Term Project Report

Class	Data Science
Time	Tue 09:00 - 13:00
Team	Team 10
Member 1	201433264 박성원
Member 2	201735877 정은서
Member 3	201835434 김주현

▶ Index ◀

1. Introduction

2. Data Curation

3. Data Inspection

4. Data Preprocessing

5. Data Analysis

  5.1 KNN(K-Nearest Neighbor) Algorithm

  5.2 Logistic Regression

    5.2.1 What is Logistic Regression?

    5.2.2 Analyze our data with Logistic Regression

  5.3 Random Forest(ensemble)

6. Evaluation – K-Fold Cross-Validataion

  6.1 Evaluate KNN(K-Nearest Neighbor) Algorithm

  6.2 Evaluate Logistic Regression

  6.3 Evaluate Random Forest

7. Conclusion

1. Introduction

Our term project object is '**Heart Disease Prediction'**. The heart is one of the most important organs in our body. It is important to check heart's health and predict disease in advance. So, we will predict heart disease or not through people's physical information and body condition like sex, age, education, current smoker, cigs per day, BPMeds, prevalent stroke, prevalent Hyp, diabetes, tot Chol, sysBP, diaBP, BMI, heart rate, glucose.

Data from Kaggle : "Heart Disease Prediction" (https://www.kaggle.com/naveengowda16/logistic-regression-heart-disease-prediction)

2. Data Curation

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	male	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabetes	totChol	sysBP	diaBP	BMI	heartRate	glucose	TenYearCHD
2	1	39	4	0	0	0	0	0	0	195	106	70	26.97	80	77	0
3	0	46	2	0	0	0	0	0	0	250	121	81	28.73	95	76	0
4	1	48	1	1	20	0	0	0	0	245	127.5	80	25.34	75	70	0
5	0	61	3	1	30	0	0	1	0	225	150	95	28.58	65	103	1
6	0	46	3	1	23	0	0	0	0	285	130	84	23.1	85	85	0
7	0	43	2	0	0	0	0	1	0	228	180	110	30.3	77	99	0
8	0	63	1	0	0	0	0	0	0	205	138	71	33.11	60	85	1
9	0	45	2	1	20	0	0	0	0	313	100	71	21.68	79	78	0
10	1	52	1	0	0	0	0	1	0	260	141.5	89	26.36	76	79	0
11	1	43	1	1	30	0	0	1	0	225	162	107	23.61	93	88	0
12	0	50	1	0	0	0	0	0	0	254	133	76	22.91	75	76	0
13	0	43	2	0	0	0	0	0	0	247	131	88	27.64	72	61	0
14	1	46	1	1	15	0	0	1	0	294	142	94	26.31	98	64	0
15	0	41	3	0	0	1	0	1	0	332	124	88	31.31	65	84	0
16	0	39	2	1	9	0	0	0	0	226	114	64	22.35	85	NA	0
17	0	38	2	1	20	0	0	1	0	221	140	90	21.35	95	70	1
18	1	48	3	1	10	0	0	1	0	232	138	90	22.37	64	72	0
19	0	46	2	1	20	0	0	0	0	291	112	78	23.38	80	89	1
20	0	38	2	1	5	0	0	0	0	195	122	84.5	23.24	75	78	0
21	1	41	2	0	0	0	0	0	0	195	139	88	26.88	85	65	0
22	0	42	2	1	30	0	0	0	0	190	108	70.5	21.59	72	85	0

↑ The data we used

- Data Shape : 4,238 rows and 16 columns

- Detail information of each columns :

#male – Male(1) or female(0)

#age - Age of the patient

#education – Final education

(1 = Some High School; 2 = High School or GED; 3 = Some College or Vocational School; 4 = college.)

#currentSmoker - Whether the patient is a current smoker

#cigsPerDay - The number of cigarettes that the person smoked on average in one day

#BPMeds - Whether the patient was on blood pressure medication

#prevalentStroke - Whether the patient had previously had a stroke

#prevalentHyp - Whether the patient was hypertensive

#diabetes - Whether the patient had diabetes

#totChol - Total cholesterol level

#sysBP - Systolic blood pressure

#diaBP - Diastolic blood pressure

#BMI - Body Mass Index

#heartRate - Heart rate

#glucose - Glucose levels

#TenYearCHD – CHD(heart disease) or not

∴ When predicting heart disease, we thought that physical information about the person, the person's current body condition, disease status, and medications being taken are necessary, and that no other unrelated information is needed. Therefore, we decided to delete information other than information about the user's body condition.

≫ So we drop the **'education'** column and proceed with data analysis.

