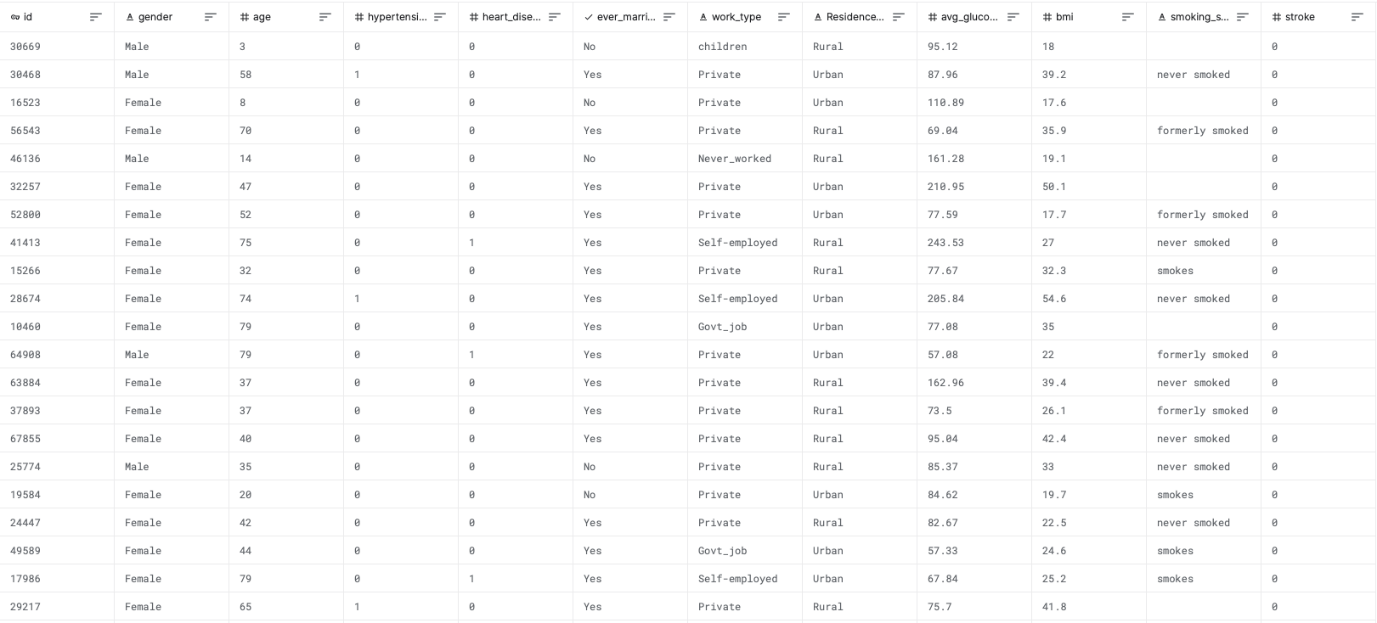
|  |
| --- |
| K-20110901-034369.pngK-20110901-034369.png  **Term project**  -Data Science- |

|  |  |  |
| --- | --- | --- |
| 안하일 |  | 201735851 |
| 이상원 |  | 201735859 |
| 오찬희 |  | 201735855 |

**Description Data Set**

Source : <https://www.kaggle.com/lirilkumaramal/heart-stroke>

Data Size : 43400 \* 12



* id: unique identifier.
* gender: "Male", "Female" or "Other".
* age: age of the patient.
* hypertension: hypertension means high blood pressure. 0 if the patient doesn't have hypertension, 1 if the patient has hypertension.
* heart\_disease: 0 if the patient doesn't have any heart diseases, 1 if the patient has a heart disease
* ever\_married: "No" or "Yes".
* work\_type: "children", "Govt\_job", "Never\_worked", "Private" or "Self-employed".
* Residence\_type: "Rural" or "Urban".
* avg\_glucose\_level: average glucose level in blood.
* bmi: body mass index, As a measure of obesity.
* smoking\_status: "formerly smoked", "never smoked", "smokes" or "Unknown"\*.
* stroke: 1 if the patient had a stroke or 0 if not.

