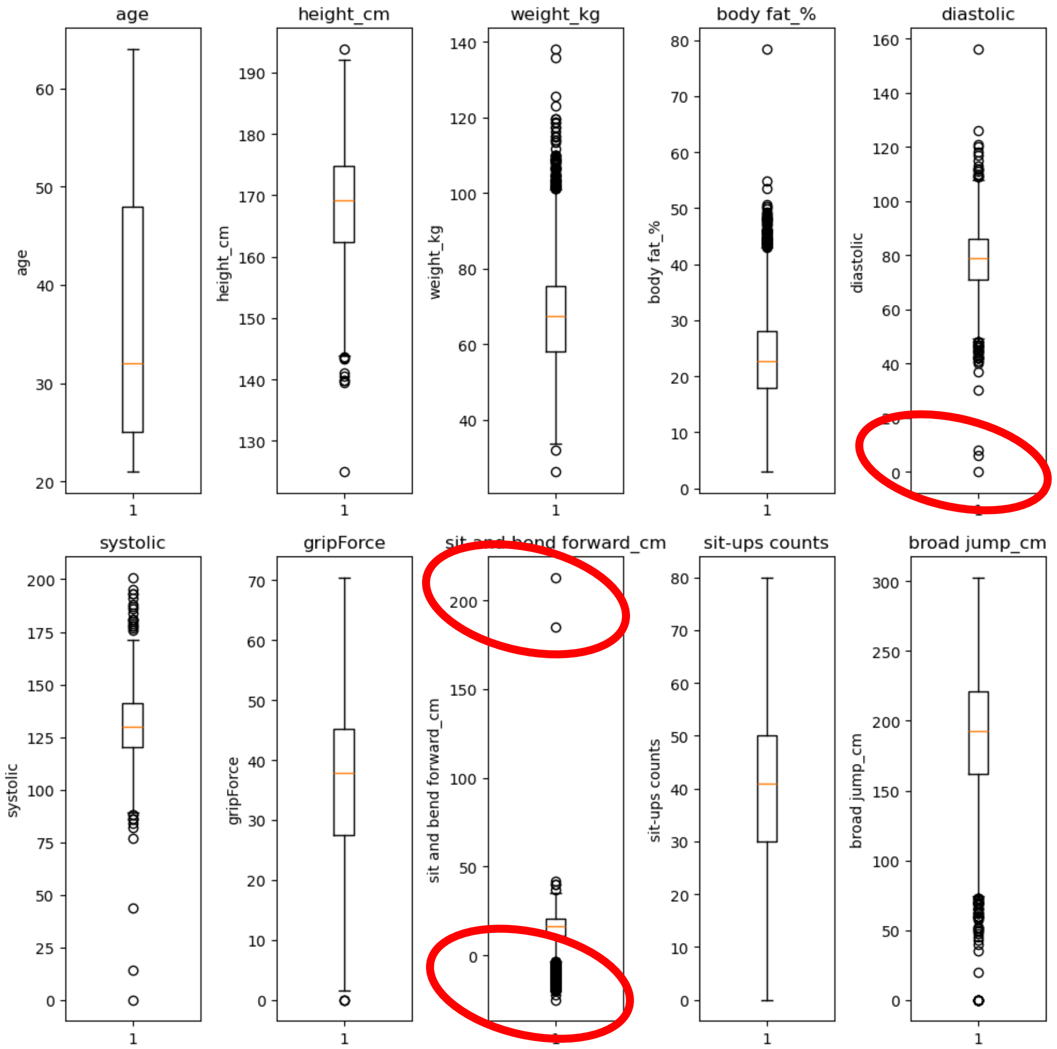
In my experience, I believe that this could be possible

In addition, there are some people who can decrease there size!!!

