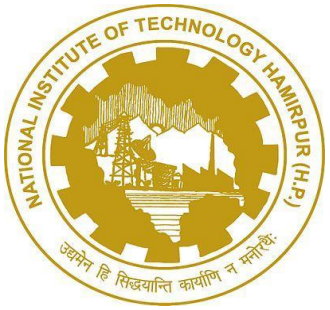
SOFTWARE FAULT PREDICTION
(Project Work Report)



Submitted to: Dr. Pooja Sharma

Submitted by: R. Kirthika(23MCS001)
Simran(23MCS019)
Smriti Gautam(23MCS010)
Richa(23MCS011)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY HAMIRPUR
HAMIRPUR, HIMACHAL PRADESH -177005, INDIA

Table of Contents

1) INTRODUCTION

This report is an implementation of research paper titled "**Software fault prediction with imbalanced datasets using SMOTE-Tomek sampling technique and Genetic Algorithm models**" by Mansi Gupta, Kumar Rajnish & Vandana Bhattacharjee, which was published in August 2023 *(will be addressed as '**the paper**' henceforth)*.

The paper discusses a method for predicting software faults that rely on choosing the most suitable machine learning and deep learning techniques from a pool of accurate and competitive learning techniques in order to construct a fault prediction model.

The paper addresses considerable discussion regarding machine learning (ML) techniques to forecast software faults and the challenges in doing the same due to variation in the prediction performance of machine learning techniques for software systems.

The paper suggests the use of mutual information formulation as a Feature Selection technique, use of a hybrid sampling technique known as SMOTE-TOMEK Link to overcome the issue with class imbalance (CI). Finally, GENETIC ALGORITHM based deep learning model Artificial Neural Networks (GA-ANN) is developed for the purpose of predicting the Software faults. To evaluate the model, accuracy is used as the performance metrics.

2) PROBLEM STATEMENT

In real world, the data collected vastly is unbalanced leading to the problem of class imbalance, next the issue is with selecting proper and relevant features which have positive affect on the outcome doesn't alter the score significantly. Lastly, during the model prediction the issue of accurate hyper-parameter needs to be addressed.

2.1) Issue with Unbalance data:

In an unbalanced dataset, the majority class have a higher weightage than the minority class, making it impossible for the model to successfully learn from the minority class. Oversampling the minority class or under sampling the majority class are two common ways to address the imbalance dataset problem. One of the concerns in software datasets is an imbalance in the number of patterns in each class label, with a low ratio of searches for the software system's most vulnerable parts. The oversampling technique seeks to duplicate certain arbitrary samples from the minority class; as a result, the data are supplemented with a fresh set of information as a result of this tactic. On the other hand, the under-sampling method involves removing some random samples from the majority class, which results in the loss of some information from the original data.

2.2) Issue with Features:

Real world collected data contains many features or attributes which will be used during the modelling process. But only handful of features out of all the features are relevant and useful the rest of the features are irrelevant or pointless. Hence these features affect the evaluation and prediction of the model. The performance and accuracy of the model might impact if the dataset is filled with all this extraneous and useless information.

2.3) Issue with accuracy of models:

To get the best model, the parameters must be adjusted properly. For each set of parameters to be analyzed, there are too many possible configurations when there are several parameter variables, each with a wide range of values. In these situations, optimization techniques should be applied to quickly identify the best input parameters. Selecting the ideal parameter for machine learning models is challenging. Not because the data is noisy or the learning method used is unsuccessful, but because the parameter values were picked wrongly, certain results can be subpar.

3) METHODOLOGY TO OVERCOME EACH ISSUES

To overcome the issue of unbalanced dataset, we can create new set of data that is derived from the current minority class. This technique is known as SMOTE (Synthetic Minority Oversampling Technique). . Though there are various versions of SMOTE, the paper uses SMOTE-Tomek Links approach to handle the issue of data imbalance.

To overcome the issue with Features, it is essential to identify and choose the most pertinent features from the data while excluding extraneous or unimportant elements, which is performed by feature selection. Feature selection is the process of choosing the most important features for the model. A feature is a quality that has an impact on an issue or is helpful for the problem. Every ML model depends on Feature Engineering which involves both Feature Selection and Feature Extraction. The main difference between the two is that feature selection selects a portion of the original feature set, whereas feature extraction creates new features.

The issue of accuracy of model can be overcome by using GENETIC ALGORITHM as one optimization solution. Genetic Algorithm is useful as it can perform effective sorting through a huge number of alternative answers. For hyper-parameter adjustment of the model the paper uses Genetic Algorithm based on Artificial Neural Network (ANN).

The paper also highlights the fact that previously attempts have been made in the field of S/W fault predictions, to over come the issue of class imbalance, issue of proper feature selection and issue of accurate hyper-parameter tuning individually and not all at once. Hence the main objective of the paper is to propose a SFP Model that deals with all the three issues simultaneously.

The implementation of aforementioned objective also serves as the purpose of this project work

4) FORMULATIONS

4.1) Feature Selection Technique

Mutual Information Gain is used to filter features.

Mutual Information Gain:

The quantity of information offered by the feature for recognizing the target value and assessing entropy reduction. It is done by calculating each variable's information gain about the target variable. In supervised learning, features with high mutual information value relating to the class are regarded ideal because they can influence the predictive model towards the correct prediction and so boost the model's accuracy.

$$I(A;B) = \sum_{a \in A} \sum_{b \in B} p(a,b) \log \frac{p(a,b)}{p(a)p(b)}$$

Where,

I (A; B) is the amount of uncertainty in 'A' due to the knowledge of B.

p (a, b) - joint probability function of A and B.

p (a) - Marginal probability distribution function of A

p (b) - marginal probability distribution function of B

4.2) Sampling Technique

SMOTE-Tomek Link is a sampling method based on the distance between each data set and the minority class's nearest neighbors. The Tomek Link method employs a set of criteria to eliminate samples from the majority class that have k nearest neighbors. This approach may be used to locate and eliminate desirable samples of data from the majority class that have the lowest distance with data from the minority class. The SMOTE-Tomek Links procedure is as follows:

- (Beginning of SMOTE) Select random data from the minority class.
- Determine the distance between the randomly generated data and its k nearest neighbors.
- Multiply the diference by a random number between 0 and 1, then add the result as a synthetic sample to the minority class.
- Repeat steps 2–3 until the appropriate proportion of the minority class is reached. (SMOTE concludes)
- (Beginning of Tomek Links) Select random data from the majority class.
- If the nearest neighbor of the random data is data from the minority class (i.e., build the Tomek Link), then delete the Tomek Link.

4.3) Machine Learning Classifier:

Genetic Algorithm: It allows us to explore the hyper-parameter space at

random while simultaneously directing the search based on prior findings. Genetic algorithms are based on Darwin's theory of natural selection. The algorithm architecture is quite flexible and very easy to create, enabling it to be used for a variety of problems. It falls under a larger category of evolutionary algorithms. Chromosomes in GA provide a population of potential solutions to the issue. The idea is that over several succeeding generations, "evolution," which is like natural selection, will fnd the problem's ideal solution. They employ a variety of processes to expand or replace the population in order to create a better fit solution.