**Object**

**Topic**

Identifying the status of disease outbreaks by indicator and calculating the probability of individual outbreaks using data

**expectation effectiveness**

Stroke is the second leading cause of death worldwide, accounting for about 11 percent of all deaths, according to the World Health Organization (WHO).

However, accurate identification requires tests such as brain MRI.

In this situation, I believe that appropriate testing needs to be provided to groups that are predicted to be at high risk by predicting the risk of development.

This dataset allows us to predict the likelihood of a patient having a stroke based on input parameters such as gender, age, various diseases and smoking conditions.

Through this machine learning, it is expected that improvements on primary prevention for stroke can be achieved by determining which groups, features, are vulnerable to stroke.

**Implementation**

**<Source Code - Visualization>**

def visualizationData(df):  
 # show missing values by heatmap  
 plt.figure(figsize=(8, 5))  
 sns.heatmap(df.isna(), cbar=False)  
 plt.xticks(rotation=45)  
 plt.yticks(ticks=[])  
 plt.subplots\_adjust(left=0.05, bottom=0.24, right=0.95, top=0.95)  
 plt.show()  
  
 # show number of values in each column  
 for feature in ["gender", "hypertension", "heart\_disease", "ever\_married", "work\_type", "Residence\_type",  
 "smoking\_status"]:  
 sns.countplot(x=df[feature], hue=df["stroke"])  
 plt.xlabel(feature)  
 plt.title(feature)  
 plt.show()  
  
 for feature in ["age", "avg\_glucose\_level", "bmi"]:  
 sns.countplot(x=df[feature], hue=df["stroke"])  
 plt.xlabel(feature)  
 plt.title(feature)  
 plt.show()  
  
 # show boxplot for columns with continuous values rather than specific values.  
 for feature in ["avg\_glucose\_level", "bmi"]:  
 sns.boxplot(x=feature, data=df)  
 plt.show()

Visualize the number of patients with or without stroke by column and get help in the preprocessing process

def visualizationCorrelation(df):  
 # compute the corr matrix  
 corr = df.corr()  
  
 # generate a mask for the upper triangle  
 mask = np.triu(np.ones\_like(corr, dtype=bool))  
  
 # set up the matplotlib figure  
 f, ax = plt.subplots(figsize=(8, 6))  
  
 # generate a custom diverging colormap  
 cmap = sns.diverging\_palette(230, 20, as\_cmap=True)  
  
 # draw the heatpmap with the mask and correct aspect ratio  
 sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0, square=True, linewidths=.5, cbar\_kws={'shrink': .5})  
 plt.subplots\_adjust(left=0, bottom=0.24, right=1, top=1)  
 plt.show()

Visualizes a correlation plot for each column in the dataset.

def visualizationAccuracy(result):  
 # bar graph the accuracy for each model.  
 sns.barplot(x='Accuracy', y='Model', data=result, color='b')  
 for i, v in enumerate(result['Accuracy']):  
 plt.text(v + 3, i + .25, str(v), color='black', fontweight='bold')  
 plt.show()  
  
 sns.barplot(x='K-Fold Mean Accuracy', y='Model', data=result, color='b')  
 for i, v in enumerate(result['K-Fold Mean Accuracy']):  
 plt.text(v + 3, i + .25, str(v), color='black', fontweight='bold')  
 plt.show()  
  
 sns.barplot(x='Precision', y='Model', data=result, color='b')  
 for i, v in enumerate(result['Precision']):  
 plt.text(v + 3, i + .25, str(v), color='black', fontweight='bold')  
 plt.show()  
  
 sns.barplot(x='Recall', y='Model', data=result, color='b')  
 for i, v in enumerate(result['Recall']):  
 plt.text(v + 3, i + .25, str(v), color='black', fontweight='bold')  
 plt.show()  
  
 sns.barplot(x='F1 Score', y='Model', data=result, color='b')  
 for i, v in enumerate(result['Score']):  
 plt.text(v + 3, i + .25, str(v), color='black', fontweight='bold')  
 plt.show()

Visualizes the results as a graph.

def visualizationROC(false\_positive\_rate, true\_positive\_rate):  
 # ROC Curve  
 roc\_auc = auc(false\_positive\_rate, true\_positive\_rate)  
  
 sns.set\_theme(style='white')  
 plt.figure(figsize=(8, 8))  
 plt.plot(false\_positive\_rate, true\_positive\_rate, color='#b01717', label='AUC = %0.3f' % roc\_auc)  
 plt.legend(loc='lower right')  
 plt.plot([0, 1], [0, 1], linestyle='--', color='#174ab0')  
 plt.axis('tight')  
 plt.ylabel('True Positive Rate')  
 plt.xlabel('False Positive Rate')  
 plt.show()

Visualizes the results as a ROC\_AUC graph.