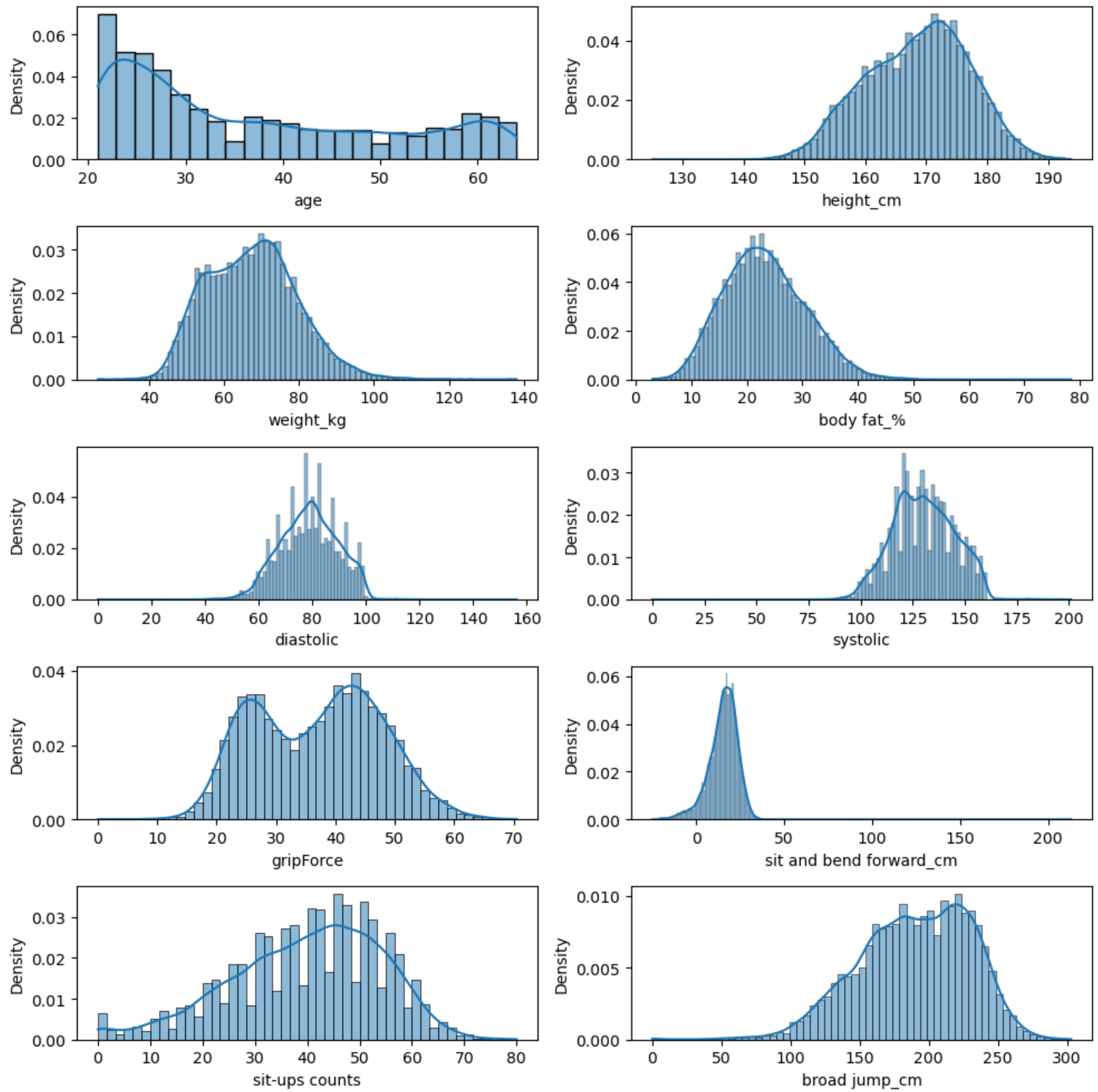
多拉A夢縮小燈～

Data statistical analysis

Box_plot of each column



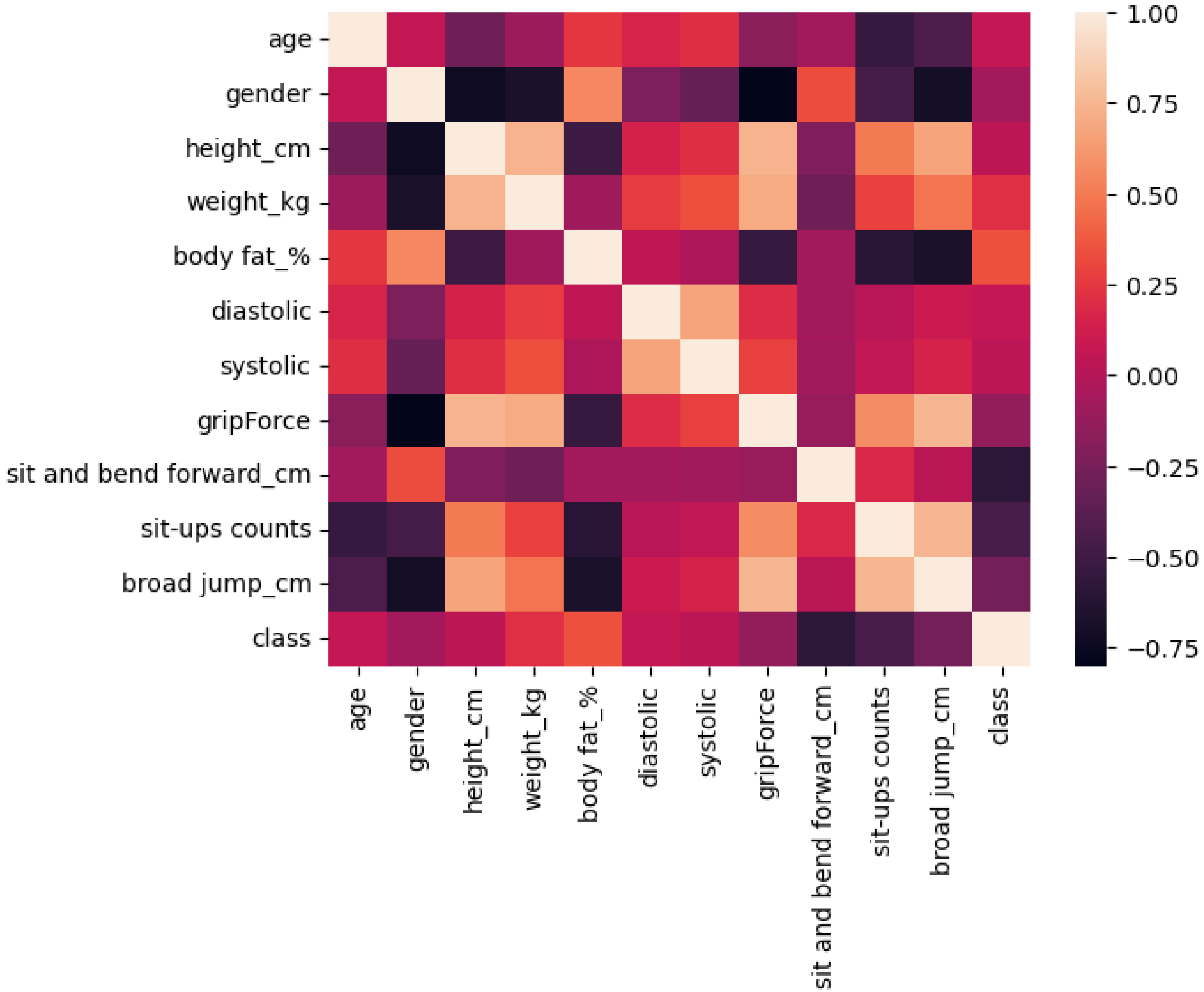
skewness of each column



Data statistical analysis

本当に牙敗です　　沒關西 咬牙硬做看看

```python
sb.heatmap(body_perform.corr())
```

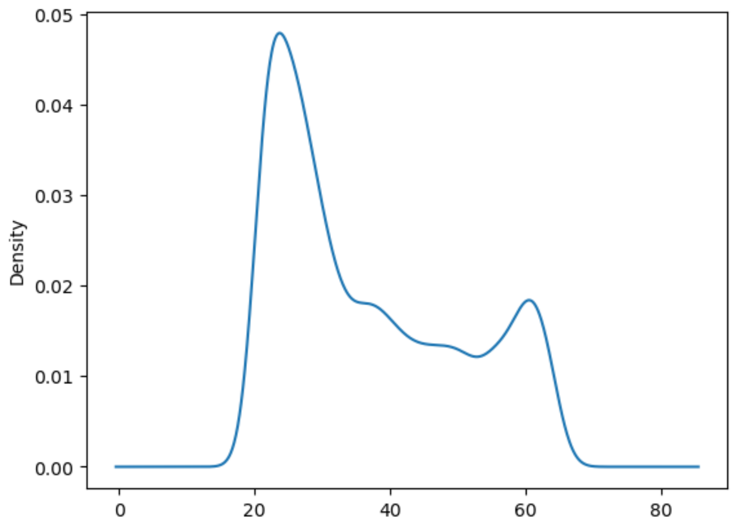



```python
body_perform.corr()["class"]
```

```
age                        0.065612
gender                    -0.075605
height_cm                  0.037753
weight_kg                  0.214129
body fat_%                 0.341956
diastolic                  0.066761
systolic                   0.035484
gripForce                 -0.136088
sit and bend forward_cm   -0.588123
sit-ups counts            -0.452832
broad jump_cm             -0.262154
class                      1.000000
Name: class, dtype: float64
```

```
body_perform['age'].plot(kind = 'density')
```

✓ 0.7s



Data initial introduction

```
