Data Science Term Project

Final Report



소프트웨어학과 201835501 이준희

소프트웨어학과 201835533 조현식

소프트웨어학과 201835534 차원우

소프트웨어학과 201934240 이민아

목차

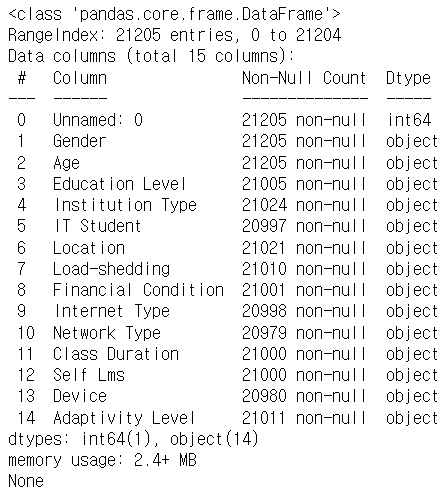
1. **Objective3**
2. **Data description3**
   1. **Data Source3**
   2. **Original Dataset Inspection3**
3. **Data preprocessing5**
   1. **Setting data for analyse6**
   2. **Handle ‘Missing data’6**
   3. **Setting data according to target6**
   4. **Handle categorical data6**
   5. **Feature selection7**
   6. **Normalization9**
4. **Algorithms11**
   1. **Linear regression11**
   2. **Decision Tree classification11**
   3. **KNN classification12**
5. **GUI14**
6. **Open Source SW Contribution14**
7. **Learning Experience17**
8. **Role17**
9. **Reference 17**
10. **Source Code 18**
11. **Objective**

Many students go private institute or study with internet lectures. Also because of Covid-19 many classes at school have been replaced online. The academy provides face-to-face and non-face-to-face lectures for students (elementary, middle, and high school). So we think adaptivity level in online education is important when parent or students choose method of education (online or offline). In order to improve students' grades, the head of the academy tries to organize a division by determining whether students are effective in face-to-face lectures or non-face-to-face lectures. Therefore, I would like to analyze how effective online education for each student is through information about the students. So we want to help under university students and private instituter to choose method of education by analyze students adaptability level in online education data.

1. **Data description**
   1. **Data Source**

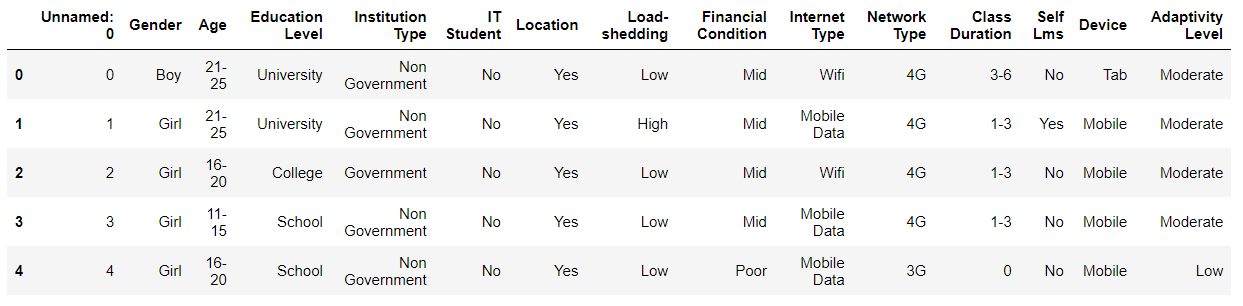
We used 'Students Adaptability Level in Online Education' dataset registered in Kaggle.

The data contains information about student who adapt in online education, for example, education level, class duration, age.

* 1. **Original Dataset Inspection**

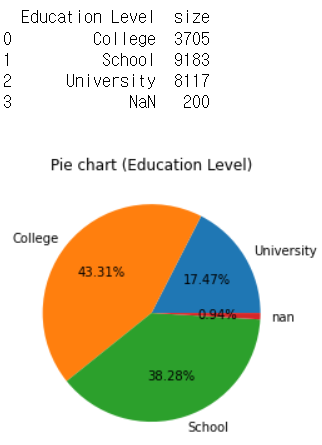
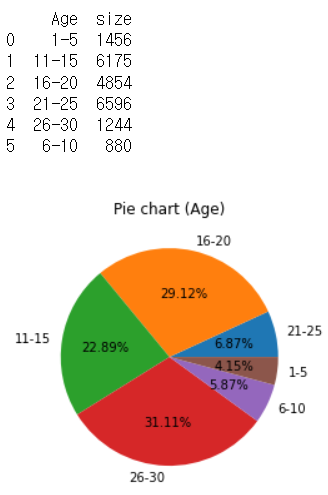
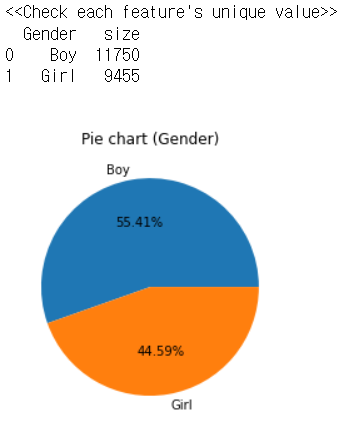
**size**: 15 columns X 21205 row

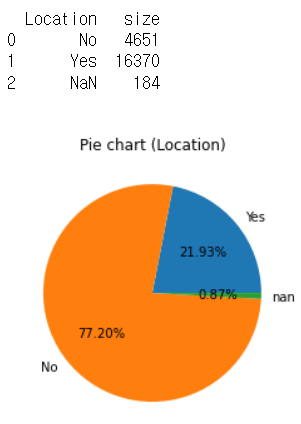
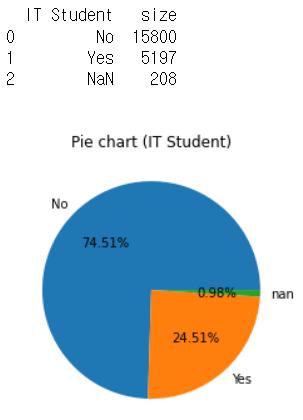
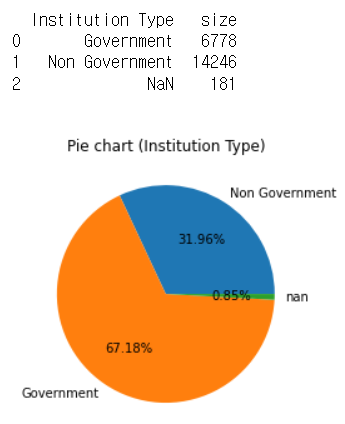
**columns** = [ ‘ ‘, 'Gender', 'Age', 'Education Level', 'Institution Type', 'IT Student', 'Location', 'Load-shedding', 'Financial Condition', 'Internet Type', 'Network Type', 'Class Duration', 'Self Lms', 'Device', 'Adaptivity Level' ]

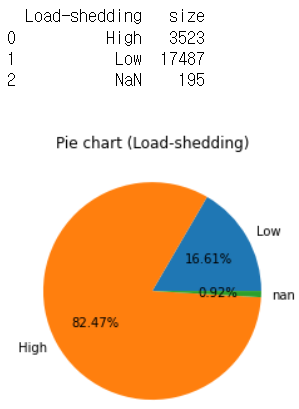
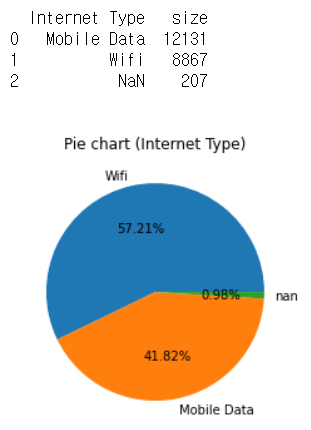
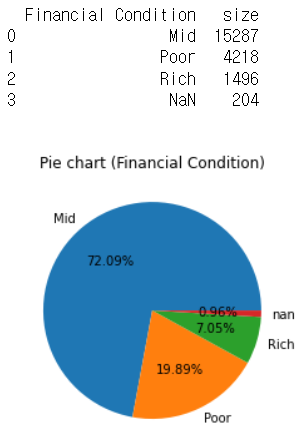


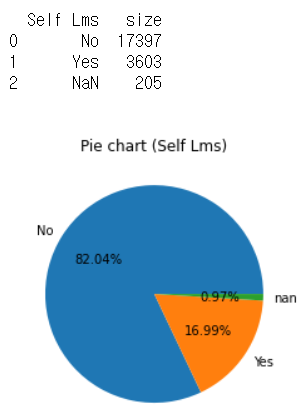
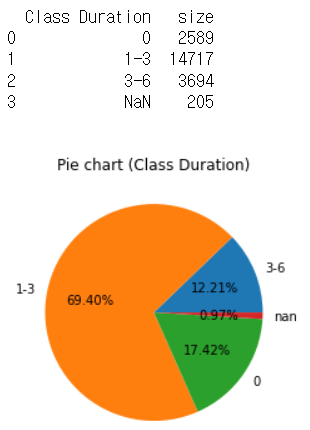
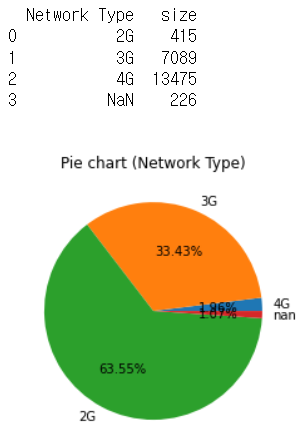
In preprocessing, we are going to handle data to analyze.

