

Machine Learning

Term Project\_Airline Passenger Satisfaction

Team 4

201731851 조해웅

201735849 안세훈

201835518 전소영

201935040 문병윤

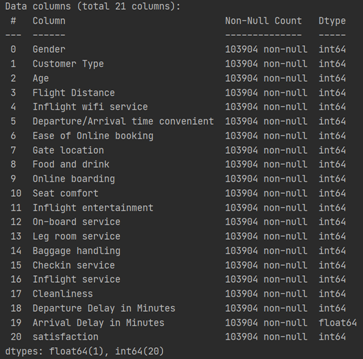
1. **Project Idea**

Currently, large airlines are annually engaged in presentations, development, and improvement activities with passengers' opinions on their services. Through this, it can be said that high-quality services are provided.

Satisfaction surveys are aimed at providing and developing services. Therefore, the purpose is to provide a clearer and more convenient service by analyzing and providing services through a passenger satisfaction survey. We tried to analyze which services affect satisfaction based on the satisfaction level reflecting the opinions of users.

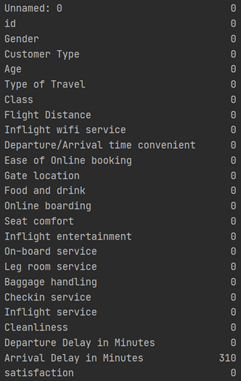
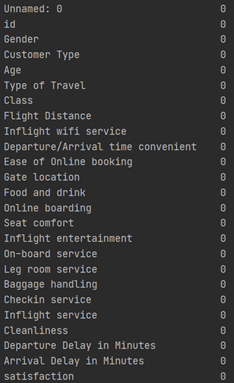
1. **Data Curation**

Data curation refers to all activities to increase the utilization value of data, such as annotation and classification in data collection and purification, and generation of data for learning. Data-based in-depth analysis and machine learning require large-scale data acquisition as well as machine readable, learnable, and understandable. Therefore, we need to analyze our dataset. Before preprocessing, we will first look at the structure and configuration of the data.



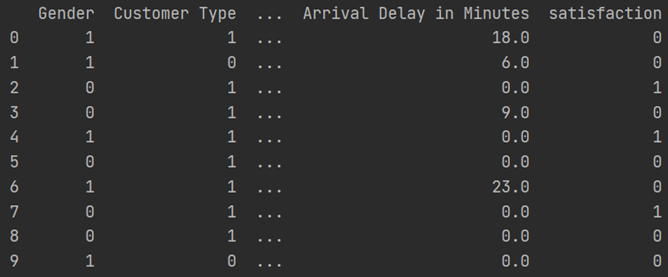
The above is the result showing the information of the selected dataset. It consists of a total of 103904 rows and 24 columns. It can be seen that it consists of 18 integers, 5 strings, and 1 Decimal.

1. **Data Preprocessing**



As a result of examining the configuration of the project dataset, a null value existed. The null data shows that there was a flight delay time. The results shown above are values indicating the existence and number of NULL values and show that they are currently present in Arrival\_delay in Minutes.

At this time, in order to do clustering later, You have to match the number of rows. However, out of approximately 100,000 data, about 300 data had NULL values, so instead of eliminating the row, they were filled with Median values. As a result, there was no NULL value as in the right result window.

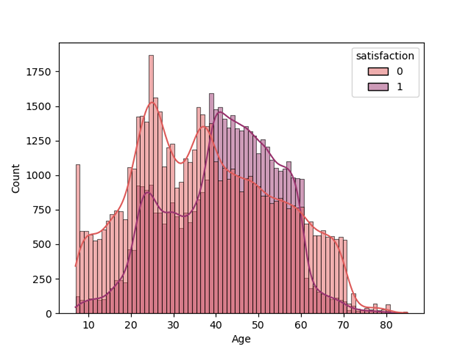


In addition, processing was performed for rows with string values, which was one way for easy classification and clustering, and label encoding was performed for categorical data. It is for processing four categorical data: gender, customer type, satisfaction, type of travel, and customer class.

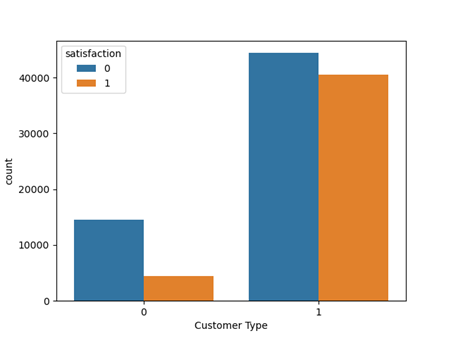
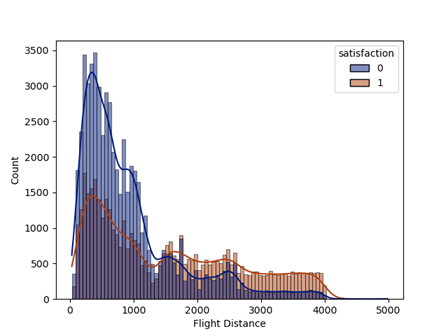
Label encoding was performed by changing data such as satisfaction/discontent, women/men, economy/business, etc. to 0/1, and the above change results were derived.

1. **Data Exploration**

Data exploration is conducted to find out the characteristics and associations of each row to perform classification and clustering. Now, We will visualize the data. (\* It is visualized data based on final satisfaction\*)



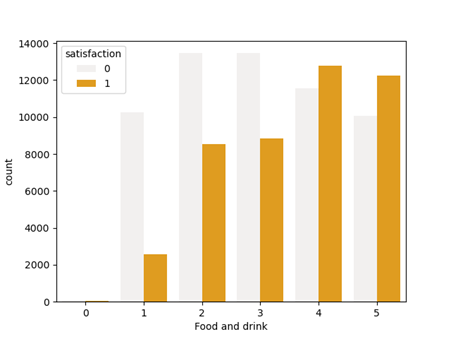
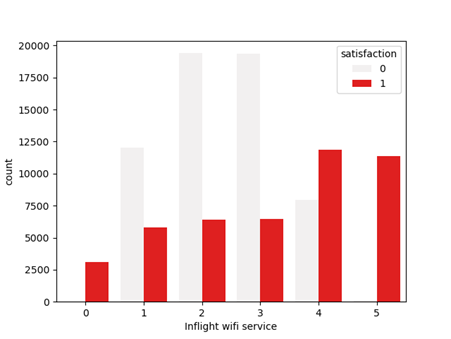
Satisfaction by age group is as follows, but it can be seen that teenagers in their 30s, 60s and 70s had more dissatisfaction than dissatisfaction, and those in their 40s and those in their 40s and 60s were more satisfied.

****

The following is the result of examining the relationship between customer type and flight distance and satisfaction. I will explain the results on the left first.

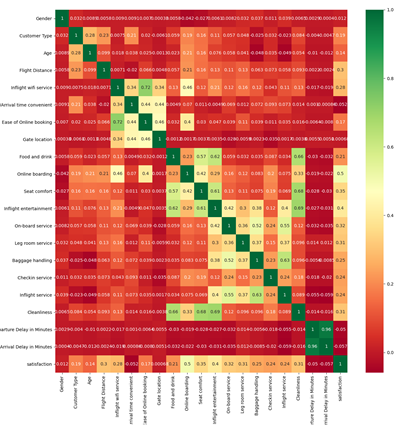
This is the result of showing that when the customer type is royal, the rate of dissatisfaction is relatively lower than when it is not royal.

The next right result shows that the shorter the flight distance, the more dissatisfaction than satisfaction, and the longer the satisfaction, the opposite of our expected results.



The following is the correlation between Wi-Fi and drinks, food, and satisfaction. The graph on the left shows the correlation between Wi-Fi service and final satisfaction, and it can be seen that the higher the Wi-Fi service satisfaction, the more clearly the overall satisfaction. The graph on the right shows the relationship between food and satisfaction, and like in-flight Wi-Fi service, satisfaction is proportional to overall satisfaction.

Until now, the correlation between each column and satisfaction has been shown. However, this method showed an overall comparison through the Heate map because it was difficult to determine the criterion that would be relatively most accurate when using a column value.

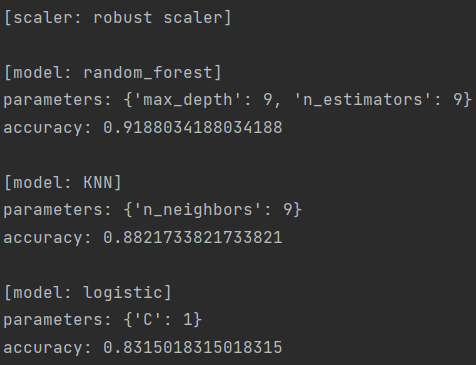


The correlation is shown in a heat map for each column, and it can be seen that the correlation is high for each similar service cluster mentioned above. Based on 0 and 1, If there are relatively many G values of RGB, it is interpreted that the relevance is high.

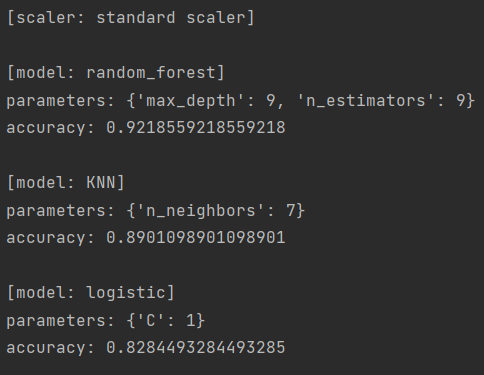
**Classification**

The reason for the classification model is to find out whether the newly entered data belongs to a related group.

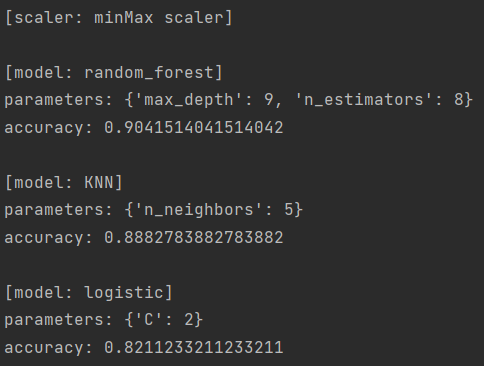
For classification, four scalar models were used: Standard Scaler, MinMax Scaler, Robust Scaler, and MaxAbs Scaler, and three classification models were used as classification models.

****

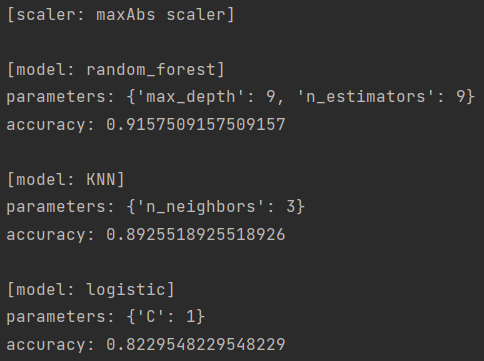
This is the result of classification using a standard scaler. It has an accuracy of about 80 percent, of which 92 percent were high in the random\_forest model. In this case, max\_depth is when parameters 9 and n\_estimator 9 are used.



