<Data Science>

Team 10

202035510 김유현

202035537 임상택

202135764 류건우

202235007 곽혜린

<Contents>

1. Big Data End-to-End Process
2. Objective Setting
3. Data Curation
4. Data Inspection

* Determine tendency, area of data
* Distribution of Data (Normal Distribution)
* Outlier check
* Correlation between variables

1. Data Preprocessing

* Data Restructuring
* Data Value Changes
* Feature Engineering
* Data Reduction

1. Data Analysis (Modeling) & Model Evaluation
2. Open Source SW
3. Function Definition
4. GitHub URL
5. End-to-End Process
6. Objective Setting

: The goal of this project is to predict suicide rates based on various characteristics such as country, year, gender, and age, and to identify key factors influencing suicides. Specifically, we aim to achieve the following objectives.

1. Develop predictive models

: Create robust predictive models by analyzing suicide rate patterns using diverse features.

* Approach
* Data Collection: Gather comprehensive datasets from reliable sources, including global and regional databases on suicide rates and demographic information.
* Feature Engineering: Generate and select key features that may impact suicide rates, including economic indicators, social factors, and health-related metrics
* Model Development: Employ various machine learning algorithms to build and validate predictive models.
* Performance Evaluation: Use cross-validation techniques and performance metrics to assess model efficacy and ensure generalizability.

1. Identify Influential Factors and Propose Interventions

: Determine the key factors influencing suicide rates and leverage these insights to propose targeted social and economic interventions and prevention policies.

* Approach
* Factor Analysis: Conduct statistical analyses (e.g., correlation analysis, principal component analysis) to identify and quantify the impact of different variables on suicide rates.
* Intervention Strategies: Develop evidence-based recommendations for policy makers and social organizations to address high-risk factors. This may include improving access to mental health services, enhancing economic support programs, and fostering community engagement initiatives.
* Policy Proposals: Draft and advocate for comprehensive prevention policies aimed at reducing suicide rates, with a focus on at-risk populations identified through the analysis.

1. Raise Awareness and Contribute to Social improvements

: Utilize data-driven insights to raise public awareness about the issue of suicides and contribute to societal responses and improvements.

* Approach
* Awareness Campaigns: Collaborate with public health organizations, NGOs, and media outlets to disseminate findings and promote awareness campaigns focused on mental health and suicide prevention.
* Educational Programs: Develop and implement educational programs and materials to inform communities about the risk factors and prevention strategies related to suicides.
* Community Engagement: Foster partnerships with local communities to create support networks and intervention programs tailored to their specific needs and challenges.
* Impact Assessment: Continuously monitor and evaluate the impact of awareness and intervention efforts to ensure they are effective and make necessary adjustments based on feedback and new data insights.

: By setting these detailed objectives, we aim to create a comprehensive approach to predicting, understanding, and addressing suicide rates, ultimately contributing to a reduction in suicides and an improvement in overall societal well-being.

2) Data Curation

: The project involves collecting the necessary data to achieve its goals. This process includes gathering, storing, and transforming the data into a format that is ready for analysis.

* Data to Analyze: Suicide Rates Overview 1985 to 2016 from Kaggle.

<https://www.kaggle.com/russellyates88/suicide-rates-overview-1985-to-2016>

* Original Data
* Columns: 12
* Country, year, sex, age, suicides\_no, population, suicides/100k pop, country-year, HDI for year, gdp\_for\_year($), gdp\_per\_capita($), generation
* 텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

  자동 생성된 설명
* Divided Data:
* master\_train.csv
* master\_test.csv
* Import & Read Data
* Code

import pandas as pd

import numpy as np

import random as rnd

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.impute import SimpleImputer

from sklearn.linear\_model import LogisticRegression

from sklearn.svm import SVC, LinearSVC

from sklearn.ensemble import RandomForestClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.naive\_bayes import GaussianNB

from sklearn.linear\_model import Perceptron

from sklearn.linear\_model import SGDClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.neural\_network import MLPClassifier

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

from sklearn.metrics import accuracy\_score

from sklearn.model\_selection import KFold

data = pd.read\_csv('master.csv')

data\_train = pd.read\_csv('master\_train.csv')

data\_test = pd.read\_csv('master\_test.csv')

1. Data Inspection

: This stage involves evaluating the quality and completeness of the collected data. During this process, we check for missing values, outliers, and duplicates, and make necessary corrections if needed.

1. Determine tendency, area of data
2. Print the format and details of the dataset

* Code
* print("Data\_train shape", data\_train.shape, "\n")
* print("Data\_test shape", data\_test.shape, "\n")
* print(data\_train.info())
* print('\_'\*40)
* print(data\_test.info())
* Output

텍스트, 메뉴, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

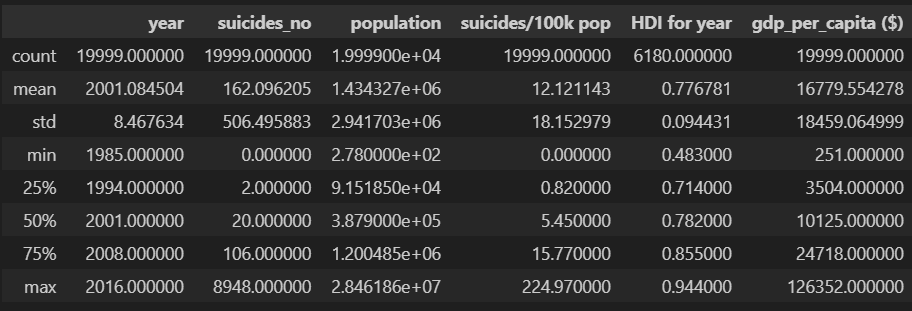
② Numerical based statistical information in the dataset

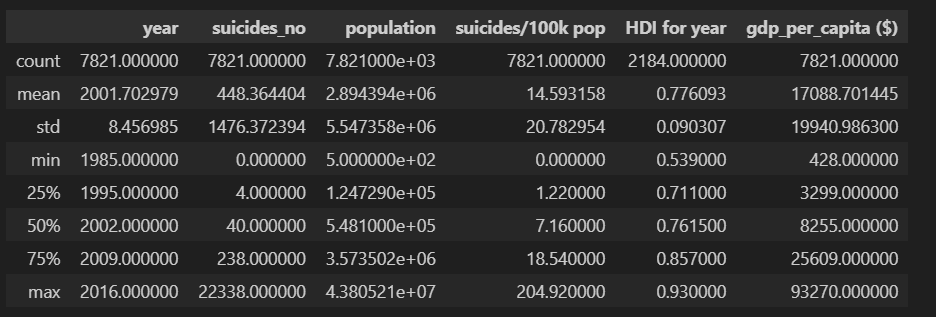
* Code

data\_train.describe()

data\_test.describe()

* Output





③ Analysis of suicide rates and data distribution by country

* Code
* #3) 국가별 자살률 확인 및 시각화
* fit = plt.figure(figsize=(30, 10))
* plt.title('Suicide rates by gdp\_per\_capita ($)')
* sns.barplot(x='country', y='suicides/100k pop', data=data[['country', 'suicides/100k pop']].groupby('country', as\_index=False).mean().sort\_values(by='suicides/100k pop', ascending=False));
* plt.xticks(rotation=90);
* data[['country', 'suicides/100k pop']].groupby('country', as\_index=False).mean().sort\_values(by='suicides/100k pop', ascending=False).head()
* Output

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

라인, 스크린샷, 그래프, 경사이(가) 표시된 사진

자동 생성된 설명

* Analysis
* Number of data points by Country: Mauritius, Austria, the Netherlands, and Iceland have the highest number of data points.
* Suicide Rates by Country: The suicide rates per 100,00 people are highest in countries such as Lithuania, Srilanka, and the Russian Federation.