# 先看看完全沒有Data cleaning 的資料
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV

X = body_perform.drop("class",axis = 1)
y = body_perform["class"]

knn_params = {'n_neighbors':[1, 2, 3, 4, 5, 6, 7]}
knn = KNeighborsClassifier()
grid = GridSearchCV(knn, knn_params)
grid.fit(X, y)
print(grid.best_score_, grid.best_params_)
```

✓ 7.1s

```
0.573284676741987 7 {'n_neighbors': 7}
```

```
# 初步cleaning 測試 drop unqualified data

from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV

X = body_drop.drop("class",axis = 1)
y = body_drop["class"]
y.shape

knn_params = {'n_neighbors':[1, 2, 3, 4, 5, 6, 7]}
knn = KNeighborsClassifier()
grid = GridSearchCV(knn, knn_params)
grid.fit(X, y)
print(grid.best_score_, grid.best_params_)
```

✓ 6.2s

```
0.5463913950824559 {'n_neighbors': 7}
```

Data initial introduction

```python
# use imputer
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy='mean')

raw_col = ["age","gender","height_cm","weight_kg","body f
body_imputed = imputer.fit_transform(body_perform)
body_imputed = pd.DataFrame(body_imputed, columns = raw_c


X_imputed = body_imputed.drop("class",axis = 1)
y_imputed = body_imputed['class']

knn_params = {'n_neighbors':[1, 2, 3, 4, 5, 6, 7]}
knn = KNeighborsClassifier()
grid = GridSearchCV(knn, knn_params)
grid.fit(X_imputed, y_imputed)
print(grid.best_score_, grid.best_params_)
```

✓ 7.7s

```
0.5470771338273702 {'n_neighbors': 7}
```

```python
# use 0 to fill None
body_zero = body_perform.fillna(0)
X_zero = body_zero.drop('class', axis = 1)
y_zero = body_zero['class']
grid.fit(X_zero, y_zero)
print("learning from {} rows".format(X_zero.
print(grid.best_score_, grid.best_params_)
```

✓ 6.5s

```
learning from 13393 rows
0.5711939821269125 {'n_neighbors': 7}
```

Data initial introduction

I also use some basic strategy to build model, here I make a table to show the performance