This is the result of classification using minMax scaler. It has an average accuracy of 85%, of which about 90% were high in the random\_forest model. In this case, max\_depth is when parameters 9 and n\_estimator 8 are used.

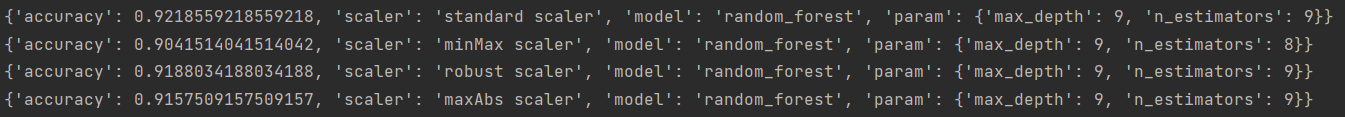


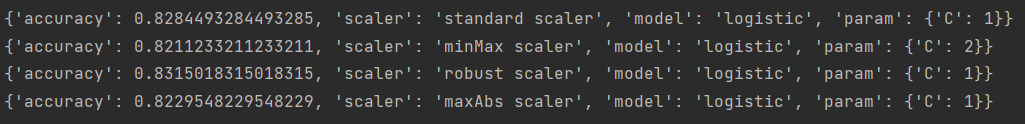
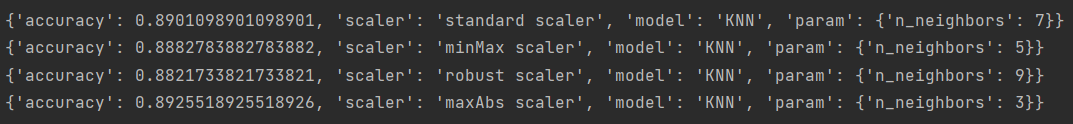
This is the result of classification using Robust Scaler. Among them, the results of using the random\_forest model also resulted in a high result of about 91%. In this case, max\_depth is when parameters 9 and n\_estimator 9 are used.



This is the result of classification using the maxAbs scaler. Among them, the results of using the random\_forest model also resulted in a high result of about 91%. In this case, max\_depth is when parameters 9 and n\_estimator 9 are used.

I will attach the results using four scalers and three classification models.





The above results show all the results, indicating that among the classification models, the accuracy of random\_forest results was the highest at an average of about 91%. In conclusion, when classification was executed, the result of the greatest accuracy was the random\_forest model using standard scaler, with max\_depth being9, n\_estimators being 9 and 92% accuracy.

**Clustering**

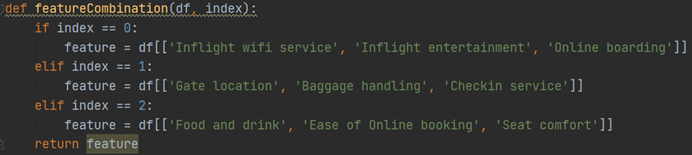
1. **Objective**

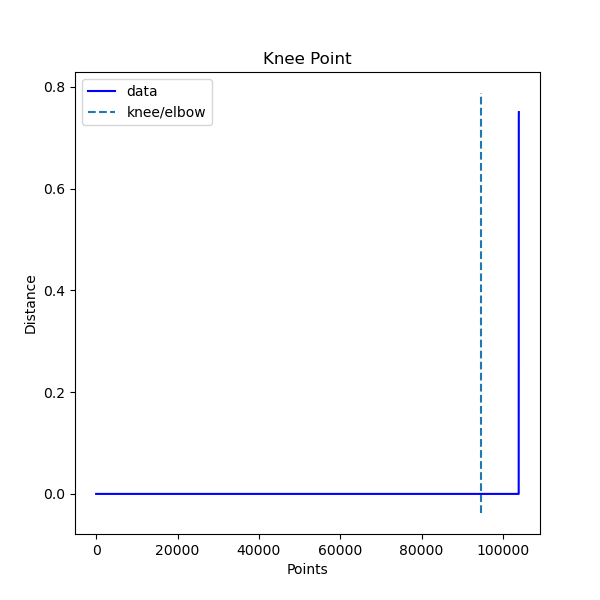
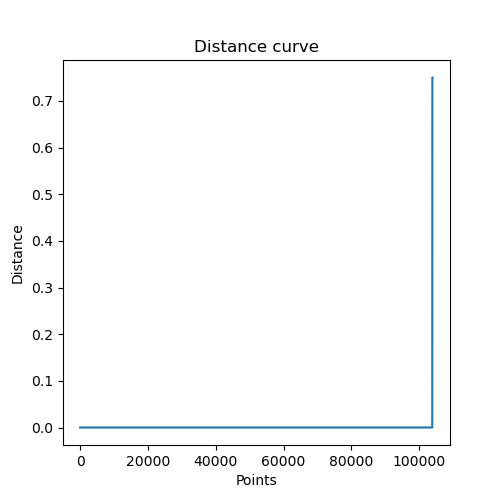
The reason for cluster analysis is to produce optimal results through optimal clusters before deriving and using results. It also aims to select a better cluster by analyzing the specificity of a specific cluster and differences between clusters.

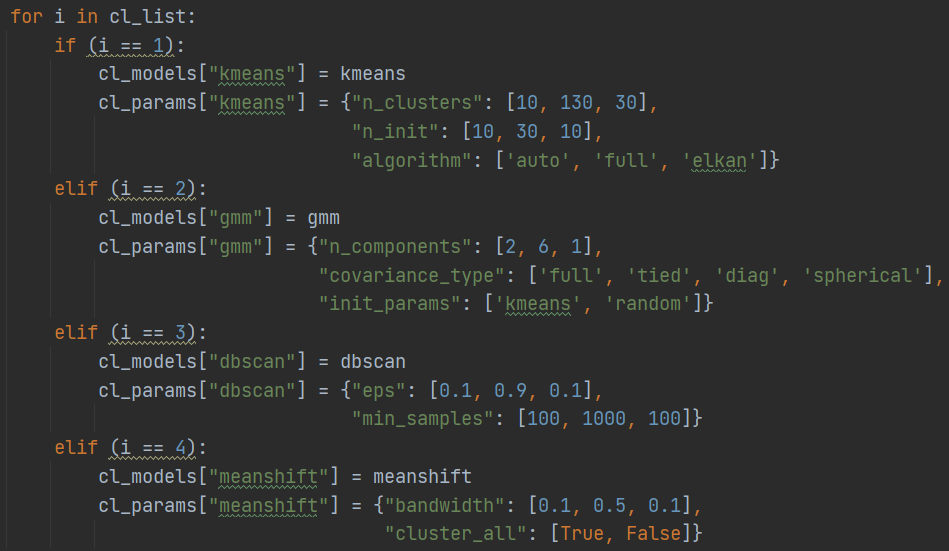
As a result, the purpose of this project is to develop, secure, and develop better services by analyzing the correlation between passenger type and service satisfaction in order to analyze and secure which services loyal customers are satisfied with.

1. **Features**

I will try clustering by dividing the current cluster into three. It is an in-flight service excluding in-flight entertainment services, ground services, and entertainment.



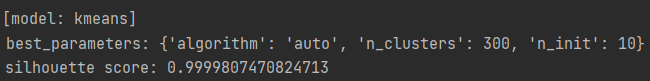
1. **Modeling**

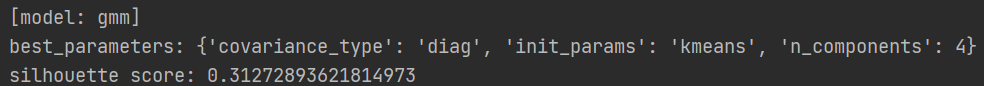


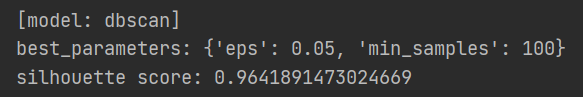
Four algorithms were used for modeling.

First, the K-means algorithm was used and proceeded using a parameter combination of n\_cluster, n\_init, and algorithm. Second, the Gmm algorithm proceeded with a combination of three parameters: n\_components, covariance\_type, and init\_params. Third, it's dbscan. It was used and proceeded with a combination of two parameters, eps and min\_sample. The last active learning was performed with a combination of two parameters, bandwidth and cluster\_all.

1. **Testing**









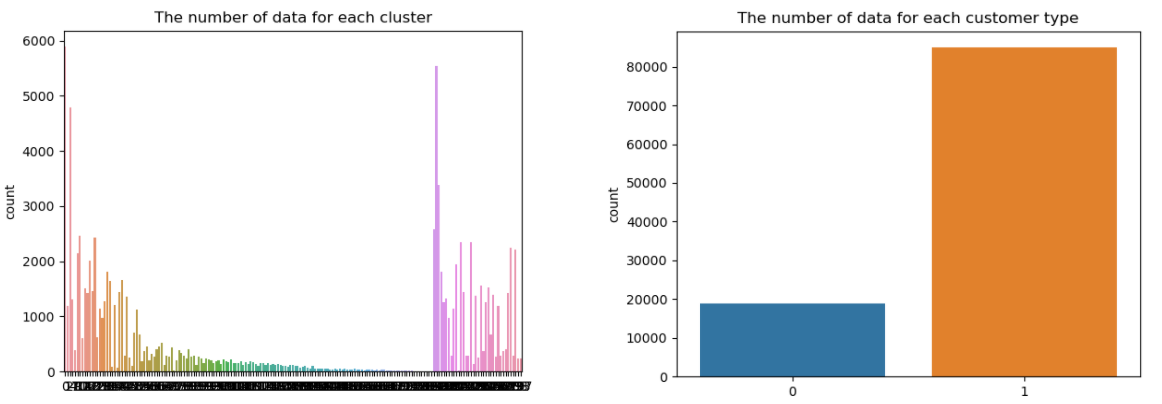
These are the results of finding the best combination for each model using Auto ML and outputting the silhouette scores.