1. Visualize changes in suicide rates by year

* Code
* #4) 연도별 자살률 시각화 및 상위 5개 년도 출력
* fig, ax = plt.subplots(figsize=(20, 5))
* plt.title('Suicide rates by year')
* sns.lineplot(x='year', y='suicides/100k pop', data=data[['year', 'suicides/100k pop']].groupby('year', as\_index=False).mean().sort\_values(by='year', ascending=True), marker='o', color='RED');
* ax.set(xticks = data['year'].unique());
* data[['year', 'suicides/100k pop']].groupby('year', as\_index=False).mean().sort\_values(by='suicides/100k pop', ascending=False).head()
* Output

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

텍스트, 도표, 라인, 그래프이(가) 표시된 사진

자동 생성된 설명

* Analysis
* Graph Analysis: Suicide rates showed an upward trend from 1985 to 1995, peaking in 1995, and then gradually declining. This decline continued through the early 2000s, but a sharp increase occurred after 2015, with a significant rise in 2016.
* Table Analysis: The years from 1995 to 1999 are recorded as the years with the highest suicide rates.

⑤ Visualize changes in suicide rates by sex

* Code
* #5) 성별에 따른 자살률 확인 및 순위 출력
* fig = plt.figure(figsize=(2,3))
* plt.title('Suicide rates by sex')
* sns.barplot(x='sex', y='suicides/100k pop', data=data[['sex', 'suicides/100k pop']].groupby('sex', as\_index=False).mean().sort\_values(by='suicides/100k pop', ascending=False));
* data[['sex', 'suicides/100k pop']].groupby('sex', as\_index=False).mean().sort\_values(by='suicides/100k pop', ascending=False)
* Output

텍스트, 폰트, 스크린샷, 번호이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 도표, 폰트이(가) 표시된 사진

자동 생성된 설명

* Analysis
* Graph Analysis: The suicide rate for men is about 20 per 100,000, while for women it is about 5 per 100,000, indicating that the male suicide rate is approximately four times higher than that of females.

⑥ Visualize changes in suicide rates by age

* Code
* #6) 연령대에 따른 자살률 확인 및 순위출력
* fig = plt.figure(figsize=(5,4));
* plt.title('Suicide rates by Age')
* sns.barplot(x='age', y='suicides/100k pop', hue='sex', data=data[['age', 'suicides/100k pop', 'sex']].groupby(['age', 'sex'], as\_index=False).mean().sort\_values(by='suicides/100k pop', ascending=False));
* data[['age', 'suicides/100k pop']].groupby('age', as\_index=False).mean().sort\_values(by='suicides/100k pop', ascending=False)
* Output

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트, 도표이(가) 표시된 사진

자동 생성된 설명

* Analysis
* Graph Analysis: Suicide rates for males are higher than those for females across all age groups, with the highest rates observed in men aged 75 and older. While women show an increasing trend in suicide rates with age, their rates remain significantly lower compared to men.

⑦ Visualize changes in suicide rates by GDP per capita

* Code
* #7) 1인당 GDP별 자살률 확인 및 시각화
* fig = plt.figure(figsize=(30, 10))
* plt.title('Suicide rates by gdp\_per\_capita ($)')
* sns.barplot(x='gdp\_per\_capita ($)', y='suicides/100k pop', data=data[['gdp\_per\_capita ($)', 'suicides/100k pop']].groupby('gdp\_per\_capita ($)', as\_index=False).mean().sort\_values(by='suicides/100k pop', ascending=False));
* data['gdp\_per\_capita ($)'].describe()
* Output

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

* Analysis
* Graph Analysis: There is a slight tendency for lower GDP per capita to be associated with higher suicide rates, but the distribution of suicide rates is similar regardless of GDP. Therefore, GDP per capita appears to have little to no correlation with suicide rates.

⑧ Visualize changes in Suicide Rates by Generation

* Code
* #8) Generation별 자살 데이터 분석
* f,ax = plt.subplots(1, 2, figsize=(15, 5))
* data['generation'].value\_counts().plot.pie(explode=[0.1, 0.1, 0.1, 0.1, 0.1, 0.1], autopct='%1.1f%%', ax=ax[0], shadow=True)
* ax[0].set\_title('Suicide Rates by Generation')
* ax[0].set\_ylabel('suicides/100k pop')
* sns.countplot(x='generation', data=data, ax=ax[1])
* ax[1].set\_title('Suicide Rates by Generation')
* plt.show()
* data[['generation', 'suicides/100k pop']].groupby('generation', as\_index=False).mean().sort\_values(by='suicides/100k pop', ascending=False)
* Output

텍스트, 도표, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

* Analysis
* Graph Analysis: The silent Generation, Generation X, and Millennials show relatively high suicide rates, suggesting that these generations may have a higher risk of suicide.
* G.I Generation – Birth years: 1901 ~ 1927
* Silent Generation – Birth years: 1928 ~ 1945
* Baby Boomers – Birth years: 1946 ~ 1964
* Generation X – Birth years: 1965 ~ 1980
* Millennials – Birth years: 1981 ~ 1996
* Generation Z – Birth years: 1997 ~ 2012

⑨ Visualize changes in suicide rates by Country and Generation

* Code
* #9)국가와 세대에 따른 자살률 변화 및 시각화
* # Assuming data has been loaded and column names verified
* generation\_order = ['G.I. Generation', 'Silent', 'Boomers', 'Generation X', 'Millenials', 'Generation Z']
* pastel\_colors = sns.color\_palette('pastel', len(generation\_order))
* sns.set(style="whitegrid")
* g = sns.FacetGrid(data, col="country", col\_wrap=4, height=4, aspect=1.5, hue="generation", palette=pastel\_colors, hue\_order=generation\_order)
* g.map(sns.barplot, "generation", "suicides/100k pop")
* g.add\_legend(title="Generation")
* g.set\_xticklabels(rotation=90)
* g.set\_axis\_labels("Generation", "Suicides/100k pop")
* g.set\_titles("{col\_name}")
* g.fig.suptitle('Suicide Rates by Country and Generation', fontsize=16)
* g.fig.subplots\_adjust(top=0.95)
* plt.show()
* Output

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

* Analysis
* Graph Analysis
* South Korea: Generation X shows a comparatively high suicide rate, with a generally similar distribution of suicide rates across different generations.
* Japan: Like South Korea, Generation X shows a comparatively high suicide rate, with a generally similar distribution of suicide rates across different generations.
* Italy: The G.I Generations shows the highest suicide rates, likely linked to issues faced by this generation.
* Bahamas: As a Caribbean Island nation heavily reliant on tourism, the Bahamas shows higher suicide rates in the G.I. Generation compared to other generations. It may be due to economic instability or the impacts of global economic crisis.

⑩ Visualize changes in suicide rates by Year and Generation

* Code
* # 10) 연도와 연령에 따른 자살률 확인 및 차트 생성
* plt.figure(figsize=(14, 8))
* pivot\_table = data.pivot\_table(values='suicides/100k pop', index='year', columns='age', aggfunc='mean')
* sns.lineplot(data=pivot\_table, dashes=False)
* plt.title('Suicide Rates by year and age group')
* plt.xlabel('year')
* plt.ylabel('suicides per 100k population')
* plt.legend(title='Age Group')
* plt.grid(True)
* plt.show()
* Output

텍스트, 그래프, 도표, 라인이(가) 표시된 사진

자동 생성된 설명

1. Distribution of Data (Normal Distribution)

* Code
* # Filter only numeric data (int and float types)
* numeric\_data = data.select\_dtypes(include=['int64', 'float64'])
* # Define pastel color array
* colors = ['powderblue', 'thistle', 'lightcoral', 'palegreen', 'lightsalmon', 'gold']
* # Set the size of the entire plot
* plt.figure(figsize=(10, 8 \* len(numeric\_data.columns)))  # Multiply the height of each plot by the number of columns to determine the total height
* # Create and position a subplot for each numeric column
* for i, column in enumerate(numeric\_data.columns):
* plt.subplot(len(numeric\_data.columns), 1, i + 1)  # Position the subplot (n rows, 1 column at position i+1)
* numeric\_data[column].hist(color=colors[i % len(colors)], bins=30)  # Draw histogram
* plt.title(column)  # Set the title for each subplot
* plt.tight\_layout()  # Automatically adjust the spacing between subplots
* plt.show()  # Display the graph
* # Normality test for a year column
* stat, p = shapiro(data['year'])
* print('year: Statistics=%.3f, p=%.3f' % (stat, p))
* # Normality test for a population column
* stat, p = shapiro(data['population'])
* print('population: Statistics=%.3f, p=%.3f' % (stat, p))
* # Normality test for a HDI for year column
* stat, p = shapiro(data['HDI for year'])
* print('HDI for year: Statistics=%.3f, p=%.3f' % (stat, p))
* # Normality test for a suicides\_no column
* stat, p = shapiro(data['suicides\_no'])
* print(
* # Normality test for a suicides/100k pop column
* stat, p = shapiro(data['suicides/100k pop'])
* print('suicides/100k pop: Statistics=%.3f, p=%.3f' % (stat, p))
* # Normality test for a gdp\_per\_capita ($) column
* stat, p = shapiro(data['gdp\_per\_capita ($)'])
* print('gdp\_per\_capita ($): Statistics=%.3f, p=%.3f' % (stat, p))
* Output

스크린샷, 도표, 텍스트, 라인이(가) 표시된 사진

자동 생성된 설명

* Analysis
* A statistic close to 1 indicates that the data is close to a normal distribution.
* A p-value of 0.005 or 0.01 or lower indicates that the data does not follow a normal distribution.