**<Source Code - Preprocessing>**

def preprocessing(data, encoder, scaler):  
 data.drop('id', axis=1, inplace=True) # 필요없는 id값 드랍  
 data['age'] = data['age'].apply(lambda x: round(x)) # 나이 반올림  
 data['bmi'] = data['bmi'].apply(lambda bmi\_value: bmi\_value if 12 < bmi\_value < 45 else np.nan) # bmi 아웃라이어 처리  
 data['gender'] = data['gender'].apply(  
 lambda gender: gender if gender == 'Female' or gender == 'Male' else np.nan) # other 값 처리  
 data.dropna(axis=0, inplace=True) # 결측값  
 data.reset\_index(drop=True, inplace=True)  
  
 # categorical data convert to numeric  
 if (encoder == LabelEncoder()):  
 data['gender'] = encoder.fit\_transform(data['gender'])  
 data['work\_type'] = encoder.fit\_transform(data['work\_type'])  
 data['Residence\_type'] = encoder.fit\_transform(data['Residence\_type'])  
 data['smoking\_status'] = encoder.fit\_transform(data['smoking\_status'])  
 data['ever\_married'] = encoder.fit\_transform(data['ever\_married'])  
 else:  
 data = pd.get\_dummies(data, columns=['gender', 'work\_type', 'Residence\_type', 'smoking\_status', 'ever\_married'])  
 print(data)  
 # OverSampling to balance the Data  
 target = data["stroke"]  
 feat = data.drop('stroke', axis=1)  
  
 feat, target = SMOTE(random\_state=2, sampling\_strategy=0.2).fit\_resample(feat, target)  
  
 target.drop(feat[feat.duplicated(keep="first") == True].index, inplace=True)  
 feat = feat.drop\_duplicates(keep="first")  
  
 feat['bmi'] = feat['bmi'].apply(lambda bmi\_value: round(bmi\_value, 1)) # bmi 아웃라이어 처리  
 feat['avg\_glucose\_level'] = feat['avg\_glucose\_level'].apply(  
 lambda avg\_glucose\_level: round(avg\_glucose\_level, 2)) # bmi 아웃라이어 처리  
  
 X\_train, X\_test, y\_train, y\_test = train\_test\_split(feat, target, test\_size=0.2, random\_state=0, stratify=target)  
  
 X\_train = scaler.fit\_transform(X\_train)  
 X\_test = scaler.fit\_transform(X\_test)  
  
 return X\_train, y\_train, X\_test, y\_test

During the preprocessing phase

1. Delete “id” column independent of data prediction

2. Delete the missing values

3. Find and delete BMI outliers

4. Round up “age” column with the decimal point

6. Encoding categorical data by encoding received parameters(LabelEncoder, OneHotEncoder)

7. Oversampling using SMOTE method

8. Delete duplicate data

9. Split into train data, test data

10. Scaling data received with parameter(Standard, MinMax, MaxAbs, Robust)

**<Source Code – Encoding&Scaling Parameters>**

encoder\_scaler = [  
 ('Onehot',StandardScaler()),  
 ('Onehot',MinMaxScaler()),  
 ('Onehot',MaxAbsScaler()),  
 ('Onehot',RobustScaler()),  
 ('Label',StandardScaler()),  
 ('Label',MinMaxScaler()),  
 ('Label',MaxAbsScaler()),  
 ('Label',RobustScaler())  
]

8 data analyses are carried out by combining 2 encoding methods and 4 scaling methods.