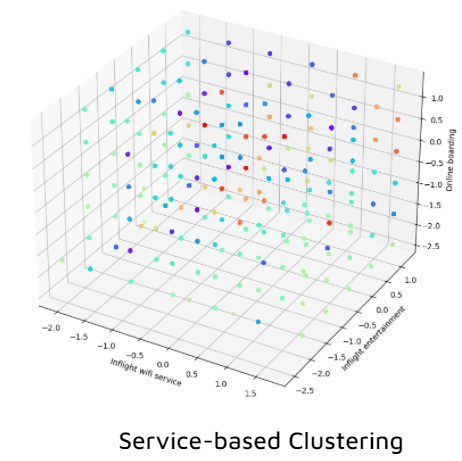
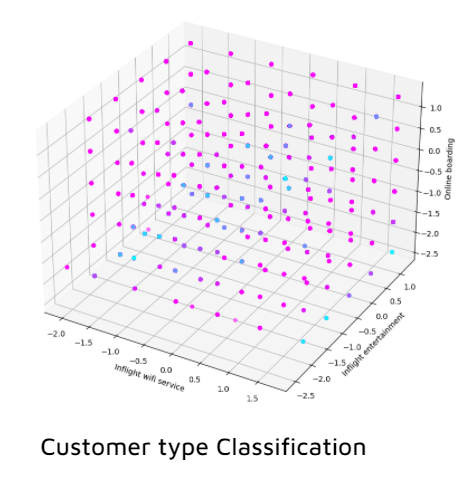
**The Best Combination**



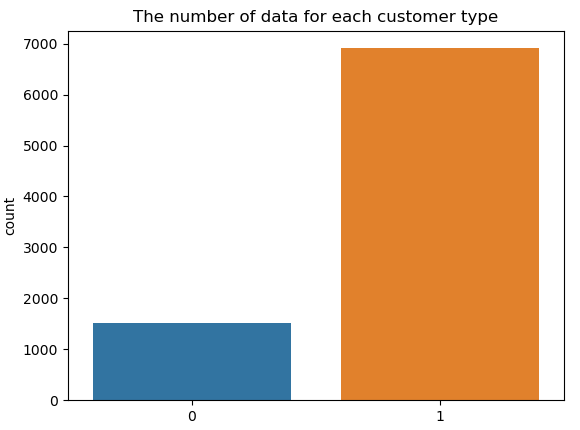
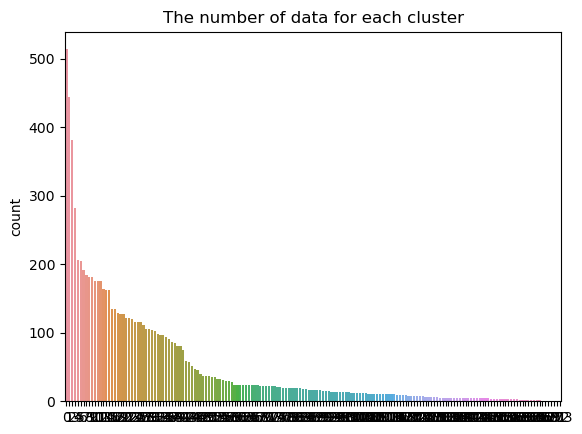
The best combination is a combination using K-Means algorithm with parameters; algorithm: auto, n\_clusters: 300, n\_init: 10, feature: inflight wifi service, inflight entertainment, online boarding. The silhouette score is 0.999, and the purity score is 0.819.

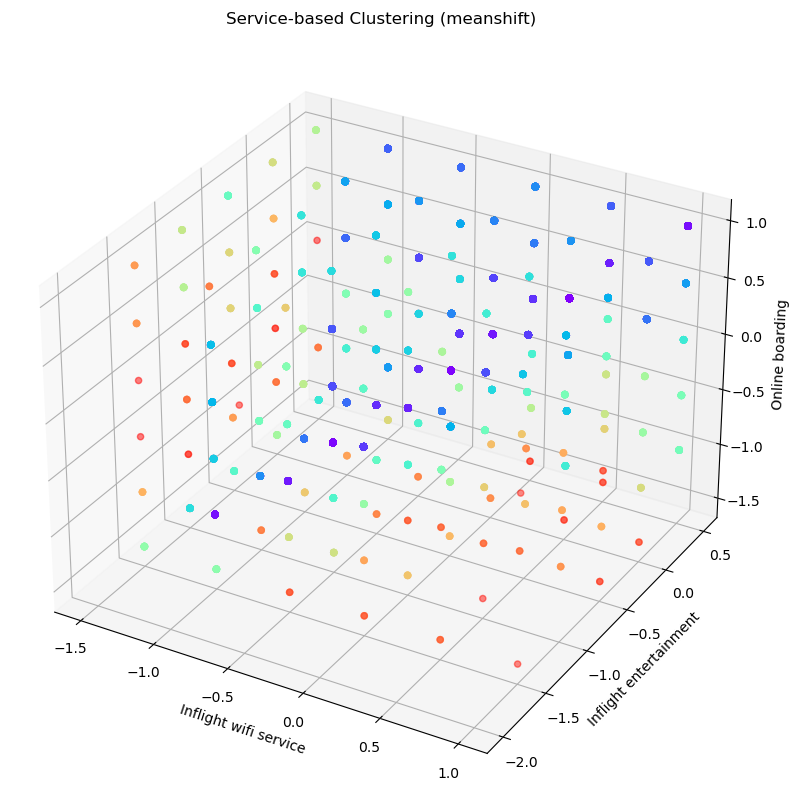
1. **Visualization**
2. The best combination with K-Means algorithm

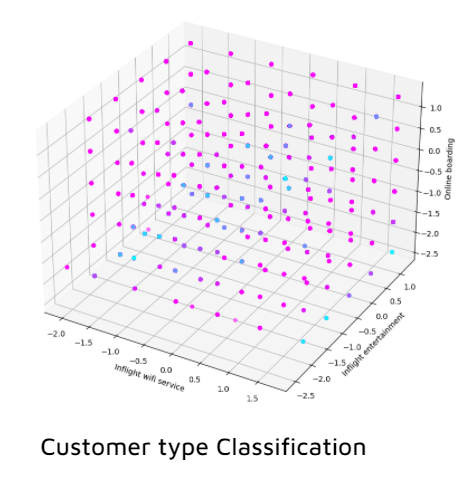




(2) The best combination with Mean-Shift algorithm



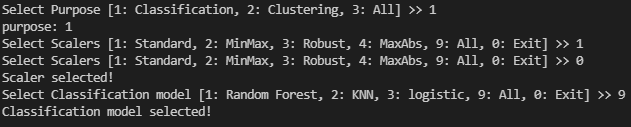




It can be seen that disloyal customers are distributed in places with relatively low satisfaction with online boarding. It is analyzed that the satisfaction of loyal customers is affected by the inflight entertainment among [inflight wifi service, inflight entertainment, online boarding], and the inflight wifi service does not significantly affect the customer type. This can be presumed to be because most of the in-flight Wi-Fi services are provided in the form of paid additional services.

**AutoML**

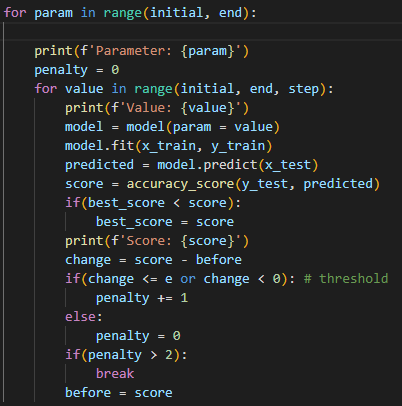
1. Customizing System



It provides users with a variety of options.  
 All options can be duplicated.

* Purpose: Classification / Clustering
* Scalers: Standard, Minmax, Robust, MaxAbs
* Classification models: RandomForest, KNN, Logistic
* Clustering models: k-means, GaussianMixture, DBSCAN, MeanShift

1. Logic



* Find Local Maximum

1. Update Parameter using step value and fit, predicted model
2. Update score and store it if it bigger than best score
3. If penalty value is more than 2, stop find parameter
4. If it reach end value, stop find parameter

* Penalty Condition

1. Accuracy change is trivial: change (current score - before score) <= e
2. Change value is negative: change < 0
3. Penalty +=1 if it meet penalty condition
4. Penalty is initialized to 0 if it didn’t meet penalty condition

3. Structure

**[Classification]**

for scaler in scalers

scaler.fit\_transform()

for model in models

Parameter tuning(model, params)

store the combination with the highest score

**[Clustering]**

for feature in featureCombination

for scaler in scalers

scaler.fit\_transform()

for model in models

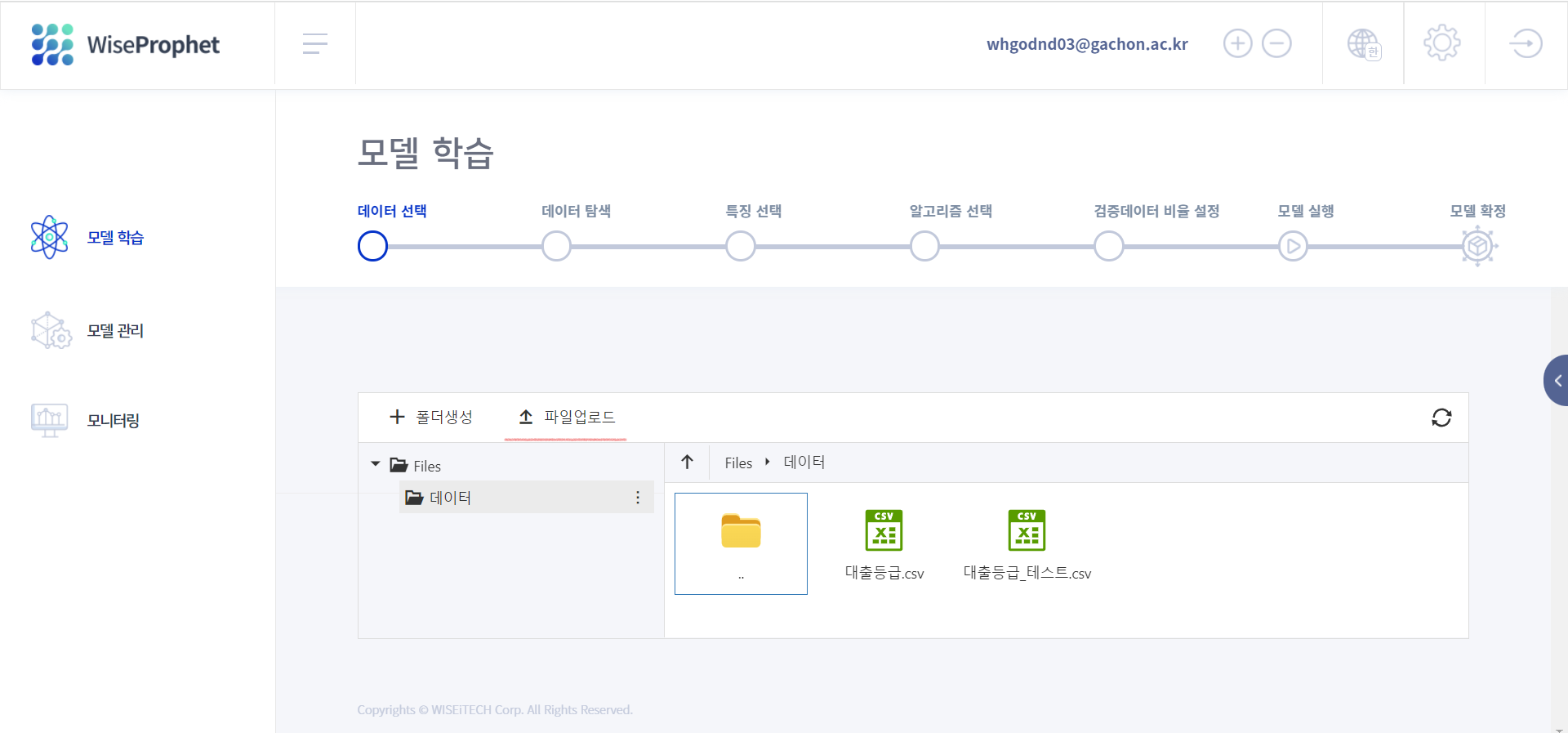
Parameter tuning(model, params)

store the combination with the highest score

Criteria for Score : classification(accuracy), clustering(silhouette score)

**Wise Prophet**

1. Select dataset



-> It brings up a dataset for learning the model. We use loan grade dataset file.