Naïve Bayes (NB): Naïve Bayes classifier. We use SKlearn's Gaussian Naive Bayes function, i.e., GaussianNB () to build the following classification model assuming features to be Gaussian:

$$P(Xi \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(Xi - \mu_y)}{2\sigma_y^2}\right)$$

Where,

y is the class variable Xi list of dependent features.

The parameters σy and μy are calculated using maximum likelihood.

Support Vector Machine (SVM): SVMs are popular and memory economical classifiers. SVM is a class provided by Scikit-learn, which is used for the study. The hyper-parameter that are tuned are kernel (this parameter specifies the kernel type to be utilized in the algorithm), gamma (kernel coefficient) and C (Regularization parameter) as linear, 10 and auto respectively (i.e., SVC (kernel = 'linear', C=10, gamma='auto.

K-Nearest Neighbor (KNN): KNN hyper-parameter tuning included leaf_size range 1 to 50, n_neighbors = 1 to 30 and p value to be either 1 or 2. Hyper-parameter tuning is done using grid search.

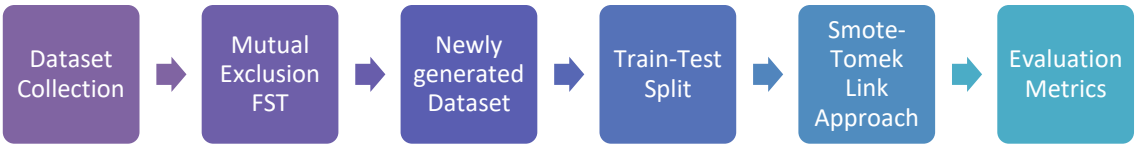
Decision Tree (DT): Parameter setting used in this work for DT model is criterion='entropy', max_depth: None, min_samples_split: 2, min_samples_leaf:1, min_weight_fraction_leaf: 0 and max_features: None. Entropy is calculated as:

$$\text{Entropy} : H(E) = -\sum_{j=1}^{c} p_j \log p_j$$

Where, 'pj ' is simply the fraction of examples in the given class.

Genetic Algorithm: We use Genetic Algorithm to tune three ANN parameters: learning rate, number of layers and neuron per layer

5) WORKFLOW



5.1) Input taken for the implementation:

For the purpose implementation of the research paper, we collect our input dataset from Eclipse Repository. We consider Eclipse bug dataset and Defect Data set Collected from Jira website.

- Eclipse Dataset Repository URL : https://bug.inf.usi.ch/download.php
- Jira Dataset Repository URL : https://zenodo.org/records/3362613

5.2) Code Implementation Workflow:

- The **Prework** includes importing necessary dependencies such as Standard libraries and some third-party libraries. It also includes loading the Dataset.
- The **Exploration Data Analysis** is an approach to visualize the data using various mathematical tools and analyze them. In this work, we plot various graphs, label classification, use histogram, and finally to establish relation between two attributes we use Heatmap which is based on Covariance.
- The process of **Data Cleaning** involves filling null values using central tendencies, removing outliers etc.
- The **Feature Selection** involves selection of relevant features using Mutual Information Gain and splitting the dataset into training data and testing data.
- SMOTE-TOMEK Link approach is used to **sample the data** set in order to deal with unbalanced dataset.
- **Classifier Techniques** such as Naïve Bayes, KNN Classification, Support Vector Classifier, Decision Tree and Genetic Algorithm(Artificial Neural Network) are implemented and corresponding scores have been calculated.

Import the datasets and libraries

In [1]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
```

In [2]:
```
df = pd.read_csv('/content/csv_result-JDT.csv')
df.head()
```

Out[2]:

	id	ck_oo_numberOfPrivateMethods	LDHH_lcom	LDHH_fanIn	numberOfNonTrivialBugsFoundUntil:	WCHU_numberOfPublicAttributes	WCHU
0	1	0	0.000934	0.000000	4	0.00	
1	2	0	0.000741	0.000000	3	0.00	
2	3	0	0.000000	0.000000	4	0.00	
3	4	0	0.001529	0.015913	35	0.00	
4	5	0	0.000000	0.000000	2	1.01	

5 rows × 63 columns

In [3]:
```
import matplotlib.pyplot as plt
%matplotlib inline

import seaborn as sns
sns.set_style('whitegrid')

from tqdm import tqdm
```

EDA

- import some dependencies to plot
- use plotly to visualization
  - label classification
    - count and plot(visualization)
  - value visualization
    - use historgram to visualization attribution
    - relationship
      - covariance
      - heatmap
  - scatter

In [4]:
```
# import some dependencies for plotting

from plotly.offline import iplot
import plotly.graph_objs as go
```

In [5]:
```
# check data
def show_info(data, is_matrix_transpose=False):
    # basic shape
    print('data shape is: {}   sample number {}   attribute number {}\n'.format(df.shape, df.shape[0], df.shape
    # attribute(key)
    print('data columns number {}  \nall columns: {}\n'.format(len(df.columns) ,df.columns))
    # value's null
    print('data all attribute count null:\n', df.isna().sum())
    # data value analysis and data demo
    if is_matrix_transpose:
        print('data value analysis: ', df.describe().T)
        print('data demo without matrix transpose: ', df.head().T)
    else:
        print('data value analysis: ', df.describe())
        print('data demo without matrix transpose: ', df.head())

show_info(df)
```

```
data shape is: (997, 63)   sample number 997   attribute number 63

data columns number 63
all columns: Index(['id', 'ck_oo_numberOfPrivateMethods', 'LDHH_lcom', 'LDHH_fanIn',
       'numberOfNonTrivialBugsFoundUntil:', 'WCHU_numberOfPublicAttributes',
       'WCHU_numberOfAttributes', 'CvsWEntropy', 'LDHH_numberOfPublicMethods',
       'WCHU_fanIn', 'LDHH_numberOfPrivateAttributes', 'CvsEntropy',
       'LDHH_numberOfPublicAttributes', 'WCHU_numberOfPrivateMethods',
       'WCHU_numberOfMethods', 'ck_oo_numberOfPublicAttributes', 'ck_oo_noc',
       'numberOfCriticalBugsFoundUntil:', 'ck_oo_wmc',
       'LDHH_numberOfPrivateMethods', 'WCHU_numberOfPrivateAttributes',
       'CvsLogEntropy', 'WCHU_noc', 'LDHH_numberOfAttributesInherited',
       'WCHU_wmc', 'ck_oo_fanOut', 'ck_oo_numberOfLinesOfCode',
```