3. Data Inspection(데이터 검사)

- Condition of data

```
>>> df.shape
(4238, 15)
```

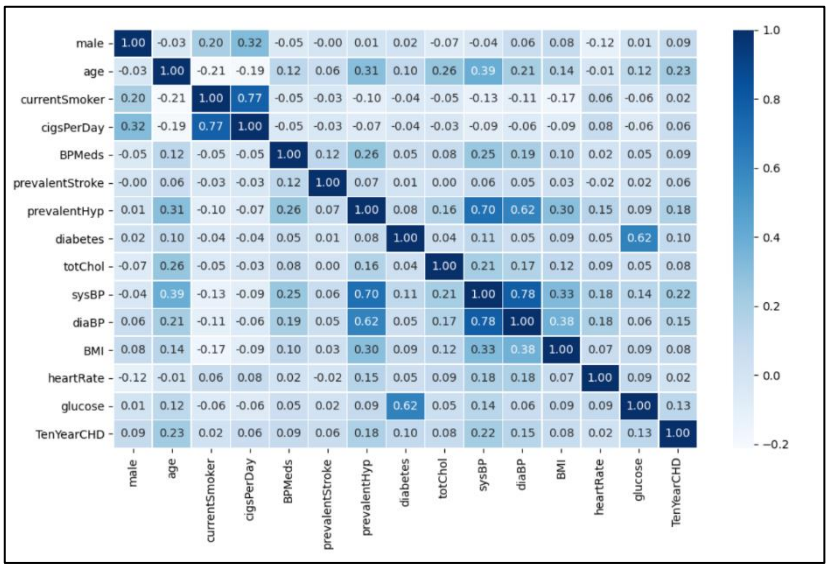
◀ Data shape(number of rows, number of columns)

```
>>> df.isnull().sum()
male             0
age              0
currentSmoker    0
cigsPerDay      29
BPMeds          53
prevalentStroke  0
prevalentHyp     0
diabetes         0
totChol         50
sysBP            0
diaBP            0
BMI             19
heartRate        1
glucose        388
TenYearCHD       0
dtype: int64
```

◀Number of missing values for each column

```
>>> (df.isnull().sum())/len(df)*100
male              0.000000
age               0.000000
currentSmoker     0.000000
cigsPerDay        0.684285
BPMeds            1.250590
prevalentStroke   0.000000
prevalentHyp      0.000000
diabetes          0.000000
totChol           1.179802
sysBP             0.000000
diaBP             0.000000
BMI               0.448325
heartRate         0.023596
glucose           9.155262
TenYearCHD        0.000000
dtype: float64
```

◀Percentage of missing value in each column(%)



◀ correlation of each column (HeatMap)

- **Target Value** : TenYearCHD

Input Value : Other values

```
Input values :
Index(['male', 'age', 'currentSmoker', 'cigsPerDay', 'BPMeds',
       'prevalentStroke', 'prevalentHyp', 'diabetes', 'totChol', 'sysBP',
       'diaBP', 'BMI', 'heartRate', 'glucose'],
      dtype='object')
```

4. Data Preprocessing - Fill missing data

Fill missing values of each column by max correlation. If the correlation is positive, the more one side increases, the more the other. (If it is negative, it is the opposite.) Therefore, we thought that the two columns with great correlation were correlated with the flow of value change. So we sorted the data by highly correlated column, and filled the missing value by ffill. Repeat this method to fill all columns with missing value.

[Step 1] Import the libraries and Define the required function.

```
import pandas as pd
import numpy as np
```

◀ Import pandas and numpy libarary

```
# Function to return a column which has max correlation.
def maxCorr(dataset, missing_col) :

  max_corr_col = dataset[missing_col].idxmax()

  return max_corr_col
```

▲ Function to get highly correlated column

```
#Function to sort the data by max_corr_col and fill missing values in missing_col by ffill
def sortCol(dataset, missing_col, max_corr_col) :

  dataset.sort_values(by = max_corr_col, inplace = True)

  dataset[missing_col].fillna(method = 'ffill',inplace = True)

  return dataset
```

▲ Function to sort dataset by highest correlated column and fill missing values by ffill

```
#Function to preprocessing
def preprocessing(df) :
  # a dataframe for the absolute value of the correlation of each column.
  corr = df.corr().abs()

  # Replace the correlation of the same column with 0
  corr = corr.replace(1, 0)

  #dataset's columns.
  df_col = df.columns

  # check every column
  for i in df_col :

    # if it has missing data fill them.
    if (df[i].isnull().sum() > 0) :

      # find a column which has max correaltion value
      max_corr_col = maxCorr(corr, i)

      # fill missing values by sortCol function
      df = sortCol(df, i, max_corr_col)

  return df
```

▲ Function to preprocessing dataset by fill all columns which have missing values

[Step 2] Import the data-set

```
# Load the dataset from csv file
df = pd.read_csv('heart_disease.csv')
df.drop(columns = ['education'], inplace = True)

#check
print(df.head(10))
```

▲ Load dataset and drop unused column('education')

	male	age	currentSmoker	cigsPerDay	...	BMI	heartRate	glucose	TenYearCHD
0	1	39	0	0.0	...	26.97	80.0	77.0	0
1	0	46	0	0.0	...	28.73	95.0	76.0	0
2	1	48	1	20.0	...	25.34	75.0	70.0	0
3	0	61	1	30.0	...	28.58	65.0	103.0	1
4	0	46	1	23.0	...	23.10	85.0	85.0	0
5	0	43	0	0.0	...	30.30	77.0	99.0	0
6	0	63	0	0.0	...	33.11	60.0	85.0	1
7	0	45	1	20.0	...	21.68	79.0	78.0	0
8	1	52	0	0.0	...	26.36	76.0	79.0	0
9	1	43	1	30.0	...	23.61	93.0	88.0	0