1. Outlier check

* Code
* # Subplot Settings
* fig, axes = plt.subplots(nrows=1, ncols=5, figsize=(20, 5))
* # Create a box plot for each variable
* columns = ['year', 'suicides\_no', 'population', 'suicides/100k pop', 'gdp\_per\_capita ($)']
* for i, col in enumerate(columns):
* axes[i].boxplot(data[col].dropna())  # Remove NA value
* axes[i].set\_title(col)
* axes[i].set\_ylabel('Values')
* # Adjust the full layout
* plt.tight\_layout()
* plt.show()
* Output

텍스트, 라인, 번호, 평행이(가) 표시된 사진

자동 생성된 설명

* Analysis
* Year: The data is evenly distributed over time, with no outliers, spanning from the mid-1980s to 2015.
* Suicides\_no: Most data points are concentrated in the lower range, but there are several outliers with very high values, indicating exceptionally high suicide cases in certain countries or years.
* Population: Most values are concentrated in the lower range, with some outliers showing high population figures, indicating the inclusion of data from countries with large populations.
* Suicides/100k pop: Most data points are between 0 and 50, with some outliers showing higher suicide rates, indicating relatively high suicide rates in specific regions or groups.
* GDP per capita ($): Most values are relatively low, but there are many outliers with very high GDP, indicating the inclusion of data from economically advanced countries.

1. Correlation between variables

* Code
* # Apply log transformation to reduce the impact of outliers
* data['log\_population'] = np.log1p(data['population'])
* data['log\_suicides\_no'] = np.log1p(data['suicides\_no'])
* data['log\_gdp\_per\_capita'] = np.log1p(data['gdp\_per\_capita ($)'])
* # Correlation matrix
* # Select numeric columns including transformed ones
* numeric\_cols = ['year', 'log\_population', 'log\_suicides\_no', 'suicides/100k pop', 'log\_gdp\_per\_capita']
* correlation\_matrix = data[numeric\_cols].corr()
* # Visualize the correlation matrix using a heatmap
* plt.figure(figsize=(10, 8))
* sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm', fmt=".2f")
* plt.title('Correlation Matrix of Variables')
* plt.show()
* # Plotting scatter plots
* plt.figure(figsize=(6, 15))
* plt.subplot(3, 1, 1)
* sns.scatterplot(x='log\_suicides\_no', y='log\_population', data=data)
* plt.title('Log suicides No vs. Log Population')
* plt.subplot(3, 1, 2)
* sns.scatterplot(x='log\_suicides\_no', y='suicides/100k pop', data=data)
* plt.title('Log Suicides No vs. Suicides/100k pop')
* plt.subplot(3, 1, 3)
* sns.scatterplot(x='year', y='log\_gdp\_per\_capita', data=data)
* plt.title('Year vs. Log Gdp Per Capita')
* plt.tight\_layout()
* plt.show()
* Output

텍스트, 스크린샷, 직사각형, 도표이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 다채로움, 그래프이(가) 표시된 사진

자동 생성된 설명

* Analysis
* Correlation Matrix of Variables
* Log population and log ‘suicides\_no’ show a correlation coefficient of 0.78, indicating a strong positive correlation between the two variables -> higher populations tend to have higher suicide numbers.
* Suicide rate (suicides/100k pop) and log ‘suicides\_no’ show a correlation coefficient of 0.48 -> higher suicide numbers are associated with higher suicide rates.
* GDP per capita (‘log\_gdp\_per\_capita’) and year have a correlation coefficient of 0.35, indicating a trend of increasing GDP per capita over time.
* Scatter Plot Analysis
* Log suicides No vs. Log Population: There is a general trend that as the number of suicides increases, so does the population -> areas with higher suicide numbers tend to have larger populations.
* Log Suicides No vs. Suicides/100k pop: At lower suicide numbers, the suicide rate is relatively low and distributed with little variation. As the number of suicides increases, the suicide rate also tends to increase, with a noticeable increase in the rate at certain points -> specific regions or groups with higher suicide numbers can see a significant increase in the suicide rate.
* Log GDP Per Capita vs. Year: Data points show a gradual increase in GDP per capita over time -> overall economic growth is evident. However, certain years show sharp increases or decreases in GDP, likely reflecting the impact of specific economic events.

1. Data Preprocessing
2. Data Restructuring

① Table Decomposition

* Code
* # Delete duplicate attribute: "suicides\_no", "population"
* # "suicides/100k pop" \* 10^5 = "suicides\_no" / "population"
* data.drop("suicides\_no", axis = 1, inplace = True)
* data.drop("population", axis = 1, inplace = True)
* print(data.info())
* # Delete duplicate attribute: "country-year"
* # "country" and "year" attributes also exist separately
* data.drop("country-year", axis = 1, inplace = True)
* print(data.info())
* # Delete duplicate attribute: "gdp\_for\_year ($)"
* # "gdp\_for\_year ($)" is proportional to "gdp\_per\_capita ($)"
* data.drop("gdp\_for\_year ($)", axis = 1, inplace = True)
* print(data.info())
* Output

텍스트, 스크린샷, 메뉴, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 메뉴, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

* Analysis
* Removal of “suicides\_no” and “population”: The suicides/100k pop is derived by dividing “suicides\_no” by the population and then multiplying by 100,000. Therefore, “suicides\_no” and population are redundant, as they are already represented through suicides/100k pop. To reduce the dimensionality of the dataset, these two columns will be removed.
* Removal of “country-year”: The country-year column provides combined information from country and year. Using these columns separately offers more flexibility in data processing, and the same information can be recreated by combining these two attributes when needed. Hence, the country-year column will be removed.
* Removal of “gdp\_for\_year ($)”: The “gdp\_for\_year ($)” represents the total annual GDP of a country, while “gdp\_per\_capita ($)” represents the GDP per capita. These two values are proportionally related, as the total GDP of a country can be calculated by multiplying the population with the GDP per capita. Therefore, “gdp\_for\_year ($)” provides redundant information and unnecessarily increases the complexity of the dataset. It will be removed.

1. Data Value Changes

① Cleaning dirty data

* Code
* data\_train.drop('HDI for year', axis=1, inplace=True)
* data\_test.drop('HDI for year', axis=1, inplace=True)
* #suicides\_no와 population을 나눈 값이 'suicides/100k pop' \* 10^-5 이므로 중복 속성 제거.
* data\_train.drop('suicides\_no', axis=1, inplace=True)
* data\_train.drop('population', axis=1, inplace=True)
* data\_train.drop('country-year', axis=1, inplace=True)
* # country 및 year 속성 별도 존재하여 삭제
* data\_train.drop(' gdp\_for\_year ($) ', axis=1, inplace=True)
* #gdp\_per\_capita($)와 비례하므로 중복 속성. 삭제
* data\_train['age']=data['age'].str.strip(" years") # years 텍스트 제거
* data\_test.drop('suicides\_no', axis=1, inplace=True)
* data\_test.drop('population', axis=1, inplace=True)
* data\_test.drop('country-year', axis=1, inplace=True)
* # country 및 year 속성 별도 존재하여 삭제
* data\_test.drop(' gdp\_for\_year ($) ', axis=1, inplace=True)
* #gdp\_per\_capita($)와 비례하므로 중복 속성. 삭제
* data\_test['age']=data['age'].str.strip(" years")
* data\_train.head()
* data\_test.head()
* Output

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

* Analysis
* Checking for missing values in the original dataset: The number of missing values in each column of the dataset was calculated, revealing that the number of missing values in the ‘HDI for year’ column amounted to 70% of the total data in that column. As a result, the ‘HDI for year’ column will be deleted.