Pipeline description	Cross-validated accuracy
Original data	0.57 % n_neighbor = 7
Drop unqualified data	0.54 % n_neighbor = 7
Impute with mean or zero	0.55 % n_neighbor = 7, 0.57 % n_neighbor = 7
logistic regression	**0.57 %**
{'classifier__C': 1.0, 'classifier__penalty': 'l1', 'classifier__solver': 'liblinear', 'imputer__strategy': 'mean', 'scalar': MinMaxScaler()}	
knn	**0.59 %**
{'classify__n_neighbors': 7, 'imputer__strategy': 'median', 'scalar': StandardScaler()}	
decisiontree	**0.67 %**
{'classify__max_depth': 12, 'imputer__strategy': 'mean', 'scalar': StandardScaler()}	
randomforest	**0.73 %**
{'classify__max_depth': None, 'classify__n_estimators': 100, 'imputer__strategy': 'mean', 'scalar': StandardScaler()}	

Data preprocessing

I replace every quantitative value be divided into 5 bins

Work flow 1：
```
pipe = Pipeline([("remover",UN),("imputer",ID)])
```

Work flow 2：
```
pipe = Pipeline([("remover",UN),("imputer",ID),("rank",PR)])
```

Work flow 3：
```
pipe = Pipeline([("remover",UN),("imputer",ID),("addBMI",ALB)])
```

I replace unqualified data to None, including systolic or diastolic pressure = 0 And bend forward = negative

```
class addleanBMI(TransformerMixin):
    def __init__(self,fat='body fat_%',height="he
        self.fat = fat
        self.height = height
        self.weight = weight
    def fit(self,*_):
        return self
    def transform(self,df):
        X = df.copy()
        body_fat = X['body fat_%'] / 100
        height_m = X['height_cm'] / 100
        lean_mass = X['weight_kg'] * (1-body_fat)
```

I add a new data here which is called Lean_BMI(瘦體組織) 類似BMI但體重的計算要去除脂肪重量

Workflow 1 :

```
pipe = Pipeline([("remover",UN),("imputer",ID)])
```

```
,('scalar', StandardScaler()), ('classify',
```

logistic regression	**0.57 %**
{'classifier__C': 1.0, 'classifier__penalty': 'l1', 'classifier__solver': 'liblinear', 'scalar': StandardScaler()}	
knn	**0.62 %**
{'classify__n_neighbors': 7, 'scalar': MinMaxScaler()}	
decisiontree	**0.67 %**
{'classify__max_depth': 11, 'scalar': StandardScaler()}	
randomforest	**0.72 %**
{'classify__max_depth': None, 'classify__n_estimators': 100, 'scalar': MinMaxScaler()}	

logistic regression

```
Best Accuracy: 0.5705219502444956
Best Parameters: {'classifier__C': 1.0, 'c
Average Time to Fit (s): 0.374
Average Time to Score (s): 0.003
```

knn

```
Best Accuracy: 0.6223398261754843
Best Parameters: {'classify__n_nei
Average Time to Fit (s): 0.016
Average Time to Score (s): 0.196
```

randomforest

```
Best Accuracy: 0.7276187624767191
Best Parameters: {'classify__max_
Average Time to Fit (s): 0.469
Average Time to Score (s): 0.013
```

decisiontree

```
Best Accuracy: 0.6665433943812704
Best Parameters: {'classify__max_
Average Time to Fit (s): 0.046
Average Time to Score (s): 0.002
```

Workflow 2 :

```
pipe = Pipeline([("remover",UN),("imputer",ID),("rank",PR)])
```

```
Pipeline([('select_feature',SL),('scalar', StandardScaler()), ('classify',
```

logistic regression	0.53 %
{'classifier__C': 1.0, 'classifier__penalty': 'l1', 'classifier__solver': 'liblinear', 'scalar': MinMaxScaler(), 'select_my_feature__threshold': 0}	
knn	0.55 %
{'classify__n_neighbors': 7, 'scalar': MinMaxScaler(), select_feature__threshold': 0}	
decisiontree	0.57%
{'classify__max_depth': 9, 'scalar': StandardScaler(), 'select_feature__threshold': 0}	
randomforest	0.59 %
{'classify__max_depth': 12, 'classify__n_estimators': 100, 'scalar': MinMaxScaler(), 'select_feature__threshold': 0}	