◀ Result

[Step 3] Check missing data and fill them

```
#check the missing data
print(df.isnull().sum())
```

▲ Check missing value

```
male             0
age              0
currentSmoker    0
cigsPerDay      29
BPMeds          53
prevalentStroke  0
prevalentHyp     0
diabetes         0
totChol         50
sysBP            0
diaBP            0
BMI             19
heartRate        1
glucose        388
TenYearCHD       0
dtype: int64
```

▲ Result

```
#fill NaN values by preprocessing function
df = preprocessing(df)

#sort the dataset by index
df.sort_index(inplace = True)

#check that all missing data is filled
print(df.isnull().sum())
```

◀ Preprocessing the dataset by preprocessing function and rearrange the preprocessed dataset by index

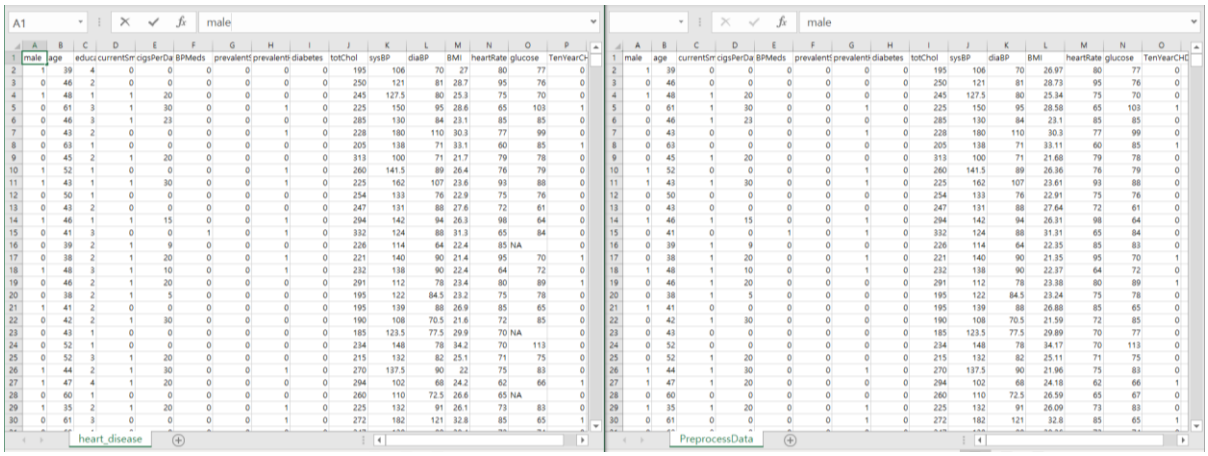
```
male             0
age              0
currentSmoker    0
cigsPerDay       0
BPMeds           0
prevalentStroke  0
prevalentHyp     0
diabetes         0
totChol          0
sysBP            0
diaBP            0
BMI              0
heartRate        0
glucose          0
TenYearCHD       0
dtype: int64
>>>
```

◀ Check that all missing values are filled.

```
#store preprocessed data by csv file
df.to_csv('PreprocessData.csv', index = False)
```

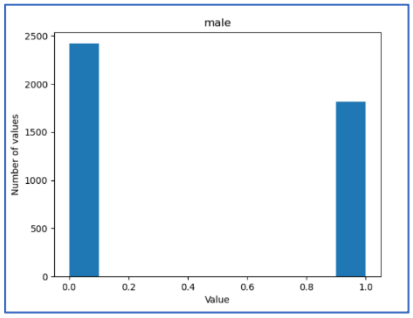
▲ Save preprocessed data by csv file. (From the next process, we proceeded with this csv file.)

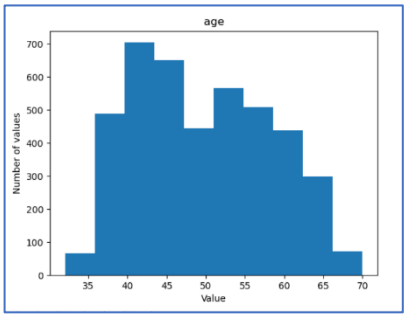
[heart_disease.csv] **[PreprocessData.csv]**

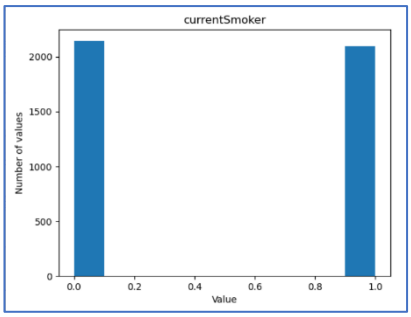


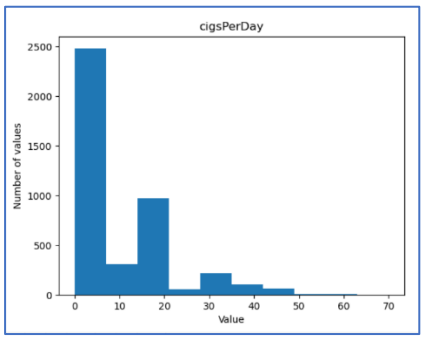
▲ Compare original data and preprocessed data