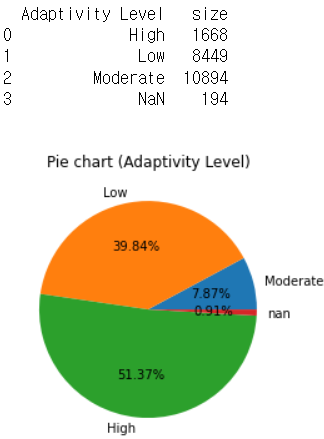
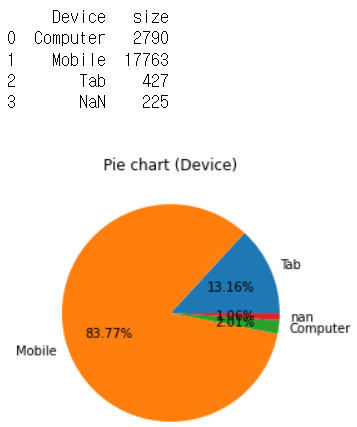
<data visualization>





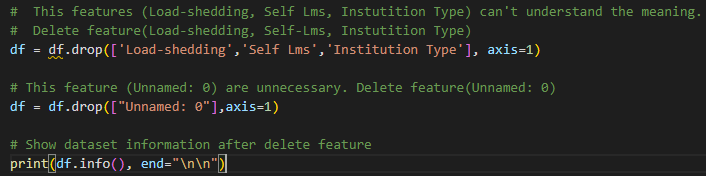


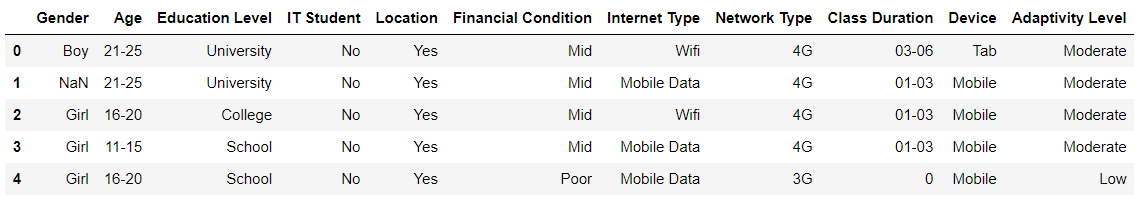


1. **Data preprocessing**
   1. **Setting data for analyse**

There are some columns that have not clear information. For example [ ‘Load-shedding’, ‘Self Lms’]. We cannot understand the meaning of that columns. So we drop those columns. And also we drop [ ‘institution type’ ] column.

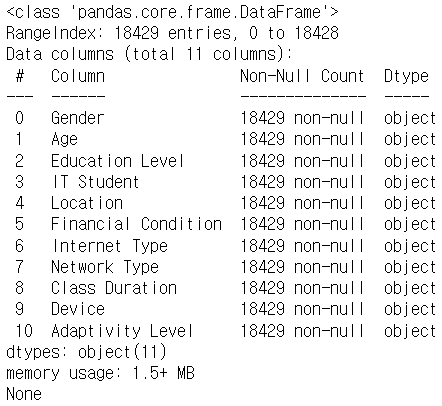
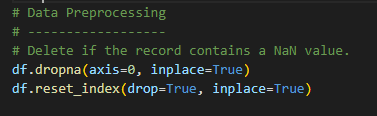
There are ‘Unnamed 0’ columns that just indexing dataset, so we drop the column.





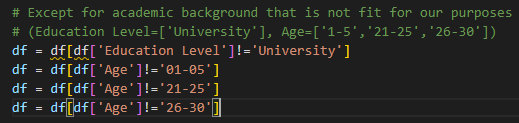
* 1. **Handle Missing data**

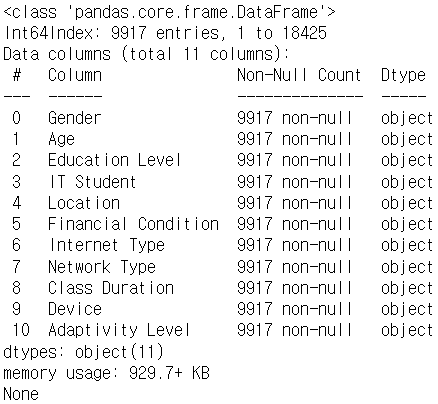
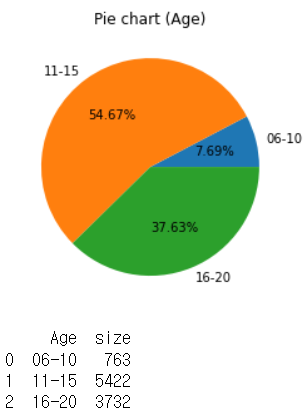
We can find there were less than 1 percent of ‘Missing data’ in several columns. Those percentage is pretty low, so we consider deleting the row which has ‘Missing data’.

* 1. **Setting data according to target**

Our project’s objective target is for under university students. So we drop data that age is ‘1~5’ and over 21.

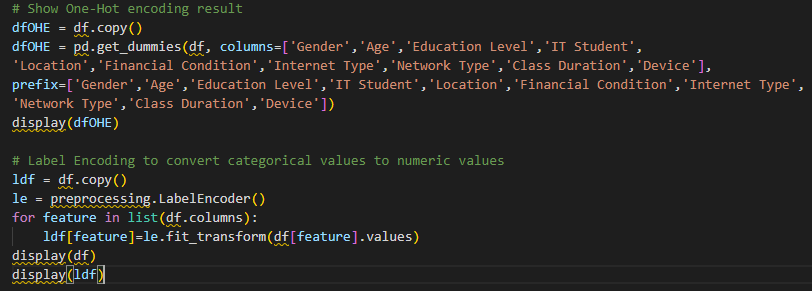


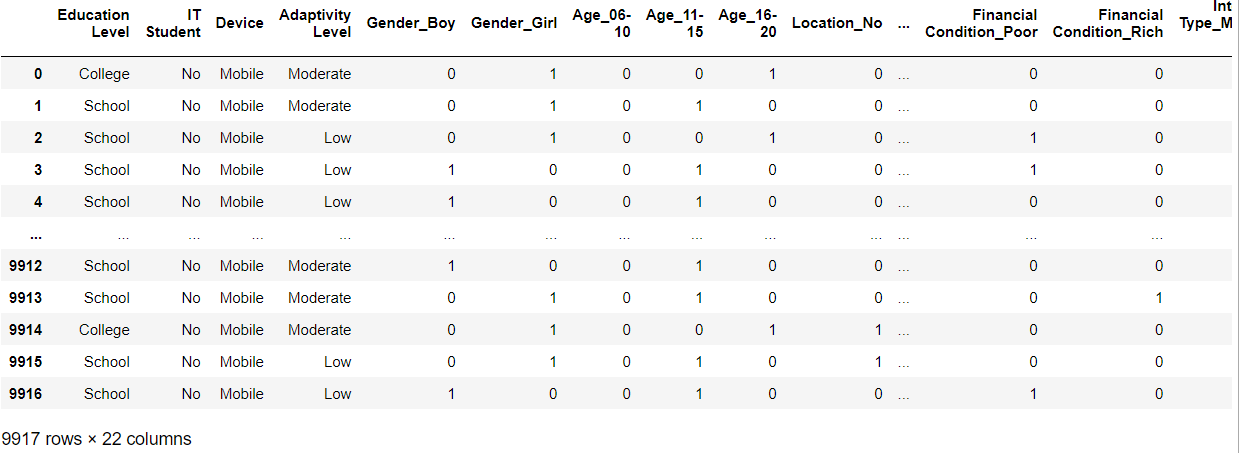
This is final data before encoding and normalizing.

* 1. **Handle categorical data**

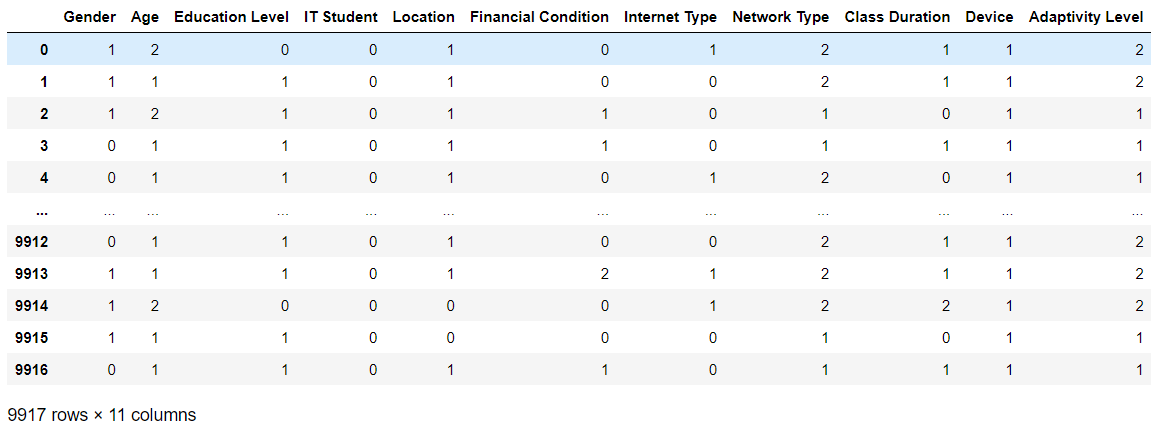
After setting data, we have to handle categorical data because machine learning cannot read categorical data.