② Text preprocessing

* Code
* # Remove 'years' text
* data["age"] = data["age"].str.strip(" years")
* print(data['age'])
* Output

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

* Analysis

: The text "years" will be removed from the values in the age column.

③ Data discretization

* Code
* #1) age 의 string을 정수로 바꾸기
* def func(dataset):
* if dataset['age'] == '75+': # 75+ ==> 80 치환
* return '80'
* elif dataset['age'] =='55-74': # 55-74 ==> 65 치환
* return '65'
* elif dataset['age'] =='35-54': # 35-54 ==> 45 치환
* return '45'
* elif dataset['age'] =='25-34': # 25-34 ==> 30 치환
* return '30'
* elif dataset['age'] =='15-24': # 15-24 ==> 20 치환
* return '20'
* else: return '10' # 5-14 = 10 치환
* #TrainData 적용
* data\_train['age'] = data\_train.apply(func, axis=1)
* data\_train[['age','suicides/100k pop']].groupby('age', as\_index=False).mean().sort\_values(by='suicides/100k pop', ascending=False)
* #TestData도 같이 적용
* data\_test['age'] = data\_test.apply(func, axis=1)
* data\_test[['age','suicides/100k pop']].groupby('age', as\_index=False).mean().sort\_values(by='suicides/100k pop', ascending=False)
* Output

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

* Analysis

: The data type will be changed from object to int, and each age group will be encoded with its median value.

④ Encoding for data mining algorithms

* Code
* # Pre-processing sex property using label encoding
* # Define the mapping function using map
* sex\_mapping = {'female': 0, 'male': 1}
* data['sex'] = data['sex'].map(sex\_mapping)
* # Verify the changes by grouping by sex and calculating the mean suicide rate
* sex\_grouped = data[['sex', 'suicides/100k pop']].groupby('sex', as\_index=False).mean().sort\_values(by='suicides/100k pop', ascending=False)
* # Display the results
* print(data['sex'])
* # Pre-processing generation property using one-hot encoding
* data\_encoded = pd.get\_dummies(data, columns=['generation'])
* # Extract the column names generated by one-hot encoding
* one\_hot\_columns = [col for col in data\_encoded.columns if 'generation\_' in col]
* # Add one-hot encoded columns to the original data frame
* data = data.join(data\_encoded[one\_hot\_columns])
* # Drop the original 'generation' column
* data.drop(columns=['generation'], inplace=True)
* print(data.info())
* Output

스크린샷, 텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 메뉴, 폰트이(가) 표시된 사진

자동 생성된 설명

* Analysis

: Using label encoding, males will be converted to 1 and females to 0.

1. Feature Engineering
2. Feature Creation

* Code
* # Calculate the mean suicide rate
* mean\_suicide\_rate = data["suicides/100k pop"].mean()
* threshold = mean\_suicide\_rate
* # Create a new attribute, Over\_threshold\_suicides
* # 1 if the suicide rate is higher than the mean suicide rate, 0 otherwise
* data["Over\_threshold\_suicides"] = np.where(data["suicides/100k pop"] > threshold, 1, 0)
* # Print the updated DataFrame information to see the new attribute and changes
* print(data.info())
* # Initialize KFold
* kf = KFold(n\_splits=10, shuffle=True, random\_state=33)
* # Prepare the new encoded column
* data['country\_encoded'] = 0.0
* # Perform target encoding using KFold
* for train\_index, val\_index in kf.split(data):
* train\_fold, val\_fold = data.iloc[train\_index], data.iloc[val\_index]
* mean\_encoded = train\_fold.groupby('country')['suicides/100k pop'].mean()
* data.loc[val\_index, 'country\_encoded'] = val\_fold['country'].map(mean\_encoded)
* # Split the data into training and testing datasets
* # Normally done for dividing data, but here it is used as an example to split part of the data arbitrarily
* train\_data = data.sample(frac=0.8, random\_state=33)
* test\_data = data.drop(train\_data.index)
* # Use mean encoding for the test dataset
* # Use the mean calculated from the entire dataset
* global\_mean = data['suicides/100k pop'].mean()
* data['country\_encoded'].fillna(global\_mean, inplace=True)
* # Drop the original 'country' column
* data.drop(columns=['country'], inplace=True)
* # Print the results
* print("Data with Target Encoding on 'country':")
* print(data['country\_encoded'])
* Output

텍스트, 스크린샷, 메뉴, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 메뉴이(가) 표시된 사진

자동 생성된 설명

* Analysis
* Create a new attribute ‘Over\_threshold\_suicides’: This attribute will indicate whether suicides/100k pop exceeds the threshold, denoted as 0 or 1.
* Target Encoding using ‘KFold Cross-Validation’: Calculate the mean of ‘suicides/100k pop’ for each country and map this to each validation set, creating a new column ‘country\_encoded’ that represents the average suicide rate per country.

1. Data Reduction
2. Data Filtering

* Code
* # Split the entire dataset into training and testing data with a 70:30 ratio
* train\_data, test\_data = train\_test\_split(data, test\_size=0.3, random\_state=33)
* # Use the properties we created ('Over\_threshold\_suicides') as predictions
* # Remove 'suicides/100k pop', a factor that affects prediction
* # Ensure that all necessary columns are present and no unnecessary columns are dropped
* columns\_to\_drop = ["Over\_threshold\_suicides", "suicides/100k pop"]
* # Drop columns in a single operation for both train and test sets
* X\_train = train\_data.drop(columns=columns\_to\_drop)
* Y\_train = ["Over\_threshold\_suicides"]
* X\_test = test\_data.drop(columns=columns\_to\_drop).copy()
* # Print the shapes of X\_train, Y\_train, and X\_test to verify
* print(f"X\_train shape: {X\_train.shape}")
* print(f"Y\_train shape: {Y\_train.shape}")
* print(f"X\_test shape: {X\_test.shape}")
* Output



* Analysis

: Split the entire dataset into training and test data at a 7:3 ratio. Use the ‘Over\_threshold\_suicides’ attribute as the target variable and remove the ‘suicides/100k pop’ column as it may influence the prediction.

1. Data Analysis (Modeling) & Model Evaluation

: Using the preprocessed data, we will build and analyze models. This involves identifying and predicting data patterns using statistical methods, machine learning algorithms, and deep learning models.

1. Linear Regression

* Analysis

: To facilitate data analysis and make it easier to compare results intuitively, the values of ‘suicides/100k pop’ were converted into percentages and encoded into a new attribute ‘suicides\_scaled’.

1. Logistic Regression

* Code
* # 1. Logistic Regression Model
* logreg = make\_pipeline(StandardScaler(), LogisticRegression(solver='lbfgs', max\_iter=1000))
* # Train the model
* logreg.fit(X\_train, Y\_train)
* # Predict on the training set and calculate accuracy
* Y\_pred = logreg.predict(X\_test)
* acc\_log = round(logreg.score(X\_train, Y\_train) \* 100, 3)
* print("Training Accuracy: ", acc\_log)
* # Evaluate the model using classification report
* print("Classification Report:")
* print(classification\_report(Y\_train, logreg.predict(X\_train)))
* # Display confusion matrix
* conf\_matrix = confusion\_matrix(Y\_train, logreg.predict(X\_train))
* sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Yes', 'No'], yticklabels=['Yes', 'No'])
* plt.xlabel('Predicted')
* plt.ylabel('Actual')
* plt.title('Confusion Matrix')
* plt.show()
* # Perform K-Fold cross-validation
* kf = KFold(n\_splits=10, shuffle=True, random\_state=33)
* cv\_results = cross\_val\_score(logreg, X\_train, Y\_train, cv=kf, scoring='accuracy')
* # Print the cross-validation results
* print(f"K-Fold Cross-Validation Scores: {cv\_results}")
* print(f"Average 10-Fold CV Score: {cv\_results.mean():.3f}, with a standard deviation of {cv\_results.std():.3f}")
* # Using different scoring metrics for cross-validation
* scores\_precision = cross\_val\_score(logreg, X\_train, Y\_train, cv=kf, scoring='precision')
* scores\_recall = cross\_val\_score(logreg, X\_train, Y\_train, cv=kf, scoring='recall')
* scores\_f1 = cross\_val\_score(logreg, X\_train, Y\_train, cv=kf, scoring='f1')
* print(f"Precision: {scores\_precision.mean():.3f}, Recall: {scores\_recall.mean():.3f}, F1-Score: {scores\_f1.mean():.3f}")
* Output
* Threshold = mean

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

* Threshold = mean \* 0.7

텍스트, 스크린샷, 소프트웨어, 컴퓨터이(가) 표시된 사진

자동 생성된 설명

* Analysis
* Training Accuracy
* Training Accuracy: 86.202%

: The model performs predictions on the training data with an accuracy of approximately 86%. Generally, high training accuracy indicates that the model has learned the data well, but it is also necessary to check for potential overfitting.