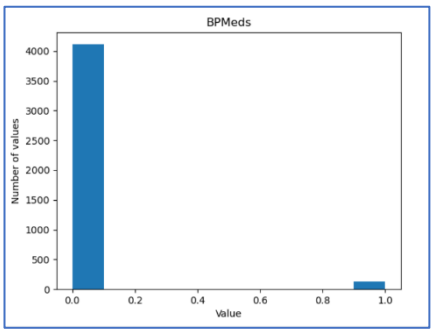
[Distribution of values in each column]

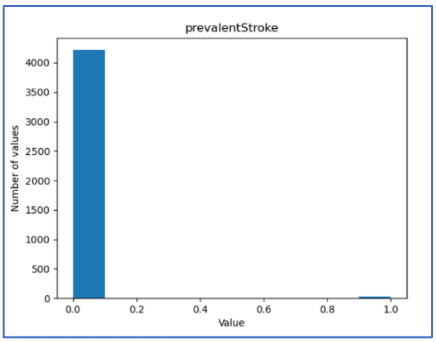


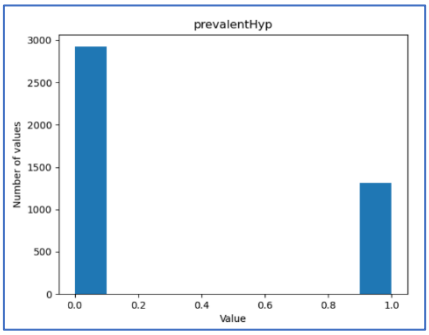


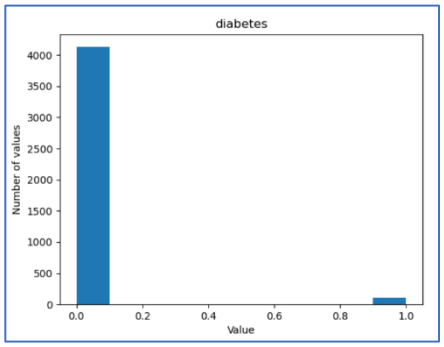


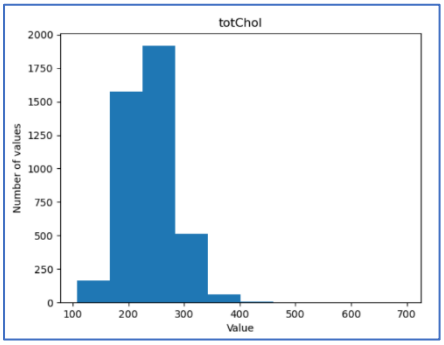


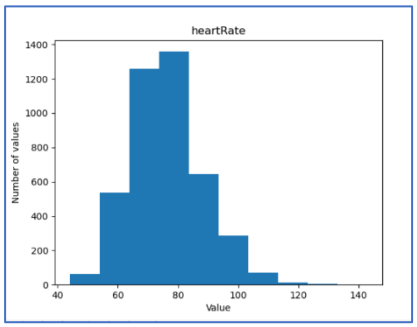


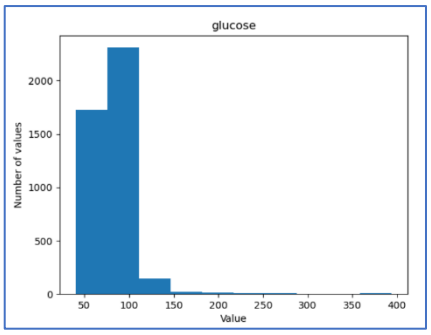


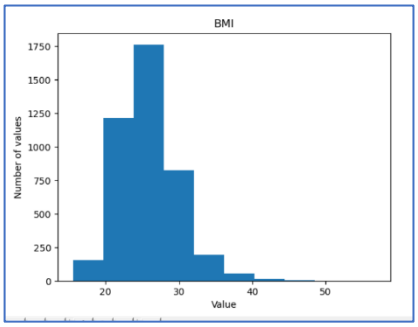


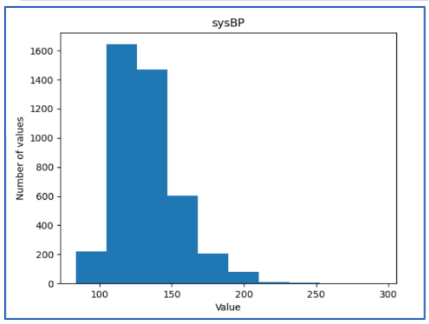


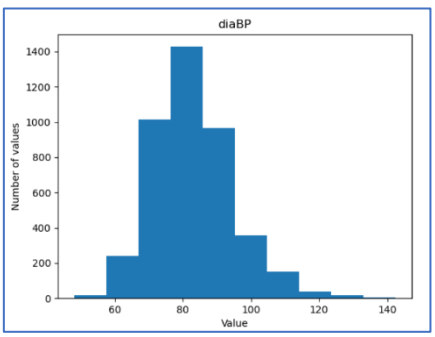


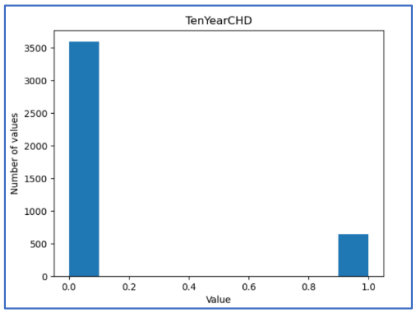












5. Data Analysis

5.1 KNN Algorithm

```
import pandas as pd
import numpy as np
import warnings
import matplotlib.pyplot as plt
from sklearn.preprocessing import RobustScaler
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
warnings.filterwarnings(action='ignore')
```

◀ Import the libraries

```
df=pd.read_csv('PreprocessedData.csv')

X=df.drop(columns=['TenYearCHD'])
y=df['TenYearCHD'].values

#Scaling data(MinMax Scaler)
scaler=MinMaxScaler()
X=scaler.fit_transform(X)

x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

◀ Read preprocessed data and scaling

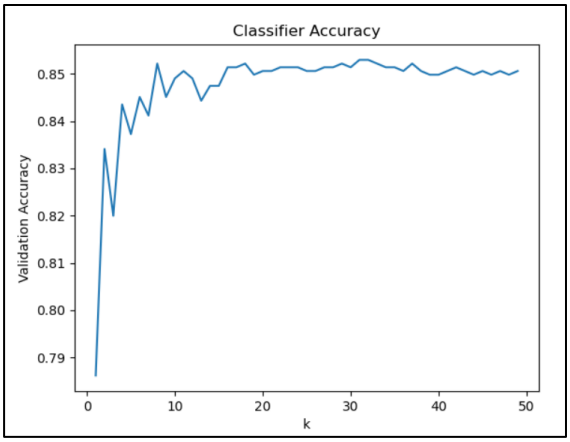
```
#k value candidate
k_list = range(1,50)
accuracies = []
max=0  #the max accuracy
index=0  #the k value which has max accuracy

#Find the best K value
for k in k_list:
    knn= KNeighborsClassifier(n_neighbors = k)
    knn.fit(x_train,y_train)
    accuracies.append(knn.score(x_test,y_test))
    if(max<knn.score(x_test,y_test)):
        max=knn.score(x_test,y_test)
        index=k

#display best k value
print("The best k value :",index)
#display best accuracy
print("** The accuracy of prediction: ", max)
```

◀ Analyze data by KNN algorithm and Find the best K value which has the largest accuracy.

```
#Draw a plot showing the accuracy of each K value.
plt.plot(k_list, accuracies)
plt.xlabel("k")
plt.ylabel("Validation Accuracy")
plt.title("Classifier Accuracy")
plt.show()
```

◀ Draw a plot to show the accuracy of each K value

```
>>>
= RESTART: C:₩Users₩김주현₩Desktop₩20SW_3학년₩데이터과학_1학기₩TermProject₩K
The best k value : 31
** The accuracy of prediction:  0.8529874213836478
```



◀ Result

5.2 Logistic Regression