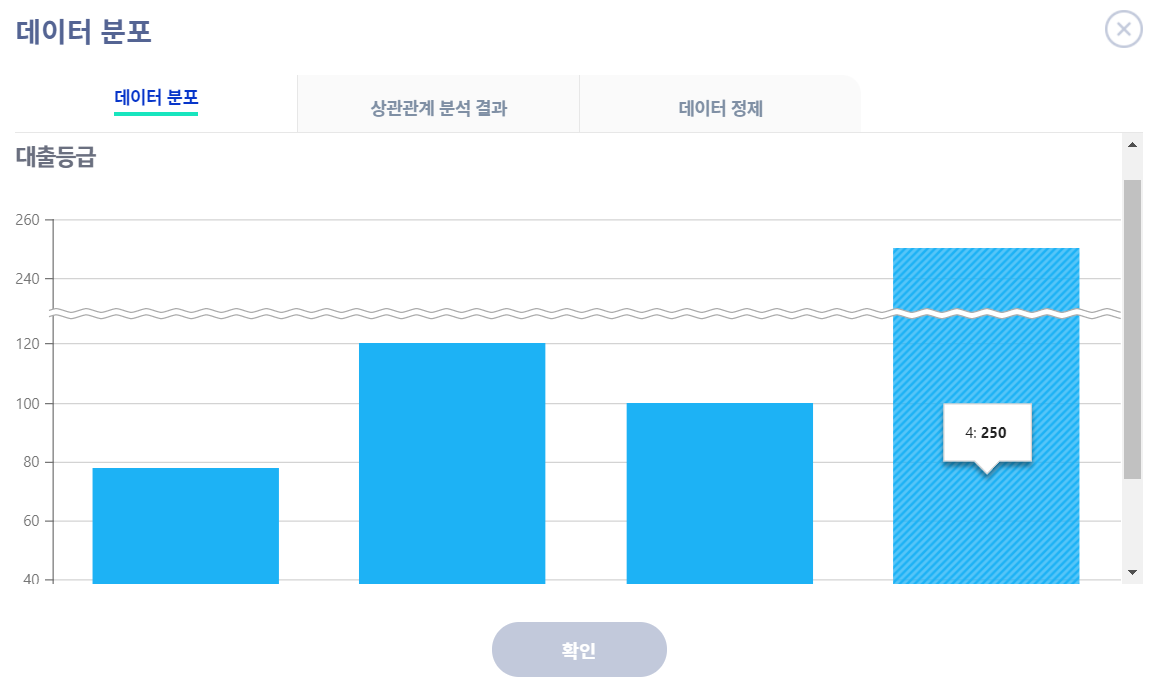
1. Data exploration



-> The loan grade was set as the target variable.

-> Meaningless variables such as customer ID were removed.

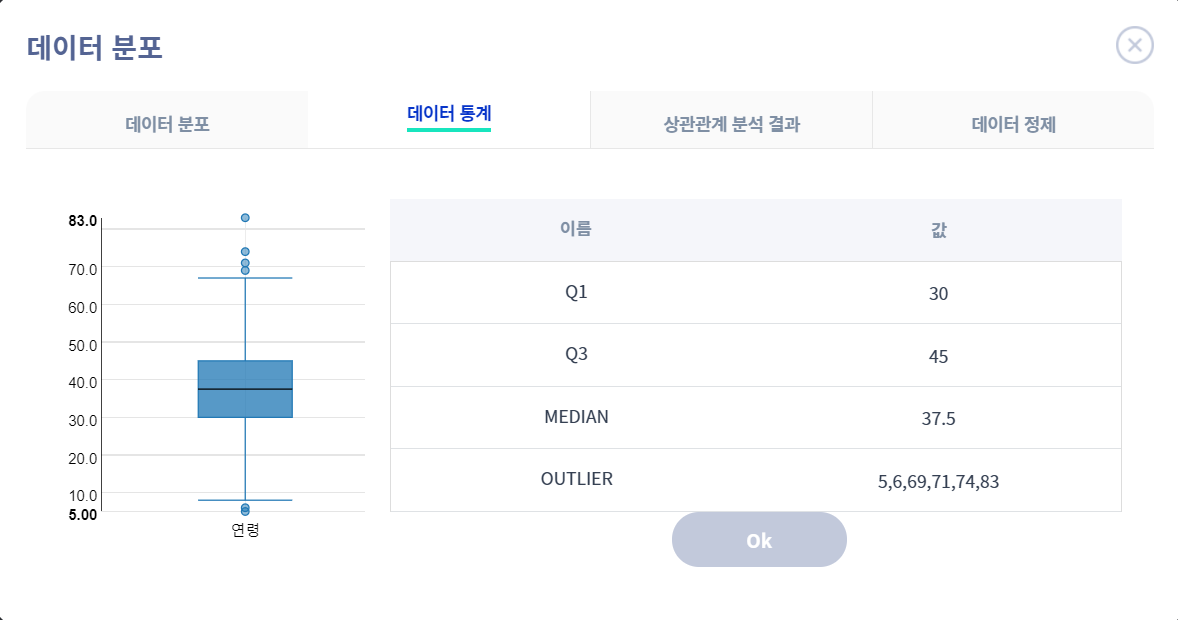
-> The type was changed to suit the characteristics of the variable.



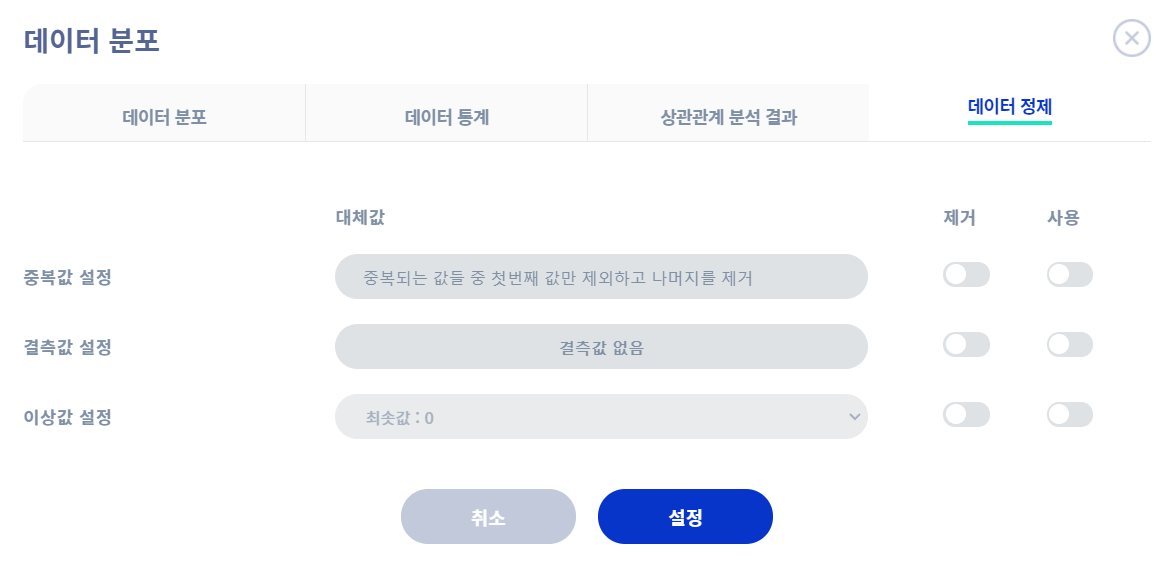
-> Loan grade : It is confirmed that the most users are distributed in grade 4.



-> As a result of performing cross-analysis between categorical variables, the target variable showed a high commercial relationship between gender and marital status.



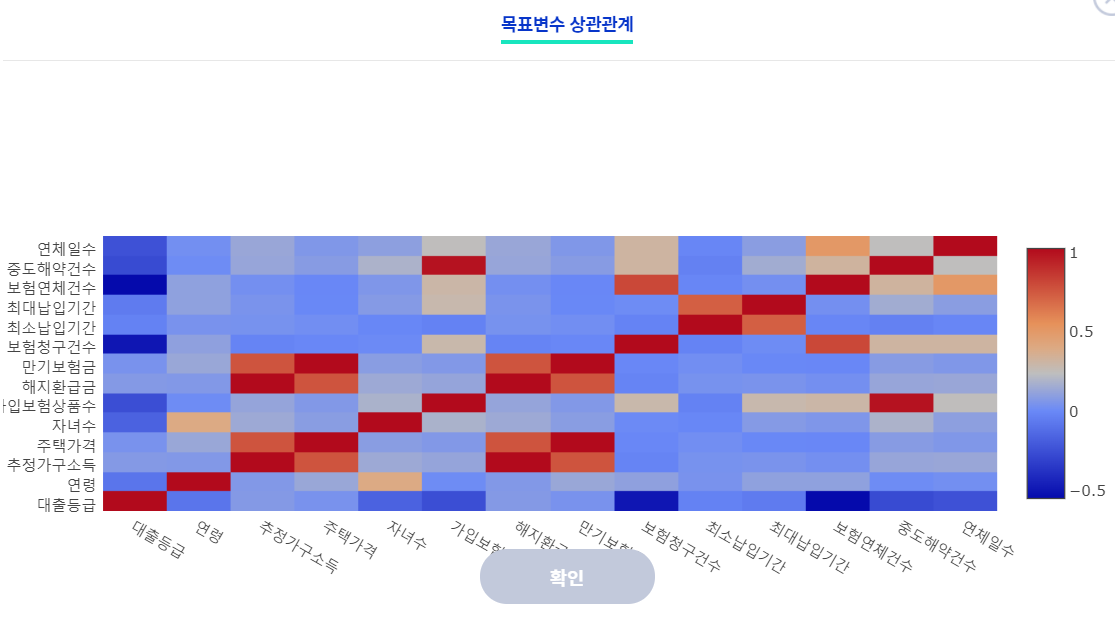
-> An approximate distribution and outliers can be identified by drawing a box plot.