logistic regression

```
Best Accuracy: 0.534532352841967
Best Parameters: {'classifier__C':
Average Time to Fit (s): 0.13
Average Time to Score (s): 0.003
```

knn

```
Best Accuracy: 0.546403847478O056
Best Parameters: {'classify__n_neighb
Average Time to Fit (s): 0.019
Average Time to Score (s): 0.122
```

decisiontree

```
Best Accuracy: 0.5667145036729397
Best Parameters: {'classify__max_d
Average Time to Fit (s): 0.027
Average Time to Score (s): 0.003
```

randomforest

```
Best Accuracy: 0.5930719693263318
Best Parameters: {'classify__max_dep
Average Time to Fit (s): 0.304
Average Time to Score (s): 0.015
```

Workflow 3 :

```
pipe = Pipeline([("remover",UN),("imputer",ID),("addBMI",ALB)])
Pipeline([('select_feature',SL),('scalar', StandardScaler()), ('classify',
```

logistic regression	**0.57 %**
{'classifier__C': 1.0, 'classifier__penalty': 'l1', 'classifier__solver': 'liblinear', 'imputer__strategy': 'mean', 'scalar': MinMaxScaler()}	
knn	**0.59 %**
{'classify__n_neighbors': 7, 'imputer__strategy': 'median', 'scalar':	
decisiontree	**0.67 %**
{'classify__max_depth': 12, 'imputer__strategy': 'mean', 'scalar':	
randomforest	**0.73 %**
{'classify__max_depth': None, 'classify__n_estimators': 100, 'imputer__strategy': 'mean', 'scalar': StandardScaler()}	

logistic regression

```
Best Accuracy: 0.5741060180682268
Best Parameters: {'classifier__C':
Average Time to Fit (s): 0.957
Average Time to Score (s): 0.003
```

knn

```
Best Accuracy: 0.6165158100469421
Best Parameters: {'classify__n_neighbo
Average Time to Fit (s): 0.018
Average Time to Score (s): 0.125
```

decisiontree

```
Best Accuracy: 0.6684843613968741
Best Parameters: {'classify__max_depth
Average Time to Fit (s): 0.04
Average Time to Score (s): 0.002
```

randomforest

```
Best Accuracy: 0.7246321554446234
Best Parameters: {'classify__max_depth': None,
Average Time to Fit (s): 0.398
Average Time to Score (s): 0.014
```

Other better model

```python
from xgboost import XGBClassifier

xg1 = XGBClassifier()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
pipe = Pipeline([('scalar', StandardScaler()),('classifier',xg1)])
pipe_params = {'scalar':[StandardScaler(), MinMaxScaler()],
               'classifier__n_estimators':[100,200,300],
               'classifier__max_depth':[6,7,8],
               }
# get_best_model_and_accuracy(pipe, pipe_params, X, y)
xg1 = xg1.fit(X_train,y_train)

print_score(xg1,X_train,y_train,X_test,y_test,train=True)
print_score(xg1,X_train,y_train,X_test,y_test,train=False)
```

```python
def print_score(clf, X_train, y_train, X_test, y_test, train=True):
    if train:
        pred = clf.predict(X_train)
        clf_report = pd.DataFrame(classification_report(y_train, pred, output_dict=True))
        print("Train Result:\n================================================")
        print(f"Accuracy Score: {accuracy_score(y_train, pred) * 100:.2f}%")
        print("_______________________________________________")
        print(f"CLASSIFICATION REPORT:\n{clf_report}")
        print("_______________________________________________")
        print(f"Confusion Matrix: \n {confusion_matrix(y_train, pred)}\n")

    elif train==False:
        pred = clf.predict(X_test)
        clf_report = pd.DataFrame(classification_report(y_test, pred, output_dict=True))
        print("Test Result:\n================================================")
        print(f"Accuracy Score: {accuracy_score(y_test, pred) * 100:.2f}%")
        print("_______________________________________________")
        print(f"CLASSIFICATION REPORT:\n{clf_report}")
        print("_______________________________________________")
        print(f"Confusion Matrix: \n {confusion_matrix(y_test, pred)}\n")
```

