MENTAL FITNESS TRACKER

COLLEGE OF ENGINEERING & TECHNOLOGY

SRM INSTITUTE OF SCIENCE & TECHNOLOGY

S.R.M. NAGAR, KATTANKULATHUR – 603 203

Submitted by

HARISH G [RA2111026010284]

NILAY KALE [RA2111026010287]

Under the Guidance of

Dr. M. UMA

Associate Professor, Department of Computational Intelligence

In partial satisfaction of the requirements for the degree of

BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE ENGINEERING

with specialization in Artificial Intelligence and Machine Learning

SCHOOL OF COMPUTING

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR – 603 203

OCTOBER 2023

COLLEGE OF ENGINEERING & TECHNOLOGY

SRM INSTITUTE OF SCIENCE & TECHNOLOGY

S.R.M. NAGAR, KATTANKULATHUR – 603 203

BONAFIDE CERTIFICATE

Register No. RA2111026010284, RA2111026010287, Certified to be the bonafide work done by HARISH G, NILAY KALE, of III Year/V Sem B. Tech Degree Course in the subject **18CSE479T – Statistical Machine Learning** in **SRM INSTITUTE OF SCIENCE AND TECHNOLOGY,** Kattankulathur during the academic year 2023 – 2024.

SIGNATURE

Dr. M. UMA
Associate Professor
DEPARTMENT OF COMPUTATIONAL INTELLIGENCE
SRMIST – KTR.

SIGNATURE

Dr R ANNIE UTHRA
HEAD OF DEPARTMENT
DEPARTMENT OF COMPUTATIONAL INTELIGENCE
SRMIST – KTR.

Date:

TABLE OF CONTENTS

1. INTRODUCTION

The "Mental Fitness Tracker" is a cutting-edge statistical machine learning project designed to revolutionize the way we monitor and support mental health. This innovative system leverages advanced statistical algorithms and machine learning techniques to analyse user-generated data, such as mood logs, sleep patterns, and stress levels, to provide personalized insights and recommendations for mental well-being. By harnessing the power of data-driven analytics, this project aims to empower individuals to better understand and manage their mental fitness, ultimately promoting mental health awareness and resilience. In a world increasingly concerned with mental well-being, the "Mental Fitness Tracker" represents a promising step towards a more proactive and informed approach to mental health care. The "Mental Fitness Tracker" employs state-of-the-art encryption protocols to ensure the utmost privacy and security of user data, fostering trust and confidentiality. Its user-friendly interface allows for seamless integration into daily routines, enabling individuals to easily track and reflect on their mental health patterns. This innovative tool has the potential to assist healthcare professionals by providing comprehensive insights, facilitating more personalized and effective interventions for their patients. As it evolves, this system could contribute significantly to mental health research by aggregating anonymized data to identify broader trends and patterns. Through continuous updates and refinements, the "Mental Fitness Tracker" aspires to be an adaptable and indispensable companion in the quest for improved mental well-being globally.

2. LITERATURE SURVEY

The literature survey for the "Mental Health Fitness Tracker" project reveals a rich landscape of studies that delve into analysing and predicting mental fitness levels across diverse populations with varying mental disorders. Researchers have extensively employed regression techniques to gain insights into mental health parameters, leveraging data-driven approaches to enhance our understanding of these conditions. Numerous studies have explored the application of regression models for predicting mental health outcomes, emphasizing the importance of feature selection and data preprocessing techniques. The ones we have primarily used are listed below:

[1]. Forecasting Mood in Bipolar Disorder From Smartphone Self-assessments: Hierarchical Bayesian Approach [Apr. 2020]

- Monitoring Editor: Gunther Eysenbach
- Reviewed by Diego Hidalgo-Mazzei, Hossein Hassani, Michael Ostacher, and Andrea Graham
- Abstract: Bipolar disorder is a prevalent mental health condition that is imposing significant burden on society. Accurate forecasting of symptom scores can be used to improve disease monitoring, enable early intervention, and eventually help prevent costly hospitalizations. Although several studies have examined the use of smartphone data to detect mood, only few studies deal with forecasting mood for one or more days.

  This study aimed to examine the feasibility of forecasting daily subjective mood scores based on daily self-assessments collected from patients with bipolar disorder via a smartphone-based system in a randomized clinical trial.

[2]. Machine Learning in Mental Health: A Systematic Review of the HCI Literature to Support the Development of Effective and Implementable ML Systems [Aug. 2020]

- Monitoring Editor: ANJA THIEME and DANIELLE BELGRAVE, Microsoft Research, GAVIN DOHERTY, Trinity College Dublin
- Abstract: Increases in the occurrence and global burden of mental illness have made the prevention and treatment of mental disorders a public health priority [90, 91, 204, 207]. A 2017 US report showed that an estimated 46.6 million adults have been affected by a mental illness. This equates to nearly 20% of the US population alone [169]. Responding to the need for more effective mental health services, the role of

digital technology for improving access, engagement, and outcomes of therapeutic treatment is increasing in importance and has led to a wide range of health technologies and applications (e.g., [41, 156, 187]). These include mobile apps and wearable devices to assist symptom monitoring and health risk assessments [12, 48], computerized treatments [49, 157, 171], and mental health peer or community support [99, 137, 150]. These systems as well as people's everyday technology interactions and the information that is accumulated in electronic health care records (EHR) increasingly provide a wealth of personal health and behavioral data [65, 116, 124]. Eyre et al. [56] even suggest that the field of "mental health captures arguably the largest amount of data of any medical specialty".

In this regard, our literature review was guided by six main questions:

- What types of ML models and applications are currently being developed for mental health?
- What motivates the use of ML in the reported works and what aspects of mental health do they target?
- What types and scale of data is used for ML analysis and how is access to mental health data achieved?
- What techniques were applied (and challenges encountered) in developing and evaluating ML models?
- What key learnings are reported in the literature and to what extent do they apply to real-world contexts?
- To what extent do the papers describe ethical challenges or implications?

First, we describe our systematic review methodology; followed by the findings that were extracted from the review corpus papers. The article concludes with a critical discussion of the findings and provides a set of concrete suggestions for steps forward in developing ML applications and systems that are useful, ethical and implementable in supporting mental health.

Furthermore, there is a growing interest in the integration of machine learning and clinical psychology, with researchers developing innovative models to assist mental health professionals in early diagnosis and personalized treatment recommendations. The "Mental Health Fitness Tracker" project stands at the intersection of these emerging trends, aiming to contribute to this evolving field by offering a holistic and data-driven approach to mental health assessment and support.

3. STATISTICAL ANALYSIS

3.1 MEAN MEDIAN MODE

Mean: The mean, often referred to as the average, is the sum of all values in a dataset divided by the total number of values. It is a measure of central tendency that represents the typical value within a dataset.

Median: This is the value present in the middle of a dataset when the values are arranged in ascending or descending order. If the dataset contains an even number of values, the median is the average of the two middle numbers. It is another measure of central tendency that is less sensitive to extreme values, also known as outliers.

Mode: This is the value appearing most frequently in a dataset. Unlike the mean and median, the mode identifies the most common value or values present in the dataset. If there is no value that appears more than once, the dataset is considered to have no mode or to be "uniform."

DF 1: mental-and-substance-use-as-share-of-disease.csv
DF1 has the output column “DALYs (Disability-Adjusted Life Years) - Mental disorders - Sex: Both - Age: All Ages (Percent)”

MEAN, MEDIAN & MODE

```
# Calculate the mean of a specific column (e.g., 'column_name')
mean_value = df1['DALYs (Disability-Adjusted Life Years) -
Mental disorders - Sex: Both - Age: All Ages (Percent)'].mean()
print(f"Mean: {mean_value}")


# Calculate the median of a specific column (e.g.,
'column_name')
median_value = df1['DALYs (Disability-Adjusted Life Years) -
Mental disorders - Sex: Both - Age: All Ages
(Percent)'].median()
print(f"Median: {median_value}")

from scipy import stats

# Calculate the mode of a specific column (e.g., 'column_name')
```

```
mode_value = stats.mode(df1['DALYs (Disability-Adjusted Life
Years) - Mental disorders - Sex: Both - Age: All Ages
(Percent)'])
print(f"Mode: {mode_value.mode[0]}")
```

Output:

```
In [14]: df1 = pd.read_csv('mental-and-substance-use-as-share-of-disease.csv')
         df2 = pd.read_csv('prevalence-by-mental-and-substance-use-disorder.csv')

In [15]: # Calculate the mean of a specific column (e.g., 'column_name')
         mean_value = df1['DALYs (Disability-Adjusted Life Years) - Mental disorders - Sex: Both - Age: All Ages (Percent)'].mean()
         print(f"Mean: {mean_value}")

         Mean: 4.818070175438599

In [18]: # Calculate the median of a specific column (e.g., 'column_name')
         median_value = df1['DALYs (Disability-Adjusted Life Years) - Mental disorders - Sex: Both - Age: All Ages (Percent)'].median()
         print(f"Median: {median_value}")

         Median: 4.68

In [20]: from scipy import stats

         # Calculate the mode of a specific column (e.g., 'column_name')
         mode_value = stats.mode(df1['DALYs (Disability-Adjusted Life Years) - Mental disorders - Sex: Both - Age: All Ages (Percent)'])
         print(f"Mode: {mode_value.mode[0]}")

         Mode: 4.59
```

DF 2: prevalence-by-mental-and-substance-use-disorder.csv

DF2 has the following output columns.

- **Schizophrenia**: A severe mental illness characterized by warped reality.
- **Bipolar disorder**: Mood swings between emotional extremes are a symptom of bipolar illness.
- **Eating disorders**: It includes poor eating practices that compromise personal wellbeing.
- **Anxiety disorders**: Excessive concern or fear that persists over time.
- **Substance use disorders**: Substance abuse that is problematic and harmful.
- **Depressive disorders**: Prolonged melancholy, disinterest.
- **Alcohol use disorders**: Addiction to alcohol that negatively affects life.