5.2.1 What is Logistic Regression?

Logistic Regression is the use of regression to predict the probability that the data fall into categories 0 or 1 and, based on these probabilities, classify them as belonging to a higher probability category. It is proceeds using the logit function rather than the linear function. Therefore, the model of logistic regression performs classification predictions by using natural logarithmic values for the ratio of probability to probability that it belongs to the target group.

Logit function is logarithmic conversion of odds ratio. Odd is the probability of 1 occurring, and Odd ratio is the ratio of probability of 1 to probability of 0.

$$\text{odds ratio} = \frac{\theta}{1-\theta} \qquad z = \text{logit}(\text{odds ratio}) = \log\left(\frac{\theta}{1-\theta}\right)$$

▲ Odd ratio and logit function (θ is probability of 1 occurring)

5.2.2 Analyze our data with Logistic Regression

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
```

◀ Import the libraries

```python
# load the preprocessed dataset
df = pd.read_csv('PreprocessedData.csv')

#divide the dataset into input feature and target attribute
x = df.drop('TenYearCHD', axis=1)
y = df['TenYearCHD']

# Scaling
standard_scaler = StandardScaler()
x = standard_scaler.fit_transform(x)

# Splitting the dataset into Training and Test set
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3)

# Parameter candidate
C = np.logspace(-3, 3, 7)

penalty = ['l2']

param_grid = dict(C=C, penalty=penalty)

# Create Logistic Regression model
logistic_model = LogisticRegression()

# Find a model which has the best parameter by Grid Search
grid_search_model = GridSearchCV(logistic_model, param_grid, cv=5)

# fit the data to the best model (training)
best_model = grid_search_model.fit(x_train, y_train)

# Measure accuracy of this model
best_score = grid_search_model.best_score_

# print the result (best accuracy, the best parameter)
print('Best Score : ', best_score)
print('Best C :', best_model.best_estimator_.get_params()['C'])
print('Best Penalty : ', best_model.best_estimator_.get_params()['penalty'])
```

np.logspace(start, end, num = N)

