ML with sklearn

Author: Simon Kim sxk190106@utdallas.edu

Date:06-11-2022

1.Read the Auto data

a. use pandas to read the data

In [139]:
```
from google.colab import files
myfile = files.upload()
```

파일 선택 선택된 파일 없음

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving Auto.csv to Auto (3).csv
```

In [140]:
```
import io
import pandas as pd
import numpy as np
```

b. output the first few rows

In [141]:
```
data = pd.read_csv(io.BytesIO(myfile['Auto.csv']))
data.head()
```

Out[141]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name
0	18.0	8	307.0	130	3504	12.0	70.0	1	chevrolet chevelle malibu
1	15.0	8	350.0	165	3693	11.5	70.0	1	buick skylark 320
2	18.0	8	318.0	150	3436	11.0	70.0	1	plymouth satellite
3	16.0	8	304.0	150	3433	12.0	70.0	1	amc rebel sst
4	17.0	8	302.0	140	3449	NaN	70.0	1	ford torino

c. output the dimensions of the data

In [142]:
```
print('The size of the DataFrame is: ', data.size)
print('The shape of the DataFrame is: ', data.shape)
print('The dimension of the DataFrame is: ', data.ndim)
```

```
The size of the DataFrame is:  3528
The shape of the DataFrame is:  (392, 9)
The dimension of the DataFrame is:  2
```

1. Data exploration with code

a. use describe() on the mpg, weight, and year columns

In [143]:
```
print('₩nDescribe  mpg, weight, and year:₩n', data.loc[:, ['mpg', 'weight', 'year'
]].describe())
```

```
Describe  mpg, weight, and year:
               mpg       weight        year
count  392.000000   392.000000  390.000000
mean    23.445918  2977.584184   76.010256
std      7.805007   849.402560    3.668093
min      9.000000  1613.000000   70.000000
25%     17.000000  2225.250000   73.000000
50%     22.750000  2803.500000   76.000000
75%     29.000000  3614.750000   79.000000
max     46.600000  5140.000000   82.000000
```

b. write comments indicating the range and average of each column

In [144]:
```
print('₩nRange of mpg:₩t₩t', data['mpg'].max() - data['mpg'].min())
print('Range of weight:₩t', data['weight'].max() - data['weight'].min())
print('Range of year:₩t₩t', data['year'].max() - data['year'].min())
print('₩n')
print('Average of mpg:₩t', data['mpg'].mean())
print('Average of weight:₩t', data['weight'].mean())
print('Average of year:₩t',data['year'].mean())
```

```
Range of mpg:            37.6
Range of weight:         3527
Range of year:           12.0


Average of mpg:  23.445918367346938
Average of weight:       2977.5841836734694
Average of year:         76.01025641025642
```

1. Explore data types

a. check the data types of all columns

```
In [145]: data.dtypes
```

```
Out[145]: mpg             float64
          cylinders         int64
          displacement    float64
          horsepower        int64
          weight            int64
          acceleration    float64
          year            float64
          origin            int64
          name             object
          dtype: object
```

b. change the cylinders column to categorical (use cat.codes)

```
In [146]: data.cylinders = data.cylinders.astype('category').cat.codes
          print(data.dtypes, "₩n")
          data.head()
```

```
          mpg             float64
          cylinders          int8
          displacement    float64
          horsepower        int64
          weight            int64
          acceleration    float64
          year            float64
          origin            int64
          name             object
          dtype: object
```

Out[146]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name
0	18.0	4	307.0	130	3504	12.0	70.0	1	chevrolet chevelle malibu
1	15.0	4	350.0	165	3693	11.5	70.0	1	buick skylark 320
2	18.0	4	318.0	150	3436	11.0	70.0	1	plymouth satellite
3	16.0	4	304.0	150	3433	12.0	70.0	1	amc rebel sst
4	17.0	4	302.0	140	3449	NaN	70.0	1	ford torino

c. change the origin column to categorical (don't use cat.codes)

In [147]:
```
data.origin = data.origin.astype('category')