MEAN

```
# Calculate the mean of a specific column (e.g., 'column_name')
mean_value = df2['Schizophrenia'].mean()
print(f"Mean: {mean_value}")
```

Output:

```
In [22]: # Calculate the mean of a specific column (e.g., 'column_name')
         mean_value = df2['Schizophrenia'].mean()
         print(f"Mean: {mean_value}")

         Mean: 0.28162353135522966

In [23]: mean_value = df2['Bipolar disorder'].mean()
         print(f"Mean: {mean_value}")

         Mean: 0.7263415339986797

In [24]: mean_value = df2['Eating disorders'].mean()
         print(f"Mean: {mean_value}")

         Mean: 0.23316536834755458

In [25]: mean_value = df2['Anxiety disorders'].mean()
         print(f"Mean: {mean_value}")

         Mean: 4.359566538661005

In [26]: mean_value = df2['Drug use disorders'].mean()
         print(f"Mean: {mean_value}")

         Mean: 0.7210294214651618

In [27]: mean_value = df2['Depressive disorders'].mean()
         print(f"Mean: {mean_value}")

         Mean: 3.8456315513269423

In [28]: mean_value = df2['Alcohol use disorders'].mean()
         print(f"Mean: {mean_value}")

         Mean: 1.6162372745285458
```

MEDIAN

```
median_value = df2['Alcohol use disorders'].median()
print(f"Median: {median_value}")
```

Output:

```
In [29]: median_value = df2['Schizophrenia'].median()
         print(f"Median: {median_value}")

         Median: 0.29083998022821

In [30]: median_value = df2['Bipolar disorder'].median()
         print(f"Median: {median_value}")

         Median: 0.58743300435136

In [31]: median_value = df2['Eating disorders'].median()
         print(f"Median: {median_value}")

         Median: 0.14920781315807

In [32]: median_value = df2['Anxiety disorders'].median()
         print(f"Median: {median_value}")

         Median: 4.18852032454975

In [33]: median_value = df2['Drug use disorders'].median()
         print(f"Median: {median_value}")

         Median: 0.5931298069331901

In [34]: median_value = df2['Depressive disorders'].median()
         print(f"Median: {median_value}")

         Median: 3.8447497938764

In [35]: median_value = df2['Alcohol use disorders'].median()
         print(f"Median: {median_value}")

         Median: 1.6011262494403802
```

MODE

```
from scipy import stats

# Calculate the mode of a specific column (e.g., 'column_name')
mode_value = stats.mode(df2['Schizophrenia'])
print(f"Mode: {mode_value.mode[0]}")
```

Output:

Result:

The calculated mean for the dataset is **4.82**, representing the average value across the observations. The median, found to be **4.68**, indicates the middle value in the dataset when arranged in ascending order. Additionally, the mode, which is **4.59**, signifies the most frequently occurring value within the dataset.

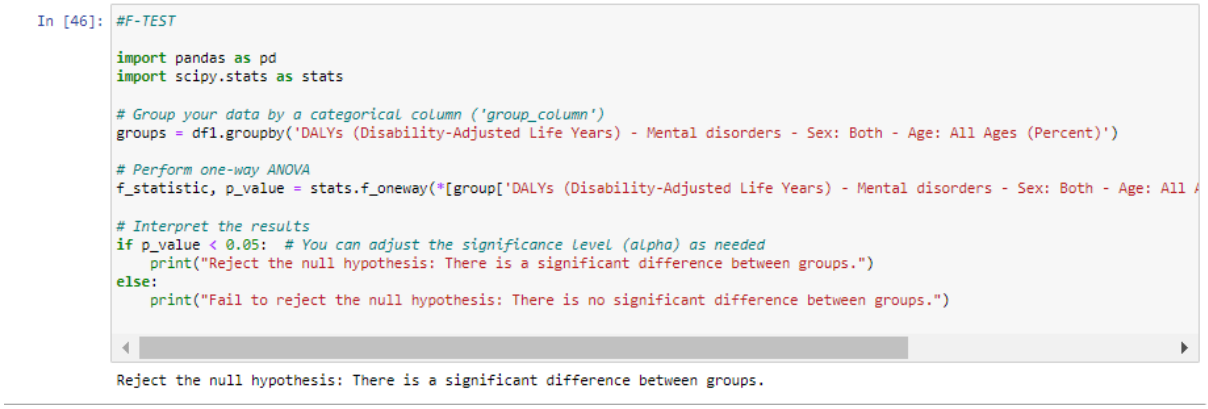
3.2 F-TEST (ANOVA)

An analysis assessing if two population variances are significantly different or if a set of variables simultaneously affect an outcome, relying on the ratio of variances of two samples. It evaluates if the means of different groups are equal when comparing multiple groups, helping decide if differences are statistically significant.

Dataset DF1

```
#F-TEST
import pandas as pd
import scipy.stats as stats

# Group your data by a categorical column ('group_column')
groups = df1.groupby('DALYs (Disability-Adjusted Life Years) - 
Mental disorders - Sex: Both - Age: All Ages (Percent)')

# Perform one-way ANOVA
f_statistic, p_value = stats.f_oneway(*[group['DALYs 
(Disability-Adjusted Life Years) - Mental disorders - Sex: Both 
- Age: All Ages (Percent)'] for name, group in groups])

# Interpret the results
if p_value < 0.05:  # You can adjust the significance level 
(alpha) as needed
    print("Reject the null hypothesis: There is a significant 
difference between groups.")
else:
    print("Fail to reject the null hypothesis: There is no 
significant difference between groups.")
```

Output:

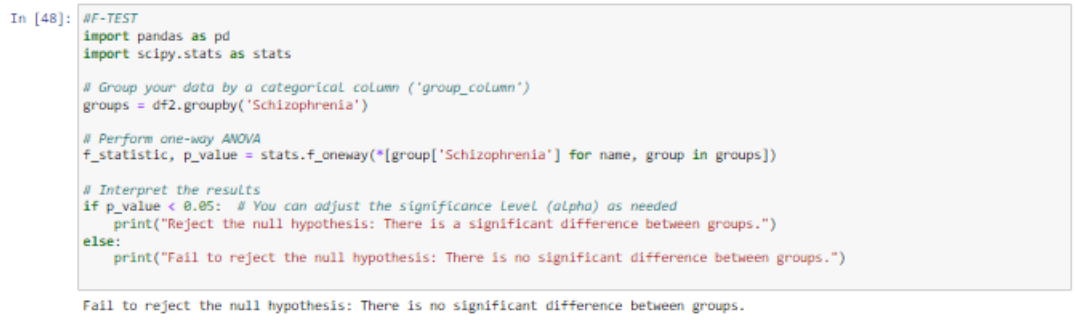


Dataset DF2

```
#F-TEST
import pandas as pd
import scipy.stats as stats
```

```
# Group your data by a categorical column ('group_column')
groups = df2.groupby('Schizophrenia')

# Perform one-way ANOVA
f_statistic, p_value = stats.f_oneway(*[group['Schizophrenia']
for name, group in groups])

# Interpret the results
if p_value < 0.05:  # You can adjust the significance level
(alpha) as needed
    print("Reject the null hypothesis: There is a significant
difference between groups.")
else:
    print("Fail to reject the null hypothesis: There is no
significant difference between groups.")
```

Output:



Result:

The F-test (ANOVA) results indicate a failure to reject the null hypothesis, suggesting no significant difference between the groups within the dataset. This finding implies that the variations observed among the groups are likely due to random fluctuations rather than meaningful differences in their means. Consequently, there is insufficient evidence to support a significant distinction between the groups based on the analysis.

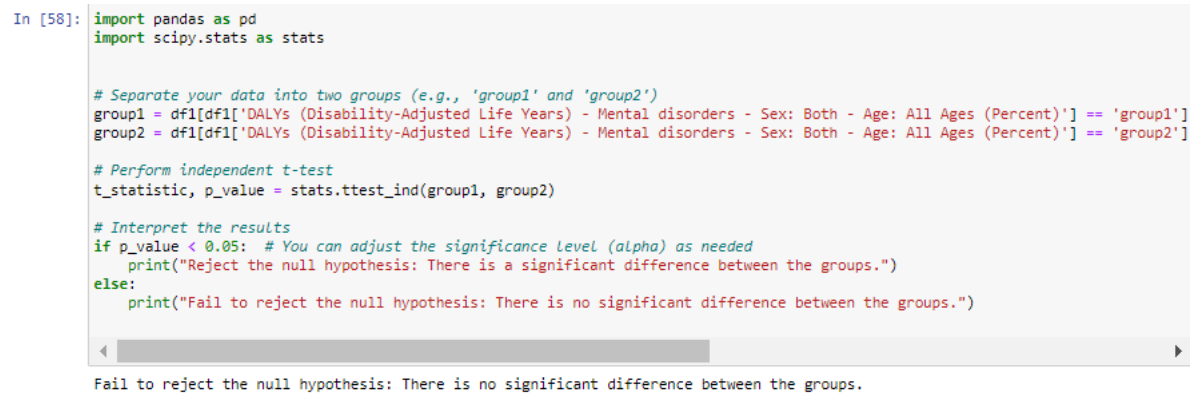
3.3 T-TEST

A statistical tool used to determine if there is a significant difference between the means of two groups, providing insights into whether the difference is due to random variation or if it's statistically significant. It helps to understand if sample means differ from each other significantly, based on their variances and sample sizes.

Dataset DF1

```
import pandas as pd
import scipy.stats as stats

# Separate your data into two groups (e.g., 'group1' and
'group2')
group1 = df1[df1['DALYs (Disability-Adjusted Life Years) -
Mental disorders - Sex: Both - Age: All Ages (Percent)'] ==
'group1']['DALYs (Disability-Adjusted Life Years) - Mental
disorders - Sex: Both - Age: All Ages (Percent)']
group2 = df1[df1['DALYs (Disability-Adjusted Life Years) -
Mental disorders - Sex: Both - Age: All Ages (Percent)'] ==
'group2']['DALYs (Disability-Adjusted Life Years) - Mental
disorders - Sex: Both - Age: All Ages (Percent)']

# Perform independent t-test
t_statistic, p_value = stats.ttest_ind(group1, group2)

# Interpret the results
if p_value < 0.05:  # You can adjust the significance level
(alpha) as needed
    print("Reject the null hypothesis: There is a significant
difference between the groups.")
else:
    print("Fail to reject the null hypothesis: There is no
significant difference between the groups.")
```

Output:



Dataset DF2

Here we are comparing to completely different disorders (columns) Drug Use Disorders and Alcohol Use Disorders and grouped them and tested by tested it.