```
       'ck_oo_numberOfAttributesInherited', 'ck_oo_numberOfMethods',
       'ck_oo_dit', 'ck_oo_fanIn', 'LDHH_noc', 'WCHU_dit', 'ck_oo_lcom',
       'WCHU_numberOfAttributesInherited', 'ck_oo_rfc', 'LDHH_wmc',
       'LDHH_numberOfAttributes', 'LDHH_numberOfLinesOfCode', 'WCHU_fanOut',
       'WCHU_lcom', 'ck_oo_cbo', 'WCHU_rfc', 'ck_oo_numberOfAttributes',
       'numberOfHighPriorityBugsFoundUntil:',
       'ck_oo_numberOfPrivateAttributes', 'numberOfMajorBugsFoundUntil:',
       'WCHU_numberOfPublicMethods', 'LDHH_dit', 'WCHU_cbo', 'CvsLinEntropy',
       'WCHU_numberOfMethodsInherited', 'numberOfBugsFoundUntil:',
       'LDHH_fanOut', 'LDHH_numberOfMethodsInherited', 'LDHH_rfc',
       'ck_oo_numberOfMethodsInherited', 'ck_oo_numberOfPublicMethods',
       'LDHH_cbo', 'WCHU_numberOfLinesOfCode', 'CvsExpEntropy',
       'LDHH_numberOfMethods', 'class'],
      dtype='object')

data all attribute count null:
 id                                 0
ck_oo_numberOfPrivateMethods        0
LDHH_lcom                           0
LDHH_fanIn                          0
numberOfNonTrivialBugsFoundUntil:   0
                                   ..
LDHH_cbo                            0
WCHU_numberOfLinesOfCode            0
CvsExpEntropy                       0
LDHH_numberOfMethods                0
class                               0
Length: 63, dtype: int64
data value analysis:                id  ck_oo_numberOfPrivateMethods   LDHH_lcom  LDHH_fanIn  \
count  997.000000                   997.000000  997.000000  997.000000
mean   499.000000                     1.297894    0.001777    0.002791
std    287.953411                     5.260343    0.004430    0.007316
min      1.000000                     0.000000    0.000000    0.000000
25%    250.000000                     0.000000    0.000000    0.000000
50%    499.000000                     0.000000    0.000000    0.000000
75%    748.000000                     1.000000    0.001192    0.001694
max    997.000000                   111.000000    0.045838    0.080233

       numberOfNonTrivialBugsFoundUntil:  WCHU_numberOfPublicAttributes  \
count                          997.000000                     997.000000
mean                            10.149448                       0.396409
std                             19.400677                       1.320920
min                              0.000000                       0.000000
25%                              1.000000                       0.000000
50%                              4.000000                       0.000000
75%                             10.000000                       0.000000
max                            200.000000                      15.200000

       WCHU_numberOfAttributes  CvsWEntropy  LDHH_numberOfPublicMethods  \
count               997.000000   997.000000                  997.000000
mean                  0.774764     0.053331                    0.001440
std                   2.317190     0.159108                    0.003932
min                   0.000000     0.000000                    0.000000
25%                   0.000000     0.002877                    0.000000
50%                   0.000000     0.007284                    0.000000
75%                   1.010000     0.035064                    0.000905
max                  32.470000     2.285260                    0.043532

       WCHU_fanIn  ...  numberOfBugsFoundUntil:  LDHH_fanOut  \
count  997.000000  ...                997.000000   997.000000
mean     1.125777  ...                 11.639920     0.002612
std      2.880374  ...                 22.126592     0.005553
min      0.000000  ...                  0.000000     0.000000
25%      0.000000  ...                  1.000000     0.000000
50%      0.000000  ...                  5.000000     0.000000
75%      1.010000  ...                 11.000000     0.002669
max     35.570000  ...                214.000000     0.049184

       LDHH_numberOfMethodsInherited    LDHH_rfc  \
count                     997.000000  997.000000
mean                        0.007039    0.006725
std                         0.009722    0.013519
min                         0.000000    0.000000
25%                         0.001487    0.000000
50%                         0.002695    0.001145
75%                         0.009737    0.007382
max                         0.060148    0.127115

       ck_oo_numberOfMethodsInherited  ck_oo_numberOfPublicMethods  \
count                      997.000000                   997.000000
mean                        49.236710                     8.952859
std                         50.367666                    19.967363
min                          0.000000                     0.000000
25%                         11.000000                     3.000000
50%                         33.000000                     5.000000
75%                         73.000000                     9.000000
max                        319.000000                   387.000000

         LDHH_cbo  WCHU_numberOfLinesOfCode  CvsExpEntropy  \
```

```
count  997.000000                997.000000     997.000000
mean     0.005242                  4.306800       0.120571
std      0.009770                  8.606297       0.132946
min      0.000000                  0.000000       0.000000
25%      0.000000                  0.000000       0.001125
50%      0.001056                  1.090000       0.039072
75%      0.006532                  4.350000       0.231202
max      0.090432                 91.440000       0.580176

       LDHH_numberOfMethods
count            997.000000
mean               0.002304
std                0.005736
min                0.000000
25%                0.000000
50%                0.000000
75%                0.001746
max                0.059817

[8 rows x 62 columns]
data demo without matrix transpose:     id  ck_oo_numberOfPrivateMethods  LDHH_lcom  LDHH_fanIn  \
0   1                             0   0.000934    0.000000
1   2                             0   0.000741    0.000000
2   3                             0   0.000000    0.000000
3   4                             0   0.001529    0.015913
4   5                             0   0.000000    0.000000

   numberOfNonTrivialBugsFoundUntil:  WCHU_numberOfPublicAttributes  \
0                                  4                          0.00
1                                  3                          0.00
2                                  4                          0.00
3                                 35                          0.00
4                                  2                          1.01

   WCHU_numberOfAttributes  CvsWEntropy  LDHH_numberOfPublicMethods  \
0                     0.00     0.004517                     0.00000
1                     0.00     0.014817                     0.00091
2                     1.01     0.024575                     0.00000
3                     0.00     0.347495                     0.00165
4                     1.01     0.012120                     0.00000

   WCHU_fanIn  ...  LDHH_fanOut  LDHH_numberOfMethodsInherited  LDHH_rfc  \
0        0.00  ...     0.000993                       0.025009  0.000969
1        0.00  ...     0.000889                       0.001487  0.002523
2        0.00  ...     0.002827                       0.021650  0.003656
3        2.03  ...     0.010602                       0.012864  0.060975
4        0.00  ...     0.005223                       0.023725  0.006207

   ck_oo_numberOfMethodsInherited  ck_oo_numberOfPublicMethods  LDHH_cbo  \
0                             127                            4  0.001007
1                               8                            9  0.000967
2                              45                            1  0.002917
3                              94                           21  0.027947
4                              73                            0  0.007285

   WCHU_numberOfLinesOfCode  CvsExpEntropy  LDHH_numberOfMethods  class
0                      1.03       0.198199              0.001036  clean
1                      2.06       0.239334              0.000923  clean
2                      1.20       0.187460              0.000000  clean
3                     24.14       0.392389              0.001669  buggy
4                      2.15       0.047726              0.000000  clean