print(data.dtypes, "₩n")
data.head()
```

```
mpg              float64
cylinders           int8
displacement     float64
horsepower         int64
weight             int64
acceleration     float64
year             float64
origin          category
name              object
dtype: object
```

Out[147]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name
0	18.0	4	307.0	130	3504	12.0	70.0	1	chevrolet chevelle malibu
1	15.0	4	350.0	165	3693	11.5	70.0	1	buick skylark 320
2	18.0	4	318.0	150	3436	11.0	70.0	1	plymouth satellite
3	16.0	4	304.0	150	3433	12.0	70.0	1	amc rebel sst
4	17.0	4	302.0	140	3449	NaN	70.0	1	ford torino

d. verify the changes with the dtypes attribute

In [148]:
```
data.dtypes
```

Out[148]:
```
mpg              float64
cylinders           int8
displacement     float64
horsepower         int64
weight             int64
acceleration     float64
year             float64
origin          category
name              object
dtype: object
```

1. Deal with NAs

a. delete rows with NAs

```
In [149]: data=data.dropna()
          data.head()
```

Out[149]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name
0	18.0	4	307.0	130	3504	12.0	70.0	1	chevrolet chevelle malibu
1	15.0	4	350.0	165	3693	11.5	70.0	1	buick skylark 320
2	18.0	4	318.0	150	3436	11.0	70.0	1	plymouth satellite
3	16.0	4	304.0	150	3433	12.0	70.0	1	amc rebel sst
6	14.0	4	454.0	220	4354	9.0	70.0	1	chevrolet impala

b. output the new dimensions

```
In [150]: print('The size of the new DataFrame is: ', data.size)
          print('The shape of the new DataFrame is: ', data.shape)
          print('The dimension of the new DataFrame is: ', data.ndim)
```