First we try to encode categorical data by ‘One Hot Encoder’. In this case dataset has too many features(26 features) so we don’t use this encoder.

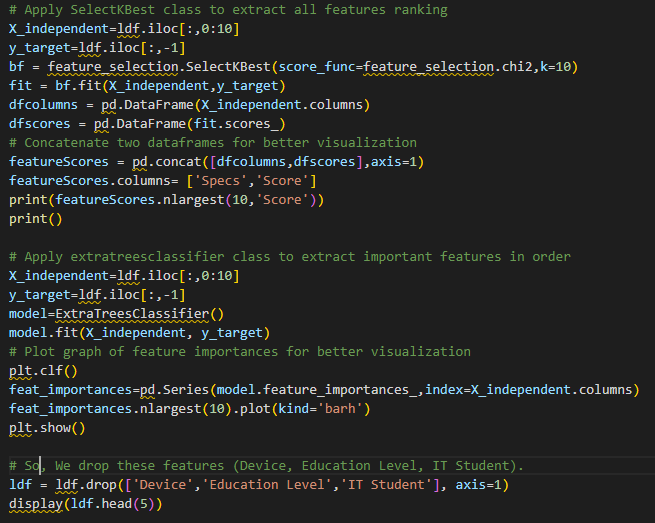


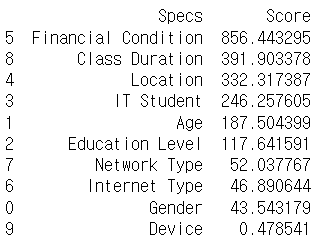
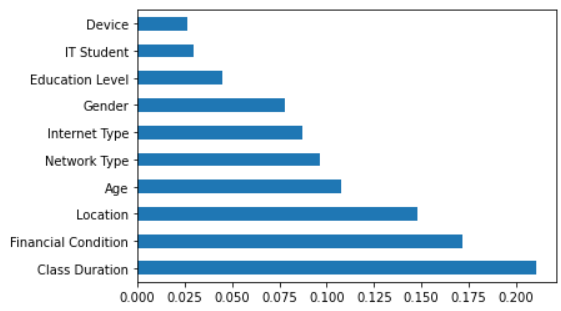
Then, we use ‘Label Encoder’.



* 1. **Feature selection**

After we handle categorical data, we have to choose independent features to predict target value. In feature selection we try 2 selection tools, SelectKBest, Extra Tree Classifier. In this process we have confuse because these two tools’ result are little different. So we try RFE and we find this result also different with those two. We want to find the reason of it. Finally we learn that 3 tools have different scoring method. And we select feature by Extra Tree Classifier’s top 7 features.

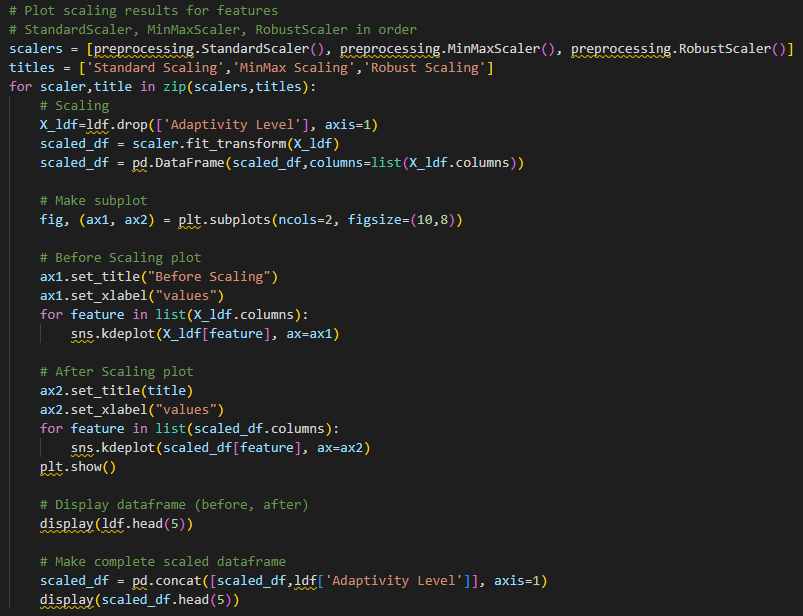


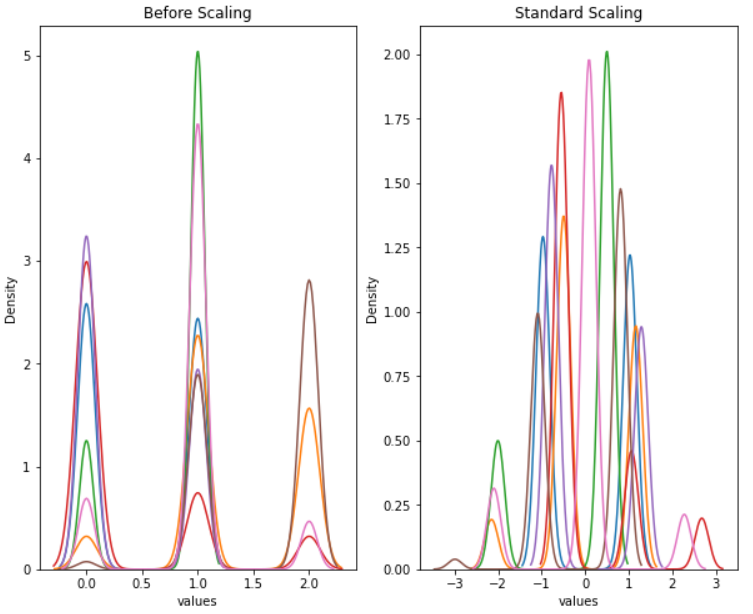
Left is result of SelectKBest and right is Extra Tree Classifier. We think Extra Tree Classifier’s result are more reasonable than SelectKBest, so we choose Extra Tree Classifier. Also there are big difference between gender and education level, so we choose features that have higher score than education level.

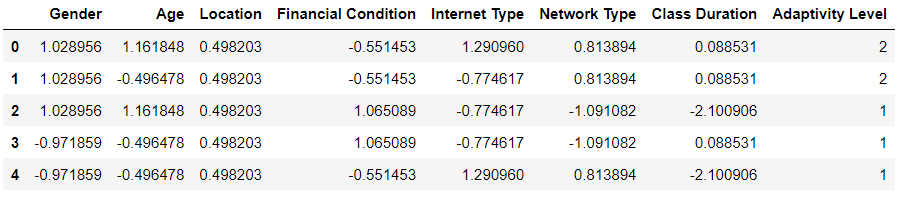
* 1. **Normalization**

After processing ‘Categorical data’ we have to Normalized data. We learn 3 kinds of scaler (standard, robust, minmax). We use all those scalers.