[5 rows x 63 columns]
```

Class label

In [6]:
```
from sklearn.preprocessing import LabelEncoder
# finding the count of different labels
df['class'].value_counts()
```

Out[6]:
```
clean    791
buggy    206
Name: class, dtype: int64
```

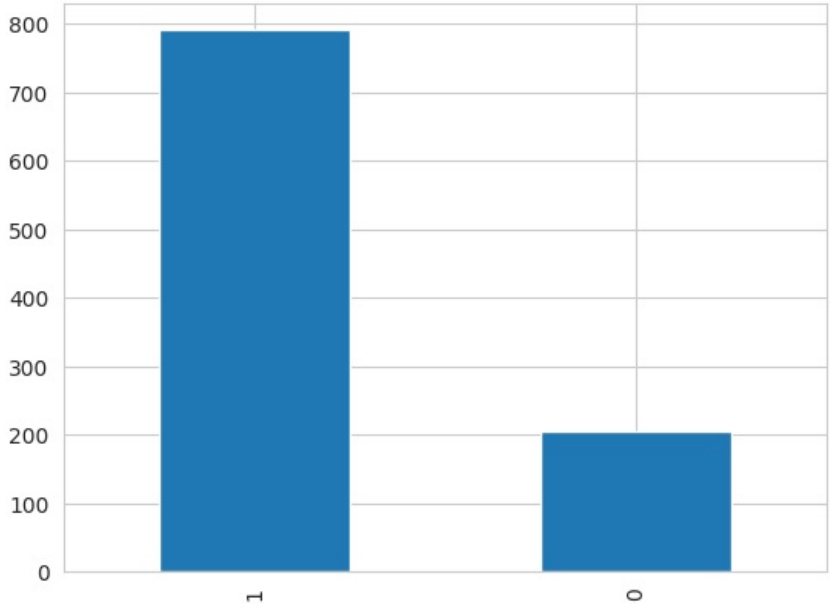
In [7]:
```
# load the Label Encoder function
label_encode = LabelEncoder()
```

In [8]:
```
labels = label_encode.fit_transform(df["class"])
```

In [9]:
```
# appending the labels to the DataFrame
df["class"] = labels
```

In [10]:
```
# label classification
df['class'].value_counts().plot.bar()
```

Out[10]:
```
<Axes: >
```



In [11]:
```
df.corr()
```

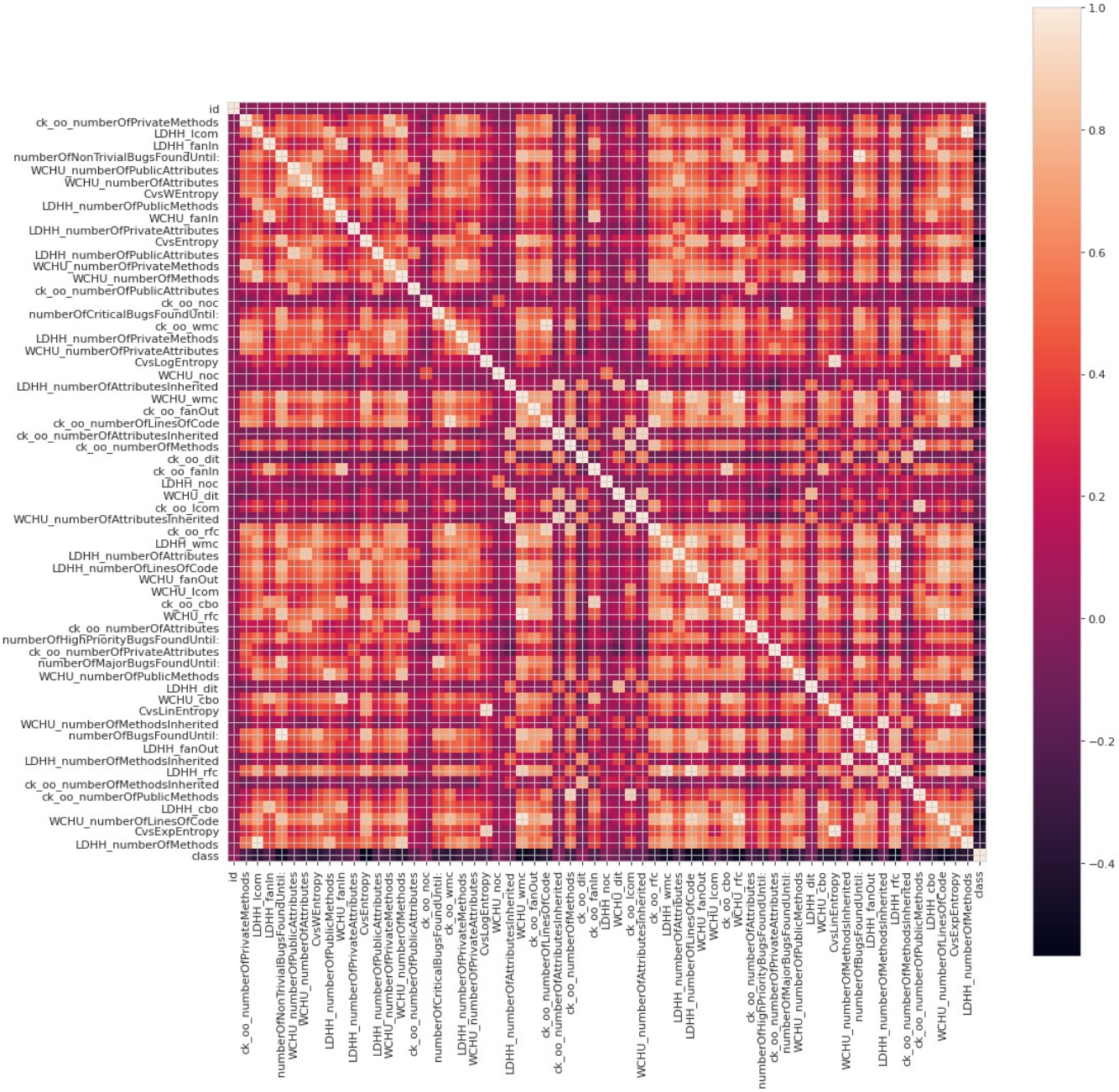
Out[11]:

	id	ck_oo_numberOfPrivateMethods	LDHH_lcom	LDHH_fanIn	numberOfNonTrivialBugsFoundUntil:
id	1.000000	-0.004050	0.020531	0.051403	0.029961
ck_oo_numberOfPrivateMethods	-0.004050	1.000000	0.546243	0.102584	0.500429
LDHH_lcom	0.020531	0.546243	1.000000	0.409128	0.610908
LDHH_fanIn	0.051403	0.102584	0.409128	1.000000	0.389176
numberOfNonTrivialBugsFoundUntil:	0.029961	0.500429	0.610908	0.389176	1.000000
...	...	...	...	...	...
LDHH_cbo	0.055331	0.281304	0.601213	0.841620	0.537211
WCHU_numberOfLinesOfCode	0.036330	0.564771	0.725557	0.401675	0.830664
CvsExpEntropy	0.038985	0.304112	0.500495	0.350257	0.595370
LDHH_numberOfMethods	0.022722	0.552963	0.996665	0.416615	0.619973
class	0.020520	-0.238242	-0.371588	-0.202935	-0.474638

63 rows × 63 columns

In [12]:
```
# plot corr
def plotCorrelationMatrix(df, graphWidth):
    df = df.dropna('columns') # drop columns with NaN
    df = df[[col for col in df if df[col].nunique() > 1]] # keep columns where there are more than 1 unique val
    if df.shape[1] < 2:
        print(f'No correlation plots shown: The number of non-NaN or constant columns ({df.shape[1]}) is less t
        return
    corr = df.corr()
    plt.figure(num=None, figsize=(graphWidth, graphWidth), dpi=80, facecolor='w', edgecolor='k')
    corrMat = plt.matshow(corr, fignum = 1)
    plt.xticks(range(len(corr.columns)), corr.columns, rotation=90)
    plt.yticks(range(len(corr.columns)), corr.columns)
    plt.gca().xaxis.tick_bottom()
    plt.colorbar(corrMat)
    plt.show()