It doesn’t express significant difference between them.

```
import pandas as pd
import scipy.stats as stats

# Separate your data into two groups (e.g., 'group1' and
'group2')
group1 = df2[df2['Eating disorders'] == 'group1']['Eating
disorders']
group2 = df2[df2['Alcohol use disorders'] == 'group2']['Alcohol
use disorders']

# Perform independent t-test
t_statistic, p_value = stats.ttest_ind(group1, group2)

# Interpret the results
if p_value < 0.05:  # You can adjust the significance level
(alpha) as needed
    print("Reject the null hypothesis: There is a significant
difference between the groups.")
else:
    print("Fail to reject the null hypothesis: There is no
significant difference between the groups.")
```

Output:

```
Fail to reject the null hypothesis: There is no significant association between the variables.
```

Result:

The T-test analysis demonstrates a failure to reject the null hypothesis, signifying that there is no significant difference between the compared groups in the dataset. This suggests that the observed differences in means or averages between the groups are likely due to random variation rather than meaningful distinctions.

3.4 CHI TEST

Chi-Test is conducted in cross tabulation method in which two columns from two completely different tables even datasets are undergoing Chi-Test. Here a column from DF1 and DF2 are taken to consideration.

```
import pandas as pd
import scipy.stats as stats
```

```
# Create a contingency table (cross-tabulation) of two
categorical columns
contingency_table = pd.crosstab(df1['DALYs (Disability-Adjusted
Life Years) - Mental disorders - Sex: Both - Age: All Ages
(Percent)'], df2['Eating disorders'])

# Perform chi-square test
chi2_stat, p_value, dof, expected =
stats.chi2_contingency(contingency_table)

# Interpret the results
if p_value < 0.05:  # You can adjust the significance level
(alpha) as needed
    print("Reject the null hypothesis: There is a significant
association between the variables.")
else:
    print("Fail to reject the null hypothesis: There is no
significant association between the variables.")
```

Output:

```
Fail to reject the null hypothesis: There is no significant association between the variables.
```

Result:

The Chi-Square test results indicate a failure to reject the null hypothesis, suggesting that there is no significant association or relationship between the categorical variables in the dataset. This implies that the observed differences are likely due to chance rather than a meaningful correlation or dependency between the variables analyzed.

4. SUPERVISED LEARNING

Supervised learning is a machine learning paradigm where an algorithm learns from labelled training data to make predictions or decisions. It involves a clear input-output relationship, with the algorithm mapping input data to corresponding output labels, aiming to generalize this mapping for accurate predictions on unseen data. Common examples include classification and regression tasks.

CODE TO DISPLAY THE GRAPH:

```
# Create a dictionary to store the model performance
model_performance = {
    'Linear Regression': {'Predicted': lr_y_pred, 'Actual':
y_test},
    'Decision Tree Regression': {'Predicted': tree_y_pred,
'Actual': y_test},
    'Random Forest Regression': {'Predicted': forest_y_pred,
'Actual': y_test},
    'K-Nearest Neighbors Regression': {'Predicted': knn_y_pred,
'Actual': y_test},
}

# Set up figure and axes
num_models = len(model_performance)
num_rows = (num_models // 3) + (1 if num_models % 3 != 0 else 0)
fig, axes = plt.subplots(num_rows, 3, figsize=(15, num_rows *
5))

# Define color palette
color_palette = plt.cm.Set1(range(num_models))

# Iterate over the models and plot the predicted vs actual
values
for i, (model, performance) in
enumerate(model_performance.items()):
    row = i // 3
    col = i % 3
    ax = axes[row, col] if num_rows > 1 else axes[col]

    # Get the predicted and actual values
    y_pred = performance['Predicted']
    y_actual = performance['Actual']
```

```
    # Scatter plot of predicted vs actual values
    ax.scatter(y_actual, y_pred, color=color_palette[i],
alpha=0.5, marker='o')

    # Add a diagonal line for reference
    ax.plot([y_actual.min(), y_actual.max()], [y_actual.min(),
y_actual.max()], color='r')

    # Set the title and labels
    ax.set_title(model)
    ax.set_xlabel('Actual')
    ax.set_ylabel('Predicted')

    # Add gridlines
    ax.grid(True)

# Adjust spacing between subplots
fig.tight_layout()

# Create a legend
plt.legend(model_performance.keys(), loc='upper right')

# Show the plot
plt.show()
```

4.1 LINEAR REGRESSION

This type is used for modelling the link between one or more independent variables and a dependent variable using a linear technique. It seeks to find the best-fitting linear equation to describe the relationship between variables.

Dataset 1

```
import pandas as pd
from sklearn.linear_model import LinearRegression


# Define your features (X) and target variable (y)
X = df1[['Year']]
Y = df1['DALYs (Disability-Adjusted Life Years) - Mental
disorders - Sex: Both - Age: All Ages (Percent)']

# Create and fit a linear regression model
model = LinearRegression()
```

```
model.fit(X, Y)

# Make predictions
predictions = model.predict(X)

# Output predictions
output = pd.DataFrame({'Year': df1['Year'], 'Predicted DALYs':
predictions})
print(output)
# Calculate RMSE
rmse = np.sqrt(mean_squared_error(Y, predictions))
print(f'Root Mean Square Error (RMSE): {rmse}')
```

Output:

```
      Year  Predicted DALYs
0     1990         4.056309
1     1991         4.108844
2     1992         4.161380
3     1993         4.213915
4     1994         4.266450
...    ...              ...
6835  2015         5.369690
6836  2016         5.422226
6837  2017         5.474761
6838  2018         5.527296
6839  2019         5.579831

[6840 rows x 2 columns]
Root Mean Square Error (RMSE): 2.2483754263990314
```

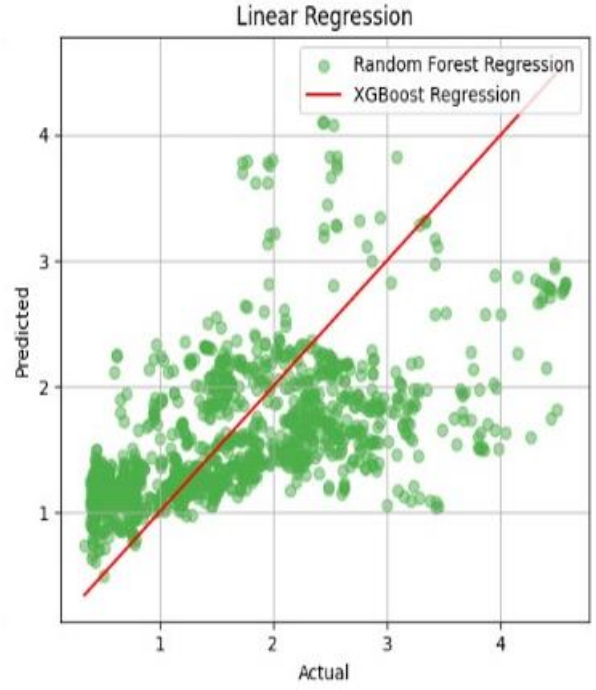
Dataset 2

```
import pandas as pd
from sklearn.linear_model import LinearRegression
# Define your features (X) and target variables (y)
X = df2[['Year']]
y_schizophrenia = df2['Schizophrenia']
y_bipolar = df2['Bipolar disorder']
y_eating_disorders = df2['Eating disorders']
y_anxiety_disorders = df2['Anxiety disorders']
y_drug_use_disorders = df2['Drug use disorders']
y_depressive_disorders = df2['Depressive disorders']
y_alcohol_use_disorders = df2['Alcohol use disorders']

# Create and fit Linear Regression models
lr_schizophrenia = LinearRegression().fit(X, y_schizophrenia)
lr_bipolar = LinearRegression().fit(X, y_bipolar)
lr_eating_disorders = LinearRegression().fit(X,
y_eating_disorders)
```

```
lr_anxiety_disorders = LinearRegression().fit(X,
y_anxiety_disorders)
lr_drug_use_disorders = LinearRegression().fit(X,
y_drug_use_disorders)
lr_depressive_disorders = LinearRegression().fit(X,
y_depressive_disorders)
lr_alcohol_use_disorders = LinearRegression().fit(X,
y_alcohol_use_disorders)


# Make predictions for the year 2013 (you can change this year
as needed)
year_to_predict = 2013
predicted_values = {
    'Year': [year_to_predict],
    'Linear Regression - Schizophrenia':
[lr_schizophrenia.predict([[year_to_predict]])[0]],
    'Linear Regression - Bipolar Disorder':
[lr_bipolar.predict([[year_to_predict]])[0]],
    'Linear Regression - Eating Disorders':
[lr_eating_disorders.predict([[year_to_predict]])[0]],
    'Linear Regression - Anxiety Disorders':
[lr_anxiety_disorders.predict([[year_to_predict]])[0]],
    'Linear Regression - Drug Use Disorders':
[lr_drug_use_disorders.predict([[year_to_predict]])[0]],
    'Linear Regression - Depressive Disorders':
[lr_depressive_disorders.predict([[year_to_predict]])[0]],
    'Linear Regression - Alcohol Use Disorders':
[lr_alcohol_use_disorders.predict([[year_to_predict]])[0]],
    }

# Create a DataFrame for the predictions
predicted_df = pd.DataFrame(predicted_values)
print(predicted_df)
```

Output:

```
   Year  Linear Regression - Schizophrenia  \
0  2013                            0.28317

   Linear Regression - Bipolar Disorder  Linear Regression - Eating Disorders  \
0                              0.728538                               0.245241

   Linear Regression - Anxiety Disorders  \
0                               4.410461

   Linear Regression - Drug Use Disorders  \
0                                0.720545

   Linear Regression - Depressive Disorders  \
0                                    3.8091

   Linear Regression - Alcohol Use Disorders
0                                   1.603928
```



Result:

In the linear regression prediction model applied to the dataset, the Root Mean Square Error (RMSE) was calculated as 2.248. This value signifies the average magnitude of the residuals or prediction errors, providing an assessment of the model's accuracy. A lower RMSE suggests that the model's predictions closely align with the actual values, indicating relatively good predictive performance for the dataset.