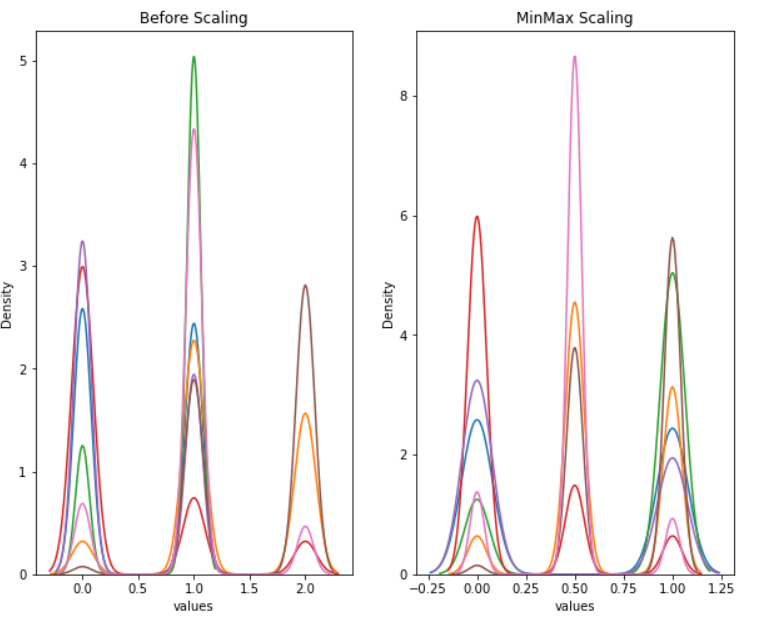


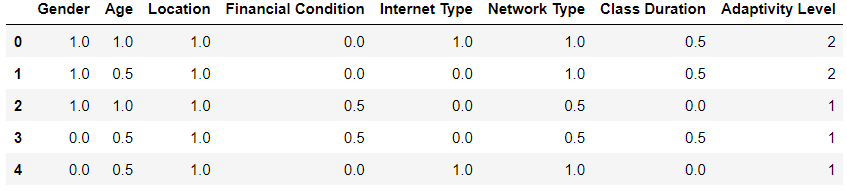
First, Standard Scaler



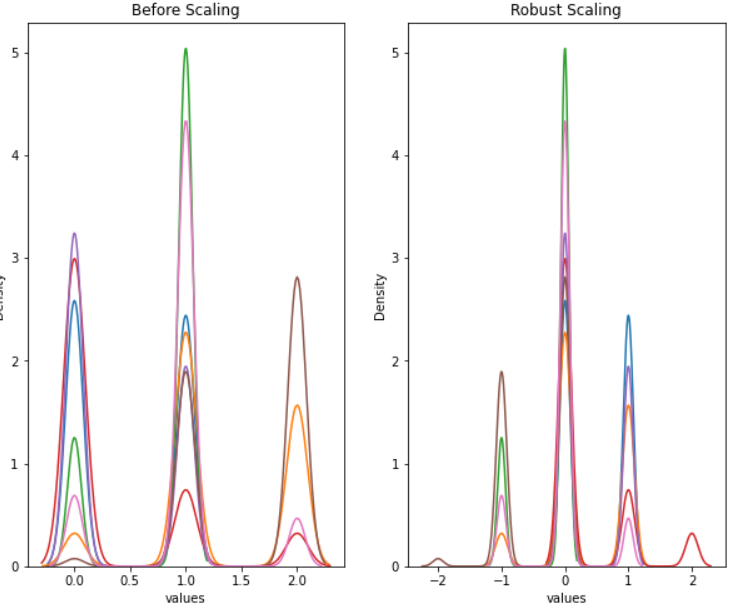


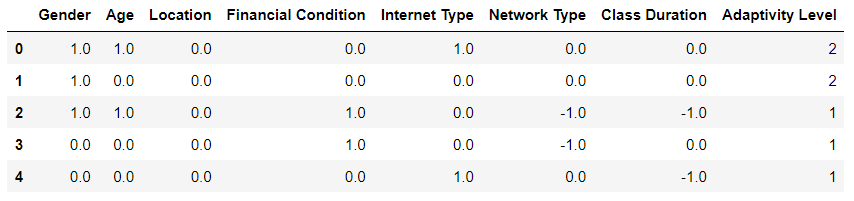
Second, MinMax Scaler





Last, Robust Scaler





1. **Modeling and Evaluation**

We are use regression and classifier to data modeling. In regression we use Linear Regression and two classifier, Decision Tree Classifier and KNN Classifier.

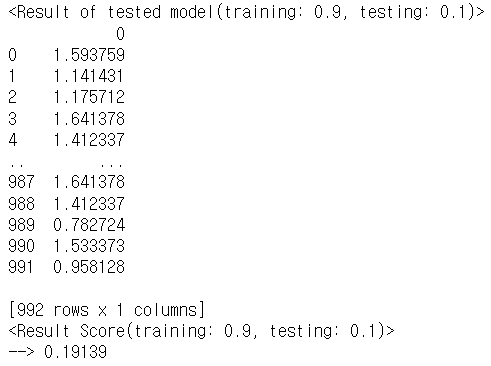
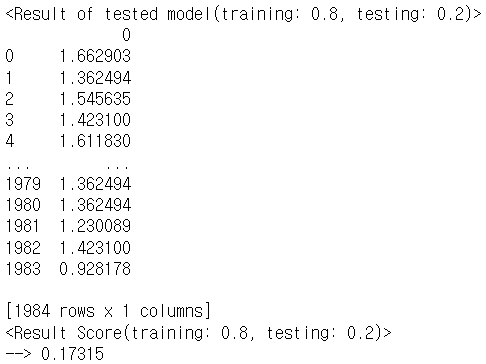
Also we train 3 times (9:1, 8:2, 7:3). We use **Shuffle method.**

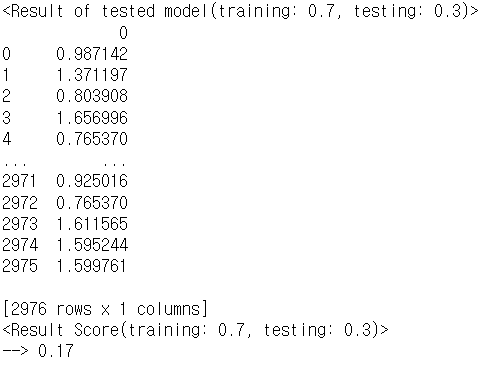
In evaluation we use R2 score in regression. And we use K-fold cross validation and classification report in classifier.

We do modeling each scaled dataset(standard, minmax, robust). But in report, we include output about standard scaling.

* 1. **Linear regression – Standard Scaling**

At first we use Linear Regression.

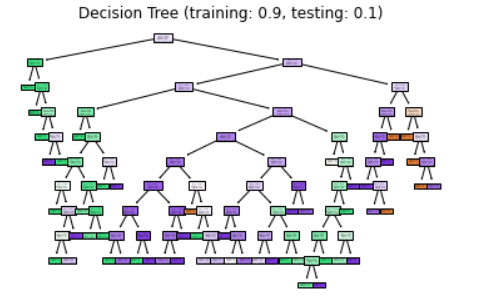
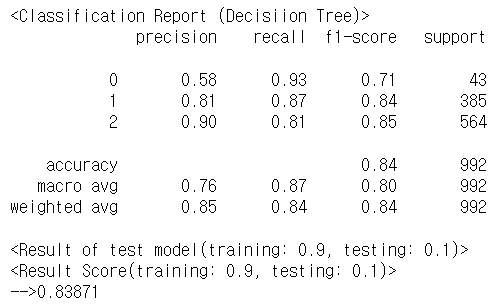
 

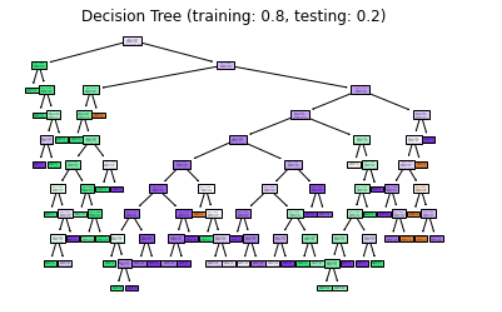
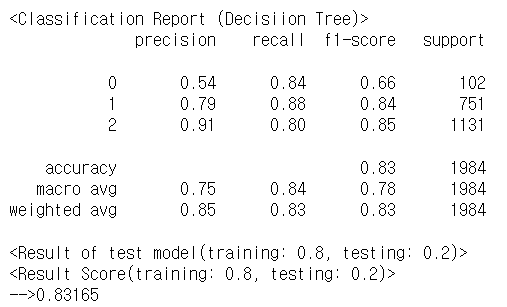


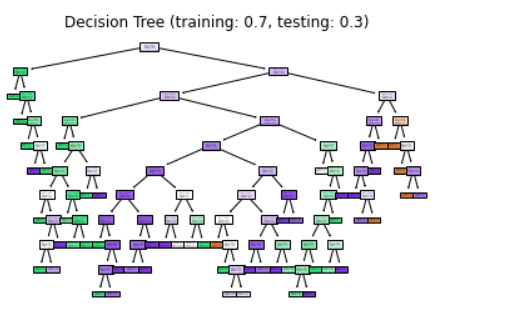
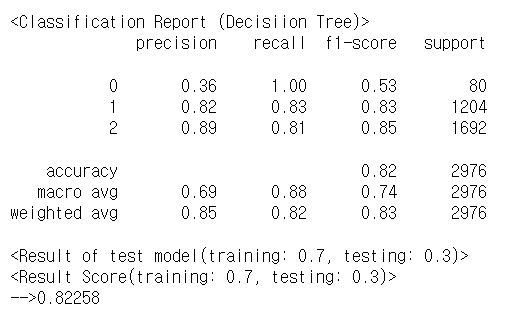
We find that linear regression model’s accuracy is very low (about 0.2, 0.17). We think the reason is our target value is non-continuous. So we are going to use another algorithm classifier.

* 1. **Decision Tree Classifier**

At first we use Decision Tree Classifier.