```
The size of the new DataFrame is:  3501
The shape of the new DataFrame is:  (389, 9)
The dimension of the new DataFrame is:  2
```

1. Modify columns

a. make a new column, mpg_high, and make it categorical: i. the column == 1 if mpg > average mpg, else == 0

In [151]:
```python
data['mpg_high'] = ['1' if t else '0' for t in list(data['mpg'] > data['mpg'].mean
())]
data.head()
```

Out[151]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name	mp
0	18.0	4	307.0	130	3504	12.0	70.0	1	chevrolet chevelle malibu	
1	15.0	4	350.0	165	3693	11.5	70.0	1	buick skylark 320	
2	18.0	4	318.0	150	3436	11.0	70.0	1	plymouth satellite	
3	16.0	4	304.0	150	3433	12.0	70.0	1	amc rebel sst	
6	14.0	4	454.0	220	4354	9.0	70.0	1	chevrolet impala	

b. delete the mpg and name columns (delete mpg so the algorithm doesn’t just learn to predict mpg_high from mpg)

In [152]:
```python
data = data.drop(['mpg', 'name'], axis=1)
```

c. output the first few rows of the modified data frame

In [153]:
```python
data.head()
```

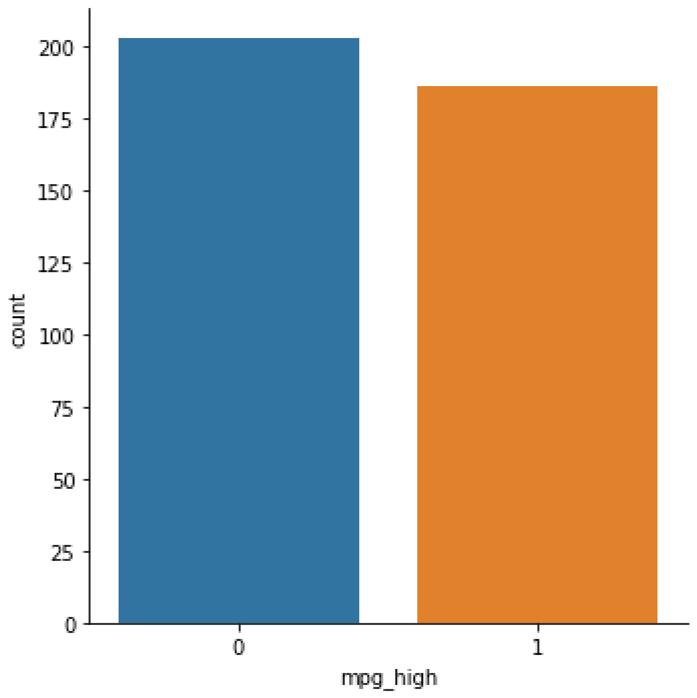
Out[153]:

	cylinders	displacement	horsepower	weight	acceleration	year	origin	mpg_high
0	4	307.0	130	3504	12.0	70.0	1	0
1	4	350.0	165	3693	11.5	70.0	1	0
2	4	318.0	150	3436	11.0	70.0	1	0
3	4	304.0	150	3433	12.0	70.0	1	0
6	4	454.0	220	4354	9.0	70.0	1	0

1. Data exploration with graphs

a. seaborn catplot on the mpg_high column

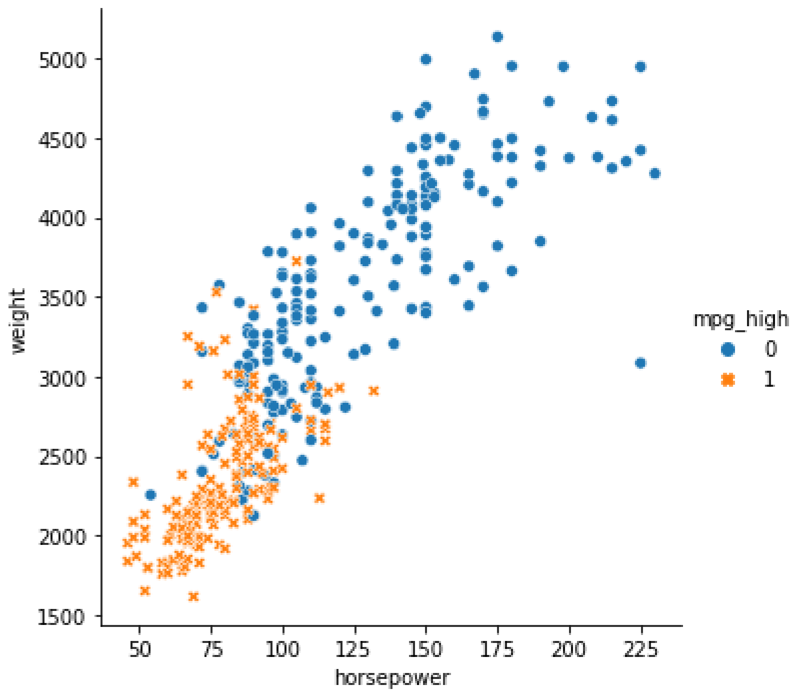
```
In [154]: import seaborn as sb
          sb.catplot(x="mpg_high", kind='count', data=data)

Out[154]: <seaborn.axisgrid.FacetGrid at 0x7f52b29ce690>
```



From this graph, the number of vehicles over average is smaller than the vehicles which are below average. Therefore, we can say that the average is higher than the median value.

b. seaborn relplot with horsepower on the x axis, weight on the y axis, setting hue or style to mpg_high