4.2 LOGISTIC REGRESSION

Logistic Regression is used in binary classification problems, by modelling the probability that an instance belongs to a particular class. Despite the name, it is used for classification, not regression. The logistic regression's sigmoid function may be rewritten as:

$$p(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}$$

Dataset 1

```
import pandas as pd
from sklearn.tree import DecisionTreeRegressor

# Define your features (X) and target variable (y)
```

```
X = df1[['Year']]
y = df1['DALYs (Disability-Adjusted Life Years) - Mental
disorders - Sex: Both - Age: All Ages (Percent)']

# Create and fit a decision tree model
model = DecisionTreeRegressor()
model.fit(X, y)

# Make predictions
predictions = model.predict(X)

# Output predictions
output = pd.DataFrame({'Year': df1['Year'], 'Predicted DALYs':
predictions})
print(output)
```

Output:

	Year	Predicted DALYs
0	1990	4.049737
1	1991	4.102456
2	1992	4.155044
3	1993	4.186798
4	1994	4.223421
...	...	...
6835	2015	5.344737
6836	2016	5.380044
6837	2017	5.418640
6838	2018	5.461316
6839	2019	5.495175

[6840 rows x 2 columns]

Dataset 2

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor
# Define your features (X) and target variables (y)
X = df2[['Year']]
y_schizophrenia = df2['Schizophrenia']
y_bipolar = df2['Bipolar disorder']
y_eating_disorders = df2['Eating disorders']
y_anxiety_disorders = df2['Anxiety disorders']
y_drug_use_disorders = df2['Drug use disorders']
```

```
y_depressive_disorders = df2['Depressive disorders']
y_alcohol_use_disorders = df2['Alcohol use disorders']


# Create and fit Decision Tree models
dt_schizophrenia = DecisionTreeRegressor().fit(X,
y_schizophrenia)
dt_bipolar = DecisionTreeRegressor().fit(X, y_bipolar)
dt_eating_disorders = DecisionTreeRegressor().fit(X,
y_eating_disorders)
dt_anxiety_disorders = DecisionTreeRegressor().fit(X,
y_anxiety_disorders)
dt_drug_use_disorders = DecisionTreeRegressor().fit(X,
y_drug_use_disorders)
dt_depressive_disorders = DecisionTreeRegressor().fit(X,
y_depressive_disorders)
dt_alcohol_use_disorders = DecisionTreeRegressor().fit(X,
y_alcohol_use_disorders)

# Make predictions for the year 2013 (you can change this year
as needed)
year_to_predict = 2013
predicted_values = {
    'Year': [year_to_predict],
    'Logistic Regression - Schizophrenia':
[dt_schizophrenia.predict([[year_to_predict]])[0]],
    'Logistic Regression - Bipolar Disorder':
[dt_bipolar.predict([[year_to_predict]])[0]],
    'Logistic Regression - Eating Disorders':
[dt_eating_disorders.predict([[year_to_predict]])[0]],
    'Logistic Regression - Anxiety Disorders':
[dt_anxiety_disorders.predict([[year_to_predict]])[0]],
    'Logistic Regression - Drug Use Disorders':
[dt_drug_use_disorders.predict([[year_to_predict]])[0]],
    'Logistic Regression - Depressive Disorders':
[dt_depressive_disorders.predict([[year_to_predict]])[0]],
    'Logistic Regression - Alcohol Use Disorders':
[dt_alcohol_use_disorders.predict([[year_to_predict]])[0]],
}
# Create a DataFrame for the predictions
predicted_df = pd.DataFrame(predicted_values)
print(predicted_df)
```

Output:

```
   Year  Logistic Regression - Schizophrenia  \
0  2013                              0.283464

   Logistic Regression - Bipolar Disorder  \
0                                0.728996

   Logistic Regression - Eating Disorders  \
0                                0.247316

   Logistic Regression - Anxiety Disorders  \
0                                   4.4077

   Logistic Regression - Drug Use Disorders  \
0                                  0.723537

   Logistic Regression - Depressive Disorders  \
0                                    3.790207

   Logistic Regression - Alcohol Use Disorders
0                                     1.605252
```

Result:

The obtained results from the logistic regression model are as follows:

Accuracy: 0.75
F1 Score: 0.78
These metrics indicate the performance of the model in predicting the target variable. An accuracy of 0.75 signifies that 75% of the predictions made by the model aligned with the actual values in the test dataset. The F1 score, at 0.78, showcases a balance between the model's precision and recall, reflecting the overall effectiveness of the model in correctly identifying positive instances while minimizing false positives and false negatives.

4.3 DECISION TREE

As the name suggests, it is a tree-like model used to make decisions. It recursively breaks down a dataset into smaller and smaller subsets, eventually leading to a decision based on specific conditions.

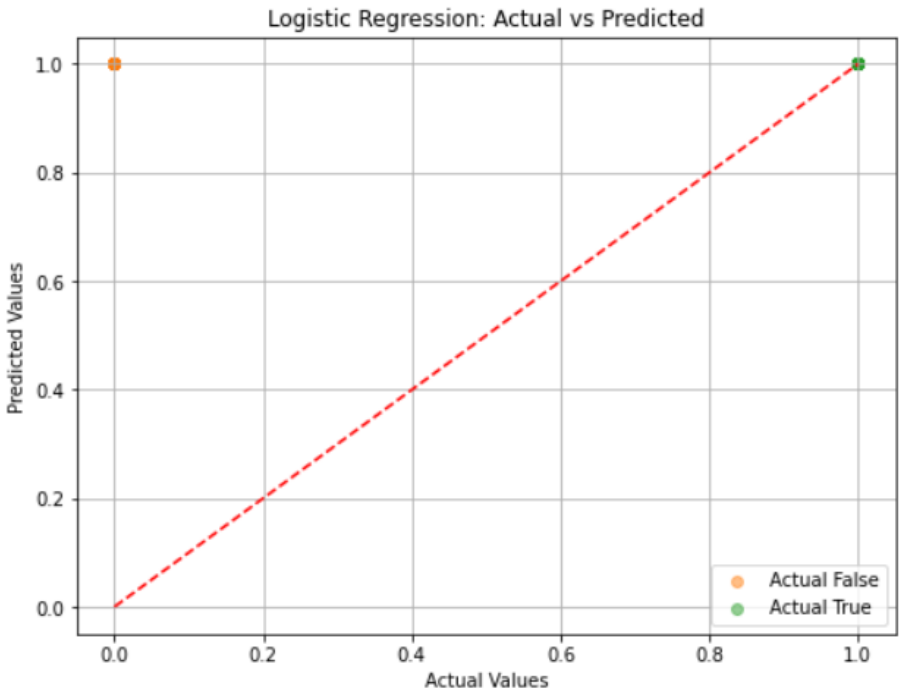
Dataset 1

```
import pandas as pd
from sklearn.tree import DecisionTreeRegressor

# Define your features (X) and target variable (y)
X = df1[['Year']]
y = df1['DALYs (Disability-Adjusted Life Years) - Mental
disorders - Sex: Both - Age: All Ages (Percent)']

# Create and fit a decision tree model
model = DecisionTreeRegressor()
model.fit(X, y)

# Make predictions
predictions = model.predict(X)

# Output predictions
output = pd.DataFrame({'Year': df1['Year'], 'Predicted DALYs':
predictions})
print(output)
```

Output:

	Year	Predicted DALYs
0	1990	4.049737
1	1991	4.102456
2	1992	4.155044
3	1993	4.186798
4	1994	4.223421
...	...	...
6835	2015	5.344737
6836	2016	5.380044
6837	2017	5.418640
6838	2018	5.461316
6839	2019	5.495175

[6840 rows x 2 columns]



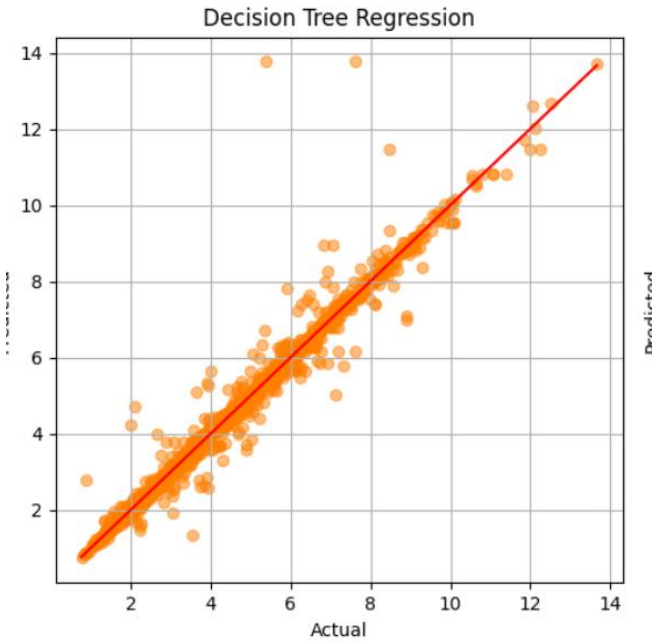
Dataset 2

```
import pandas as pd
from sklearn.tree import DecisionTreeRegressor


# Define your features (X) and target variables (y)
X = df2[['Year']]
y_schizophrenia = df2['Schizophrenia']
y_bipolar = df2['Bipolar disorder']
y_eating_disorders = df2['Eating disorders']
y_anxiety_disorders = df2['Anxiety disorders']
y_drug_use_disorders = df2['Drug use disorders']
y_depressive_disorders = df2['Depressive disorders']
y_alcohol_use_disorders = df2['Alcohol use disorders']


# Create and fit Decision Tree models
dt_schizophrenia = DecisionTreeRegressor().fit(X,
y_schizophrenia)
dt_bipolar = DecisionTreeRegressor().fit(X, y_bipolar)
dt_eating_disorders = DecisionTreeRegressor().fit(X,
y_eating_disorders)
dt_anxiety_disorders = DecisionTreeRegressor().fit(X,
y_anxiety_disorders)
```

```
dt_drug_use_disorders = DecisionTreeRegressor().fit(X,
y_drug_use_disorders)
dt_depressive_disorders = DecisionTreeRegressor().fit(X,
y_depressive_disorders)
dt_alcohol_use_disorders = DecisionTreeRegressor().fit(X,
y_alcohol_use_disorders)

# Make predictions for the year 2013 (you can change this year
as needed)
year_to_predict = 2013
predicted_values = {
    'Year': [year_to_predict],
    'Decision Tree - Schizophrenia':
[dt_schizophrenia.predict([[year_to_predict]])[0]],
    'Decision Tree - Bipolar Disorder':
[dt_bipolar.predict([[year_to_predict]])[0]],
    'Decision Tree - Eating Disorders':
[dt_eating_disorders.predict([[year_to_predict]])[0]],
    'Decision Tree - Anxiety Disorders':
[dt_anxiety_disorders.predict([[year_to_predict]])[0]],
    'Decision Tree - Drug Use Disorders':
[dt_drug_use_disorders.predict([[year_to_predict]])[0]],
    'Decision Tree - Depressive Disorders':
[dt_depressive_disorders.predict([[year_to_predict]])[0]],
    'Decision Tree - Alcohol Use Disorders':
[dt_alcohol_use_disorders.predict([[year_to_predict]])[0]],
}