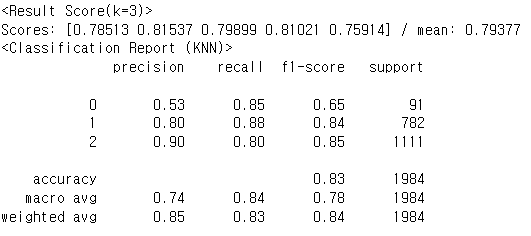
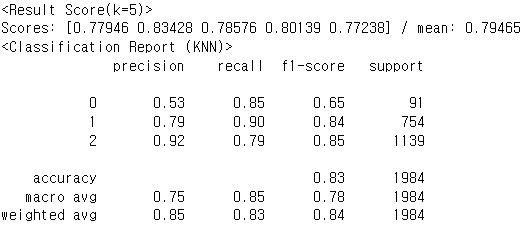
 

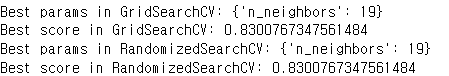
 

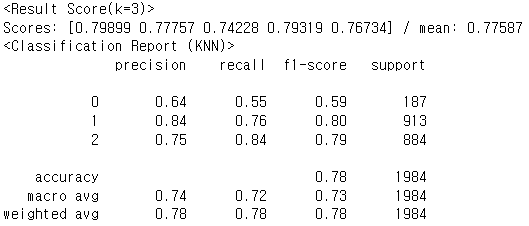
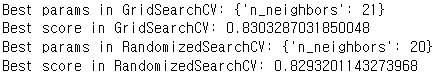
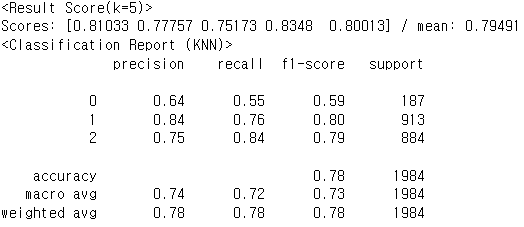
 

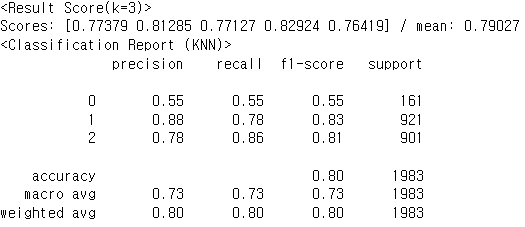
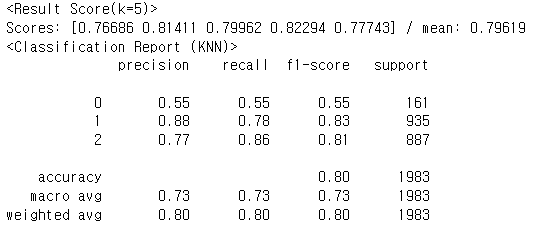
* 1. **KNN Classifier**

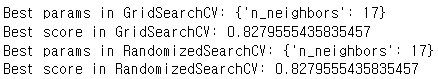
Second we try KNN Classifier. In this case we use GridSearchCV and RandomizedGridSearchCV to find hyper parameter. We divide 5 split and do modeling each split.

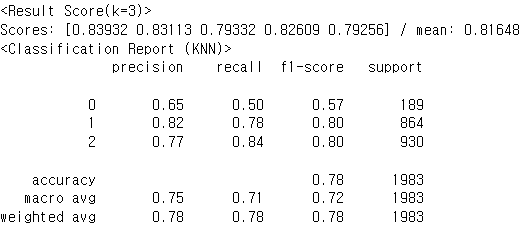
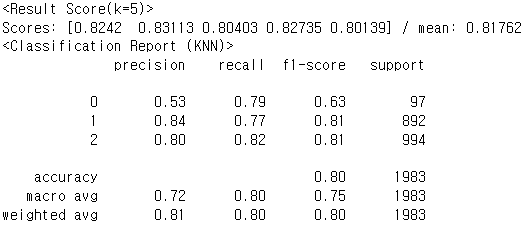
 

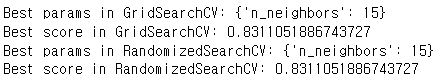


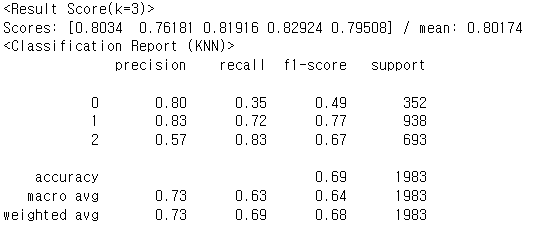
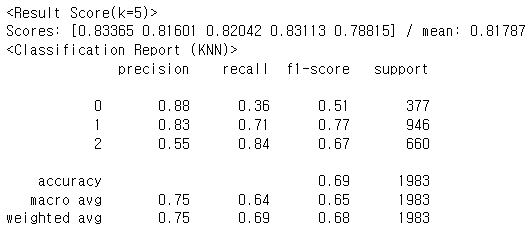
 

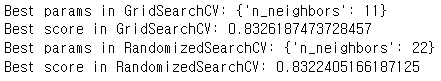
 



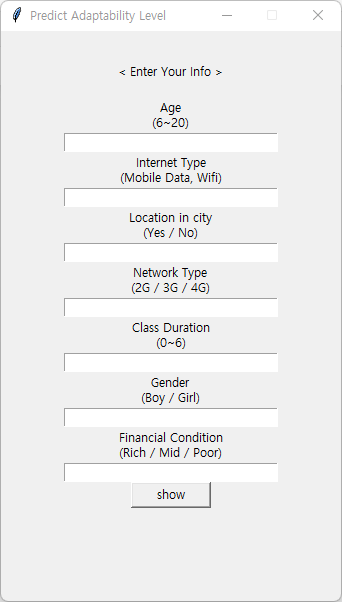
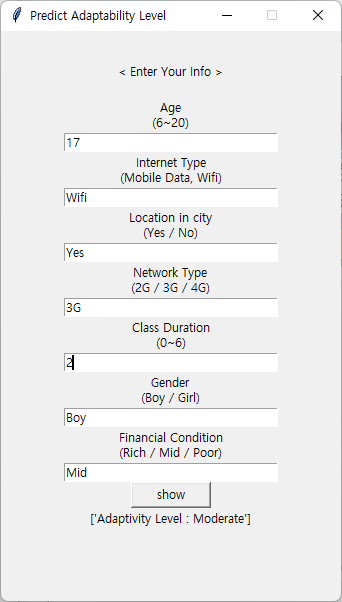




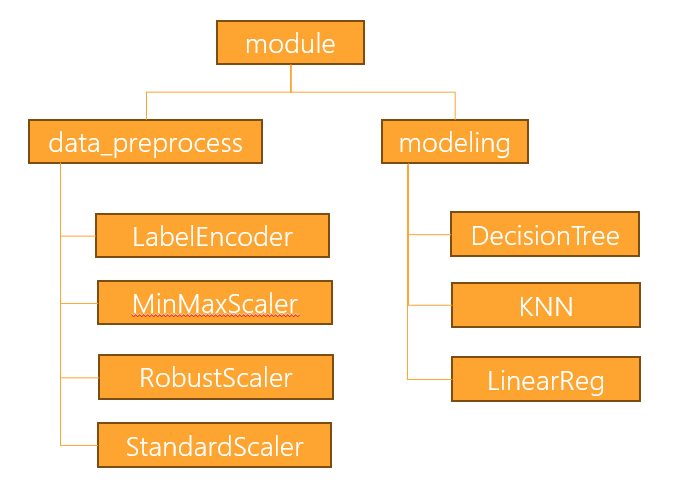
1. **GUI**

We created a GUI-based program that outputs the results of the Internet class accuracy prediction when entering student information using the Decision Tree Classifier, which shows the best acuity among the above modeling.

1. **Open Source SW Contribution**

We make module about scaling, encoding and modeling. Source code about modules are in ’ 10. Source Code ’ part.



* 1. **LabelEncoder**

input parameter : org\_df

org\_df (type: DataFrame) >> Target of LabelEncoding

output : Return DataFrame after LabelEncoding

* 1. **MinMaxScaler**

input parameter : org\_df, target, showPlot

org\_df (type: DataFrame) >> Target of MinMaxScaling

target (type: String) >> Feature name of target value

showPlot (type: bool, default=False) >> Whether to plot the result

output : Return DataFrame that after MinMaxScaling. Drawing graph based on whether or not plotting

* 1. **RobustScaler**

input parameter : org\_df, target, showPlot

org\_df (type: DataFrame) >> Target of RobustScaling

target (type: String) >> Feature name of target value

showPlot (type: bool, default=False) >> Whether to plot the result

output : Return DataFrame that after RobustScaling. Drawing graph based on whether or not plotting

* 1. **StandardScaler**

input parameter : org\_df, target, showPlot

org\_df (type: DataFrame) >> Target of StandardScaling

target (type: String) >> Feature name of target value

showPlot (type: Bool, default=False) >> Whether to plot the result

output : Return DataFrame that after StandardScaling. Drawing graph based on whether or not plotting

* 1. **Decision Tree**

input parameter : scaled\_df, target, test\_size, shuffle, criterion, showPlot

scaled\_df (type: DataFrame) >> Target of DecisionTree that after Scaling

target (type: String) >> Feature name of target value

test\_size (type: Float, default: 0.25) >> Specify testset ratio when training\_test\_split

shuffle (type: Bool, default: False) >> Specify whether shuffle when training\_test\_split

criterion (type: String, default: 'gini') >> Determine the type of criterion used in Decision Tree

showPlot (type: bool, default=False) >> Whether to plot the result

output : Show DecisionTree score and confision matrix. Drawing tree based on whether or not plotting

* 1. **KNN**

input parameter : scaled\_df, target, test\_size, shuffle, k

scaled\_df (type: DataFrame) >> Target of DecisionTree that after Scaling

target (type: String) >> Feature name of target value

test\_size (type: Float, default: 0.25) >> Specify testset ratio when training\_test\_split

shuffle (type: Bool, default: False) >> Specify whether shuffle when training\_test\_split

k (type: Int, default: 3) >> Specify a value for n\_neighbors in KNN

output : Show KNN score evaluated by cross\_validation=5 and confision matrix

* 1. **Linear Regression**

input parameter : scaled\_df, target, test\_size, shuffle

scaled\_df (type: DataFrame) >> Target of DecisionTree that after Scaling

target (type: String) >> Feature name of target value

test\_size (type: Float, default: 0.25) >> Specify testset ratio when training\_test\_split

shuffle (type: Bool, default: False) >> Specify whether shuffle when training\_test\_split

output : Show regression score

1. **Learning Experience**

While we doing term project, we experience big data end-to-end process. And we learn and feel a lot of things from project.