**<Source Code – Parameters>**

# Models that have undergone hyperparameter tuning to be used in grid search.  
g\_models = [  
 (XGBClassifier(use\_label\_encoder=False, eval\_metric='merror'),  
 [{'learning\_rate': [0.1, 0.5, 1], 'n\_estimators': [10, 100, 1000],  
 'booster': ['gbtree', 'gblinear']}]),  
 (KNeighborsClassifier(), [{'n\_neighbors': [2, 3, 4], 'leaf\_size': [15, 30],  
 'weights': ['uniform', 'distance'], 'algorithm': ['ball\_tree', 'kd\_tree', 'brute']}]),  
 (DecisionTreeClassifier(), [{'criterion': ['gini', 'entropy'], 'splitter': ['best', 'random'],  
 'max\_depth': [None, 2, 3], 'max\_features': [None, 'sqrt', 'log2']}]),  
 (RandomForestClassifier(),  
 [{'n\_estimators': [10, 100, 1000], 'criterion': ['gini', 'entropy'], 'max\_depth': [None, 2, 3],  
 'max\_features': [None, 'sqrt', 'log2']}]),  
 (ExtraTreesClassifier(),  
 [{'n\_estimators': [10, 100, 1000], 'criterion': ['gini', 'entropy'], 'max\_depth': [None, 2, 3],  
 'max\_features': [None, 'sqrt', 'log2']}]),  
 (AdaBoostClassifier(), [{'algorithm': ['SAMME', 'SAMME.R'],  
 'learning\_rate': [0.1, 0.5, 1], 'n\_estimators': [10, 50, 100]}]),  
 (GradientBoostingClassifier(),  
 [{'n\_estimators': [10, 100, 1000], 'criterion': ['friedman\_mse', 'mse'], 'learning\_rate': [0.1, 0.5, 1],  
 'loss': ['deviance', 'exponential']}])]

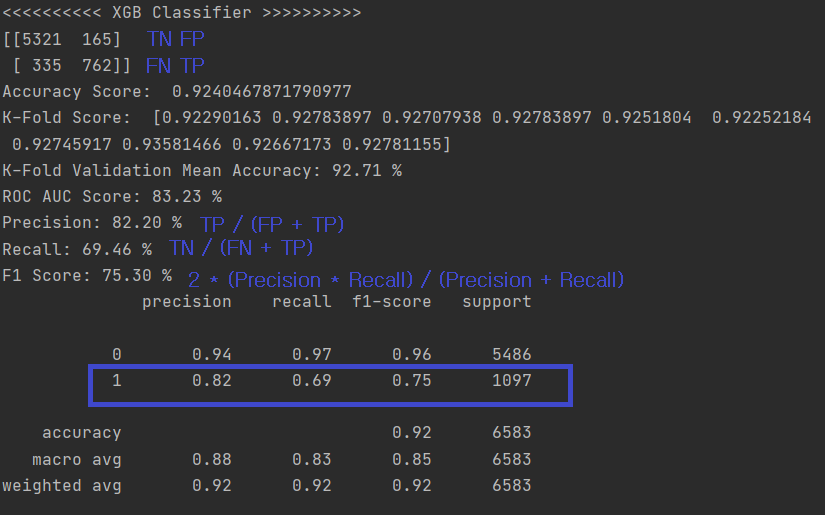
Parameters for hyperparameter tuning

**<Source Code – Analysis & GridSearch>**

def gridsearch(enc\_scal, g\_models):  
 result\_list = []  
 m = 0  
 # Proceed with hyperparameter tuning through grid search.  
 for encoder, scaler in enc\_scal:  
 X\_train\_res, y\_train\_res, X\_test, y\_test = preprocessing(df.copy(), encoder, scaler)  
 for model, param in g\_models:  
 result = []  
 grid = GridSearchCV(estimator=model, param\_grid=param, scoring='accuracy', cv=5, n\_jobs=-1)  
 grid.fit(X\_train\_res, y\_train\_res)  
 print(' {}: \n Best Accuracy: {:.2f} %'.format(model, grid.best\_score\_ \* 100))  
 print('\n Best Parameter : {}', grid.best\_params\_)  
  