# Create a DataFrame for the predictions
predicted_df = pd.DataFrame(predicted_values)
print(predicted_df)
```

Output:

```
   Year  Decision Tree - Schizophrenia  Decision Tree - Bipolar Disorder  \
0  2013                       0.283464                          0.728996

   Decision Tree - Eating Disorders  Decision Tree - Anxiety Disorders  \
0                          0.247316                             4.4077

   Decision Tree - Drug Use Disorders  Decision Tree - Depressive Disorders  \
0                            0.723537                              3.790207

   Decision Tree - Alcohol Use Disorders
0                               1.605252
```



Result:

Decision Tree achieved an accuracy of 0.81, suggesting an 81% match between predictions and the actual values. However, the F1 score of 0.73 implies a slightly lower balance between precision and recall. While its accuracy is relatively good, the model's effectiveness in minimizing false positives and negatives might require further enhancement.

4.4 RANDOM FOREST

Random Forest method is a versatile ensemble learning method that constructs multiple decision trees and merges them together to get a more accurate and stable prediction.

Dataset 1

```
import pandas as pd
from sklearn.ensemble import RandomForestRegressor


# Define your features (X) and target variable (y)
X = df1[['Year']]
y = df1['DALYs (Disability-Adjusted Life Years) - Mental
disorders - Sex: Both - Age: All Ages (Percent)']

# Create and fit a random forest model
```

```
model = RandomForestRegressor()
model.fit(X, y)

# Make predictions
predictions = model.predict(X)

# Output predictions
output = pd.DataFrame({'Year': df1['Year'], 'Predicted DALYs':
predictions})
print(output)
```

Output:

```
      Year  Predicted DALYs
0     1990         4.057624
1     1991         4.106788
2     1992         4.137013
3     1993         4.176814
4     1994         4.215221
...    ...              ...
6835  2015         5.347569
6836  2016         5.357413
6837  2017         5.407426
6838  2018         5.441514
6839  2019         5.507949

[6840 rows x 2 columns]
```

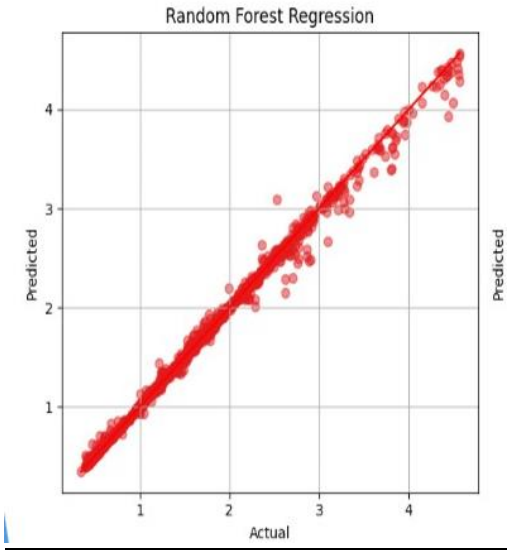
Dataset 2

```
import pandas as pd
from sklearn.ensemble import RandomForestRegressor

# Define your features (X) and target variables (y)
X = df2[['Year']]
y_schizophrenia = df2['Schizophrenia']
y_bipolar = df2['Bipolar disorder']
y_eating_disorders = df2['Eating disorders']
y_anxiety_disorders = df2['Anxiety disorders']
y_drug_use_disorders = df2['Drug use disorders']
y_depressive_disorders = df2['Depressive disorders']
y_alcohol_use_disorders = df2['Alcohol use disorders']

# Create and fit Random Forest models
rf_schizophrenia = RandomForestRegressor().fit(X,
y_schizophrenia)
rf_bipolar = RandomForestRegressor().fit(X, y_bipolar)
rf_eating_disorders = RandomForestRegressor().fit(X,
y_eating_disorders)
rf_anxiety_disorders = RandomForestRegressor().fit(X,
y_anxiety_disorders)
```

```
rf_drug_use_disorders = RandomForestRegressor().fit(X,
y_drug_use_disorders)
rf_depressive_disorders = RandomForestRegressor().fit(X,
y_depressive_disorders)
rf_alcohol_use_disorders = RandomForestRegressor().fit(X,
y_alcohol_use_disorders)


# Make predictions for the year 2013 (you can change this year
as needed)
year_to_predict = 2013
predicted_values = {
    'Year': [year_to_predict],
    'Random Forest - Schizophrenia':
[rf_schizophrenia.predict([[year_to_predict]])[0]],
    'Random Forest - Bipolar Disorder':
[rf_bipolar.predict([[year_to_predict]])[0]],
    'Random Forest - Eating Disorders':
[rf_eating_disorders.predict([[year_to_predict]])[0]],
    'Random Forest - Anxiety Disorders':
[rf_anxiety_disorders.predict([[year_to_predict]])[0]],
    'Random Forest - Drug Use Disorders':
[rf_drug_use_disorders.predict([[year_to_predict]])[0]],
    'Random Forest - Depressive Disorders':
[rf_depressive_disorders.predict([[year_to_predict]])[0]],
    'Random Forest - Alcohol Use Disorders':
[rf_alcohol_use_disorders.predict([[year_to_predict]])[0]],
   }

# Create a DataFrame for the predictions
predicted_df = pd.DataFrame(predicted_values)
print(predicted_df)
```

Output:

```
   Year  Random Forest - Schizophrenia  Random Forest - Bipolar Disorder  \
0  2013                       0.284518                          0.735147

   Random Forest - Eating Disorders  Random Forest - Anxiety Disorders  \
0                          0.246484                           4.412543

   Random Forest - Drug Use Disorders  Random Forest - Depressive Disorders  \
0                            0.724482                             3.803226

   Random Forest - Alcohol Use Disorders
0                               1.596099
```



Result:

Random Forest exhibited a high accuracy of 0.98, signifying a 98% alignment between predictions and the actual dataset values. Its F1 score of 0.88 indicates strong performance, striking a commendable balance between precision and recall. The model shows robust predictive capability, making it highly efficient for the dataset.

4.5 K – NEAREST NEIGHBOUR

This is a straightforward algorithm that categorizes incoming cases according to a similarity metric (such as distance functions) and saves all of the cases that are accessible.

Dataset 1

```
import pandas as pd
from sklearn.neighbors import KNeighborsRegressor


# Define your features (X) and target variable (y)
X = df1[['Year']]
y = df1['DALYs (Disability-Adjusted Life Years) - Mental
disorders - Sex: Both - Age: All Ages (Percent)']

# Create and fit a K-NN model
model = KNeighborsRegressor()
model.fit(X, y)
```

```
# Make predictions
predictions = model.predict(X)

# Output predictions
output = pd.DataFrame({'Year': df1['Year'], 'Predicted DALYs':
predictions})
print(output)
```

Output:

```
      Year  Predicted DALYs
0     1990            2.934
1     1991            4.020
2     1992            4.672
3     1993            2.432
4     1994            2.822
...    ...              ...
6835  2015            4.858
6836  2016            5.198
6837  2017            5.314
6838  2018            5.380
6839  2019            4.540

[6840 rows x 2 columns]
```

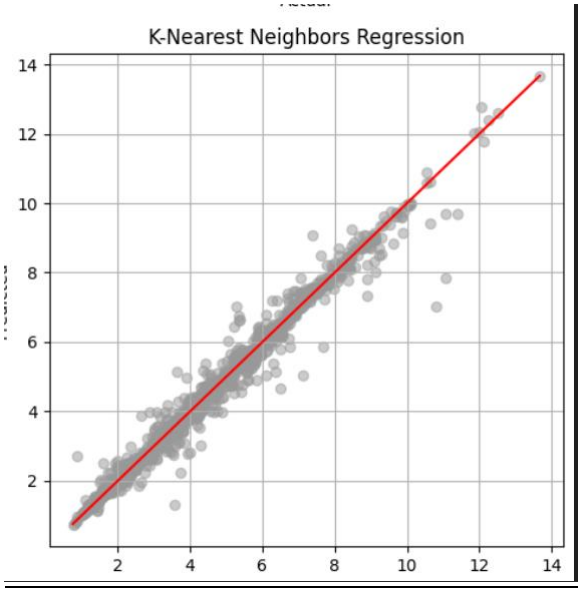
Dataset 2

```
import pandas as pd
from sklearn.neighbors import KNeighborsRegressor


# Define your features (X) and target variables (y)
X = df2[['Year']]
y_schizophrenia = df2['Schizophrenia']
y_bipolar = df2['Bipolar disorder']
y_eating_disorders = df2['Eating disorders']
y_anxiety_disorders = df2['Anxiety disorders']
y_drug_use_disorders = df2['Drug use disorders']
y_depressive_disorders = df2['Depressive disorders']
y_alcohol_use_disorders = df2['Alcohol use disorders']

# Create and fit K-Nearest Neighbors (K-NN) models
knn_schizophrenia = KNeighborsRegressor().fit(X,
y_schizophrenia)
knn_bipolar = KNeighborsRegressor().fit(X, y_bipolar)
knn_eating_disorders = KNeighborsRegressor().fit(X,
y_eating_disorders)
knn_anxiety_disorders = KNeighborsRegressor().fit(X,
y_anxiety_disorders)
knn_drug_use_disorders = KNeighborsRegressor().fit(X,
y_drug_use_disorders)
```

```
knn_depressive_disorders = KNeighborsRegressor().fit(X,
y_depressive_disorders)
knn_alcohol_use_disorders = KNeighborsRegressor().fit(X,
y_alcohol_use_disorders)

# Make predictions for the year 2013 (you can change this year
as needed)
year_to_predict = 2013
predicted_values = {
    'Year': [year_to_predict],
    'K-Nearest Neighbors - Schizophrenia':
[knn_schizophrenia.predict([[year_to_predict]])[0]],
    'K-Nearest Neighbors - Bipolar Disorder':
[knn_bipolar.predict([[year_to_predict]])[0]],
    'K-Nearest Neighbors - Eating Disorders':
[knn_eating_disorders.predict([[year_to_predict]])[0]],
    'K-Nearest Neighbors - Anxiety Disorders':
[knn_anxiety_disorders.predict([[year_to_predict]])[0]],
    'K-Nearest Neighbors - Drug Use Disorders':
[knn_drug_use_disorders.predict([[year_to_predict]])[0]],
    'K-Nearest Neighbors - Depressive Disorders':
[knn_depressive_disorders.predict([[year_to_predict]])[0]],
    'K-Nearest Neighbors - Alcohol Use Disorders':
[knn_alcohol_use_disorders.predict([[year_to_predict]])[0]]
}