plotCorrelationMatrix(df, 15)
```

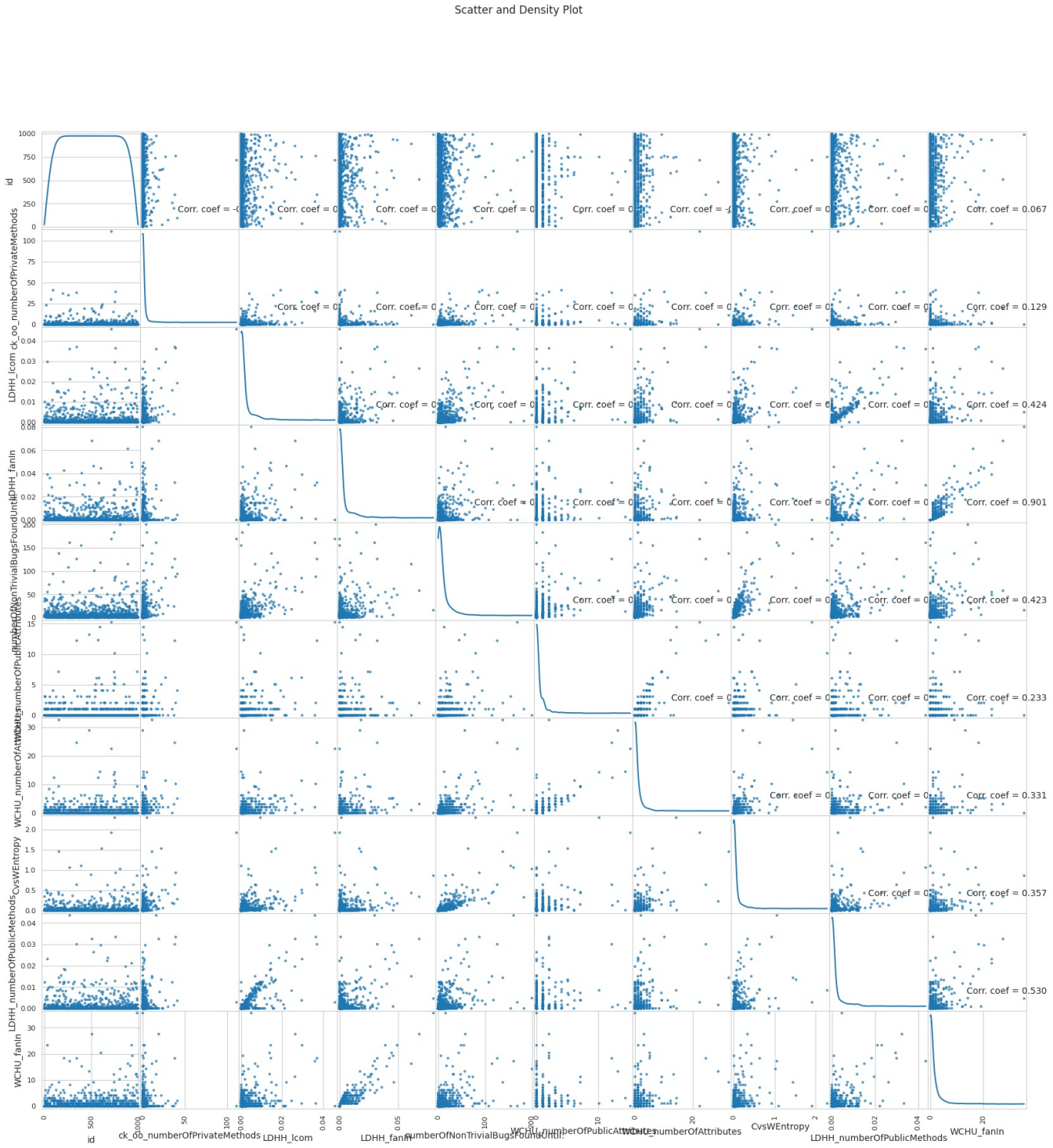
```
<ipython-input-12-4abd0cdf933f>:3: FutureWarning:

In a future version of pandas all arguments of DataFrame.dropna will be keyword-only.
```



In [13]:

```
# Scatter and density plots
def plotScatterMatrix(df, plotSize, textSize):
    df = df.select_dtypes(include =[np.number]) # keep only numerical columns
    # Remove rows and columns that would lead to df being singular
    df = df.dropna('columns')
    df = df[[col for col in df if df[col].nunique() > 1]] # keep columns where there are more than 1 unique val
    columnNames = list(df)
    if len(columnNames) > 10: # reduce the number of columns for matrix inversion of kernel density plots
        columnNames = columnNames[:10]
    df = df[columnNames]
    ax = pd.plotting.scatter_matrix(df, alpha=0.75, figsize=[plotSize, plotSize], diagonal='kde')
    corrs = df.corr().values
    for i, j in zip(*plt.np.triu_indices_from(ax, k = 1)):
        ax[i, j].annotate('Corr. coef = %.3f' % corrs[i, j], (0.8, 0.2), xycoords='axes fraction', ha='center',
    plt.suptitle('Scatter and Density Plot')
    plt.show()

plotScatterMatrix(df, 20, 10)
```

```
<ipython-input-13-6ec3fc198971>:5: FutureWarning:

In a future version of pandas all arguments of DataFrame.dropna will be keyword-only.
```



NORMALIZATION

In [14]:
```
from sklearn import preprocessing
def change(X):
    length = X.shape[0]
    d=pd.Series(np.ones((length)))
    for i in range(0, length, 73):
        d[i]=0
    return d
def a(X):
    length = X.shape[0]
    d=pd.Series(np.zeros((length)))
    for i in range(0, length, 20):
        d[i]=1
    return d
```

In [15]:
```
def change(X):
    length = X.shape[0]
    d=pd.Series(np.ones((length)))
    for i in range(0, length, 73):
        d[i]=0
    return d
```

```
def a(X):
    length = X.shape[0]
    d=pd.Series(np.zeros((length)))
    for i in range(0, length, 20):
        d[i]=1
    return d
```

SMOTE-TOMEK

In [16]:
```
pip install -U imbalanced-learn
```

```
Requirement already satisfied: imbalanced-learn in /usr/local/lib/python3.10/dist-packages (0.10.1)
Collecting imbalanced-learn
  Downloading imbalanced_learn-0.11.0-py3-none-any.whl (235 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 235.6/235.6 kB 3.8 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn)
(1.23.5)
Requirement already satisfied: scipy>=1.5.0 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn)
(1.11.4)
Requirement already satisfied: scikit-learn>=1.0.2 in /usr/local/lib/python3.10/dist-packages (from imbalanced-
learn) (1.2.2)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn)
(1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from imbalanced
-learn) (3.2.0)
Installing collected packages: imbalanced-learn
  Attempting uninstall: imbalanced-learn
    Found existing installation: imbalanced-learn 0.10.1
    Uninstalling imbalanced-learn-0.10.1:
      Successfully uninstalled imbalanced-learn-0.10.1
Successfully installed imbalanced-learn-0.11.0
```

In [17]:
```
pip install imblearn
```