```
In [155]: sb.relplot(x='horsepower', y='weight', data=data, hue=data.mpg_high, style=data.mpg
          _high)
Out[155]: <seaborn.axisgrid.FacetGrid at 0x7f5252650690>
```



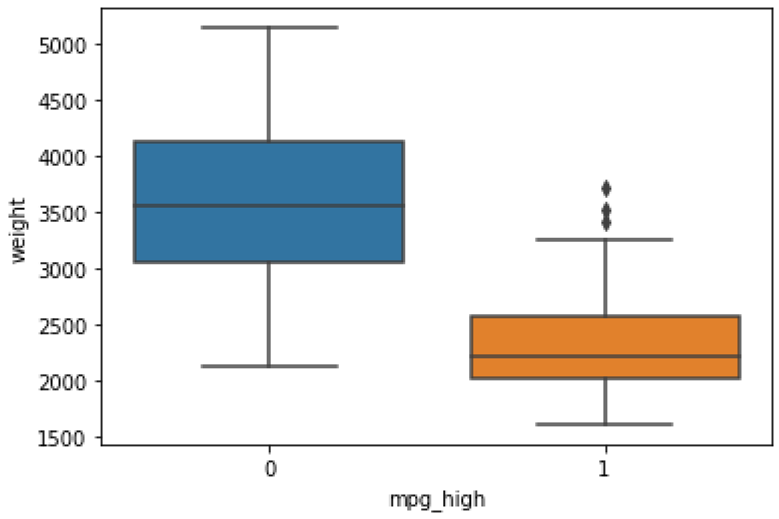
From the graph, smaller weight, and smaller horsepower, then the vehicle might exceed the average.

c. seaborn boxplot with mpg_high on the x axis and weight on the y axis

In [156]:
```python
sb.boxplot('mpg_high', y='weight', data=data)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pas
s the following variable as a keyword arg: x. From version 0.12, the only valid posi
tional argument will be `data`, and passing other arguments without an explicit keyw
ord will result in an error or misinterpretation.
  FutureWarning
```

Out[156]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f525263a7d0>
```



From the graph, we can say that the larger the weight, the more likely it is to be in a range smaller than average

1. Train/test split

a. 80/20

b. use seed 1234 so we all get the same results

c. train /test X data frames consists of all remaining columns except mpg_high

d. output the dimensions of train and test

In [157]:
```python
from sklearn.model_selection import train_test_split
X = data.loc[:, ['cylinders', 'displacement', 'horsepower','weight','acceleration',
'year','origin']]
y = data.mpg_high

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
te=1234)

print('train size:', X_train.shape)
print('test size:', X_test.shape)
```

```
train size: (311, 7)
test size: (78, 7)
```

1. Logistic Regression

a. train a logistic regression model using solver lbfgs

In [158]:
```
from sklearn.linear_model import LogisticRegression

lr_model = LogisticRegression(solver='lbfgs',max_iter=400).fit(X_train, y_train)
lr_model.score(X_train, y_train)
```

Out[158]:
```
0.9035369774919614
```

b. test and evaluate

In [159]:
```
# make predictions
pred = lr_model.predict(X_test)

# evaluate
from sklearn.metrics import accuracy_score

print('accuracy score: ', accuracy_score(y_test, pred))
```

```
accuracy score:  0.8974358974358975
```

c. print metrics using the classification report

In [160]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, pred)
```

Out[160]:
```
array([[42,  8],
       [ 0, 28]])
```

1. Decision Tree

a. train a decision tree

In [161]:
```
from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)
```

Out[161]:
```
DecisionTreeClassifier()
```

b. test and evaluate

In [162]:
```
# make predictions

pred = clf.predict(X_test)
# evaluate
from sklearn.metrics import accuracy_score
print('accuracy score: ', accuracy_score(y_test, pred))
```
```
accuracy score:  0.8846153846153846
```

c. print the classification report metrics

In [166]:
```
# confusion matrix
from sklearn.metrics import confusion_matrix

confusion_matrix(y_test, pred)
```
Out[166]:
```
array([[46,  4],
       [ 5, 23]])
```

1. Neural Network

a. train a neural network, choosing a network topology of your choice

In [167]:
```
# normalize the data
from sklearn import preprocessing

scaler = preprocessing.StandardScaler().fit(X_train)

X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)

# train
from sklearn.neural_network import MLPClassifier

clf2 = MLPClassifier(solver='lbfgs', hidden_layer_sizes=(5, 2), max_iter=500, rando
m_state=1234)
clf2.fit(X_train_scaled, y_train)
```
Out[167]:
```
MLPClassifier(hidden_layer_sizes=(5, 2), max_iter=500, random_state=1234,
              solver='lbfgs')
```

b. test and evaluate

In [168]:
```
# make predictions

pred = clf2.predict(X_test_scaled)
# output results

print('accuracy = ', accuracy_score(y_test, pred))

confusion_matrix(y_test, pred)
```

```
accuracy =  0.8717948717948718
```

Out[168]:
```
array([[43,  7],
       [ 3, 25]])
```

c. train a second network with a different topology and different settings

In [169]:
```
import keras

# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, 2)
y_test = keras.utils.to_categorical(y_test, 2)
```

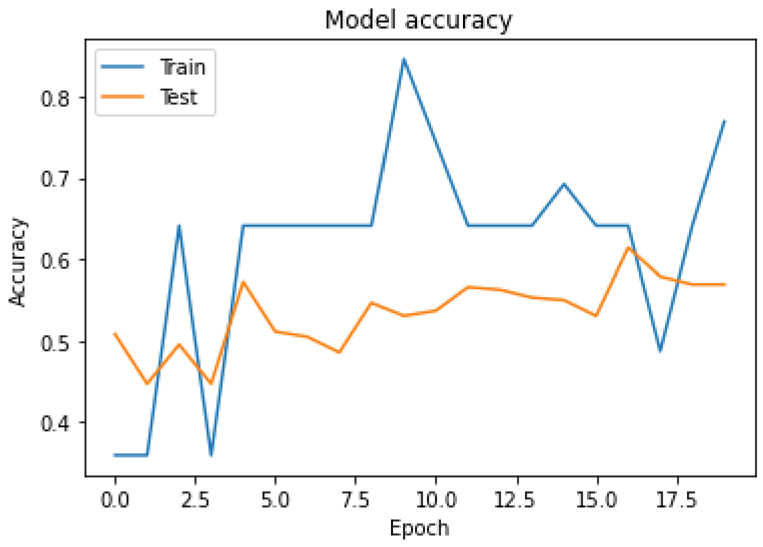
In [170]:
```
from __future__ import print_function

import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.optimizers import RMSprop

batch_size = 128
epochs = 20

model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(7,)))
model.add(Dropout(0.2))
#model.add(Dense(512, activation='relu'))
#model.add(Dropout(0.2))
model.add(Dense(2, activation='sigmoid'))
model.compile(loss='binary_crossentropy',
              optimizer=RMSprop(),
              metrics=['accuracy'])

history = model.fit(X_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(X_test, y_test))
```

```
Epoch 1/20
3/3 [==============================] - 1s 135ms/step - loss: 57.9060 - accuracy: 0.5
080 - val_loss: 18.2723 - val_accuracy: 0.3590
Epoch 2/20
3/3 [==============================] - 0s 33ms/step - loss: 41.3568 - accuracy: 0.44
69 - val_loss: 29.3937 - val_accuracy: 0.3590
Epoch 3/20
3/3 [==============================] - 0s 22ms/step - loss: 36.1711 - accuracy: 0.49
52 - val_loss: 29.0305 - val_accuracy: 0.6410
Epoch 4/20
3/3 [==============================] - 0s 23ms/step - loss: 48.5874 - accuracy: 0.44
69 - val_loss: 15.1257 - val_accuracy: 0.3590
Epoch 5/20
3/3 [==============================] - 0s 22ms/step - loss: 31.0299 - accuracy: 0.57
23 - val_loss: 5.1042 - val_accuracy: 0.6410
Epoch 6/20
3/3 [==============================] - 0s 23ms/step - loss: 35.5256 - accuracy: 0.51
13 - val_loss: 10.0880 - val_accuracy: 0.6410
Epoch 7/20
3/3 [==============================] - 0s 28ms/step - loss: 33.7642 - accuracy: 0.50
48 - val_loss: 7.4312 - val_accuracy: 0.6410
Epoch 8/20
3/3 [==============================] - 0s 24ms/step - loss: 39.3910 - accuracy: 0.48
55 - val_loss: 16.2014 - val_accuracy: 0.6410
Epoch 9/20
3/3 [==============================] - 0s 25ms/step - loss: 30.9643 - accuracy: 0.54
66 - val_loss: 12.2723 - val_accuracy: 0.6410
Epoch 10/20
3/3 [==============================] - 0s 37ms/step - loss: 34.1679 - accuracy: 0.53
05 - val_loss: 12.4899 - val_accuracy: 0.8462
Epoch 11/20
3/3 [==============================] - 0s 25ms/step - loss: 29.6464 - accuracy: 0.53
70 - val_loss: 4.2760 - val_accuracy: 0.7436
Epoch 12/20
3/3 [==============================] - 0s 21ms/step - loss: 27.9873 - accuracy: 0.56
59 - val_loss: 13.6753 - val_accuracy: 0.6410
Epoch 13/20
3/3 [==============================] - 0s 24ms/step - loss: 32.4994 - accuracy: 0.56
27 - val_loss: 10.3959 - val_accuracy: 0.6410
Epoch 14/20
3/3 [==============================] - 0s 22ms/step - loss: 31.6041 - accuracy: 0.55
31 - val_loss: 12.9684 - val_accuracy: 0.6410
Epoch 15/20
3/3 [==============================] - 0s 21ms/step - loss: 30.4635 - accuracy: 0.54
98 - val_loss: 12.1871 - val_accuracy: 0.6923
Epoch 16/20
3/3 [==============================] - 0s 31ms/step - loss: 31.2342 - accuracy: 0.53
05 - val_loss: 15.3447 - val_accuracy: 0.6410
Epoch 17/20
3/3 [==============================] - 0s 24ms/step - loss: 27.1544 - accuracy: 0.61
41 - val_loss: 7.9609 - val_accuracy: 0.6410
Epoch 18/20
3/3 [==============================] - 0s 31ms/step - loss: 25.5068 - accuracy: 0.57
88 - val_loss: 8.3496 - val_accuracy: 0.4872
Epoch 19/20
3/3 [==============================] - 0s 25ms/step - loss: 28.0283 - accuracy: 0.56
91 - val_loss: 12.2256 - val_accuracy: 0.6410
```