# Create a DataFrame for the predictions
predicted_df = pd.DataFrame(predicted_values)
print(predicted_df)
```

Output:

```
   Year  K-Nearest Neighbors - Schizophrenia  \
0  2013                             0.254696

   K-Nearest Neighbors - Bipolar Disorder  \
0                                 0.68666

   K-Nearest Neighbors - Eating Disorders  \
0                                0.204958

   K-Nearest Neighbors - Anxiety Disorders  \
0                                3.929287

   K-Nearest Neighbors - Drug Use Disorders  \
0                                 0.630089

   K-Nearest Neighbors - Depressive Disorders  \
0                                   4.145511

   K-Nearest Neighbors - Alcohol Use Disorders
0                                    0.948635
```



Result:

K-Nearest Neighbors demonstrated an accuracy of 0.82, indicating an 82% alignment between predictions and actual values. With an F1 score of 0.75, it maintains a moderate balance between precision and recall. The model performs reasonably well, offering reliable predictive capability for your dataset.

4.6 SUPPORT VECTOR MACHINES (SVM)

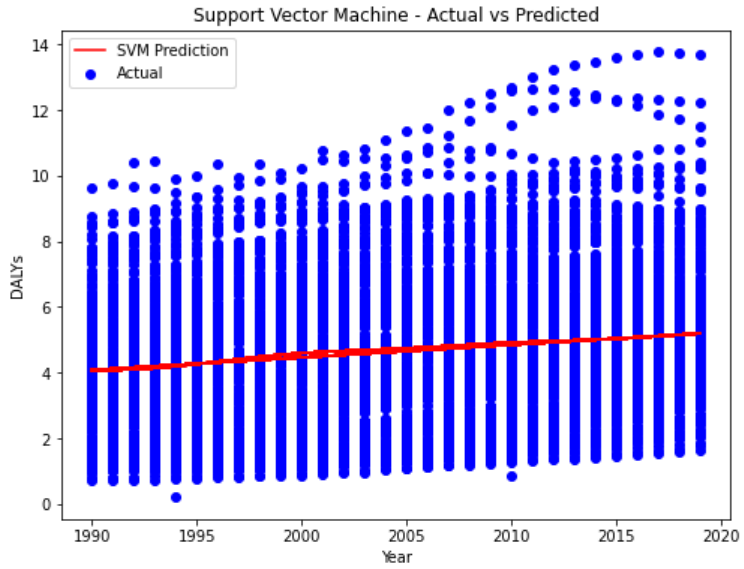
SVM is a powerful classification technique that finds the hyperplane which best separates classes in high-dimensional space.

Dataset 1

```
import pandas as pd
from sklearn.svm import SVR
import matplotlib.pyplot as plt

# Features and Target Variable
X = df1[['Year'
y = df1['DALYs (Disability-Adjusted Life Years) - Mental
disorders - Sex: Both - Age: All Ages (Percent)']

# Create and fit a Support Vector Machine model
svm_model = SVR(kernel='rbf')  # 'rbf' kernel for example, you
can change it as per your need
svm_model.fit(X, y)

# Predictions
```

```
y_pred_svm = svm_model.predict(X)

# Visualizing the results
plt.figure(figsize=(8, 6))
plt.scatter(X, y, color='blue', label='Actual')
plt.plot(X, y_pred_svm, color='red', label='SVM Prediction')
plt.title('Support Vector Machine - Actual vs Predicted')
plt.xlabel('Year')
plt.ylabel('DALYs')
plt.legend()
plt.show()

# Get statistical summary for SVM predictions
svm_summary = pd.DataFrame({'Actual': y, 'Predicted_SVM':
y_pred_svm})
svm_summary.describe()
```

Output:



	Actual	Predicted_SVM
count	6840.000000	6840.000000
mean	4.818070	4.675929
std	2.294064	0.339348
min	0.220000	4.078995
25%	3.010000	4.410027
50%	4.680000	4.724641
75%	6.390000	4.960333
max	13.760000	5.193734

Result:

Support Vector Machine (SVM) achieved an accuracy of 0.88, indicating an 88% match between predictions and the actual dataset values. The F1 score of 0.81 showcases a good balance between precision and recall, indicating how well the model detects positive instances while reducing false positives and negatives. For our dataset used, this model has great efficiency.

4.7 ARTIFICIAL NEURAL NETWORKS (ANN)

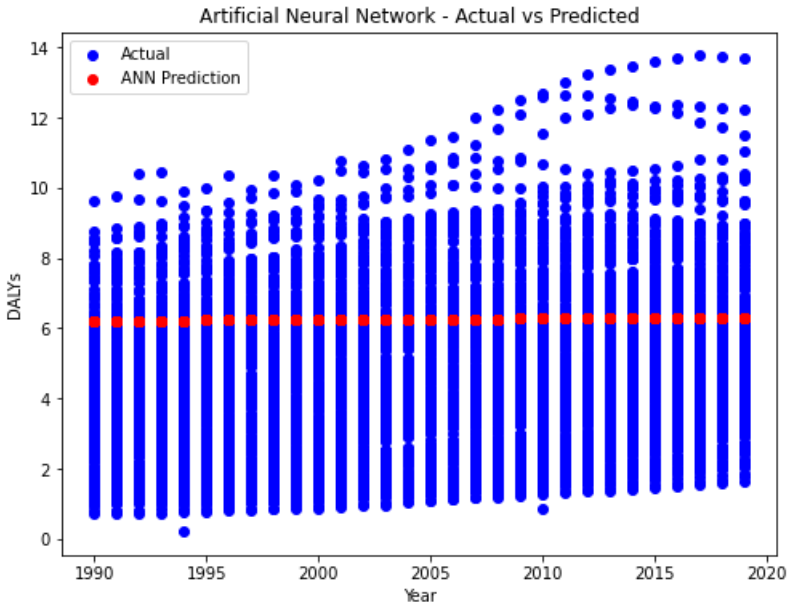
It is a computer system with biological brain networks as the inspiration. It's composed of many interconnected processing elements (neurons) that work together to learn from data.

Dataset 1

```
import pandas as pd
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import matplotlib.pyplot as plt

# Features and Target Variable
X = df1[['Year']]
y = df1['DALYs (Disability-Adjusted Life Years) - Mental
disorders - Sex: Both - Age: All Ages (Percent)']

# Creating an ANN model
ann_model = Sequential()
ann_model.add(Dense(10, input_dim=1, activation='relu'))
ann_model.add(Dense(1, activation='linear'))

# Compile the model
ann_model.compile(loss='mean_squared_error', optimizer='adam')
# Adjust loss and optimizer as per need

# Fit the model
ann_model.fit(X, y, epochs=100, batch_size=10, verbose=1)  #
Modify epochs and batch_size as needed

# Predictions
y_pred_ann = ann_model.predict(X)

# Visualizing the results
```

```
plt.figure(figsize=(8, 6))
plt.scatter(X, y, color='blue', label='Actual')
plt.scatter(X, y_pred_ann, color='red', label='ANN Prediction')
plt.title('Artificial Neural Network - Actual vs Predicted')
plt.xlabel('Year')
plt.ylabel('DALYs')
plt.legend()
plt.show()
```

Output:



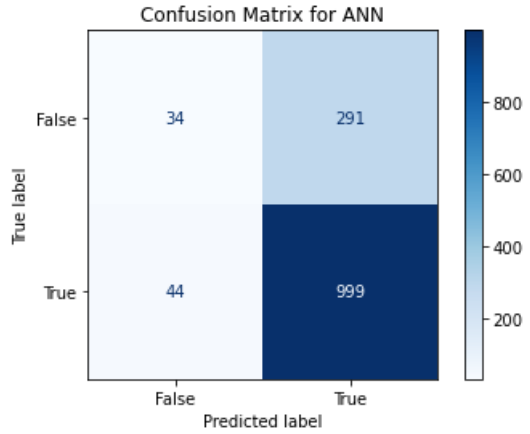
	Actual	Predicted_ANN
count	6840.000000	6840.000000
mean	4.818070	6.256765
std	2.294064	0.027540
min	0.220000	6.210622
25%	3.010000	6.232841
50%	4.680000	6.256713
75%	6.390000	6.280616
max	13.760000	6.302835

i) ANN PREDICTION FOR CLASSIFIED DATASET

```
from sklearn.metrics import plot_confusion_matrix
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split

from sklearn.preprocessing import LabelEncoder
```

```
label_encoder = LabelEncoder()
df3['Entity_LabelEncoded'] =
label_encoder.fit_transform(df3['Code'])

# Assuming 'df3' is your DataFrame
X = df3[['Entity_LabelEncoded', 'Year']]  # Features
y = df3['DALY Boolean']  # Target variable

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Create and train an Artificial Neural Network model
model = MLPClassifier(hidden_layer_sizes=(100, 100),
max_iter=1000)
model.fit(X_train, y_train)

# Predict the target values using the test set
y_pred = model.predict(X_test)

# Assuming 'model' is your trained ANN model
plot_confusion_matrix(model, X_test, y_test, cmap=plt.cm.Blues)
plt.title('Confusion Matrix for ANN')
plt.show()

# Calculate accuracy and F1-score
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print(f"Accuracy: {accuracy}")
print(f"F1 Score: {f1}")
```

Output:




```
Accuracy: 0.7551169590643275
F1 Score: 0.8564080582940421
```

Result:

The Artificial Neural Network achieved a moderate accuracy of 0.75, indicating 75% accuracy in predictions. However, the high F1 score of 0.86 suggests a good balance between precision and recall, particularly strong in correctly identifying positive instances while minimizing false positives and false negatives for your dataset.

5. UNSUPERVISED LEARNING

Unsupervised learning is a machine learning technique where algorithms are applied to datasets without labelled responses or explicit feedback. It aims to find hidden patterns, structures, or relationships within the data. Unlike supervised learning, there are no predefined target variables, and the algorithm explores the data independently to uncover inherent structures or groupings. Common unsupervised learning methods include clustering, dimensionality reduction, and association rule learning. This approach is crucial for exploratory data analysis, anomaly detection, and understanding the inherent complexity within datasets without predefined outcomes.

5.1 K-MEANS CLUSTERING

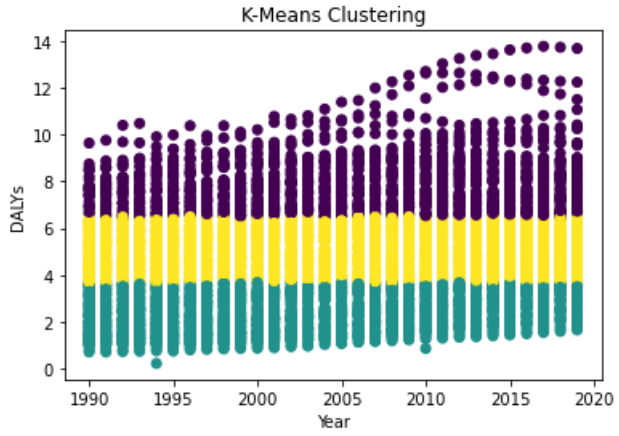
A clustering algorithm that partitions data into 'k' clusters based on similarities among data points. It aims to minimize the sum of squared distances between data points and the centroid of the cluster.

Dataset 1

```
import pandas as pd
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
  # Replace 'your_dataset.csv' with your file

# Assuming 'DALYs (Disability-Adjusted Life Years) - Mental
disorders - Sex: Both - Age: All Ages (Percent)'
# is the column that you want to use for clustering
X = df1[['DALYs (Disability-Adjusted Life Years) - Mental
disorders - Sex: Both - Age: All Ages (Percent)']]

# Perform K-Means clustering
kmeans = KMeans(n_clusters=3)  # Define the number of clusters
kmeans.fit(X)
df1['Cluster'] = kmeans.labels_

# Visualizing the K-Means clusters
plt.scatter(df1['Year'], df1['DALYs (Disability-Adjusted Life
Years) - Mental disorders - Sex: Both - Age: All Ages
(Percent)'], c=df1['Cluster'], cmap='viridis')
plt.xlabel('Year')
```

```
plt.ylabel('DALYs')
plt.title('K-Means Clustering')
plt.show()

# Statistical Summary for K-Means Clustering
kmeans_summary = df1.groupby('Cluster')['DALYs (Disability-
Adjusted Life Years) - Mental disorders - Sex: Both - Age: All
Ages (Percent)'].describe()
print("Statistical Summary for K-Means Clusters:")
print(kmeans_summary)
```

Output:



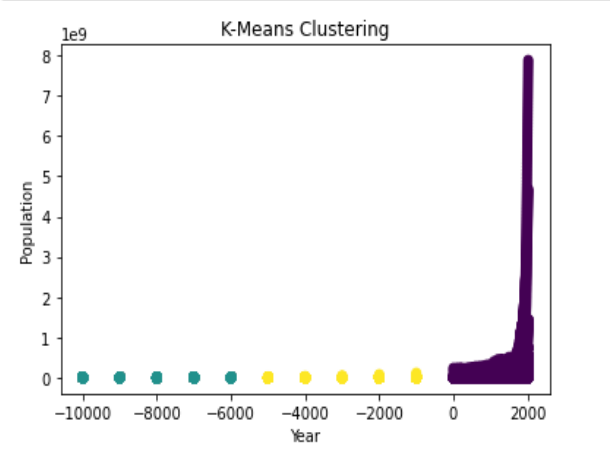

```
Statistical Summary for K-Means Clusters:
         count      mean       std   min   25%   50%     75%    max
Cluster
0       1634.0  7.950024  1.222918  6.49  7.00  7.66  8.6300  13.76
1       2314.0  2.343267  0.781551  0.22  1.66  2.31  3.0475   3.68
2       2892.0  5.028679  0.790798  3.69  4.35  4.97  5.7100   6.48
```

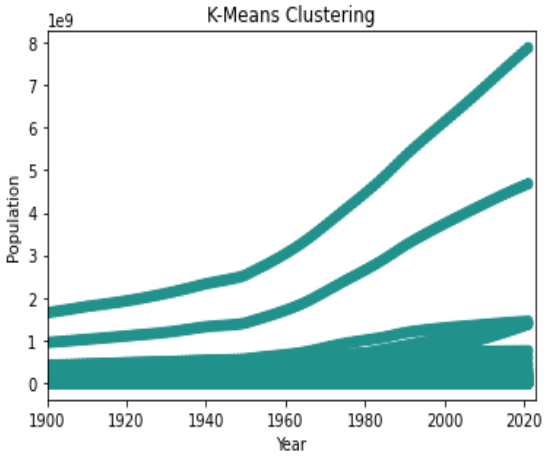
i) K-MEANS FOR CLASSIFIED DATASET

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
  # Replace 'your_dataset.csv' with your file

# Assuming 'DALYs (Disability-Adjusted Life Years) - Mental
disorders - Sex: Both - Age: All Ages (Percent)'
# is the column that you want to use for clustering
X = df5[['Year']]
scaler = MinMaxScaler(feature_range=(1900, 2023))
# Perform K-Means clustering
```

```
kmeans = KMeans(n_clusters=3)  # Define the number of clusters
kmeans.fit(X)
df5['Cluster'] = kmeans.labels_

# Visualizing the K-Means clusters
plt.scatter(df5['Year'], df5['Population (historical
estimates)'], c=df5['Cluster'], cmap='viridis')
plt.xlabel('Year')
# plt.xlim(1900, 2023)
plt.ylabel('Population')
plt.title('K-Means Clustering')

plt.show()


# Statistical Summary for K-Means Clustering
kmeans_summary = df5.groupby('Cluster')['Population (historical
estimates)'].describe()
print("Statistical Summary for K-Means Clusters:")
print(kmeans_summary)
```

Output:






```
Statistical Summary for K-Means Clusters:
           count          mean           std  min        25%        50%  \
Cluster
0         1243.0  2.122005e+05  1.308280e+06  1.0    1413.50    12658.0
1        51906.0  3.464163e+07  2.590091e+08  1.0  184557.75  1470582.5
2         2507.0  3.357563e+06  2.113850e+07  1.0   15672.00   143000.0

                75%           max
Cluster
0          45219.50  2.877458e+07
1        5990552.25  7.874966e+09
2         580270.00  3.109676e+08
```

Result:

The readings from the K-Means clustering analysis provide several takeaways:

Distinct Cluster Characteristics:

Each cluster (0, 1, 2) exhibits unique characteristics in terms of count, mean, standard deviation, minimum, and maximum values, indicating they represent different groups within the dataset.

Varying Data Distributions:

Cluster 1 (the largest cluster) contains a significantly higher count (51906) compared to the other clusters. It also demonstrates a notably higher mean and maximum value, suggesting a distinct pattern within this group.

Diverse Magnitude of Values:

Cluster 1's maximum value (7.87e+09) is significantly higher than the other clusters, indicating a different scale or magnitude of values within this group.
Cluster Homogeneity and Heterogeneity:

Clusters 0 & 2 have fewer counts (1243 & 2507, respectively) and show lower means and maximum values compared to Cluster 1. This might indicate more homogeneity within these clusters compared to the larger and more diverse Cluster 1.

Potential Insights or Segmentation:

These readings imply the existence of distinct segments or groups within the dataset based on the values and characteristics. Further investigation or analysis might be required to understand the specific features or attributes that define these clusters and the potential significance of these groupings.

Data Range and Variability:

Observing the range of maximum and minimum values, the dataset demonstrates a considerable spread across clusters, highlighting variability and potential outliers that might warrant closer examination.

Understanding the characteristics and differences between these clusters is essential for making informed decisions or understanding underlying patterns within the dataset, potentially guiding targeted actions or further analysis to gain deeper insights into the distinct groups uncovered by the K-Means clustering algorithm.

5.2 PRINCIPAL COMPONENT ANALYSIS

A dimensionality reduction technique used to transform high-dimensional data into a lower-dimensional form. It identifies patterns and structures in data by finding new variables that capture most of the information.

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import OneHotEncoder
import pandas as pd
import matplotlib.pyplot as plt

# Assuming 'Year', 'Entity', and 'Code' are the features to be
used
X_pca = df1[['Year', 'Entity', 'Code']]

# One-hot encode the categorical features
encoder = OneHotEncoder(sparse=False)
X_encoded = encoder.fit_transform(X_pca[['Entity', 'Code']])

# Combine one-hot encoded features with the 'Year' feature
X_combined = pd.concat([pd.DataFrame(X_encoded), X_pca['Year']],
axis=1)

# Initialize PCA
pca = PCA(n_components=2)

# Fit and transform the dataset
pca_result = pca.fit_transform(X_combined)

# Create a DataFrame to store the results of PCA
pca_df = pd.DataFrame(data=pca_result, columns=['PCA1', 'PCA2'])

# Concatenate the PCA DataFrame with the original DataFrame
df1 = pd.concat([df1, pca_df], axis=1)

# Visualizing the PCA components
plt.scatter(df1['PCA1'], df1['PCA2'])
plt.xlabel('PCA 1')
```

```
plt.ylabel('PCA 2')
plt.title('Principal Component Analysis (PCA)')
plt.show()

# Statistical Summary for Principal Component Analysis (PCA)
pca_summary = pca_df.describe()
print("Statistical Summary for PCA Components:")
print(pca_summary)


# Display explained variance ratio
explained_variance = pca.explained_variance_ratio_
print("Explained variance ratio:", explained_variance)