* Classification Report
* Class 0 (Positive)
* Precision: 0.89 – 89% of the model’s ‘Yes’ predictions are actually ‘Yes’.
* Recall: 0.91 – The model correctly predicts ‘Yes’ for 91% of actual ‘Yes’ cases.
* F1-Score: 0.90 – The harmonic means of precision and recall
* Class 1 (Negative)
* Precision: 0.80 – 80% of the model’s ‘No’ predictions are actually ‘No’.
* Recall: 0.75 – The model correctly predicts ‘No’ for 75% of actual ‘No’ cases.
* F1-Score: 0.77 – The harmonic means of precision and recall.
* The model shows higher accuracy and consistency in predicting ‘Class 0 (positive)’ and relatively lower performance in predicting ‘Class 1 (Negative)’.
* K-Fold Cross-Validation
* Cross-Validation Score: Mean 0.862, Standard Deviation: 0.008
* In evaluating the model’s generalization ability through cross-validation, it shows an average accuracy of about 86%, and a relatively low ‘\*’ standard deviation (0.008)’ indicates the stability of the model’s performance.
* This demonstrates that the model can maintain consistent performance across various sub-datasets.
* Various Cross-Validation Metrics
* Precision: 0.799
* Recall: 0.745
* F1-Score: 0.771
* The results from cross-validation show that the values for precision and recall have generally decreased, suggesting that the model may not be treating all classes equally well. Specifically, it indicates that while the model performs well for certain classes, it may not do so for others.

1. Decision Tree

* Code
* # 2. Decision Tree
* decision\_tree = DecisionTreeClassifier(random\_state=33)
* decision\_tree.fit(X\_train, Y\_train)
* # Predict on the training set
* Y\_pred = decision\_tree.predict(X\_test)
* acc\_decision\_tree = round(decision\_tree.score(X\_train, Y\_train) \* 100, 3)
* print("Training Accuracy: ", acc\_decision\_tree)
* # Evaluate the model using classification report
* print("Classification Report:")
* print(classification\_report(Y\_train, decision\_tree.predict(X\_train)))
* # Display confusion matrix
* conf\_matrix = confusion\_matrix(Y\_train, decision\_tree.predict(X\_train))
* sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Yes', 'No'], yticklabels=['Yes', 'No'])
* plt.xlabel('Predicted')
* plt.ylabel('Actual')
* plt.title('Confusion Matrix')
* plt.show()
* # Perform K-Fold cross-validation
* kf = KFold(n\_splits=10, shuffle=True, random\_state=33)
* cv\_results = cross\_val\_score(decision\_tree, X\_train, Y\_train, cv=kf, scoring='accuracy')
* # Print the cross-validation results
* print(f"K-Fold Cross-Validation Scores: {cv\_results}")
* print(f"Average 10-Fold CV Score: {cv\_results.mean():.3f}, with a standard deviation of {cv\_results.std():.3f}")
* # Using different scoring metrics for cross-validation
* scores\_precision = cross\_val\_score(decision\_tree, X\_train, Y\_train, cv=kf, scoring='precision')
* scores\_recall = cross\_val\_score(decision\_tree, X\_train, Y\_train, cv=kf, scoring='recall')
* scores\_f1 = cross\_val\_score(decision\_tree, X\_train, Y\_train, cv=kf, scoring='f1')
* print(f"Precision: {scores\_precision.mean():.3f}, Recall: {scores\_recall.mean():.3f}, F1-Score: {scores\_f1.mean():.3f}")
* Output
* Threshold = mean

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

* Threshold = mean \* 0.7

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

* Analysis
* Training Accuracy
* Training Accuracy: 100.0%

: This indicates that the model fits the training data ‘perfectly’, accurately classifying every data point. This often implies overfitting.

* Classification Report
* Class 0 (Yes)
* Precision: 1.00 – All ‘Yes’ predictions are correct.
* Recall: 1.00 – All actual ‘Yes’ instances are correctly predicted.
* F1-Score: 1.00 – A perfect balance of Precision and Recall.
* Class 1 (No)
* Precision: 1.00 – All ‘No’ predictions are correct.
* Recall: 1.00 – All actual ‘No’ instances are correctly predicted.
* F1-Score: 1.00 – A perfect balance of Precision and Recall.
* These ‘perfect metrics’ suggest the model is overfitted to the training data.
* K-Fold Cross-Validation
* Cross-Validation Score: Average: 0.912, Standard Deviation: 0.005

: The cross-validation results are ‘lower’ compared to the training accuracy, indicating a possibility of the model being overfitted to the training data.

* Various Cross-Validation Metrics
* Precision: 0.856
* Recall: 0.862
* F1-Score: 0.859
* Additional cross-validation metrics show good performance but do not match the perfect training performance. This indicates that the model is ‘performing reasonably well’ in real-world scenarios, but not to the extent of its perfect training performance.

<-> Q. How can the Dataset accuracy of Decision Tree be 100%?

1. <https://www.kaggle.com/code/fooenglow/suicide-prediction-decision-tree-random-forest> :

* Output텍스트, 스크린샷, 폰트이(가) 표시된 사진

  자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

* Refer to this result, training data analysis result accuracy of classifier decision tree shows 100%. Also, validation data analysis results show 88%. -> Above this, we can notice the evaluation is similar with our search.
* Analysis about why our decision tree’s training value is 100%.
* During the preprocessing stage, a "country encoded" attribute was added. It was identified that the "suicides/100k pop" attribute used for encoding this attribute affected the result, causing the accuracy of the decision tree to slightly drop to 99.995.
* As you can see from the outlier analysis graph of the "suicides/100k pop" column, most of the data in this attribute is distributed between 0 and 20, with an average value of approximately 10. However, when examining the identified outliers, there are values significantly higher than the average, ranging from 40 to a maximum of around 250. Given the nature of classifier models, which predict 1 if a threshold is exceeded and 0 otherwise, the presence of several outliers influences the decision of the threshold. As a result, most data points do not exceed the average value of the "suicides/100k pop" attribute used to determine the threshold, which has been assessed to impact the accuracy of the decision tree. By setting the bounds to 1.5 times the interquartile range (IQR), with Q1 at 0.25 and Q3 at 0.75, we identified approximately 2000 outliers. After removing each row containing outliers and training the model, we observed that the accuracy of the training data slightly decreased to 99.983.
* Finally, in out code, we trained the decision tree model without modifying the parameter values, using ‘decision\_tree = DecisionTreeClassifier()’. To prevent overfitting, we then trained the model by adjusting various parameter values.

For example, ‘decision\_tree = DecisionTreeClassifier(max\_depth = 10, min\_samples\_split=2, min\_samples\_leaf = 5, max\_leaf\_nodes=1000)’. As a result, we observed that the accuracy significantly decreased to ‘Training Accuracy: 82.085’.