-> N logarithmic scale from 'start' to 'end'

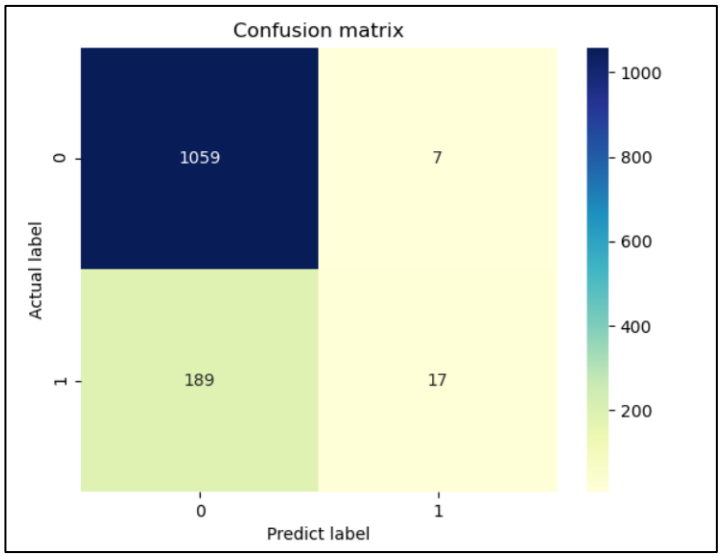
◀ Analyze data by logistic regression

```python
# predict y_test values by x_test
y_pred = grid_search_model.predict(x_test)

# make confusion matrix by y_test and y_pred
cnf_metrix = metrics.confusion_matrix(y_test, y_pred)

# calculate accurancy by confusion matrix
total = np.sum(cnf_metrix)

TP = cnf_metrix[0][0]
TN = cnf_metrix[1][1]
FP = cnf_metrix[1][0]
FN = cnf_metrix[0][1]

confusion_accurancy = round((TP + TN) / total, 2)


#the model's score
best_score = round(grid_search_model.best_score_, 2)

#display the score of model and the accurancy which calculated by confusion matrix
print('Confusion_matrix_score :', confusion_accurancy)
print('Model_score : ', best_score)


# draw heatmap by confusion matrix
sns.heatmap(pd.DataFrame(cnf_metrix), annot=True, cmap='YlGnBu', fmt='g')
plt.title('Confusion matrix')
plt.ylabel('Actual label')
plt.xlabel('Predict label')
plt.show()
```

◀ Make confusion matrix by predict result

```
'''
=== RESTART: C:₩Users₩김주현₩Desktop₩20SW_3학년₩데이터과학_1학기₩TermProject₩Logistic.py ==
=
Best Score : 0.8567127145541985
Best C : 1.0
Best Penalty : l2
Confusion_matrix_score : 0.85
Model_score : 0.86
```

◀ Result



◀ Heatmap of confusion matrix

5.3 Random Forest(ensemble)

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import metrics
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

#Read preprocessed data from csv file
df = pd.read_csv('PreprocessedData.csv')

# divide the dataset into input feature and targe attribute
x = df.drop('TenYearCHD', axis=1)
y = df['TenYearCHD']
```

◀Import libraries and read data from csv file

```
# Scaling
standard_scaler = StandardScaler()
x = standard_scaler.fit_transform(x)

# Split the dataset into train and test dataset
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3)
```

◀Scaling and split it into train and test set

```
# Create random forest model
random_forest_model = RandomForestClassifier()


# fit the data to random forest model
ensemble_model = random_forest_model.fit(x_train, y_train)
```

◀Create Random Forest model and fit train data

```
# predict target value with test input data
y_pred = ensemble_model.predict(x_test)

# Make confusion matrix by predict result
cnf_metrix = metrics.confusion_matrix(y_test, y_pred)

# calcualte accuracy by confusion matrix
total = np.sum(cnf_metrix)

TP = cnf_metrix[0] [0]
TN = cnf_metrix[1] [1]
FP = cnf_metrix[1] [0]
FN = cnf_metrix[0] [1]

confusion_accurancy = round((TP + TN) / total, 2)
```

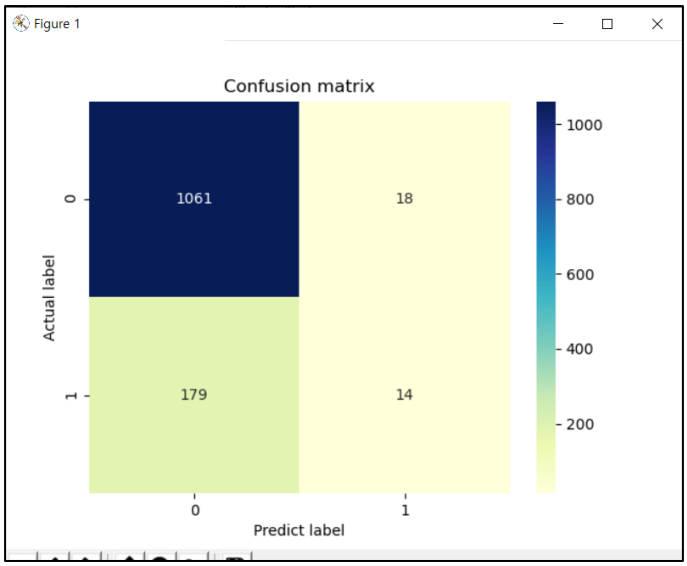
◀Predict target value with test input data and calculate the model's accuracy by confusion matrix