```
Collecting imblearn
  Downloading imblearn-0.0-py2.py3-none-any.whl (1.9 kB)
Requirement already satisfied: imbalanced-learn in /usr/local/lib/python3.10/dist-packages (from imblearn) (0.1
1.0)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn-
>imblearn) (1.23.5)
Requirement already satisfied: scipy>=1.5.0 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn->
imblearn) (1.11.4)
Requirement already satisfied: scikit-learn>=1.0.2 in /usr/local/lib/python3.10/dist-packages (from imbalanced-
learn->imblearn) (1.2.2)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn-
>imblearn) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from imbalanced
-learn->imblearn) (3.2.0)
Installing collected packages: imblearn
Successfully installed imblearn-0.0
```

In [18]:
```
pip install seaborn
```

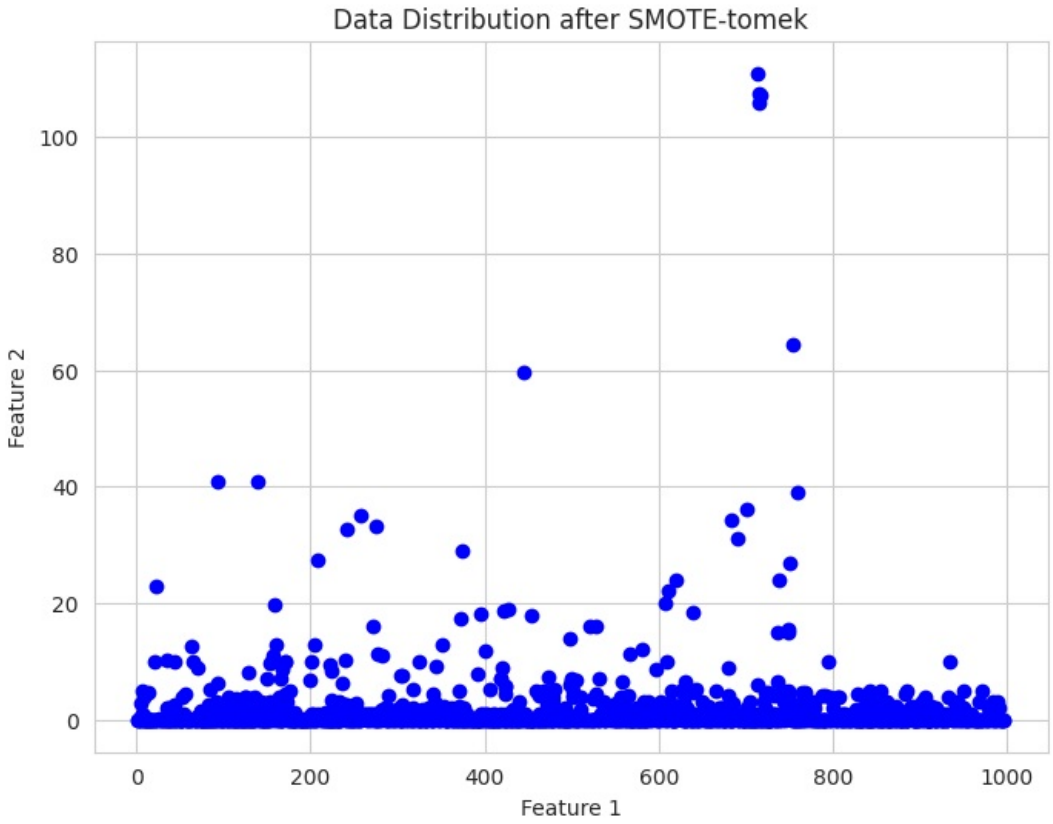
```
Requirement already satisfied: seaborn in /usr/local/lib/python3.10/dist-packages (0.12.2)
Requirement already satisfied: numpy!=1.24.0,>=1.17 in /usr/local/lib/python3.10/dist-packages (from seaborn) (
1.23.5)
Requirement already satisfied: pandas>=0.25 in /usr/local/lib/python3.10/dist-packages (from seaborn) (1.5.3)
Requirement already satisfied: matplotlib!=3.6.1,>=3.1 in /usr/local/lib/python3.10/dist-packages (from seaborn
) (3.7.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.
6.1,>=3.1->seaborn) (1.2.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,
>=3.1->seaborn) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3
.6.1,>=3.1->seaborn) (4.46.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3
.6.1,>=3.1->seaborn) (1.4.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6
.1,>=3.1->seaborn) (23.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1
,>=3.1->seaborn) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.
6.1,>=3.1->seaborn) (3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib
!=3.6.1,>=3.1->seaborn) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.25->seab
orn) (2023.3.post1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->
matplotlib!=3.6.1,>=3.1->seaborn) (1.16.0)
```

In [19]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from imblearn.combine import SMOTETomek
from collections import Counter
```

```
# Assuming the last column is the target variable
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values


# 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)

# Apply SMOTE-Tomek only on the training data
smt = SMOTETomek(random_state=42)
X_resampled, y_resampled = smt.fit_resample(X_train, y_train)
```

In [20]:
```
from imblearn.combine import SMOTETomek

# Visualize the data distribution after applying SMOTE
plt.figure(figsize=(8, 6))
plt.scatter(X_resampled[:, 0], X_resampled[:, 1], c='blue', cmap=plt.cm.Paired, marker='o')
plt.title("Data Distribution after SMOTE-tomek")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```

```
<ipython-input-20-996aeb56b06b>:5: UserWarning:

No data for colormapping provided via 'c'. Parameters 'cmap' will be ignored
```



KNN

In [21]:
```
pip install sklearn
```

```
Collecting sklearn
  Downloading sklearn-0.0.post12.tar.gz (2.6 kB)
  error: subprocess-exited-with-error

  × python setup.py egg_info did not run successfully.
  │ exit code: 1
  └─> See above for output.

  note: This error originates from a subprocess, and is likely not a problem with pip.
  Preparing metadata (setup.py) ... error
error: metadata-generation-failed

× Encountered error while generating package metadata.
└─> See above for output.

note: This is an issue with the package mentioned above, not pip.
hint: See above for details.
```

In [22]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
```

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score
```

In [23]:
```
import math
math.sqrt(len(y_test))
```

Out[23]:
```
14.142135623730951
```

In [24]:
```
#Define the model
knn_model = KNeighborsClassifier (n_neighbors=13, p=2, metric='euclidean')

#fit model
knn_model.fit(X_resampled, y_resampled)
```

Out[24]:
```
KNeighborsClassifier(metric='euclidean', n_neighbors=13)
```

In [25]:
```
X_test.shape
```

Out[25]:
```
(200, 62)
```

In [26]:
```
# Predict the test set results
knn_pred = knn_model.predict(X_test)
y_pred = knn_model.predict(X_test)
y_pred
```

Out[26]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1,
       0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1,
       1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1,
       0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1,
       0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1,
       1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1,
       1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1,
       1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0,
       0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 1])
```

In [27]:
```
# Evaluate Model
cm = confusion_matrix(y_test, y_pred)
print (cm)
```

```
[[ 29  16]
 [ 37 118]]
```

In [28]:
```
print(accuracy_score(y_test, y_pred))
```

```
0.735
```

DECISION TREE

In [29]:
```
#import the necessary packages
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
from sklearn import tree
import matplotlib.pyplot as plt
```