-> Duplicate values, missing values, and outliers can be removed by utilizing the data purification function.

1. Feature selection



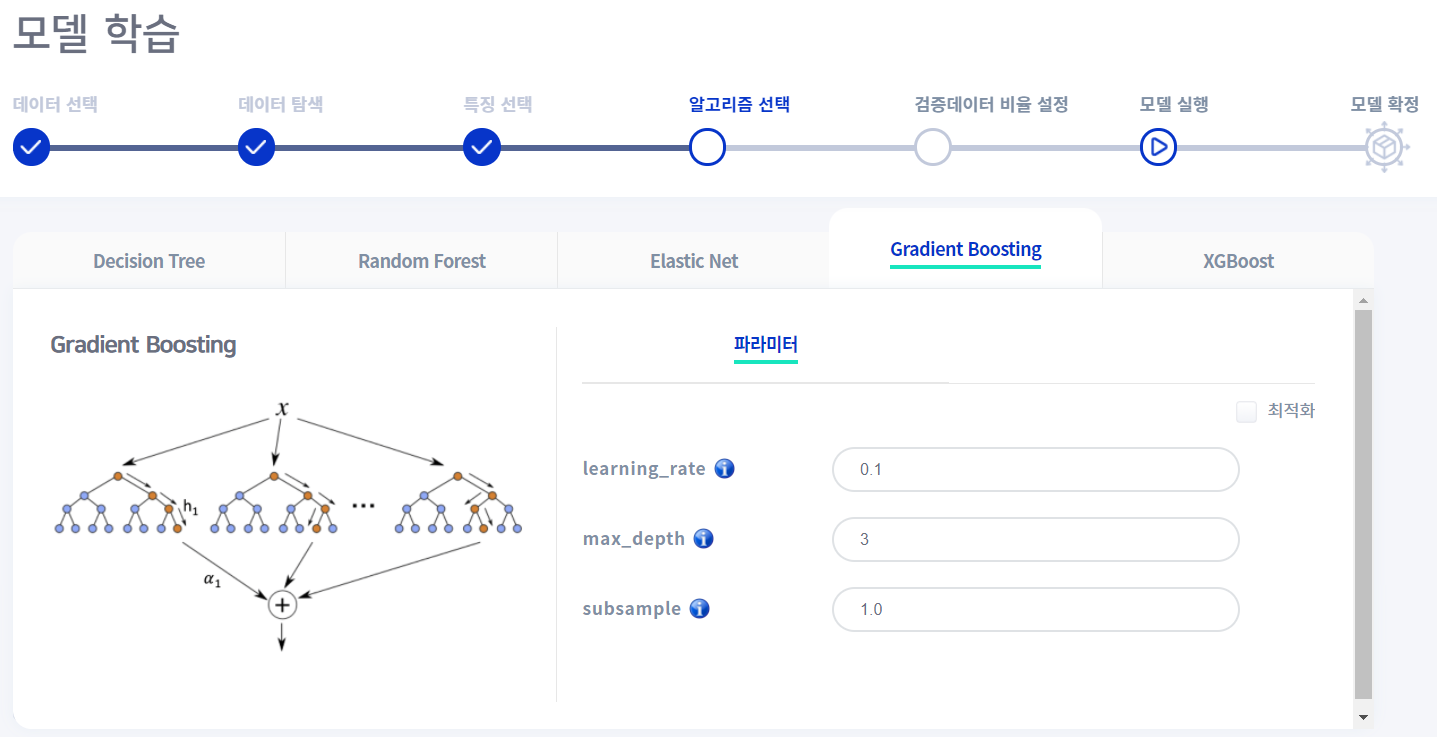


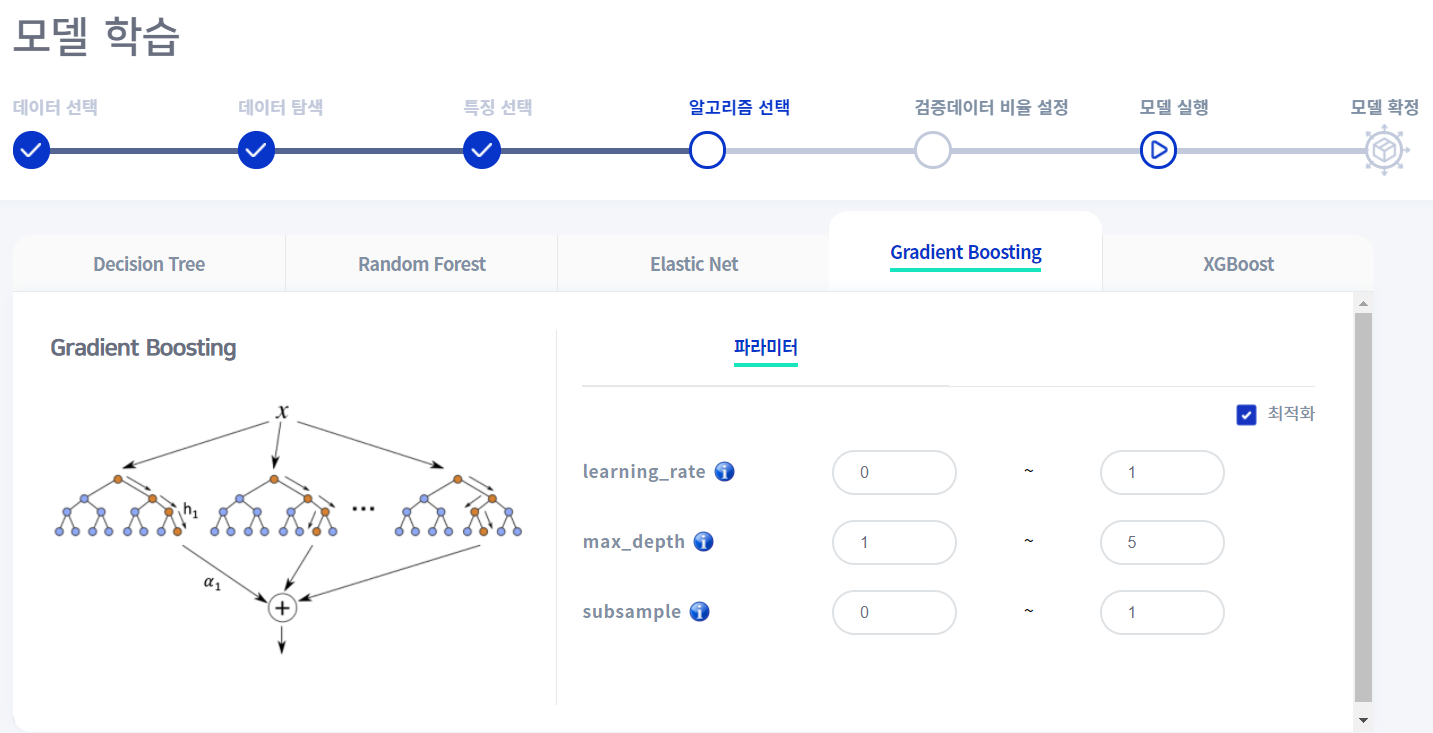
-> As a result of the correlation analysis, it can be confirmed that the number of overdue insurance and the number of insurance claims has a relatively high importance with the target variable.

1. Algorithm selection



-> Since the target variable is a categorical variable, a classification algorithm is used.





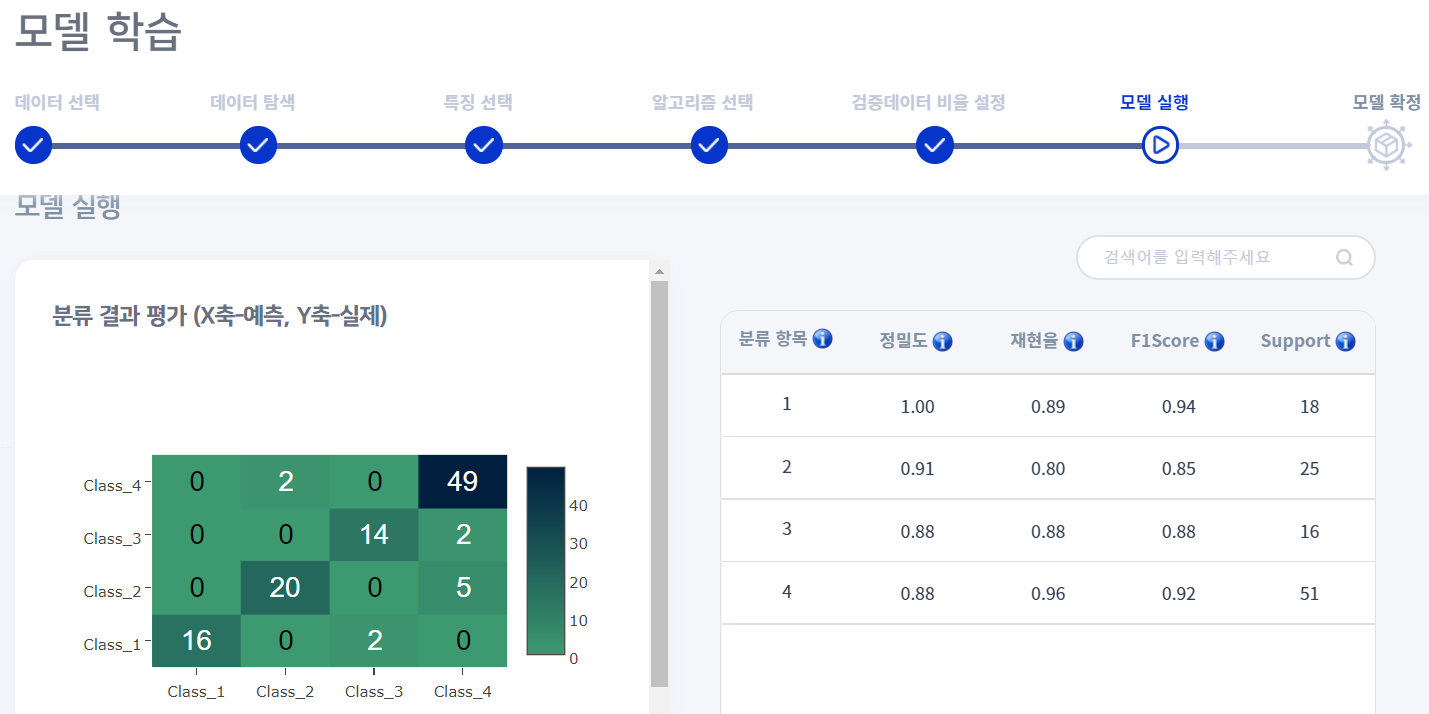
-> Among the ensemble models, the Gradient boosting algorithm was used.