```
# display the result(the model's accuracy)
print('Confusion_matrix_score :', confusion_accurancy)

# Draw heatmap with confusion matrix
sns.heatmap(pd.DataFrame(cnf_metrix), annot=True, cmap='YlGnBu', fmt='g')
plt.title("Confusion matrix", y=1.1)
plt.ylabel("Actual label")
plt.xlabel("Predict label")
plt.show()
```

◀Display the result

```
orest.py
Confusion_matrix_score : 0.85
>>>
```

◀Result(Accuracy of the Random Forest model)



◀Heatmap of confusion matrix

6. Evaluation

6.1 Evaluate KNN Algorithm

```
import pandas as pd
import numpy as np
import warnings
import matplotlib.pyplot as plt
from sklearn.preprocessing import RobustScaler
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedShuffleSplit

warnings.filterwarnings(action='ignore')

df=pd.read_csv('PreprocessedData.csv')

X=df.drop(columns=['TenYearCHD'])
y=df['TenYearCHD'].values
```

◀Import libraries and read data from csv file

```
# MinMaxScaler
mmscaler=MinMaxScaler()
mmX=mmscaler.fit_transform(X)


# StandardScaler
stdscaler=StandardScaler()
stdX=stdscaler.fit_transform(X)


# RobustScaler
robscaler=RobustScaler()
robX=robscaler.fit_transform(X)
```

◀Create Min-Max / Standard / Robust Scaler(to compare) and fit the input data

```
#train/test
stratified_shuffle_split = StratifiedShuffleSplit(train_size=0.7, test_size=0.3, n_splits=10)
```

◀Create Stratified Shuffle Split to split by test and train ten times in the same distribution

```
knn_cv = KNeighborsClassifier (n_neighbors = 31)
 #10-fold cross validation
mm_scores = cross_val_score (knn_cv, mmX, y, cv = stratified_shuffle_split,scoring='accuracy')
mm_avg_score = mm_scores.mean()

std_scores=cross_val_score (knn_cv, stdX, y, cv = stratified_shuffle_split,scoring='accuracy')
std_avg_score = std_scores.mean()

rob_scores=cross_val_score (knn_cv, robX, y, cv = stratified_shuffle_split,scoring='accuracy')
rob_avg_score = rob_scores.mean()
```

◀ Make the KNN model (which we use to analysis the data) and do K-Fold validation with each scaler(K = 10)

```
print('MinMax Average score :', round(mm_avg_score, 2))
print('Scores :', mm_scores)
print()

print('Standard Average score :', round(std_avg_score, 2))
print('Scores :', std_scores)
print()

print('Robust Average score :', round(rob_avg_score, 2))
print('Scores :', rob_scores)
```

◀ Print the result

```
MinMax Average score : 0.85
Scores : [0.85062893 0.84748428 0.84748428 0.8490566  0.84748428 0.84669811
 0.84827044 0.84984277 0.84984277 0.84827044]

Standard Average score : 0.85
Scores : [0.84748428 0.85062893 0.84984277 0.84827044 0.8490566  0.84748428
 0.8490566  0.84669811 0.84984277 0.84827044]

Robust Average score : 0.85
Scores : [0.84827044 0.84669811 0.8490566  0.84984277 0.85220126 0.85062893
 0.84984277 0.84669811 0.84827044 0.84748428]
>>>
```

◀ Result

6.2 Evaluate Logistic Regression

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import RobustScaler
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedShuffleSplit

df = pd.read_csv('PreprocessedData.csv')

#divide data  by input feature and  target attribute
x = df.drop('TenYearCHD', axis=1)
y = df['TenYearCHD']
```

◀Import libraries and read data from csv file

```
#Standard Scaler
standard_scaler = StandardScaler()
std_x = standard_scaler.fit_transform(x)

#Robust Scaler
robust_scaler = RobustScaler()
rob_x = robust_scaler.fit_transform(x)
```

◀Create Min-Max / Standard / Robust Scaler(to compare) and fit the input data

```
# Create Logistic Regression model
logistic_model = LogisticRegression(C = 1.0, penalty = "l2")

#use StratifiedShuffleSplit for K-Fold(K = 10)
stratified_shuffle_split = StratifiedShuffleSplit(
  train_size=0.7, test_size=0.3, n_splits=10)

#10-fold cross validataion
std_scores = cross_val_score(logistic_model, std_x, y, cv=stratified_shuffle_split)
std_avg_score = std_scores.mean()

rob_scores = cross_val_score(logistic_model, rob_x, y, cv=stratified_shuffle_split)
rob_avg_score = rob_scores.mean()
```