In [30]:
```
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values

# 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)
```

In [31]:
```
feature_names = X_train
dt_model = DecisionTreeClassifier(criterion='entropy',random_state=42)
dt_model.fit(X_train, y_train)
```

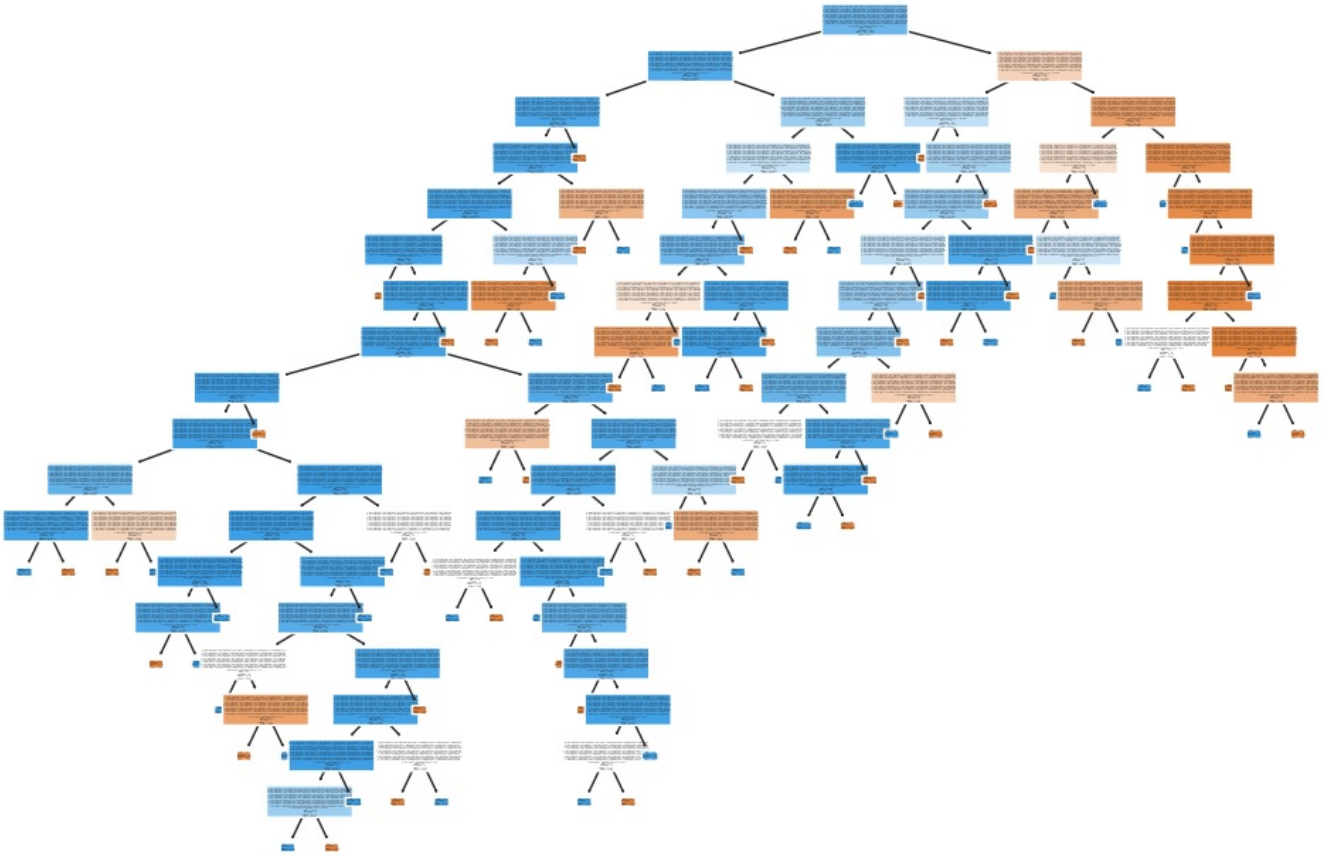
Out[31]:
```
DecisionTreeClassifier(criterion='entropy', random_state=42)
```

In [32]:
```
dt_model = DecisionTreeClassifier()
dt_model.fit(X_train, y_train)
dt_pred = dt_model.predict(X_test)
```

In [33]:
```
# Make predictions on the test set
y_pred = dt_model.predict(X_test)

# Calculate and print the accuracy of the classifier on the test set
accuracy = accuracy_score(y_test, y_test)
```

```
print(f"Accuracy: {accuracy:.2f}")

# Visualize the decision tree
plt.figure(figsize=(12, 8))
tree.plot_tree(dt_model, feature_names=feature_names, class_names=['True','False'], filled=True, rounded=True)
plt.show()
```

```
Accuracy: 1.00
```



In [34]:
```
#function to perform training with entropy
dt_model = DecisionTreeClassifier(criterion='entropy')
dt_model= dt_model.fit(X_resampled, y_resampled)
```