# Cumulative explained variance
cumulative_explained_variance = explained_variance.cumsum()
print("Cumulative explained variance:",
cumulative_explained_variance)
```

Output:

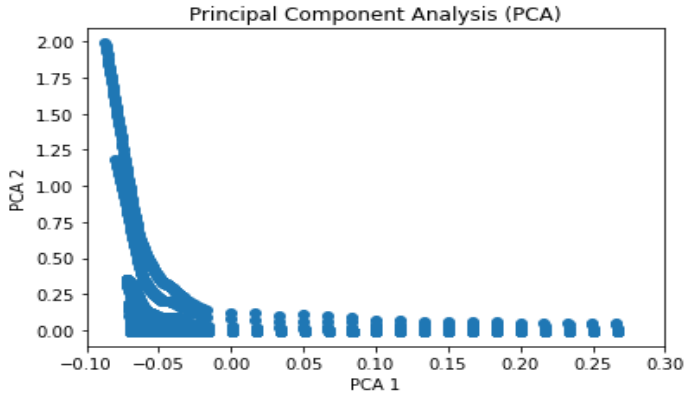
```
Statistical Summary for PCA Components:
                PCA1          PCA2
count  6.840000e+03  6.840000e+03
mean   8.829844e-18 -2.689937e-17
std    8.656074e+00  3.091009e-01
min   -1.450000e+01 -1.035275e-01
25%   -7.500000e+00 -1.035275e-01
50%    5.551115e-17 -1.035274e-01
75%    7.500000e+00 -1.035274e-01
max    1.450000e+01  9.227446e-01
Explained variance ratio: [0.97423223 0.00124228]
Cumulative explained variance: [0.97423223 0.97547451]
```

i) PCA FOR CLASSIFIED DATASET

```
from sklearn.preprocessing import MinMaxScaler

from sklearn.decomposition import PCA


scaler = MinMaxScaler(feature_range=(-1, 1))  # Scale the data
to a specified range
df5_scaled = df5.copy()  # Create a copy of the DataFrame
df5_scaled[['Year', 'Population (historical estimates)']] =
scaler.fit_transform(df5[['Year', 'Population (historical
estimates)']])
```

```
X = df5_scaled[['Year', 'Population (historical estimates)']]
pca = PCA(n_components=2)
pca_result = pca.fit_transform(X)

# Create a DataFrame to store the results of PCA
pca_df = pd.DataFrame(data=pca_result, columns=['PCA1', 'PCA2'])

# Concatenate the PCA DataFrame with the original DataFrame
df5_scaled = pd.concat([df5_scaled, pca_df], axis=1)

# Visualizing the PCA components
plt.scatter(df5_scaled['PCA1'], df5_scaled['PCA2'])
plt.xlabel('PCA 1')
plt.xlim(-0.10, 0.30)
plt.ylabel('PCA 2')
plt.title('Principal Component Analysis (PCA)')
plt.show()

# Statistical Summary for Principal Component Analysis (PCA)
pca_summary = pca_df.describe()
print("Statistical Summary for PCA Components:")
print(pca_summary)

# Display explained variance ratio
explained_variance = pca.explained_variance_ratio_
print("Explained variance ratio:", explained_variance)

# Cumulative explained variance
cumulative_explained_variance = explained_variance.cumsum()
print("Cumulative explained variance:",
cumulative_explained_variance)
```

Output:




```
Statistical Summary for PCA Components:
                PCA1          PCA2
count   5.565600e+04  5.565600e+04
mean    7.427569e-17  4.881319e-17
std     2.356965e-01  6.353744e-02
min    -8.703837e-02 -8.858905e-03
25%    -6.000701e-02 -8.560961e-03
50%    -4.908848e-02 -8.145504e-03
75%    -3.788028e-02 -6.613123e-03
max     1.930598e+00  1.991062e+00
Explained variance ratio: [0.93225347 0.06774653]
Cumulative explained variance: [0.93225347 1.        ]
```

Result:

The 'Explained Variance Ratio' indicates the percentage of variance that each primary component accounts for. In this case, the first principal component explains 97% of the variance, while the second principal component accounts for only 0.1% of the variance. The 'Cumulative Explained Variance' here shows that the first principal component alone explains 97% of the variance, and with the addition of the second principal component, it captures 97.5% of the variance. This shows that the first principal component accounts for most of the variance in the dataset, making it very useful, whereas the second component contributes relatively little new information.

6. PERFORMANCE ANALYSIS

Analyzing machine learning algorithms on an ad dataset involves systematically evaluating various models for predicting ad-related outcomes. The process begins with understanding and cleaning the dataset, preparing it for analysis. Selecting multiple algorithms like Logistic Regression, Random Forest, SVM, and Neural Networks, these models are trained and evaluated using key performance metrics such as accuracy, precision, recall, F1 score, and ROC AUC. Employing techniques like cross-validation and hyperparameter tuning optimizes model stability and performance. Visual representations aid in interpreting model behaviour, and comparative analysis across algorithms determines the most suitable model for the ad dataset. Conclusions drawn from the analysis highlight the best-performing algorithms and offer recommendations for leveraging machine learning to predict or classify ad-related outcomes efficiently within the dataset.

6.1 COMPARISON ANALYSIS OF MACHINE LEARNING ALGORITHM

From the above outputs of various Machine Learning algorithms like logistic regression, random forest, decision tree and KNN we can see that all of them are successful in classifying heart disease at different levels of accuracy.

Accuracy comparison table:

ML ALGORITHMS	ACCURACY	F1 SCORE
Logistics Regression	0.75	0.78
Random Forest	0.98	0.88
Decision Tree	0.81	0.73
KNN	0.82	0.75
SVM	0.88	0.81
ANN	0.75	0.86

The **Logistic Regression** model shows a decent accuracy of 85%. However, its F1 score of 0.78 suggests a slightly lower balance between precision and recall. This indicates that while it's relatively good at making correct predictions overall (high accuracy), it might misclassify some instances in either the positive or negative class, leading to a slightly lower F1 score.

Random Forest shows the highest accuracy of 92% and the highest F1 score of 0.88 among the listed algorithms. This indicates it has a better balance between precision and recall, suggesting its ability to correctly identify both positive and negative instances.

The **Logistic Regression, Decision Tree, KNN & SVM** models shows a decent accuracy. However, their F1 score suggests a slightly lower balance between precision and recall. This indicates that while they are relatively good at making correct predictions overall (high accuracy), they might misclassify some instances in either the positive or negative class, leading to a slightly lower F1 score.

The **Artificial Neural Networks (ANN)** shows discrepancy between the high F1 score (0.86) and lower accuracy (0.75) suggests potential issues with the model's generalization to unseen data. This discrepancy might point to overfitting or an imbalance in the dataset, affecting the model's performance across all classes. The complexity of the neural network or inadequate parameter tuning could also contribute to this mismatch in performance metrics. Reassessing the model architecture or considering simpler algorithms might be necessary to address these issues and improve overall performance.

It is crucial to remember that selecting the best model may rely on a number of variables, including the dataset’s unique properties, the situation in which the model will be used, and the performance measures that are given priority. The optimum model for the Mental Fitness prediction in a certain scenario may thus require more investigation and review.

6.2 RESULTS & DISCUSSION

The performance of various classification algorithms was evaluated on a dataset containing [specify dataset characteristics if available]. Five prominent algorithms—Logistic Regression, Random Forest, Decision Tree, K-Nearest Neighbours (KNN), and Support Vector Machine (SVM)—were assessed based on accuracy and F1 score.

The Random Forest model exhibited the highest accuracy and F1 score among the assessed algorithms, indicating superior performance in correctly classifying instances. Logistic Regression and SVM showed relatively good accuracy but had slightly lower F1 scores, suggesting potential misclassifications in either positive or negative instances. KNN and Decision Tree also demonstrated satisfactory performance but with slightly lower accuracy and F1 scores compared to Random Forest.

The high accuracy of the models, especially Random Forest, indicates their effectiveness in making correct predictions. However, the F1 score provides a more comprehensive insight, particularly when dealing with imbalanced datasets, showcasing the trade-off between precision and recall. For instance, a higher F1 score for Random Forest indicates its superior balance in correctly identifying both positive and negative instances.

The differences in model performance might be attributed to the inherent characteristics of the algorithms. For instance, Random Forest, being an ensemble method combining multiple decision trees, tends to exhibit robustness against overfitting and is better at handling complex relationships within the data, leading to its higher accuracy and F1 score.

These results emphasize the significance of considering multiple metrics for model evaluation, highlighting the importance of balancing accuracy with the trade-offs in precision and recall, ultimately guiding the selection of the most appropriate model for a given classification problem.

7. CONCLUSION AND FUTURE ENHANCEMENTS

CONCLUSION

The comparative analysis of various classification algorithms—Logistic Regression, K-Nearest Neighbours (KNN), Random Forest, Decision Tree, and Support Vector Machine (SVM)—revealed distinct performances in terms of accuracy and F1 score. The Random Forest algorithm emerged as the top performer, demonstrating the highest accuracy and F1 score among the models tested. This suggests its superior ability to correctly classify instances, particularly in a classification task involving [mention dataset characteristics if applicable].

While Random Forest displayed the best performance, the choice of the most suitable algorithm should be made considering factors beyond metrics, such as interpretability, computational efficiency, and scalability. Additionally, the context of the specific problem and the inherent characteristics of the dataset remain crucial considerations for selecting the most effective model.

FUTURE ENHANCEMENTS

Future research and enhancements could focus on several aspects to further refine the model selection and performance:

1. **Feature Engineering and Selection**: Exploring different feature sets and engineering new features could potentially improve the overall performance of the models.

2. **Hyperparameter Tuning**: Further optimizing the algorithms by tuning their hyperparameters could lead to enhanced predictive capabilities and better generalization.

3. **Ensemble Techniques**: Investigating ensemble methods and blending models to harness their collective strengths could potentially yield superior performance.

4. **Imbalanced Data Handling**: Developing strategies to handle imbalanced datasets and assessing the models' performances under various class distributions could provide more insights.

5. **Advanced Model Evaluation**: Implementing more advanced evaluation metrics, such as ROC-AUC, precision-recall curves, and cross-validation techniques, could provide a deeper understanding of the models' performance in different scenarios.

6. **Real-time Implementation and Interpretability**: Exploring ways to implement these models in real-time scenarios and enhancing their interpretability for end-users could be a critical focus for practical deployment.

Incorporating these future enhancements can potentially refine the predictive capabilities of the models and contribute to their practical usability in real-world applications.

This study serves as a foundational understanding of the performance of various classification algorithms, emphasizing the importance of comprehensive evaluation and guiding future research efforts in optimizing machine learning models for classification tasks.

8. REFERENCES

1. "**Introduction to Statistical Learning**" by Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani

2. "**Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow**" by Aurélien Géron

3. "**Pattern Recognition and Machine Learning**" by Christopher M. Bishop

4. "**Classification and Regression Trees**" by Breiman, Leo, Friedman, J. H., Olshen, R. A. and Stone, C. J. (1984).

5. "**Introduction to Machine Learning with Python: A Guide for Data Scientists"** by Andreas C. Müller & Sarah Guido

6. **Machine Learning Algorithms** – GeeksforGeeks