1. Split data

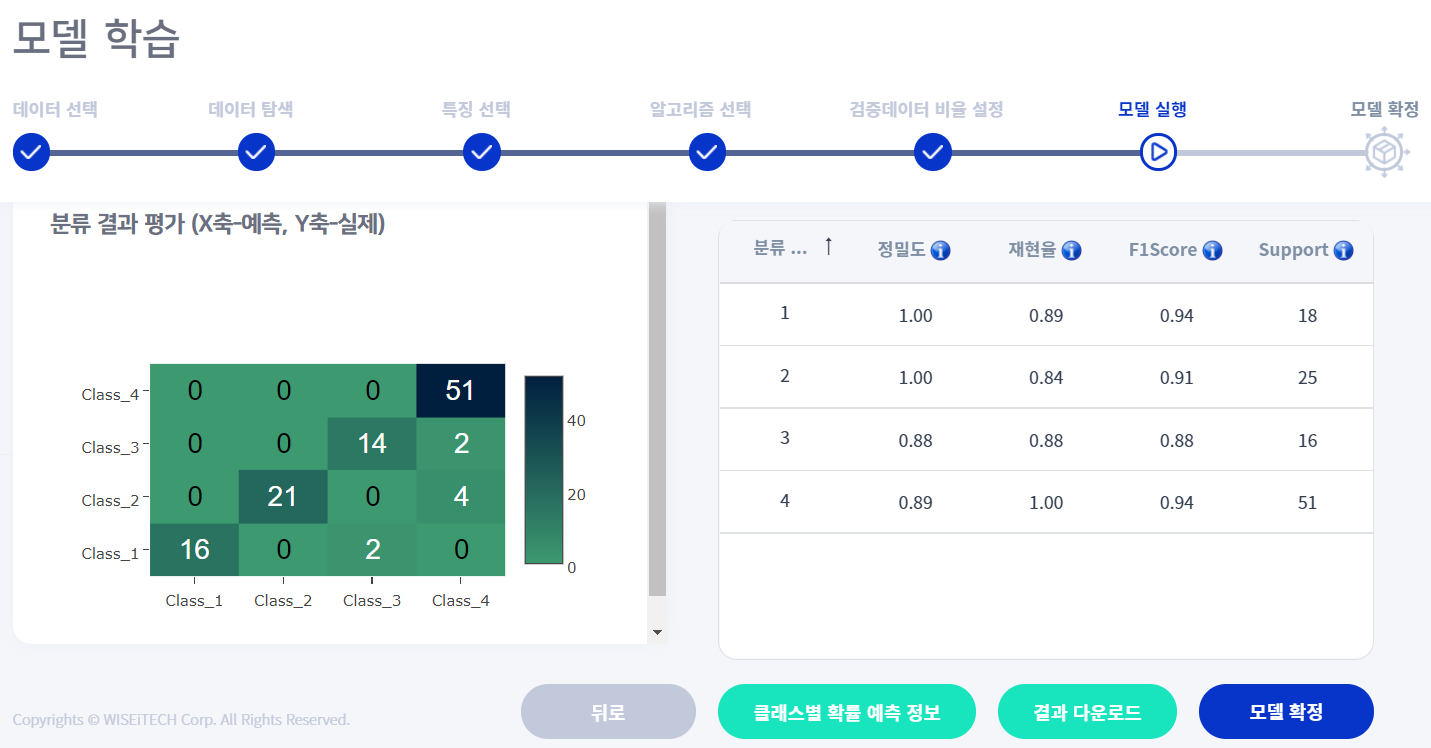


-> The evaluation data was divided into 8:2.

1. Model evaluation

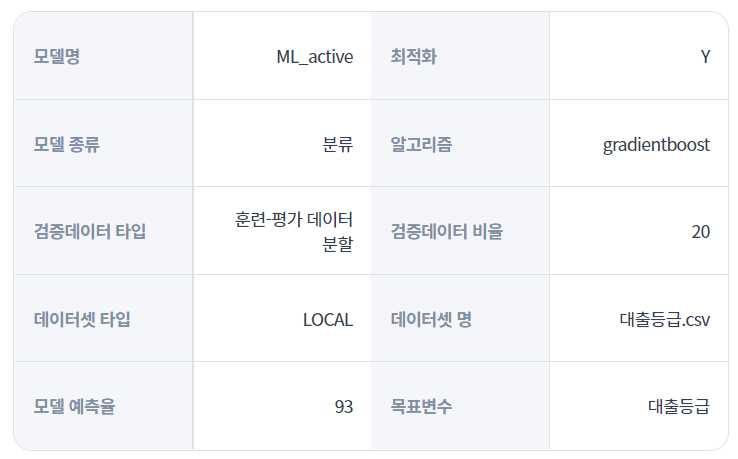


-> It shows a somewhat high level of predictive power.



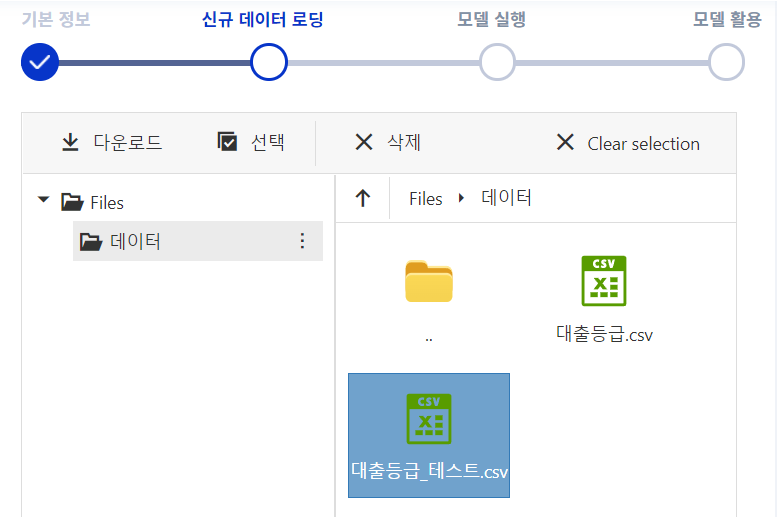
-> After removing four lower importance levels and learning, it can be seen that the predictive power slightly increased in classes 2 and 4.

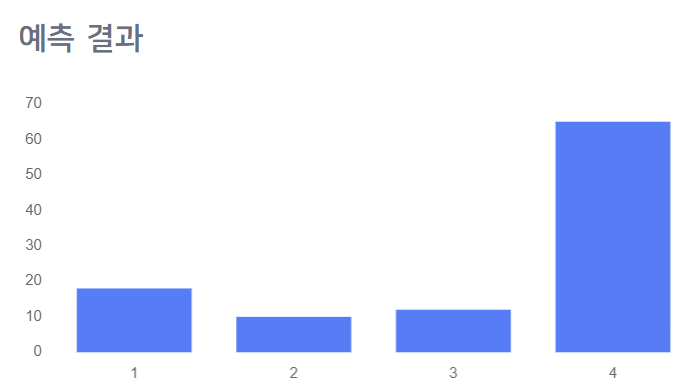
1. Final model



-> The previously learned model was selected as the final model. It shows a somewhat high level with a prediction rate of 93 percent.

1. predict using model & result





-> It was used to predict new data using the final model.

**Team Member’s contribution**

201731851 조해웅: 25%

201735849 안세훈: 35%

201835518 전소영: 30%

201935040 문병윤: 10%

**GitHub URL**[**https://github.com/ashpurple/Machine-Learning-Project**](https://github.com/ashpurple/Machine-Learning-Project)

**Source Code**

import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
import seaborn as sns  
from matplotlib import cm  
  
# Scaler  
from sklearn.preprocessing import MinMaxScaler  
from sklearn.preprocessing import StandardScaler  
from sklearn.preprocessing import RobustScaler  
from sklearn.preprocessing import MaxAbsScaler  
# Classification  
from sklearn import tree  
from sklearn.linear\_model import LogisticRegression  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn import svm  
from sklearn.ensemble import RandomForestClassifier  
# Cluster  
from sklearn.cluster import KMeans  
from sklearn.cluster import DBSCAN  
from sklearn.cluster import MeanShift  
from sklearn.mixture import GaussianMixture  
from sklearn.cluster import estimate\_bandwidth  
# Cluster Evaluation  
from sklearn import metrics  
from sklearn.metrics import silhouette\_score  
from sklearn.metrics import silhouette\_samples  
from kneed import KneeLocator  
from sklearn.neighbors import NearestNeighbors  
from mpl\_toolkits.mplot3d import Axes3D  
# Validation  
from sklearn.model\_selection import GridSearchCV  
from sklearn.model\_selection import KFold  
from sklearn.metrics import accuracy\_score  
from sklearn.model\_selection import cross\_val\_score  
  
import warnings  
import os  
warnings.filterwarnings(action='ignore')  
  
purpose = 0 # global variable  
  
# -----------------Data Inspection----------------- #  
def dataExploration(df):  
 print(df.head())  
 print(df.shape)  
 print(df.info())  
 print(df.isnull().sum())  
 plt.figure(figsize=(10,10))  
 sns.countplot(x='Online boarding',hue="satisfaction",data=df,color="green")  
 plt.show()  
 sns.histplot(x='Age',hue="satisfaction",data=df,kde=True,palette="flare")  
 plt.show()  
 sns.countplot(x='Customer Type',hue="satisfaction",data=df)  
 plt.show()  
 sns.histplot(x='Flight Distance',hue="satisfaction",data=df,kde=True,palette="dark")  
 plt.show()  
 sns.countplot(x='Inflight wifi service',hue="satisfaction",data=df,color="red")  
 plt.show()  
 sns.countplot(x='Food and drink',hue="satisfaction",data=df,color="orange")  
 plt.show()  
 plt.figure(figsize = (15,15))  
 sns.heatmap(df.corr(), annot = True, cmap = "RdYlGn")  
 plt.show()  
  