We learned how to use Big Data process. And we learned that importance of preprocessing because of modeling results are have huge different by human’s preprocessing.

At first, we just use all records and features of data. After presentation we get feedback about objective and ‘Who use this analysis?’. From this feedback we recognized that objective setting and use correct data about that objective is very important. So we classifying unnecessary feature values according to the subjects to be analyzed (e.g., drop values over 20 years of age based on adolescents, not people of all ages), a slightly more accurate analysis result was obtained.

While we doing term project, we have several challenge. Most confusion challenge is the results of several feature selection tools(SelectKBest, Extra Tree Classifier, RFE) are all different. Because of this we search internet and discussing. Finally we find that tools scoring methods are different so some features’ score can be different.

We were able to develop communication and collaboration skills with different people by coordinating schedules and sharing roles while carrying out projects with randomly composed teams

1. **Role**

|  |  |  |  |
| --- | --- | --- | --- |
| **이준희 (25%)** | **조현식 (35%)** | **차원우 (20%)** | **이민아 (20%)** |
| **Coding** | **Coding**  **(most contributed)** | **Coding** | **Coding** |
| **Writing report** | **GUI base program** | **PPT** | **Writing report** |
| **Make module** | **Make module** | **Presentation** | **PPT** |

1. **Reference**

**Project Git URL**

[**https://github.com/CHOHYUNSIK/data\_science\_module**](https://github.com/CHOHYUNSIK/data_science_module)

[**https://www.kaggle.com/datasets/mdmahmudulhasansuzan/students-adaptability-level-in-online-education**](https://www.kaggle.com/datasets/mdmahmudulhasansuzan/students-adaptability-level-in-online-education)

1. **Source Code**

**End-to-End Process Full Code**

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn import preprocessing

from sklearn import feature\_selection

from sklearn.ensemble import ExtraTreesClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn import tree

from sklearn.metrics import accuracy\_score

from sklearn.neighbors import KNeighborsClassifier

from sklearn.model\_selection import KFold

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import GridSearchCV

from sklearn.model\_selection import RandomizedSearchCV

from sklearn import linear\_model

from sklearn.metrics import classification\_report

# get data

df = pd.read\_csv("students\_adaptability\_level\_online\_education.csv")

# Data Inspection

# ---------------

# Show dataset information

print(df.info(), end="\n\n")

# Check the first 5 rows of the dataframe

display(df.head(5))

print("\n")

# Data Preprocessing

# ------------------

# This features (Load-shedding, Self Lms, Instutition Type) can't understand the meaning.

# Delete feature(Load-shedding, Self-Lms, Instutition Type)

df = df.drop(['Load-shedding','Self Lms','Institution Type'], axis=1)

# This feature (Unnamed: 0) are unnecessary. Delete feature(Unnamed: 0)

df = df.drop(["Unnamed: 0"],axis=1)

# Show dataset information after delete features

print(df.info(), end="\n\n")

# Check the first 5 rows of the dataframe

display(df.head(5))

print("\n")

# Check the feature names & dataset shape

print("Feature names:",list(df.columns), end="\n\n")

print("Dataset shape:",df.shape, end="\n\n")

# Check feature's unique value & value\_counts -> visualize (pie chart)

print("<<Check each feature's unique value>>")

for feature in df.columns.values:

    group\_df=df.groupby([feature], dropna=False, as\_index=False)

    plt.clf()

    plt.pie(group\_df.size()['size'],labels=group\_df.size()[feature].unique(),autopct="%1.2f%%")

    plt.title("Pie chart ("+feature+")")

    plt.show()

    print(group\_df.size(), end="\n\n")

# Delete if the record contains a NaN value.

df.dropna(axis=0, inplace=True)

df.reset\_index(drop=True, inplace=True)

# Except for academic background that is not fit for our purposes (Education Level=['University'], Age=['1-5','21-25','26-30'])

df = df[df['Education Level']!='University']

df = df[df['Age']!='01-05']

df = df[df['Age']!='21-25']

df = df[df['Age']!='26-30']

# Show dataset information

df.reset\_index(drop=True, inplace=True)

print(df.info(), end="\n\n")

# Label Encoding to convert categorical values to numeric values

ldf = df.copy()

le = preprocessing.LabelEncoder()

for feature in list(df.columns):

    ldf[feature]=le.fit\_transform(df[feature].values)

display(df)

display(ldf)

# Show One-Hot encoding result

dfOHE = df.copy()

dfOHE = pd.get\_dummies(df, columns=['Gender','Age','Location','Financial Condition','Internet Type','Network Type','Class Duration'], prefix=['Gender','Age','Location','Financial Condition','Internet Type','Network Type','Class Duration'])

display(dfOHE)

# Check feature's unique value & value\_counts -> visualize (pie chart)

print("<<Check each feature's unique value>>")

for feature in df.columns.values:

    group\_df=df.groupby([feature], dropna=False, as\_index=False)

    plt.clf()

    plt.pie(group\_df.size()['size'],labels=group\_df.size()[feature].unique(),autopct="%1.2f%%")

    plt.title("Pie chart ("+feature+")")

    plt.show()

    print(group\_df.size(), end="\n\n")

# Apply SelectKBest class to extract all features ranking

X\_independent=ldf.iloc[:,0:10]

y\_target=ldf.iloc[:,-1]

bf = feature\_selection.SelectKBest(score\_func=feature\_selection.chi2,k=10)

fit = bf.fit(X\_independent,y\_target)

dfcolumns = pd.DataFrame(X\_independent.columns)

dfscores = pd.DataFrame(fit.scores\_)

# Concatenate two dataframes for better visualization

featureScores = pd.concat([dfcolumns,dfscores],axis=1)

featureScores.columns= ['Specs','Score']

print(featureScores.nlargest(10,'Score'))

print()

# Apply extratreesclassifier class to extract important features in order

X\_independent=ldf.iloc[:,0:10]

y\_target=ldf.iloc[:,-1]

model=ExtraTreesClassifier()

model.fit(X\_independent, y\_target)

# Plot graph of feature importances for better visualization

plt.clf()

feat\_importances=pd.Series(model.feature\_importances\_,index=X\_independent.columns)

feat\_importances.nlargest(10).plot(kind='barh')

plt.show()

# So, We drop these features (Device, Education Level, IT Student).

ldf = ldf.drop(['Device','Education Level','IT Student'], axis=1)

display(ldf.head(5))

# Save clean dataset

df.to\_csv("clean.csv",header=True,index=False)

# Plot scaling results for features

# StandardScaler, MinMaxScaler, RobustScaler in order

scalers = [preprocessing.StandardScaler(), preprocessing.MinMaxScaler(), preprocessing.RobustScaler()]

titles = ['Standard Scaling','MinMax Scaling','Robust Scaling']

for scaler,title in zip(scalers,titles):

    # Scaling

    X\_ldf=ldf.drop(['Adaptivity Level'], axis=1)

    scaled\_df = scaler.fit\_transform(X\_ldf)

    scaled\_df = pd.DataFrame(scaled\_df,columns=list(X\_ldf.columns))

    # Make subplot

    fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(10,8))

    # Before Scaling plot

    ax1.set\_title("Before Scaling")

    ax1.set\_xlabel("values")

    for feature in list(X\_ldf.columns):

        sns.kdeplot(X\_ldf[feature], ax=ax1)

    # After Scaling plot

    ax2.set\_title(title)

    ax2.set\_xlabel("values")

    for feature in list(scaled\_df.columns):

        sns.kdeplot(scaled\_df[feature], ax=ax2)

    plt.show()

    # Display dataframe (before, after)

    display(ldf.head(5))

    # Make complete scaled dataframe

    scaled\_df = pd.concat([scaled\_df,ldf['Adaptivity Level']], axis=1)

    display(scaled\_df.head(5))

    # Data Analysis

    # -------------

    # Regression

    X = scaled\_df.drop(columns=['Adaptivity Level']).values

    y = scaled\_df['Adaptivity Level'].values

    # Split the dataset 9/10(8/10,7/10) for training and 1/10(2/10,3/10) for testing)

    for a,b in zip([0.9,0.8,0.7],[0.1,0.2,0.3]):

        X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,train\_size=a,test\_size=b,shuffle=True)

        # Linear regression -> fit & predict

        reg = linear\_model.LinearRegression()

        reg.fit(X\_train, y\_train)

        y\_pred=reg.predict(X\_test)