Workflow 1 :

```
pipe = Pipeline([("remover",UN),("imputer",ID)])
```

```
,('scalar', StandardScaler()), ('classify',
```

```
Train Result:
================================================
Accuracy Score: 96.70%
_______________________________________________
CLASSIFICATION REPORT:
                      0            1            2            3  accuracy  \
precision     0.998663     0.983102     0.958510     0.931329   0.96704
recall        0.971813     0.953361     0.950861     0.993068   0.96704
f1-score      0.985055     0.968003     0.954670     0.961208   0.96704
support    2306.000000  2380.000000  2381.000000  2308.000000   0.96704

             macro avg  weighted avg
precision     0.967901      0.967938
recall        0.967276      0.967040
f1-score      0.967234      0.967138
support    9375.000000   9375.000000
```

```
Test Result:
================================================
Accuracy Score: 75.29%
_______________________________________________
CLASSIFICATION REPORT:
                     0           1           2            3  accuracy  \
precision     0.913866    0.711721    0.624360     0.762697  0.752862
recall        0.834132    0.695562    0.631470     0.837500  0.752862
f1-score      0.872180    0.703549    0.627895     0.798350  0.752862
support    1043.000000  969.000000  966.000000  1040.000000  0.752862

             macro avg  weighted avg
precision     0.753161      0.756385
recall        0.749666      0.752862
f1-score      0.750494      0.753672
support    4018.000000   4018.000000
```

Workflow 3 :

```
pipe = Pipeline([("remover",UN),("imputer",ID),("addBMI",ALB)])
```

```
Pipeline([('select_feature',SL),('scalar', StandardScaler()), ('classify',
```

```
Train Result:
================================================
Accuracy Score: 96.70%
_______________________________________________
CLASSIFICATION REPORT:
                     0            1            2            3  accuracy  \
precision     0.998663     0.983102     0.958510     0.931329   0.96704
recall        0.971813     0.953361     0.950861     0.993068   0.96704
f1-score      0.985055     0.968003     0.954670     0.961208   0.96704
support    2306.000000  2380.000000  2381.000000  2308.000000   0.96704

             macro avg  weighted avg
precision     0.967901      0.967938
recall        0.967276      0.967040
f1-score      0.967234      0.967138
support    9375.000000   9375.000000
```

```
Test Result:
================================================
Accuracy Score: 75.29%
_______________________________________________
CLASSIFICATION REPORT:
                     0           1           2            3  accuracy  \
precision     0.913866    0.711721    0.624360     0.762697  0.752862
recall        0.834132    0.695562    0.631470     0.837500  0.752862
f1-score      0.872180    0.703549    0.627895     0.798350  0.752862
support    1043.000000  969.000000  966.000000  1040.000000  0.752862

             macro avg  weighted avg
precision     0.753161      0.756385
recall        0.749666      0.752862
f1-score      0.750494      0.753672
support    4018.000000   4018.000000
```

conclusion

```
body_perform.corr()["class"]
```

```
age                        0.065612
gender                    -0.075605
height_cm                  0.037753
weight_kg                  0.214129
body fat_%                 0.341956
diastolic                  0.066761
systolic                   0.035484
gripForce                 -0.136088
sit and bend forward_cm   -0.588123
sit-ups counts            -0.452832
broad jump_cm             -0.262154
class                      1.000000
Name: class, dtype: float64
```

- 在查看data的correlation時有很奇妙的現象
- 訓練結果發現，即便做了許多的資料處理都不夠好，我認為要不是feature或imputer做的不好，就是選定的人群，因為老年人和年輕人的分級標準應該要做區分才對
- 未來的改善方針我認為可以更換其他model來做，或是使用其他的特徵萃取方法

希望CC出道當V tuber