* max\_depth: Specifies the maximum depth of the tree. Setting this value prevents the tree from growing deeper than the specified depth. A tree that is too deep can lead to overfitting, so this parameter helps control the complexity of the model.
* min\_samples\_split: Specifies the minimum number of samples required to split a node. Nodes with fewer samples than this value will not be split. Setting a higher value results in fewer splits, which makes the tree shallower and helps reduce overfitting.
* min\_samples\_leaf: Specifies the minimum number of samples required to be at a leaf node. A leaf node is the final node of the tree, used to make predictions. This parameter prevents leaf nodes from being created with too few samples, which can improve the generalization ability of the model.
* max\_leaf\_nodes: Specifies the maximum number of leaf nodes allowed in the tree. By limiting the number of leaf nodes, this parameter controls the size and complexity of the tree. A smaller value reduces the complexity of the model, which helps mitigate the risk of overfitting.

1. K-Nearest Neighbors (KNN)

* Code
* # 3. K-Nearest Neighbors
* knn = KNeighborsClassifier(n\_neighbors=3)
* knn.fit(X\_train, Y\_train)
* # Predict on the training set
* Y\_pred = knn.predict(X\_test)
* acc\_knn = round(knn.score(X\_train, Y\_train) \* 100, 3)
* print("Training Accuracy: ", acc\_knn)
* # Evaluate the model using classification report
* print("Classification Report:")
* print(classification\_report(Y\_train, knn.predict(X\_train)))
* # Display confusion matrix
* conf\_matrix = confusion\_matrix(Y\_train, knn.predict(X\_train))
* sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Yes', 'No'], yticklabels=['Yes', 'No'])
* plt.xlabel('Predicted')
* plt.ylabel('Actual')
* plt.title('Confusion Matrix')
* plt.show()
* # Perform K-Fold cross-validation
* kf = KFold(n\_splits=10, shuffle=True, random\_state=33)
* cv\_results = cross\_val\_score(knn, X\_train, Y\_train, cv=kf, scoring='accuracy')
* # Print the cross-validation results
* print(f"K-Fold Cross-Validation Scores: {cv\_results}")
* print(f"Average 10-Fold CV Score: {cv\_results.mean():.3f}, with a standard deviation of {cv\_results.std():.3f}")
* # Using different scoring metrics for cross-validation
* scores\_precision = cross\_val\_score(knn, X\_train, Y\_train, cv=kf, scoring='precision')
* scores\_recall = cross\_val\_score(knn, X\_train, Y\_train, cv=kf, scoring='recall')
* scores\_f1 = cross\_val\_score(knn, X\_train, Y\_train, cv=kf, scoring='f1')
* print(f"Precision: {scores\_precision.mean():.3f}, Recall: {scores\_recall.mean():.3f}, F1-Score: {scores\_f1.mean():.3f}")
* Output
* Threshold = mean

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

* Threshold = mean \* 0.7

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

* Analysis
* Training Accuracy
* Training Accuracy: 86.752%
* This model makes correct predictions on the training data with an accuracy of approximately 87%. This indicates that the model is relatively effective at learning the patterns in the data.
* Classification Report
* Class 0 (Positive)
* Precision: 0.88 - 88% of the predicted 'Yes' instances were actually 'Yes'
* Recall: 0.93 - 93% of the actual 'Yes' instances were correctly predicted
* F1-Score: 0.91 - The harmonic means of precision and recall
* Class 1 (Negative)
* Precision: 0.83 - 83% of the predicted 'No' instances were actually 'No'
* Recall: 0.72 - 72% of the actual 'No' instances were correctly predicted
* F1-Score: 0.77 - The harmonic means of precision and recall
* This model is more effective at predicting ‘Class 0 (Positive)’, while showing a need for improvement in predicting ‘Class 1 (Negative)’.
* K-Fold Cross-Validation
* Cross-Validation Scores
* Mean: 0.700, Standard Deviation: 0.010
* The cross-validation results indicate that the model exhibits some inconsistency and generally shows lower generalization performance compared to its performance on the training data.
* Various Cross-Validation Metrics
* Precision: 0.523
* Recall: 0.452
* F1-Score: 0.485
* Additional cross-validation metrics show that the model's precision, recall, and F1 score are relatively ‘low’. This suggests that the model might be struggling with ‘imbalanced data’ and indicates a need for improvement in predicting specific classes.

1. Gaussian Naïve Bayes

* Code
* # 4. Gaussian Naive Bayes
* gaussian = GaussianNB()
* gaussian.fit(X\_train, Y\_train)
* # Predict on the training set
* Y\_pred = gaussian.predict(X\_test)
* acc\_gaussian = round(gaussian.score(X\_train, Y\_train) \* 100, 3)
* print("Training Accuracy: ", acc\_gaussian)
* # Evaluate the model using classification report
* print("Classification Report:")
* print(classification\_report(Y\_train, gaussian.predict(X\_train)))
* # Display confusion matrix
* conf\_matrix = confusion\_matrix(Y\_train, gaussian.predict(X\_train))
* sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Yes', 'No'], yticklabels=['Yes', 'No'])
* plt.xlabel('Predicted')
* plt.ylabel('Actual')
* plt.title('Confusion Matrix')
* plt.show()
* # Perform K-Fold cross-validation
* kf = KFold(n\_splits=10, shuffle=True, random\_state=33)
* cv\_results = cross\_val\_score(gaussian, X\_train, Y\_train, cv=kf, scoring='accuracy')
* # Print the cross-validation results
* print(f"K-Fold Cross-Validation Scores: {cv\_results}")
* print(f"Average 10-Fold CV Score: {cv\_results.mean():.3f}, with a standard deviation of {cv\_results.std():.3f}")
* # Using different scoring metrics for cross-validation
* scores\_precision = cross\_val\_score(gaussian, X\_train, Y\_train, cv=kf, scoring='precision')
* scores\_recall = cross\_val\_score(gaussian, X\_train, Y\_train, cv=kf, scoring='recall')
* scores\_f1 = cross\_val\_score(gaussian, X\_train, Y\_train, cv=kf, scoring='f1')
* print(f"Precision: {scores\_precision.mean():.3f}, Recall: {scores\_recall.mean():.3f}, F1-Score: {scores\_f1.mean():.3f}")
* Output
* Threshold = mean

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

* Threshold = mean \* 0.7

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

* Analysis
* Training Accuracy
* Training Accuracy: 86.752%
* This model makes correct predictions on the training data with an accuracy of approximately ‘87%’. This indicates that the model is relatively effective at learning the patterns in the data.
* Classification Report
* Class 0 (Positive)
* Precision: 0.84 – ‘84%’ of the predicted ‘Yes’ instances were actually ‘Yes’.
* Recall: 0.93 – ‘93%’ of the actual ‘Yes’ instances were correctly predicted.
* F1-Score: 0.91 – The harmonic means of precision and recall.
* Class 1 (Negative)
* Precision: 0.83 – ‘83%’ of the predicted ‘No’ instances were actually ‘No’.
* Recall: 0.72 – ‘72%’ of the actual ‘No’ instances were correctly predicted.
* F1-Score: 0.77 – The harmonic means of precision and recall.
* This model is more effective at predicting ‘Class 0 (Positive)’, while showing a need for improvement in predicting ‘Class 1 (Negative)’.
* K-Fold Cross-Validation
* Cross-Validation Scores
* Mean: 0.700, Standard Deviation: 0.010
* The cross-validation results indicate that the model exhibits some inconsistency and generally shows lower generalization performance compared to its performance on the training data.
* Various Cross-Validation Metrics
* Precision: 0.523
* Recall: 0.452
* F1-Score: 0.485
* Additional cross-validation metrics show that the model's precision, recall, and F1 score are relatively ‘low’. This suggests that the model might be struggling with ‘imbalanced data’ and indicates a need for improvement in predicting specific classes.

1. Analysis of Model Performance changes

: Overall, it can be observed that the values for ‘Class 0 (Positive)’ have decreased while the values for ‘Class 1 (Negative)’ have increased. Additionally, for models like ‘Decision Tree’ and ‘Random Forest’, the overall performance has slightly decreased.

* Changes in Class Balance

: By lowering the threshold to 70% for the suicide rate, the number of 'Positive' class instances decreased, and the number of 'Negative' class instances increased.