# -----------------Preprocessing---------------- #  
def findMissingValue(df):  
 # check missing value  
 # only 'Arrival Delay in Minutes' has missing values  
 #df.dropna(inplace=True)  
 df.fillna(df.mean(), inplace = True)  
 return df  
  
def encoding(df):  
 df.drop(["Unnamed: 0","id"],axis=1,inplace=True)  
 df["Gender"] = df["Gender"].map({"Male":1,"Female":0})  
 df["Customer Type"] = df["Customer Type"].map({"Loyal Customer":1,"disloyal Customer":0})  
 # Type of class  
 df.drop(["Type of Travel","Class"],axis=1,inplace=True)  
 # Statisfaction  
 df["satisfaction"] = df["satisfaction"].map({"satisfied":1,"neutral or dissatisfied":0})  
   
 return df  
  
# -----------------Classification---------------- #  
def classification(x\_train, y\_train, x\_test, y\_test, scalers, models, params\_dict):  
 best\_accuracy = {}  
 # find the best parameter by using grid search  
 for scaler\_key, scaler in scalers.items():  
 x\_train = scaler.fit\_transform(x\_train)  
 x\_test = scaler.fit\_transform(x\_test)  
 print(f'\n[scaler: {scaler\_key}]')  
 for model\_key, model in models.items():  
 print(f'\n[model: {model\_key}]')  
  
 new\_dict = dict()  
 for param, value\_list in params\_dict[model\_key].items():  
 start = value\_list[0]  
 end = value\_list[1]  
 step = value\_list[2]  
 new\_dict[param] = [x for x in np.arange(start, end, step)]  
   
 # grid search  
 grid = GridSearchCV(model, param\_grid=new\_dict)  
 grid.fit(x\_train, y\_train)  
 print(f'parameters: {grid.best\_params\_}')  
 best\_model = grid.best\_estimator\_  
 predicted = best\_model.predict(x\_test)  
 accuracy = accuracy\_score(y\_test, predicted)  
  
 # parameter tuning  
 best\_score = 0  
 e = 0.1  
 before = 0  
 initial= 0  
 step = 1  
 end = 10  
 for param in range(initial, end):  
 if(accuracy < 100):  
 continue  
 print(f'Parameter: {param}')  
 penalty = 0  
 for value in range(initial, end, step):  
 print(f'Value: {value}')  
 model = model(param = value)  
 model.fit(x\_train, y\_train)  
 predicted = model.predict(x\_test)  
 score = accuracy\_score(y\_test, predicted)  
 if(best\_score < score):  
 best\_score = score  
 print(f'Score: {score}')  
 change = score - before  
 if(change <= e or change < 0): # threshold  
 penalty += 1  
 else:  
 penalty = 0  
 if(penalty > 2):  
 break  
 before = score  
  
 # save the 10 highest accuracy and parameters each models  
 list\_size = 10  
 list\_size -= 1  
 flag = False  
  
 target\_dict = {'accuracy': accuracy,   
 'scaler': scaler\_key,  
 'model': model\_key,   
 'param': grid.best\_params\_}  
  
 # save accuracy  
 if model\_key not in best\_accuracy.keys():  
 best\_accuracy[model\_key] = []  
 if len(best\_accuracy[model\_key]) <= list\_size:  
 best\_accuracy[model\_key].append(target\_dict)  
  
 # insert accuracy  
 elif best\_accuracy[model\_key][-1]['accuracy'] < accuracy:  
 for i in range(1, list\_size):  
 if best\_accuracy[model\_key][list\_size - 1 - i]['accuracy'] > accuracy:  
 best\_accuracy[model\_key].insert(list\_size - i, target\_dict)  
 best\_accuracy[model\_key].pop()  
 flag = True  
 break  
 if flag is False:  
 best\_accuracy[model\_key].insert(0, target\_dict)  
 best\_accuracy[model\_key].pop()  
  
 print(f'accuracy: {accuracy}', end='')  
 print()  
  
 return best\_accuracy  
  
# -----------------Clustering----------------- #  
def featureCombination(df, index):  
 if index == 0:  
 feature = df[['Inflight wifi service', 'Inflight entertainment', 'Online boarding']]  
 elif index == 1:  
 feature = df[['Gate location', 'Baggage handling', 'Checkin service']]  
 elif index == 2:  
 feature = df[['Food and drink', 'Ease of Online booking', 'Seat comfort']]  
 return feature  
  
# function for store combination that has the best accuracy  
def clustering(df, scalers, models, params\_dict):  
 best\_combination = {}  
 best\_score = 0  
 best\_X = 0  
 best\_label = 0  
   
 # Sample Data  
 for index in range(1):  
 X = featureCombination(df, index)  
 feature = X.columns.tolist()  
 print(f'\n[feature: {feature}]')  
  
 num\_cols = X.select\_dtypes(include=['int64', 'float64']).columns.to\_list() # numerical value  
  
 # find the best parameter by using grid search  
 for scaler\_key, scaler in scalers.items():  
 scaled\_X = scaler.fit\_transform(X[num\_cols])  
 print(f'\n[scaler: {scaler\_key}]')  
  
 knee\_method(scaled\_X)  
   
 for model\_key, model in models.items():  
 print(f'\n[model: {model\_key}]')  
 cv = [(slice(None), slice(None))]  
  
 new\_dict = dict()  
 for param, value\_list in params\_dict[model\_key].items():  
 if type(value\_list[0]) is int:  
 start = value\_list[0]  
 end = value\_list[1]  
 step = value\_list[2]  
 new\_dict[param] = [x for x in np.arange(start, end, step)]  
 else:  
 new\_dict[param] = value\_list  
  
 # Grid Search  
 if (model\_key == 'meanshift'): # mean-shift  
 grid = GridSearchCV(estimator=model,  
 # param\_grid=estimate\_bandwidth(scaled\_X),  
 param\_grid=new\_dict,  
 scoring=silhouette\_scorer,  
 cv=cv)  
 else: # other models  
 grid = GridSearchCV(estimator=model,  
 param\_grid=new\_dict,  
 scoring=silhouette\_scorer,  
 cv=cv)  
 grid.fit(scaled\_X)  
   
 print(f'best\_parameters: {grid.best\_params\_}')  
 score = grid.best\_score\_  
 if (best\_score < score):  
 best\_score = score  
 best\_X = scaled\_X  
 best\_label = grid.best\_estimator\_  
   
 target\_dict = {'silhouette': score,  
 'scaler': scaler\_key,  
 'model': model\_key,  
 'param': grid.best\_params\_,   
 'feature': feature  
 }  
   
 list\_size = 10  
 list\_size -= 1  
 flag = False  
  
 # save accuracy  
 if model\_key not in best\_combination.keys():  
 best\_combination[model\_key] = []  
 if len(best\_combination[model\_key]) <= list\_size:  
 best\_combination[model\_key].append(target\_dict)  
  
 # insert accuracy  
 elif best\_combination[model\_key][-1]['silhouette'] < score:  
 for i in range(1, list\_size):  
 if best\_combination[model\_key][list\_size - 1 - i]['silhouette'] > score:  
 best\_combination[model\_key].insert(list\_size - i, target\_dict)  
 best\_combination[model\_key].pop()  
 flag = True  
 break  
 if flag is False:  
 best\_combination[model\_key].insert(0, target\_dict)  
 best\_combination[model\_key].pop()  
  
 print(f'silhouette score: {score}', end='')  
 print()  
  
 return best\_combination, best\_X, best\_label  
  
def clustering\_with\_best(df, best\_scaler, best\_model, best\_params):  
 # Sample Data  
 for index in range(1):  
 X = featureCombination(df, index)  
 feature = X.columns.tolist()  
 #print(f'\n[feature: {feature}]')  
  
 num\_cols = X.select\_dtypes(include=['int64', 'float64']).columns.to\_list() # numerical value  
  
 if (best\_scaler == 'standard scaler'):  
 scaler = StandardScaler()  
 elif (best\_scaler == 'minMax scaler'):  
 scaler = MinMaxScaler()  
 elif (best\_scaler == 'robust scaler'):  
 scaler = RobustScaler()  
 elif (best\_scaler == 'maxAbs scaler'):  
 scaler = MaxAbsScaler()  
  
 scaled\_X = scaler.fit\_transform(X[num\_cols])  
  
 if (best\_model == 'kmeans'):  
 kmeans = KMeans(n\_clusters=best\_params['n\_clusters'],  
 n\_init=best\_params['n\_init'],  
 algorithm=best\_params['algorithm'])  
 kmeans.fit(scaled\_X)  
 labels = kmeans.labels\_  
  
 elif (best\_model == 'gmm'):  
 gmm = GaussianMixture(n\_components=best\_params['n\_components'],  
 covariance\_type=best\_params['covariance\_type'],  
 init\_params=best\_params['init\_params'])  
 labels = gmm.fit\_predict(scaled\_X)  
  
 elif (best\_model == 'dbscan'):  
 dbscan = DBSCAN(eps=best\_params['eps'],  
 min\_samples=best\_params['min\_samples'])  
 dbscan.fit(scaled\_X)  
 labels = dbscan.labels\_  
 elif (best\_model == 'meanshift'):  
 meanshift = MeanShift(bandwidth=best\_params['bandwidth'],  
 cluster\_all=best\_params['cluster\_all'])  
 meanshift.fit(scaled\_X)  
 labels = meanshift.labels\_  
  
 display\_scatter\_purity(best\_model, scaled\_X, labels, df.loc[:]['Customer Type'], feature)  
  
# -----------------Clustering Evaluation----------------- #  
def clustering\_result\_analysis(train, best\_combi):  
 best\_scaler = best\_combi.get('scaler')  
 best\_model = best\_combi.get('model')  
 best\_param = best\_combi.get('param')  
  
 clustering\_with\_best(train, best\_scaler, best\_model, best\_param)  
  
def display\_scatter\_purity(model\_key, x, labels, type, feature):  
  
 df = pd.DataFrame(x, columns=feature)  
 df.insert(0, "Cluster", labels, True)  
 df.insert(1, "Customer Type", type, True)  
 customerType = type.to\_numpy()  
 # print(df.head(30))  
  
 sns.countplot(labels)  
 plt.title("The number of data for each cluster")  
 plt.show()  
  
 sns.countplot(customerType)  
 plt.title("The number of data for each customer type")  
 plt.show()  
  
 fig = plt.figure(figsize=(10, 10))  
 ax = fig.add\_subplot(111, projection='3d')  
 ax.set\_title('Service-based Clustering ({})'.format(model\_key))  
 ax.set\_xlabel(df.columns[2], fontsize=10)  
 ax.set\_ylabel(df.columns[3], fontsize=10)  
 ax.set\_zlabel(df.columns[4], fontsize=10)  
 x = df.iloc[:, 2]  
 y = df.iloc[:, 3]  
 z = df.iloc[:, 4]  
 ax.scatter(x, y, z, c=labels, s=20, alpha=0.5, cmap='rainbow')  
  
 fig = plt.figure(figsize=(10, 10))  
 ax = fig.add\_subplot(111, projection='3d')  
 ax.set\_title('Customer type Classification ({})'.format(model\_key))  
 ax.set\_xlabel(df.columns[2], fontsize=10)  
 ax.set\_ylabel(df.columns[3], fontsize=10)  
 ax.set\_zlabel(df.columns[4], fontsize=10)  
 x = df.iloc[:, 2]  
 y = df.iloc[:, 3]  
 z = df.iloc[:, 4]  
 ax.scatter(x, y, z, c=customerType, s=20, alpha=0.5, cmap='cool')  
 plt.show()  
  