```
Epoch 20/20
3/3 [==============================] - 0s 27ms/step - loss: 24.8539 - accuracy: 0.56
91 - val_loss: 11.4830 - val_accuracy: 0.7692
```

d. test and evaluate

In [171]:
```python
import matplotlib.pyplot as plt

# Plot training & validation accuracy values
plt.plot(history.history['val_accuracy'])
plt.plot(history.history['accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```



In [172]:
```python
score = model.evaluate(X_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Test loss: 11.4829683303833
Test accuracy: 0.7692307829856873
```

e. compare the two models and why you think the performance was same/different

The first NN model got accuracy = 0.8717948717948718 accuracy and the second one got 0.7692307829856873 accuracy.

I used MLP for first NN model and Keras sequential model for the second NN model. And Keras showed the lower accuracty for the same input. Sequential modelis appropriate for a plain stack of layers where each layer has exactly one input tensor and one output tensor. I think this made the different performance. The dataset we used and the hypothesis we tried wasn't appropriate for sequantial model.

1. Analysis

a. which algorithm performed better?

In my case, the logistic regression showed better performance. and NN models the worst.

b. compare accuracy, recall and precision metrics by class

Both accuracy and other indicators were higher in the order of logistic regression>DT>MLPCclassifier>Sequential model.

c. give your analysis of why the better-performing algorithm might have outperformed the other

I think this result is because the logistic regression is easier and simpler to code, resulting in fewer mistakes. When using Neural Network, there are too many variables and conditions to be reviewed, so I think it can produce different results as in this case because it takes a lot of work compared to strong performance. On the other hand, classification using logistic regression is a very simple algorithm compared to other methods, so I think a more suitable result came out. I put target at mpg_high, so I think the performance of the neural network is poor.

d. write a couple of sentences comparing your experiences using R versus sklearn. Feel free to express strong preferences.

Comparing R and sklearn, R required another resource called time to analyze data. In particular, in my case, the glm function was used to utilize the logistic regression, and the fast speed of sklearn came to be more attractive because it did not stop running forever. Nevertheless, I think the advantage of R is its accessibility. Sklearn is no different from R in terms of difficulty, but I think it is difficult to use due to the disadvantages of Python's unique easy to code and hard to use. In the end, preferences will depend on which side you are more proficient, but at this point I prefer R. This is because it has been used for a long time compared to sklearn.