* Changes in Performance Metrics
* Class 0 (Positive): Precision and Recall have decreased. This means that the criteria for predicting the 'Positive' class have become stricter, leading to an increase in missed actual 'Positive' instances.
* Class 1 (Negative): Precision and Recall have improved. This indicates that the criteria for determining the 'Negative' class have been relaxed, allowing the model to better predict 'Negative' instances -> ‘A positive outcome for a model predicting suicide rates’
* Decrease in Decision Tree Performance

: The performance of Decision Tree model has generally decreased slightly. The adjustment of the threshold seems to have impacted the data splitting methods of these tree-based models, leading to an increase in errors particularly in handling the 'Negative' class.

1. Open Source SW
2. Function Definition

* Data Analysis / Machine Learning / Preprocessing
* Code
* # Data Analysis
* import pandas as pd
* import numpy as np
* # Machine Learning
* from sklearn.linear\_model import LogisticRegression
* from sklearn.ensemble import RandomForestClassifier
* from sklearn.neighbors import KNeighborsClassifier
* from sklearn.naive\_bayes import GaussianNB
* from sklearn.tree import DecisionTreeClassifier
* from sklearn.model\_selection import KFold, train\_test\_split
* from sklearn.model\_selection import cross\_val\_score
* from sklearn.preprocessing import StandardScaler
* from sklearn.pipeline import make\_pipeline
* from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score
* # Load the dataset
* data = pd.read\_csv('master.csv')
* # Initial count of rows
* initial\_count = data.shape[0]
* # Detecting outliers using IQR
* Q1 = data['suicides/100k pop'].quantile(0.25)
* Q3 = data['suicides/100k pop'].quantile(0.75)
* IQR = Q3 - Q1
* lower\_bound = Q1 - 1.5 \* IQR
* upper\_bound = Q3 + 1.5 \* IQR
* # Counting outliers
* outliers = data[(data['suicides/100k pop'] < lower\_bound) | (data['suicides/100k pop'] > upper\_bound)]
* outlier\_count = outliers.shape[0]
* # Plotting outliers
* sns.boxplot(x=data['suicides/100k pop'])
* plt.title('Box Plot before Removing Outliers')
* plt.show()
* # Removing outliers
* data = data[(data['suicides/100k pop'] >= lower\_bound) & (data['suicides/100k pop'] <= upper\_bound)]
* # Final count of rows
* final\_count = data.shape[0]
* print("Initial number of rows:", initial\_count)
* print("Outliers detected:", outlier\_count)
* print("Number of rows after removing outliers:", final\_count)
* # Optionally, plot to confirm outlier removal
* sns.boxplot(x=data['suicides/100k pop'])
* plt.title('Box Plot after Removing Outliers')
* plt.show()
* def preprocess\_classification(dataset, threshold\_factor=1.0):
* data = pd.read\_csv(dataset)
* # Remove duplicate attributes
* data.drop(["suicides\_no", "population", "country-year", "gdp\_for\_year ($)"], axis=1, inplace=True)
* data.drop("HDI for year", axis=1, inplace=True)
* # Remove 'years' text from 'age' and convert age strings to integers
* data["age"] = data["age"].str.strip(" years")
* def convert\_age(age):
* if '75+' in age:
* return 80
* elif '55-74' in age:
* return 65
* elif '35-54' in age:
* return 45
* elif '25-34' in age:
* return 30
* elif '15-24' in age:
* return 20
* elif '5-14' in age:
* return 10
* else:
* return None
* data['age'] = data['age'].apply(convert\_age)
* # Encode sex using a mapping
* sex\_mapping = {'female': 0, 'male': 1}
* data['sex'] = data['sex'].map(sex\_mapping)
* # Apply one-hot encoding to 'generation'
* data = pd.get\_dummies(data, columns=['generation'])
* # Calculate the mean suicide rate and adjust by the threshold\_factor
* mean\_suicide\_rate = data["suicides/100k pop"].mean()
* adjusted\_threshold = mean\_suicide\_rate \* threshold\_factor
* # Create a new attribute, Over\_threshold\_suicides
* # 1 if the suicide rate is higher than the adjusted threshold, 0 otherwise
* data["Over\_threshold\_suicides"] = np.where(data["suicides/100k pop"] > adjusted\_threshold, 1, 0)
* # Remove the original 'country' column
* data.drop(columns=['country'], inplace=True)
* print(data.info())
* train\_data, test\_data = train\_test\_split(data, test\_size=0.3, random\_state=33)
* # Drop unnecessary columns
* columns\_to\_drop = ["suicides/100k pop", "Over\_threshold\_suicides"]
* X\_train = train\_data.drop(columns=columns\_to\_drop)
* Y\_train = train\_data["Over\_threshold\_suicides"]
* X\_test = test\_data.drop(columns=columns\_to\_drop)
* Y\_test = test\_data["Over\_threshold\_suicides"]
* print(data.info())
* return X\_train, Y\_train, X\_test, Y\_test
* Analysis

: Code simplified by converting the data preprocessing part into functions / Completing the data preprocessing code

* Adjust Weight
* Code
* def adjust\_weights(weights):
* # Calculate the sum of all provided weights
* total = sum(weights.values())
* # Normalize each weight by dividing by the total sum to adjust their proportions
* adjusted\_weights = {key: value / total for key, value in weights.items()}
* return adjusted\_weights
* # Example weights input from the user
* input\_weights = {
* 'test\_accuracy': 1,  # Weight for test accuracy
* 'precision': 3,      # Weight for precision
* 'recall': 3,         # Weight for recall
* 'f1\_positive': 1,    # Weight for F1 score of the positive class
* 'f1\_negative' : 4,   # Weight for F1 score of the negative class
* 'cv\_accuracy': 1     # Weight for cross-validation accuracy
* }
* # Adjust the weights to ensure their sum equals 1
* WEIGHTS = adjust\_weights(input\_weights)
* Analysis

: In the following code, we will calculate the integrated scores of each model by multiplying each score by its respective weight. This is the necessary code for that purpose. Here, we determine which weights to assign to which scores. For example, if the test accuracy is 1 and the weight of f1 negative is 4, then we will later multiply the test accuracy and f1 negative values by 1 / (1 + 3 + 3 + 1 + 1 + 4 + 1) and 4 / (1 + 3 + 3 + 1 + 1 + 4 + 1) respectively. The reason why the weight of f1 negative is higher than that of f1 positive is because the goal of our project is to predict suicides, so the model should more leniently capture those who are expected to be at risk of suicide.

* Logistic regression model
* Code
* def evaluate\_logistic\_regression(X\_train, Y\_train, X\_test, Y\_test):
* # Create a logistic regression pipeline with feature scaling
* logreg = make\_pipeline(StandardScaler(), LogisticRegression(solver='lbfgs', max\_iter=1000))
* logreg.fit(X\_train, Y\_train)  # Train the model
* # Make predictions and calculate accuracy on the test set
* Y\_pred\_test = logreg.predict(X\_test)
* test\_accuracy = accuracy\_score(Y\_test, Y\_pred\_test)
* # Compute precision and recall for the model using macro averaging
* precision = precision\_score(Y\_test, Y\_pred\_test, average='macro')
* recall = recall\_score(Y\_test, Y\_pred\_test, average='macro')
* # Compute F1 scores for each class and extract separately
* f1\_scores = f1\_score(Y\_test, Y\_pred\_test, average=None)
* f1\_positive, f1\_negative = f1\_scores[0], f1\_scores[1]
* # Conduct 10-fold cross-validation to estimate model stability
* kf = KFold(n\_splits=10, shuffle=True, random\_state=33)
* cv\_accuracy = cross\_val\_score(logreg, X\_train, Y\_train, cv=kf, scoring='accuracy').mean()
* # Calculate integrated score using weighted averages from WEIGHTS
* integrated\_score = (
* test\_accuracy \* WEIGHTS['test\_accuracy'] +
* precision \* WEIGHTS['precision'] +
* recall \* WEIGHTS['recall'] +
* f1\_positive \* WEIGHTS['f1\_positive'] +
* f1\_negative \* WEIGHTS['f1\_negative'] +
* cv\_accuracy \* WEIGHTS['cv\_accuracy']
* )
* return integrated\_score \* 100 # Convert to percentage
* Analysis

: The integrated score is calculated by multiplying the ratio of weights obtained earlier by the final integrated score. \*100 is used to convert the value to a percentage for better readability. The integrated score is measured using the accuracy, precision, recall, F1 positive, F1 negative of each test data, and the accuracy of K-Fold CV (K=10).