 purityScore = purity\_scorer(customerType, labels)  
 print("'purity score' :",purityScore)  
  
def purity\_scorer(target, y\_pred):  
 contingency\_matrix = metrics.cluster.contingency\_matrix(target, y\_pred)  
 score = np.sum(np.amax(contingency\_matrix, axis=0)) / np.sum(contingency\_matrix)  
 return score  
  
def silhouette\_scorer(estimator, X):  
 labels = estimator.fit\_predict(X)  
 score = silhouette\_score(X, labels, metric='euclidean')  
 return score  
  
def display\_silhouette\_plot(X, labels):  
 sil\_score = metrics.silhouette\_score(X, labels, metric='euclidean')  
 n\_clusters\_ = len(set(labels)) - (1 if -1 in labels else 0)  
 print("For n\_clusters =", n\_clusters\_, "The average silhouette score is :", sil\_score)  
 # compute the silhouette scores for each sample  
 sample\_silhouette\_values = silhouette\_samples(X, labels)  
  
 fig, ax1 = plt.subplots()  
 fig.set\_size\_inches(18, 7)  
 ax1.set\_xlim([-0.1, 1])  
 ax1.set\_ylim([0, len(X) + (n\_clusters\_ + 1) \* 10])  
 y\_lower = 10  
 for i in range(n\_clusters\_):  
 ith\_cluster\_silhouette\_values = sample\_silhouette\_values[labels == i]  
 ith\_cluster\_silhouette\_values.sort()  
 size\_cluster\_i = ith\_cluster\_silhouette\_values.shape[0]  
 y\_upper = y\_lower + size\_cluster\_i  
 color = cm.nipy\_spectral(float(i) / n\_clusters\_)  
 ax1.fill\_betweenx(np.arange(y\_lower, y\_upper), 0, ith\_cluster\_silhouette\_values, facecolor=color,  
 edgecolor=color, alpha=0.7)  
 ax1.text(-0.05, y\_lower + 0.5 \* size\_cluster\_i, str(i))  
 y\_lower = y\_upper + 10  
 ax1.set\_title("The silhouette plot for the various clusters")  
 ax1.set\_xlabel("The silhouette coefficient values")  
 ax1.set\_ylabel("Cluster label")  
  
 ax1.axvline(x=sil\_score, color="red", linestyle='--')  
 ax1.set\_yticks([])  
 ax1.set\_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])  
  
 plt.suptitle(("Silhouette analysis for clustering on sample data with n\_clusters = %d" % n\_clusters\_),  
 fontsize=14, fontweight='bold')  
 plt.show()  
 return sil\_score  
  
def knee\_method(X):  
 nearest\_neighbors = NearestNeighbors(n\_neighbors=11)  
 neighbors = nearest\_neighbors.fit(X)  
 distances, indices = neighbors.kneighbors(X)  
 distances = np.sort(distances[:, 10], axis=0)  
 fig = plt.figure(figsize=(5, 5))  
 plt.plot(distances)  
 plt.xlabel("Points")  
 plt.ylabel("Distance")  
 plt.savefig("Distance\_curve.png", dpi=300)  
 plt.title("Distance curve")  
 plt.show()  
 i = np.arange(len(distances))  
 knee = KneeLocator(i, distances, S=1, curve='convex', direction='increasing', interp\_method='polynomial')  
 fig = plt.figure(figsize=(5, 5))  
 knee.plot\_knee()  
 plt.xlabel("Points")  
 plt.ylabel("Distance")  
 plt.show()  
 print(distances[knee.knee])  
  
# -------------------------Auto ML------------------------- #  
def selectPurpose():  
 global purpose  
 while(True): # Select Purpose  
 purpose = int(input("Select Purpose [1: Classification, 2: Clustering, 3: All] >> "))  
 if(0 < purpose and purpose < 4):  
 break  
 else:  
 print("Invalid Value!")  
 print(f'purpose: {purpose}')  
  
def selectModel():  
 scaler\_list = []  
 cf\_model\_list = []  
 cl\_model\_list = []  
  
 while(True): # Select Scalers  
 value = int(input("Select Scalers [1: Standard, 2: MinMax, 3: Robust, 4: MaxAbs, 9: All, 0: Exit] >> "))  
 if(value == 0): # exit  
 if(0 < len(scaler\_list)):  
 break  
 else:  
 print("Please choose at least one!")  
 if(value == 9): # all  
 for i in range (1, 5):  
 scaler\_list.append(i)  
 break  
 if(0 < value and value < 5):  
 scaler\_list.append(value)  
 else:  
 print("Invalid Value!")  
 print('Scaler selected!')  
  
 if(purpose == 1 or purpose == 3):  
 while(True): # Select Classification model  
 value = int(input("Select Classification model [1: Random Forest, 2: KNN, 3: logistic, 9: All, 0: Exit] >> "))  
 if(value == 0): # exit  
 if(0 < len(cf\_model\_list)):  
 break  
 else:  
 print("Please choose at least one!")  
 if(value == 9): # all  
 for i in range (1, 4):  
 cf\_model\_list.append(i)  
 break  
 if(0 < value and value < 4):  
 cf\_model\_list.append(value)  
 else:  
 print("Invalid Value!")  
 print('Classification model selected!')  
  
 if(purpose == 2 or purpose == 3):  
 while(True): # Select Clustering model  
 value = int(input("Select Clustering model [1: kmeans, 2: gmm, 3: dbscan, 4: mean-shift, 9: All, 0: Exit] >> "))  
 if(value == 0): # exit  
 if(0 < len(cl\_model\_list)):  
 break  
 else:  
 print("Please choose at least one!")  
 if(value == 9): # all  
 for i in range (1, 5):  
 cl\_model\_list.append(i)  
 break  
 if(0 < value and value < 5):  
 cl\_model\_list.append(value)  
 else:  
 print("Invalid Value!")  
 print('Clustering model selected!')  
  
 return scaler\_list, cf\_model\_list, cl\_model\_list  
  
def setCombination(scaler\_list, cf\_list = [], cl\_list = []):  
  
 # Scaler List  
 standard = StandardScaler() #1  
 minMax = MinMaxScaler() #2  
 robust = RobustScaler() #3  
 maxAbs = MaxAbsScaler() #4  
  
 scalers = {}  
 cf\_models = {}  
 cl\_models = {}  
 cf\_params = {}  
 cl\_params = {}  
  
 for i in scaler\_list:  
 if (i == 1):  
 scalers["standard scaler"] = standard  
 elif (i == 2):  
 scalers["minMax scaler"] = minMax  
 elif (i == 3):  
 scalers["robust scaler"] = robust  
 elif (i == 4):  
 scalers["maxAbs scaler"] = maxAbs  
  
 # Classification Model List  
 random\_forest = RandomForestClassifier()  
 knn = KNeighborsClassifier()  
 logistic = LogisticRegression()  
  
 for i in cf\_list:  
 if (i == 1):  
 cf\_models["random\_forest"] = random\_forest  
 cf\_params["random\_forest"] = {"n\_estimators": [3, 10, 1],  
 "max\_depth": [2, 10, 1]}  
 elif (i == 2):  
 cf\_models["KNN"] = knn  
 cf\_params["KNN"] = {"n\_neighbors": [2, 10, 1]}  
  
 elif (i == 3):  
 cf\_models["logistic"] = logistic  
 cf\_params["logistic"] = {"C": [1, 5, 1]}  
  
 # Clustering Model List  
 kmeans = KMeans() #1  
 gmm = GaussianMixture() #2  
 dbscan = DBSCAN() #3  
 meanshift = MeanShift() #4  
  
 for i in cl\_list:  
 if (i == 1):  
 cl\_models["kmeans"] = kmeans  
 cl\_params["kmeans"] = {"n\_clusters": [10, 130, 30],  
 "n\_init": [10, 30, 10],  
 "algorithm": ['auto', 'full', 'elkan']}  
 elif (i == 2):  
 cl\_models["gmm"] = gmm  
 cl\_params["gmm"] = {"n\_components": [2, 6, 1],  
 "covariance\_type": ['full', 'tied', 'diag', 'spherical'],  
 "init\_params": ['kmeans', 'random']}  
 elif (i == 3):  
 cl\_models["dbscan"] = dbscan  
 cl\_params["dbscan"] = {"eps": [0.1, 0.9, 0.1],  
 "min\_samples": [100, 1000, 100]}  
 elif (i == 4):  
 cl\_models["meanshift"] = meanshift  
 cl\_params["meanshift"] = {"bandwidth": [0.1, 0.5, 0.1],  
 "cluster\_all": [True, False]}  
   
 return scalers, cf\_models, cf\_params, cl\_models, cl\_params  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 # Read data  
 dir = os.path.dirname(os.path.realpath(\_\_file\_\_)).replace('\\', '/') + '/'  
 train = pd.read\_csv(dir+"train.csv")  
 test = pd.read\_csv(dir+"test.csv")  
  
 # Handle Missing value  
 train = findMissingValue(train)  
 test = findMissingValue(test)  
  
 # Encoding  
 train = encoding(train)  
 test = encoding(test)  
  
 # Data exploration  
 dataExploration(train)  
 dataExploration(test)  
  
 # Split Target  
 x\_train = train.drop(["satisfaction"],axis=1)  
 y\_train = train['satisfaction']  
 x\_test = test.drop(["satisfaction"],axis=1)  
 y\_test = test["satisfaction"]  
  
 # Auto ML  
 selectPurpose()  
 selected\_scaler, selected\_cf, selected\_cl = selectModel()  
 scalers, cf\_models, cf\_params, cl\_models, cl\_params = setCombination(selected\_scaler, selected\_cf, selected\_cl)  
  
 # Classification  
 if(purpose == 1 or purpose == 3):  
 result\_dict = classification(x\_train, y\_train, x\_test, y\_test, scalers, cf\_models, cf\_params)  
 print("\n-----[Best Classification Result]-----")  
 for model\_name, result\_list in result\_dict.items():  
 for result in result\_list:  
 print(result)  
 print()  
  
 # Clustering  
 if(purpose == 2 or purpose == 3):  
 ## get best combination dictionary  
 best\_result, best\_X, best\_label = clustering(train, scalers, cl\_models, cl\_params)  
 print("\n-----[Best Clustering Results]-----")  
 best\_score = 0  
 best\_combi = 0  
 for model\_name, result\_list in best\_result.items():  
 print(model\_name)  
 for result in result\_list:  
 print(result)  
 if (best\_score < result['silhouette']):  
 best\_score = result['silhouette']  
 best\_combi = result  
 print()  
  
 print("[Best Combination]")  
 print(best\_combi)  
 #display\_silhouette\_plot(best\_X, best\_label.fit\_predict(best\_X))  
 clustering\_result\_analysis(train, best\_combi)  
 print('\nDone!')