Use **Stratified Shuffle Split** to split data by test and train 10 times in the same distribution

◀ Make the Logistic Regression model (which we use to analysis the data) and do K-Fold validation with each scaler(K = 10)

```
print('Standard Average score :', round(std_avg_score, 2))
print('Scores :', std_scores)
print('₩n')
print('Robust Average score :', round(rob_avg_score, 2))
print('Scores :', rob_scores)
```

◀ Print the result

```
Standard Average score : 0.85
Scores : [0.85534591 0.86006289 0.8577044  0.8490566  0.85377358 0.85691824
0.85141509 0.85298742 0.85534591 0.85062893]


Robust Average score : 0.85
Scores : [0.85298742 0.85377358 0.85377358 0.85220126 0.85141509 0.85455975
0.85534591 0.85141509 0.85220126 0.8577044 ]
>>>
```

◀ Result

6.2 Evaluate Random Forest(ensemble)

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import RobustScaler
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedShuffleSplit

df = pd.read_csv('PreprocessedData.csv')

#divide data  by input feature and  target attribute
x = df.drop('TenYearCHD', axis=1)
y = df['TenYearCHD']
```

◀Import libraries and read data from csv file

```
#Standard Scaler
standard_scaler = StandardScaler()
std_x = standard_scaler.fit_transform(x)

#Robust Scaler
robust_scaler = RobustScaler()
rob_x = robust_scaler.fit_transform(x)
```

◀Create Min-Max / Standard / Robust Scaler(to compare) and fit the input data

```
# Create Random Forest model
random_forest_model = RandomForestClassifier()

#use StratifiedShuffleSplit for K-Fold(K = 10)
stratified_shuffle_split = StratifiedShuffleSplit(
  train_size=0.7, test_size=0.3, n_splits=10)
```

◀Create Random Forest model and split data using **Startified Shuffle Split to split data by test and train 10 times in the same distribution**

```
#10-fold cross validataion
std_scores = cross_val_score(random_forest_model, std_x, y, cv=stratified_shuffle_split)
std_avg_score = std_scores.mean()

rob_scores = cross_val_score(random_forest_model, rob_x, y, cv=stratified_shuffle_split)
rob_avg_score = rob_scores.mean()

print('Standard Average score :', round(std_avg_score, 2))
print('Scores :', std_scores)
print('₩n')
print('Robust Average score :', round(rob_avg_score, 2))
print('Scores :', rob_scores)
```

◀ Do K-Fold validation with each scaler(K = 10) and print result

```
RandomForest_Kfold.py
Standard Average score : 0.85
Scores : [0.85141509 0.85298742 0.84748428 0.8427673  0.84984277 0.84984277
 0.84591195 0.8490566  0.84433962 0.84984277]


Robust Average score : 0.85
Scores : [0.84748428 0.84827044 0.84669811 0.85377358 0.84198113 0.84355346
 0.85220126 0.85377358 0.84748428 0.84827044]
>>>
```

◀ Result

7. Conclusion

We've analyzed the heart disease data and predict the possibility of heart disease. We generated algorithms, and evaluated them. The results are as follows :

- KNN algorithm accuracy

```
MinMax Average score : 0.85
Scores : [0.85062893 0.84748428 0.84748428 0.8490566  0.84748428 0.84669811
 0.84827044 0.84984277 0.84984277 0.84827044]

Standard Average score : 0.85
Scores : [0.84748428 0.85062893 0.84984277 0.84827044 0.8490566  0.84748428
 0.8490566  0.84669811 0.84984277 0.84827044]

Robust Average score : 0.85
Scores : [0.84827044 0.84669811 0.8490566  0.84984277 0.85220126 0.85062893
 0.84984277 0.84669811 0.84827044 0.84748428]
>>>
```

- Logistic regression accuracy

```
Standard Average score : 0.85
Scores : [0.85534591 0.86006289 0.8577044  0.8490566  0.85377358 0.85691824
 0.85141509 0.85298742 0.85534591 0.85062893]


Robust Average score : 0.85
Scores : [0.85298742 0.85377358 0.85377358 0.85220126 0.85141509 0.85455975
 0.85534591 0.85141509 0.85220126 0.8577044 ]
>>>
```

- Random forest accuracy

```
RandomForest_Kfold.py
Standard Average score : 0.85
Scores : [0.85141509 0.85298742 0.84748428 0.8427673  0.84984277 0.84984277
 0.84591195 0.8490566  0.84433962 0.84984277]


Robust Average score : 0.85
Scores : [0.84748428 0.84827044 0.84669811 0.85377358 0.84198113 0.84355346
 0.85220126 0.85377358 0.84748428 0.84827044]
>>>
```

∴ The analysis result's accuracy = 85%

Even when predicting and evaluating the results using random forest, one of the ensemble algorithms, the accuracy was equally 85%.

This accuracy can be thought to be quite high. With effective preprocessing, We thought both algorithms had quite high accuracy.

However, since it is a heart disease prediction, even a slight error can be dangerous to one's life, so it needs to be more accurate than this.