        # Evaluation

        print("<Result of tested model(training: ",a,", testing: ",b,")>", sep="")

        print(pd.DataFrame(y\_pred))

        print("<Result Score(training: ",a,", testing: ",b,")>", sep="")

        print("-->",np.round(reg.score(X\_test, y\_test),5))

        print()

    # Decision Tree

    X = scaled\_df.drop(columns=['Adaptivity Level']).values

    y = scaled\_df['Adaptivity Level'].values

    # Split the dataset 9/10(8/10,7/10) for training and 1/10(2/10,3/10) for testing)

    for a,b in zip([0.9,0.8,0.7],[0.1,0.2,0.3]):

        X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,train\_size=a,test\_size=b,shuffle=True)

        # Decision tree -> fit & predict

        tr = tree.DecisionTreeClassifier(criterion='entropy')

        tr.fit(X\_train,y\_train)

        y\_pred\_tr = tr.predict(X\_test)

        # Plotting

        plt.figure()

        tree.plot\_tree(tr,filled=True)

        plt.title("Decision Tree (training: "+str(a)+", testing: "+str(b)+")")

        plt.show()

        # Show matrix

        print("<Classification Report (Decisiion Tree)>")

        print(classification\_report(tr.predict(X\_test),y\_test))

        # Evaluation

        print("<Result of test model(training: ",a,", testing: ",b,")>", sep="")

        print("<Result Score(training: ",a,", testing: ",b,")>", sep="")

        print('-->%.5f' % accuracy\_score(y\_test, y\_pred\_tr))

        print()

    # KNN

    X = ldf.drop(columns=['Adaptivity Level']).values

    y = ldf['Adaptivity Level'].values

    # Split the dataset (4/5 for training and 1/5 for testing)

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,train\_size=0.8,test\_size=0.2,shuffle=True,stratify=y)

    # Prepare cross validation

    kfold = KFold(5,shuffle=True,random\_state=1)

    # For Check Loop

    idx=0

    # Split the dataset into 5 subsets of equal size

    for train, test in kfold.split(X):

        # For Check Loop

        idx+=1

        print("Split[",idx,"]",sep="")

        for k in [3,5]:

            # Set train data & test data

            X\_train, X\_test, y\_train, y\_test = X[train],X[test],y[train],y[test]

            # KNN -> fit & predict

            knn = KNeighborsClassifier(n\_neighbors=k)

            knn.fit(X\_train, y\_train)

            cv\_scores = cross\_val\_score(knn,X\_train,y\_train,cv=5)

            # Evaluation

            print("<Result Score(k=",k,")>",sep="")

            print("Scores:",np.round(cv\_scores,5),"/ mean:",np.round(np.mean(cv\_scores),5))

            # Show matrix

            print("<Classification Report (KNN)>")

            print(classification\_report(knn.predict(X\_test),y\_test))

            # GridSearchCV

            param\_grid={'n\_neighbors':np.arange(1,25)}

            knn\_gscv = GridSearchCV(knn,param\_grid,cv=5)

            knn\_gscv.fit(X\_train,y\_train)

            # Evaluation

            print("Best params in GridSearchCV:",knn\_gscv.best\_params\_)

            print("Best score in GridSearchCV:",knn\_gscv.best\_score\_)

            # RandomizedSearchCV

            param\_r\_grid={'n\_neighbors':np.arange(1,25)}

            knn\_rgscv = RandomizedSearchCV(knn,param\_r\_grid,cv=5,scoring='accuracy')

            knn\_rgscv.fit(X\_train,y\_train)

            # Evaluation

            print("Best params in RandomizedSearchCV:",knn\_rgscv.best\_params\_)

            print("Best score in RandomizedSearchCV:",knn\_rgscv.best\_score\_)

        print()

**GUI**

import numpy as np

import pandas as pd

from sklearn.preprocessing import MinMaxScaler

from sklearn import tree

from sklearn.metrics import accuracy\_score

# Show all data

pd.set\_option('display.max\_columns',None)

# Read data

data\_org = pd.read\_csv("clean.csv")

# Function to predict result

def predict\_result():

    # Get data

    user\_age=entry\_age.get()

    user\_loc=entry\_loc.get()

    user\_nt=entry\_nt.get()

    user\_it=entry\_it.get()

    user\_cd=entry\_cd.get()

    user\_gender=entry\_gender.get()

    user\_fc=entry\_fc.get()

    # Modify data (Age, Class Duration)

    if(int(user\_age)<=10): user\_age = '06-10'

    elif(11<=int(user\_age)<=15): user\_age = '11-15'

    elif(16<=int(user\_age)): user\_age = '16-20'

    if(int(user\_cd)==0): user\_cd='0'

    elif(1<=int(user\_cd)<=3): user\_cd='01-03'

    elif(4<=int(user\_cd)): user\_cd='03-06'

    # Add user data into dataset

    user = np.array([[str(user\_gender),str(user\_age),str(user\_loc),str(user\_fc),str(user\_it),str(user\_nt),str(user\_cd)]])

    X\_df = data\_org.drop(columns=['Education Level','IT Student','Device','Adaptivity Level'])

    df = X\_df.to\_numpy()

    print(df[0])

    print(user)

    print("\noriginal data + user")

    df = np.append(df,user, axis=0)

    print(df)

    # Normalization (dataset & user)

    from sklearn.preprocessing import LabelEncoder

    le = LabelEncoder()

    org\_df = pd.DataFrame(df,columns=['Gender','Age','Location','Financial Condition','Internet Type','Network Type','Class Duration'])

    display(org\_df)

    for feature in ['Gender','Age','Location','Financial Condition','Internet Type','Network Type','Class Duration']:

        org\_df[feature]=le.fit\_transform(org\_df[feature].values)

    scaler = MinMaxScaler()

    df = scaler.fit\_transform(org\_df)

    print("\nMinMaxScaler result")

    print(df)

    # Check user input

    user = df[-1]

    print("user",user)

    # Drop user input

    df = df[:len(df)-1]

    df = pd.DataFrame(df,columns=['Gender','Age','Location','Financial Condition','Internet Type','Network Type','Class Duration'])

    X = df.values

    y = data\_org['Adaptivity Level'].values

    tr = tree.DecisionTreeClassifier(criterion='entropy')

    tr.fit(X,y)

    y\_pred\_tr = tr.predict(user.reshape(1,-1))

    # Predicted result

    print(y\_pred\_tr)

    #Set text at GUI

    label\_final.config(text="Adaptivity Level : "+y\_pred\_tr)

from tkinter import \*

# Setting window

root = Tk()

root.title("Predict Adaptability Level")

root.geometry("340x570-500+140")

root.resizable(False, False)

# Make text

label = Label(root, text = "\n\n< Enter Your Info >\n")

label.pack()

# Age

label\_age = Label(root, text = "Age\n(6~20)")

label\_age.pack()

entry\_age = Entry(root, width=30)

entry\_age.bind("<Return>", predict\_result)

entry\_age.pack()

# Internet Type

label\_it = Label(root, text = "Internet Type\n(Mobile Data, Wifi)")

label\_it.pack()

entry\_it = Entry(root, width=30)

entry\_it.bind("<Return>", predict\_result)

entry\_it.pack()

# Location

label\_loc = Label(root, text = "Location in city\n(Yes / No)")

label\_loc.pack()

entry\_loc = Entry(root, width=30)

entry\_loc.bind("<Return>", predict\_result)

entry\_loc.pack()

# Network Type

label\_nt = Label(root, text = "Network Type\n(2G / 3G / 4G)")

label\_nt.pack()

entry\_nt = Entry(root, width=30)

entry\_nt.bind("<Return>", predict\_result)

entry\_nt.pack()

# Class Duration

label\_cd = Label(root, text = "Class Duration\n(0~6)")

label\_cd.pack()

entry\_cd = Entry(root, width=30)

entry\_cd.bind("<Return>", predict\_result)

entry\_cd.pack()

# Gender

label\_gender = Label(root, text = "Gender\n(Boy / Girl)")

label\_gender.pack()

entry\_gender = Entry(root, width=30)

entry\_gender.bind("<Return>", predict\_result)

entry\_gender.pack()

# Financial Condition

label\_fc = Label(root, text = "Financial Condition\n(Rich / Mid / Poor)")

label\_fc.pack()

entry\_fc = Entry(root, width=30)

entry\_fc.bind("<Return>", predict\_result)

entry\_fc.pack()

# Make button

button = Button(root, width=10, text="show",overrelief="solid",command=predict\_result)

button.pack()

# Make text

label\_final = Label(root,text=" ")

label\_final.pack()

# Set on the window

root.mainloop()

**Module**

**LabelEncoder.py**

import numpy as np

import pandas as pd

from sklearn.preprocessing import LabelEncoder

''' LabelEncoder

input parameter : org\_df

org\_df (type: DataFrame) >> Target of LabelEncoding

output : Return DataFrame after LabelEncoding '''

def do\_encoding(org\_df):

    le = LabelEncoder()

    for feature in list(org\_df.columns):

        org\_df[feature]=le.fit\_transform(org\_df[feature].values)

    return org\_df

**StandardScaler.py**

import numpy as np

import pandas as pd

from sklearn.preprocessing import StandardScaler

''' StandardScaler

input parameter : org\_df, target, showPlot

org\_df (type: DataFrame) >> Target of StandardScaling

target (type: String) >> Feature name of target value

showPlot (type: Bool, default=False) >> Whether to plot the result

output : Return DataFrame that after StandardScaling. Drawing graph based on whether or not plotting'''

def do\_scaling(org\_df, target, showPlot=False):