* Decision Tree
* Code
* def evaluate\_decision\_tree(X\_train, Y\_train, X\_test, Y\_test):
* # Initialize and train a Decision Tree model
* decision\_tree = DecisionTreeClassifier(random\_state=33)
* decision\_tree.fit(X\_train, Y\_train)
* # Predict and evaluate accuracy on both training and test sets
* Y\_pred\_test = decision\_tree.predict(X\_test)
* test\_accuracy = accuracy\_score(Y\_test, Y\_pred\_test)
* # Calculate precision and recall using macro averaging to treat all classes equally
* precision = precision\_score(Y\_test, Y\_pred\_test, average='macro', zero\_division=0)
* recall = recall\_score(Y\_test, Y\_pred\_test, average='macro', zero\_division=0)
* # Compute F1 scores for positive and negative classes individually
* f1\_scores = f1\_score(Y\_test, Y\_pred\_test, average=None)
* f1\_positive, f1\_negative = f1\_scores[0], f1\_scores[1]  # F1 scores for class 0 and class 1
* # Perform 10-fold cross-validation to estimate model robustness
* kf = KFold(n\_splits=10, shuffle=True, random\_state=33)
* cv\_accuracy = cross\_val\_score(decision\_tree, X\_train, Y\_train, cv=kf, scoring='accuracy').mean()
* # Calculate an integrated score based on predefined weights
* integrated\_score = (test\_accuracy \* WEIGHTS['test\_accuracy'] +
* precision \* WEIGHTS['precision'] +
* recall \* WEIGHTS['recall'] +
* f1\_positive \* WEIGHTS['f1\_positive'] +
* f1\_negative \* WEIGHTS['f1\_negative'] +
* cv\_accuracy \* WEIGHTS['cv\_accuracy'])
* return integrated\_score \* 100 # Convert to percentage
* Analysis

: Function to evaluate total score of decision tree model

* KNN / Gaussian
* code
* def evaluate\_knn(X\_train, Y\_train, X\_test, Y\_test):
* # Initialize and train a K-Nearest Neighbors model with 3 neighbors
* knn = KNeighborsClassifier(n\_neighbors=3)
* knn.fit(X\_train, Y\_train)
* # Predict and evaluate accuracy on both training and test sets
* Y\_pred\_train = knn.predict(X\_train)
* Y\_pred\_test = knn.predict(X\_test)
* train\_accuracy = accuracy\_score(Y\_train, Y\_pred\_train)
* test\_accuracy = accuracy\_score(Y\_test, Y\_pred\_test)
* # Calculate precision, recall, and F1 scores using macro averaging
* precision = precision\_score(Y\_test, Y\_pred\_test, average='macro')
* recall = recall\_score(Y\_test, Y\_pred\_test, average='macro')
* # Compute F1 scores for positive and negative classes individually
* f1\_scores = f1\_score(Y\_test, Y\_pred\_test, average=None)
* f1\_positive, f1\_negative = f1\_scores[0], f1\_scores[1]  # F1 scores for class 0 and class 1
* # Perform 10-fold cross-validation to estimate model robustness
* kf = KFold(n\_splits=10, shuffle=True, random\_state=33)
* cv\_accuracy = cross\_val\_score(knn, X\_train, Y\_train, cv=kf, scoring='accuracy').mean()
* # Calculate an integrated score based on predefined weights
* integrated\_score = (test\_accuracy \* WEIGHTS['test\_accuracy'] +
* precision \* WEIGHTS['precision'] +
* recall \* WEIGHTS['recall'] +
* f1\_positive \* WEIGHTS['f1\_positive'] +
* f1\_negative \* WEIGHTS['f1\_negative'] +
* cv\_accuracy \* WEIGHTS['cv\_accuracy'])
* return integrated\_score \* 100 # Convert to percentage
* def evaluate\_gaussian\_nb(X\_train, Y\_train, X\_test, Y\_test):
* # Initialize and train Gaussian Naive Bayes model
* gaussian = GaussianNB()
* gaussian.fit(X\_train, Y\_train)
* # Predict on the test set and calculate accuracy
* Y\_pred\_test = gaussian.predict(X\_test)
* test\_accuracy = accuracy\_score(Y\_test, Y\_pred\_test)
* # Calculate precision, recall, and F1 score using macro averaging
* precision = precision\_score(Y\_test, Y\_pred\_test, average='macro', zero\_division=0)
* recall = recall\_score(Y\_test, Y\_pred\_test, average='macro', zero\_division=0)
* # Compute F1 scores for positive and negative classes individually
* f1\_scores = f1\_score(Y\_test, Y\_pred\_test, average=None)
* f1\_positive, f1\_negative = f1\_scores[0], f1\_scores[1]  # F1 scores for class 0 and class 1
* # Perform 10-fold cross-validation to estimate the model's robustness
* kf = KFold(n\_splits=10, shuffle=True, random\_state=33)
* cv\_accuracy = cross\_val\_score(gaussian, X\_train, Y\_train, cv=kf, scoring='accuracy').mean()
* # Calculate an integrated score based on predefined weights
* integrated\_score = (test\_accuracy \* WEIGHTS['test\_accuracy'] +
* precision \* WEIGHTS['precision'] +
* recall \* WEIGHTS['recall'] +
* f1\_positive \* WEIGHTS['f1\_positive'] +
* f1\_negative \* WEIGHTS['f1\_negative'] +
* cv\_accuracy \* WEIGHTS['cv\_accuracy'])
* return integrated\_score \* 100  # Convert to percentage
* Analysis

: Function to evaluate total score of KNN, Gaussian

* Model Evaluator
* Code
* class ModelEvaluator:
* def \_\_init\_\_(self, dataset, models, threshold\_factors, top\_n=5):
* self.dataset = dataset
* self.models = models
* self.threshold\_factors = threshold\_factors
* self.top\_n = top\_n
* def evaluate\_models(self):
* results = []
* for threshold in self.threshold\_factors:
* X\_train, Y\_train, X\_test, Y\_test = preprocess\_classification(self.dataset, threshold)
* for model\_name, model\_func in self.models.items():
* score = model\_func(X\_train, Y\_train, X\_test, Y\_test)
* results.append({
* 'model': model\_name,
* 'threshold': threshold,
* 'score': score
* })
* # Sorting and returning top n results
* return sorted(results, key=lambda x: x['score'], reverse=True)[:self.top\_n]
* # Model functions need to be defined elsewhere in the script or as imports
* # Configuration
* dataset\_path = 'master.csv'
* threshold\_range = np.linspace(0.5, 1.5, 21)  # Range of thresholds from 0.5 to 1.5
* top\_results\_count = 20  # Number of top models to return
* # Dictionary of models to evaluate
* models\_to\_evaluate = {
* 'Logistic Regression': evaluate\_logistic\_regression,
* 'Decision Tree': evaluate\_decision\_tree,
* }
* # Creating an instance of the evaluator
* evaluator = ModelEvaluator(dataset\_path, models\_to\_evaluate, threshold\_range, top\_results\_count)
* best\_models = evaluator.evaluate\_models()
* # Display results
* for idx, result in enumerate(best\_models, start=1):
* print(f"Rank {idx}:")
* print(f"\tModel - {result['model']}")
* print(f"\tThreshold Factor - {result['threshold']:.5f}")
* print(f"\tIntegrated Score - {result['score']:.5f}")
* print()
* Analysis

: The final code saves the integrated scores from running the models into the results list and returns them sorted in descending order. This is followed by code that changes the threshold from 0.5 to 1.5, 21 times, to find the model that returns the optimal score. If the threshold is 0.5, it means the model predicts a suicide if the data exceeds 50% of the average suicide rate.

* Final output
* Output
* Analysis

: "As a result of running the code, all the top 1 to 5 models turned out to be decision tree models. Each had thresholds of 0.7, 0.75, 0.55. It was found that the models with thresholds generally lower than the average suicide rate predicted better."

1. GitHub URL

<https://github.com/Kim-Yoo-Hyun/-10>