 # predict with best model and calculate MSE  
 best\_model = grid.best\_estimator\_  
 y\_pred = best\_model.predict(X\_test)  
  
 print(confusion\_matrix(y\_test, y\_pred))  
 print(classification\_report(y\_test, y\_pred))   
  
 result.append(models[m][0])  
 result.append(accuracy\_score(y\_test, y\_pred) \* 100)  
 result.append(roc\_auc\_score(y\_test, y\_pred) \* 100)  
 result.append(precision\_score(y\_test, y\_pred) \* 100)  
 result.append(recall\_score(y\_test, y\_pred) \* 100)  
 result.append(f1\_score(y\_test, y\_pred) \* 100)  
 result\_list.append(result)  
 m = m + 1  
 result\_df = pd.DataFrame(result\_list,  
 columns=['Model', 'Accuracy', 'ROC\_AUC', 'Precision', 'Recall', 'F1 Score'])  
 print(encoder, scaler)  
 print(result\_df)

We use the XGBoost, KNN, Decision Tree, RandomForest, ExtraTrees, AdaBoost, GradientBoost models to output optimal parameters and their accuracy through hyperparameter tuning in each model.

**<Overview of Results>**

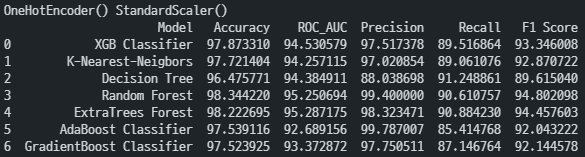


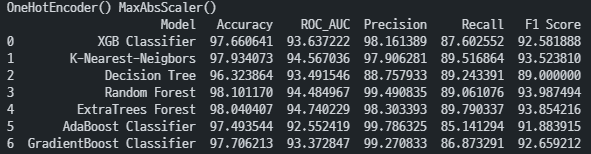
In general, accuracy is an evaluation indicator that can most intuitively represent the performance of a model. However, there is something that needs to be considered here.

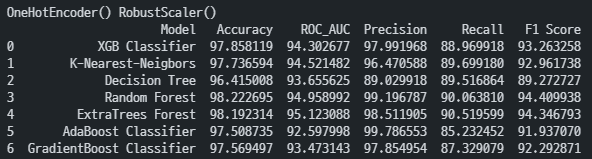
Suppose that rainy days are uncommon when we predict the weather for a month. In this case, the domain of the data is disproportionate, so the performance of predicting the clear is high, but the performance of predicting the rain is very low. Our dataset is also the same as this because the proportion of stroke patients is significantly lower than that of those who do not.

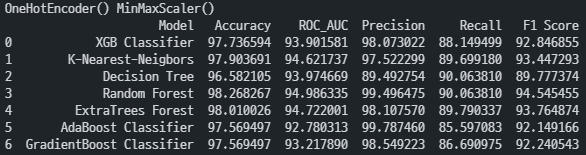
So, we focused not only on Accuracy but also on F1 scores, especially when Stroke is 1.

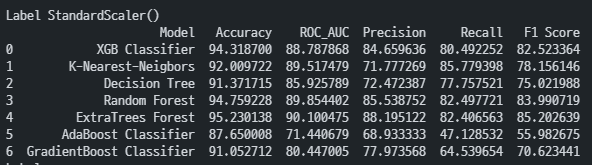
**<Results for Each Case>**

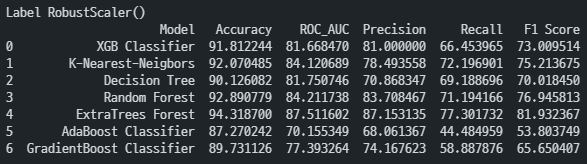
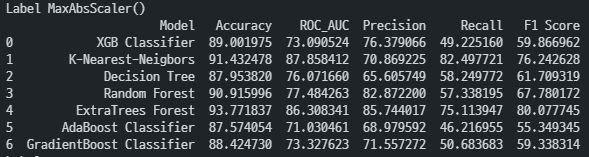
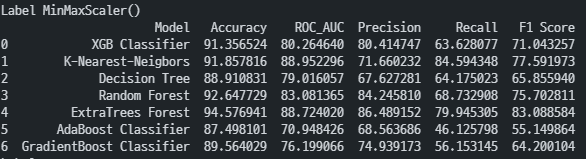












**<**

**<Appendix : Team 7 Dataset Result>**