    # Init variables

    scaler = StandardScaler()

    title = 'Standard Scaling'

    # Scaling

    X\_df=org\_df.drop([target], axis=1)

    scaled\_df = scaler.fit\_transform(X\_df)

    scaled\_df = pd.DataFrame(scaled\_df,columns=list(X\_df.columns))

    # show plot

    if(showPlot == True):

        import seaborn as sns

        from matplotlib import pyplot as plt

        # Make subplot

        fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(10,8))

        # Before Scaling plot

        ax1.set\_title("Before Scaling")

        ax1.set\_xlabel("values")

        for feature in list(X\_df.columns):

            sns.kdeplot(X\_df[feature], ax=ax1)

        # After Scaling plot

        ax2.set\_title(title)

        ax2.set\_xlabel("values")

        for feature in list(scaled\_df.columns):

            sns.kdeplot(scaled\_df[feature], ax=ax2)

        plt.show()

    # Make complete scaled dataframe

    scaled\_df = pd.concat([scaled\_df,org\_df[target]], axis=1)

    return scaled\_df

**MinMaxScaler.py**

import numpy as np

import pandas as pd

from sklearn.preprocessing import MinMaxScaler

''' MinMaxScaler

input parameter : org\_df, target, showPlot

org\_df (type: DataFrame) >> Target of RobustScaling

target (type: String) >> Feature name of target value

showPlot (type: bool, default=False) >> Whether to plot the result

output : Return DataFrame that after RobustScaling. Drawing graph based on whether or not plotting

'''

def do\_scaling(org\_df, target, showPlot=False):

    # Init variables

    scaler = MinMaxScaler()

    title = 'MinMax Scaling'

    # Scaling

    X\_df=org\_df.drop([target], axis=1)

    scaled\_df = scaler.fit\_transform(X\_df)

    scaled\_df = pd.DataFrame(scaled\_df,columns=list(X\_df.columns))

    # show plot

    if(showPlot == True):

        import seaborn as sns

        from matplotlib import pyplot as plt

        # Make subplot

        fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(10,8))

        # Before Scaling plot

        ax1.set\_title("Before Scaling")

        ax1.set\_xlabel("values")

        for feature in list(X\_df.columns):

            sns.kdeplot(X\_df[feature], ax=ax1)

        # After Scaling plot

        ax2.set\_title(title)

        ax2.set\_xlabel("values")

        for feature in list(scaled\_df.columns):

            sns.kdeplot(scaled\_df[feature], ax=ax2)

        plt.show()

    # Make complete scaled dataframe

    scaled\_df = pd.concat([scaled\_df,org\_df[target]], axis=1)

    return scaled\_df

**RobustScaler.py**

import numpy as np

import pandas as pd

from sklearn.preprocessing import RobustScaler

''' RobustScaler

input parameter : org\_df, target, showPlot

org\_df (type: DataFrame) >> Target of RobustScaling

target (type: String) >> Feature name of target value

showPlot (type: bool, default=False) >> Whether to plot the result

output : Return DataFrame that after RobustScaling. Drawing graph based on whether or not plotting

'''

def do\_scaling(org\_df, target, showPlot=False):

    # Init variables

    scaler = RobustScaler()

    title = 'Robust Scaling'

    # Scaling

    X\_df=org\_df.drop([target], axis=1)

    scaled\_df = scaler.fit\_transform(X\_df)

    scaled\_df = pd.DataFrame(scaled\_df,columns=list(X\_df.columns))

    # show plot

    if(showPlot == True):

        import seaborn as sns

        from matplotlib import pyplot as plt

        # Make subplot

        fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(10,8))

        # Before Scaling plot

        ax1.set\_title("Before Scaling")

        ax1.set\_xlabel("values")

        for feature in list(X\_df.columns):

            sns.kdeplot(X\_df[feature], ax=ax1)

        # After Scaling plot

        ax2.set\_title(title)

        ax2.set\_xlabel("values")

        for feature in list(scaled\_df.columns):

            sns.kdeplot(scaled\_df[feature], ax=ax2)

        plt.show()

    # Make complete scaled dataframe

    scaled\_df = pd.concat([scaled\_df,org\_df[target]], axis=1)

    return scaled\_df

**LinearReg.py**

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn import linear\_model

''' LinearReg

input parameter : scaled\_df, target, test\_size, shuffle

scaled\_df (type: DataFrame) >> Target of DecisionTree that after Scaling

target (type: String) >> Feature name of target value

test\_size (type: Float, default: 0.25) >> Specify testset ratio when training\_test\_split

shuffle (type: Bool, default: False) >> Specify whether shuffle when training\_test\_split

output : Show regression score

'''

def do\_modeling(scaled\_df, target, test\_size=0.25, shuffle=False):

    # Split dataset (Independent / Target)

    X = scaled\_df.drop(columns=[target]).values

    y = scaled\_df[target].values

    # Split dataset (train / test)

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,train\_size=1-test\_size,test\_size=test\_size,shuffle=shuffle)

    # Linear regression -> fit & predict

    reg = linear\_model.LinearRegression()

    reg.fit(X\_train, y\_train)

    y\_pred=reg.predict(X\_test)

    # Evaluation

    print("<Result Score(training: ",str(1-test\_size),", testing: ",str(test\_size),")>", sep="")

    print("-->",np.round(reg.score(X\_test, y\_test),5))

    print()

**DecisionTree.py**

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import classification\_report

from sklearn.metrics import accuracy\_score

''' DecisionTree

input parameter : scaled\_df, target, test\_size, shuffle, criterion, showPlot

scaled\_df (type: DataFrame) >> Target of DecisionTree that after Scaling

target (type: String) >> Feature name of target value

test\_size (type: Float, default: 0.25) >> Specify testset ratio when training\_test\_split

shuffle (type: Bool, default: False) >> Specify whether shuffle when training\_test\_split

criterion (type: String, default: 'gini') >> Determine the type of criterion used in Decision Tree

showPlot (type: bool, default=False) >> Whether to plot the result

output : Show DecisionTree score and confision matrix. Drawing tree based on whether or not plotting'''

def do\_modeling(scaled\_df, target, test\_size=0.25, shuffle=False, criterion='gini', showPlot=False):

    # Split dataset (Independent / Target)

    X = scaled\_df.drop(columns=[target]).values

    y = scaled\_df[target].values

    # Split dataset (train / test)

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,train\_size=1-test\_size,test\_size=test\_size,shuffle=shuffle)

    # Decision tree -> fit & predict

    tr = DecisionTreeClassifier(criterion=criterion)

    tr.fit(X\_train,y\_train)

    y\_pred\_tr = tr.predict(X\_test)

    # show plot

    if showPlot == True:

        import matplotlib.pyplot as plt

        from sklearn.tree import plot\_tree

        # Plotting

        plt.figure()

        plot\_tree(tr,filled=True)

        plt.title("Decision Tree (training: "+str(1-test\_size)+", testing: "+str(test\_size)+")")

        plt.show()

    # Show matrix

    print("<Classification Report (Decisiion Tree)>")

    print(classification\_report(tr.predict(X\_test),y\_test))

    # Evaluation

    print("<Result of test model(training: ",str(1-test\_size),", testing: ",str(test\_size),")>", sep="")

    print("<Result Score(training: ",str(1-test\_size),", testing: ",str(test\_size),")>", sep="")

    print('-->%.5f' % accuracy\_score(y\_test, y\_pred\_tr))

    print()

**KNN.py**

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import classification\_report

from sklearn.model\_selection import cross\_val\_score

''' KNN

input parameter : scaled\_df, target, test\_size, shuffle, k

scaled\_df (type: DataFrame) >> Target of DecisionTree that after Scaling

target (type: String) >> Feature name of target value

test\_size (type: Float, default: 0.25) >> Specify testset ratio when training\_test\_split

shuffle (type: Bool, default: False) >> Specify whether shuffle when training\_test\_split

k (type: Int, default: 3) >> Specify a value for n\_neighbors in KNN

output : Show KNN score evaluated by cross\_validation=5 and confision matrix

'''

def do\_modeling(scaled\_df, target, test\_size=0.25, shuffle=False, k=3):

    # Split dataset (Independent / Target)

    X = scaled\_df.drop(columns=[target]).values

    y = scaled\_df[target].values

    # Split dataset (train / test)

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,train\_size=1-test\_size,test\_size=test\_size,shuffle=shuffle)

    # KNN -> fit & predict

    knn = KNeighborsClassifier(n\_neighbors=k)

    knn.fit(X\_train, y\_train)

    cv\_scores = cross\_val\_score(knn,X\_train,y\_train,cv=5)

    # Evaluation

    print("<Result Score(k=",k,")>",sep="")

    print("Scores:",np.round(cv\_scores,5),"/ mean:",np.round(np.mean(cv\_scores),5))

    # Show matrix

    print("<Classification Report (KNN)>")

    print(classification\_report(knn.predict(X\_test),y\_test))