In [35]:
```
y_pred_en = dt_model.predict(X_test)
y_pred_en
```

Out[35]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
       0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1,
       1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0,
       0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1,
       1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1,
       0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1,
       1, 1])
```

In [36]:
```
print ("Accuracy is "),metrics.accuracy_score(y_test,y_pred_en)*100
```

Out[36]:
```
Accuracy is
(None, 80.5)
```

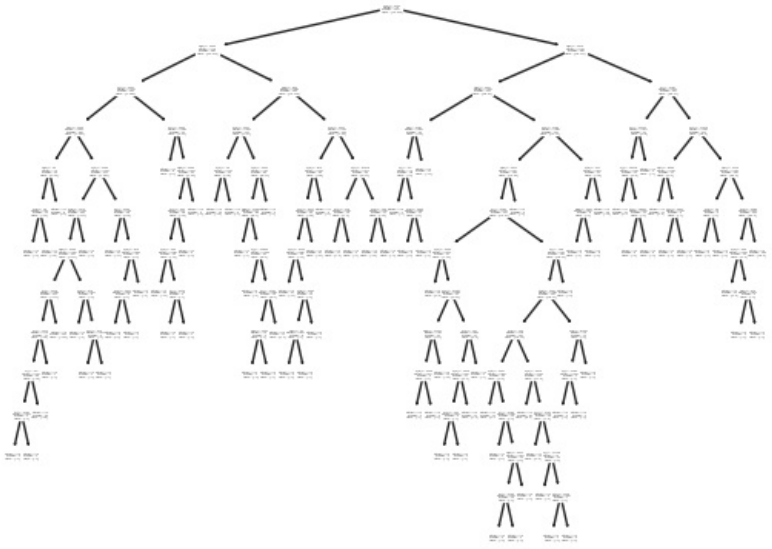
In [37]:
```
from sklearn import tree
tree.plot_tree(dt_model)
```

Out[37]:
```
[Text(0.5084206586826348, 0.9642857142857143, 'x[24] <= 2.194\nentropy = 1.0\nsamples = 1246\nvalue = [623, 623
]'),
 Text(0.26721556886227543, 0.8928571428571429, 'x[54] <= 0.005\nentropy = 0.78\nsamples = 666\nvalue = [154, 51
2]'),
 Text(0.1601796407185629, 0.8214285714285714, 'x[17] <= 0.002\nentropy = 0.474\nsamples = 433\nvalue = [44, 389
]'),
 Text(0.09281437125748503, 0.75, 'x[57] <= 0.903\nentropy = 0.333\nsamples = 391\nvalue = [24, 367]'),
 Text(0.059880239520958084, 0.6785714285714286, 'x[53] <= 0.0\nentropy = 1.0\nsamples = 20\nvalue = [10, 10]'),
 Text(0.04790419161676647, 0.6071428571428571, 'x[35] <= 1.5\nentropy = 0.65\nsamples = 12\nvalue = [2, 10]'),
 Text(0.03592814371257485, 0.5357142857142857, 'entropy = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(0.059880239520958084, 0.5357142857142857, 'entropy = 0.0\nsamples = 10\nvalue = [0, 10]'),
 Text(0.0718562874251497, 0.6071428571428571, 'entropy = 0.0\nsamples = 8\nvalue = [8, 0]'),
 Text(0.12574850299401197, 0.6785714285714286, 'x[23] <= 0.006\nentropy = 0.232\nsamples = 371\nvalue = [14, 35
7]'),
 Text(0.09580838323353294, 0.6071428571428571, 'x[35] <= 199.5\nentropy = 0.133\nsamples = 323\nvalue = [6, 317
]'),
 Text(0.08383233532934131, 0.5357142857142857, 'x[50] <= 0.156\nentropy = 0.097\nsamples = 321\nvalue = [4, 317
```

]'),
 Text(0.059880239520958084, 0.4642857142857143, 'x[0] <= 243.5\nentropy = 0.056\nsamples = 313\nvalue = [2, 311
]'),
 Text(0.04790419161676647, 0.39285714285714285, 'x[0] <= 241.5\nentropy = 0.187\nsamples = 70\nvalue = [2, 68]'
),
 Text(0.03592814371257485, 0.32142857142857145, 'x[7] <= 0.0\nentropy = 0.109\nsamples = 69\nvalue = [1, 68]'),
 Text(0.023952095808383235, 0.25, 'x[3] <= 0.001\nentropy = 0.918\nsamples = 3\nvalue = [1, 2]'),
 Text(0.011976047904191617, 0.17857142857142858, 'entropy = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(0.03592814371257485, 0.17857142857142858, 'entropy = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.04790419161676647, 0.25, 'entropy = 0.0\nsamples = 66\nvalue = [0, 66]'),
 Text(0.059880239520958084, 0.32142857142857145, 'entropy = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.0718562874251497, 0.39285714285714285, 'entropy = 0.0\nsamples = 243\nvalue = [0, 243]'),
 Text(0.10778443113772455, 0.4642857142857143, 'x[18] <= 35.0\nentropy = 0.811\nsamples = 8\nvalue = [2, 6]'),
 Text(0.09580838323353294, 0.39285714285714285, 'entropy = 0.0\nsamples = 5\nvalue = [0, 5]'),
 Text(0.11976047904191617, 0.39285714285714285, 'x[27] <= 97.0\nentropy = 0.918\nsamples = 3\nvalue = [2, 1]'),
 Text(0.10778443113772455, 0.32142857142857145, 'entropy = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(0.1317365269461078, 0.32142857142857145, 'entropy = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.10778443113772455, 0.5357142857142857, 'entropy = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(0.15568862275449102, 0.6071428571428571, 'x[1] <= 0.192\nentropy = 0.65\nsamples = 48\nvalue = [8, 40]'),
 Text(0.1437125748502994, 0.5357142857142857, 'entropy = 0.0\nsamples = 37\nvalue = [0, 37]'),
 Text(0.16766467065868262, 0.5357142857142857, 'x[18] <= 46.5\nentropy = 0.845\nsamples = 11\nvalue = [8, 3]'),
 Text(0.15568862275449102, 0.4642857142857143, 'x[24] <= 2.027\nentropy = 0.503\nsamples = 9\nvalue = [8, 1]'),
 Text(0.1437125748502994, 0.39285714285714285, 'entropy = 0.0\nsamples = 8\nvalue = [8, 0]'),
 Text(0.16766467065868262, 0.39285714285714285, 'entropy = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.17964071856287425, 0.4642857142857143, 'entropy = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(0.2275449101796407, 0.75, 'x[17] <= 0.993\nentropy = 0.998\nsamples = 42\nvalue = [20, 22]'),
 Text(0.2155688622754491, 0.6785714285714286, 'entropy = 0.0\nsamples = 8\nvalue = [8, 0]'),
 Text(0.23952095808383234, 0.6785714285714286, 'x[58] <= 0.005\nentropy = 0.937\nsamples = 34\nvalue = [12, 22]
'),
 Text(0.2275449101796407, 0.6071428571428571, 'x[37] <= 0.01\nentropy = 0.75\nsamples = 28\nvalue = [6, 22]'),
 Text(0.2155688622754491, 0.5357142857142857, 'x[49] <= 2.05\nentropy = 0.258\nsamples = 23\nvalue = [1, 22]'),
 Text(0.20359281437125748, 0.4642857142857143, 'entropy = 0.0\nsamples = 21\nvalue = [0, 21]'),
 Text(0.2275449101796407, 0.4642857142857143, 'x[7] <= 0.015\nentropy = 1.0\nsamples = 2\nvalue = [1, 1]'),
 Text(0.2155688622754491, 0.39285714285714285, 'entropy = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.23952095808383234, 0.39285714285714285, 'entropy = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.23952095808383234, 0.5357142857142857, 'entropy = 0.0\nsamples = 5\nvalue = [5, 0]'),
 Text(0.25149700598802394, 0.6071428571428571, 'entropy = 0.0\nsamples = 6\nvalue = [6, 0]'),
 Text(0.37425149700598803, 0.8214285714285714, 'x[56] <= 58.5\nentropy = 0.998\nsamples = 233\nvalue = [110, 12
3]'),
 Text(0.31137724550898205, 0.75, 'x[34] <= 0.047\nentropy = 0.804\nsamples = 110\nvalue = [83, 27]'),
 Text(0.2874251497005988, 0.6785714285714286, 'x[45] <= 0.5\nentropy = 0.792\nsamples = 21\nvalue = [5, 16]'),
 Text(0.2754491017964072, 0.6071428571428571, 'entropy = 0.0\nsamples = 16\nvalue = [0, 16]'),
 Text(0.2994011976047904, 0.6071428571428571, 'entropy = 0.0\nsamples = 5\nvalue = [5, 0]'),
 Text(0.33532934131736525, 0.6785714285714286, 'x[23] <= 0.031\nentropy = 0.54\nsamples = 89\nvalue = [78, 11]'
),
 Text(0.32335329341317365, 0.6071428571428571, 'x[56] <= 16.562\nentropy = 0.41\nsamples = 85\nvalue = [78, 7]'
),
 Text(0.31137724550898205, 0.5357142857142857, 'entropy = 0.0\nsamples = 4\nvalue = [0, 4]'),
 Text(0.33532934131736525, 0.5357142857142857, 'x[0] <= 33.605\nentropy = 0.229\nsamples = 81\nvalue = [78, 3]'
),
 Text(0.32335329341317365, 0.4642857142857143, 'entropy = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(0.3473053892215569, 0.4642857142857143, 'x[4] <= 0.431\nentropy = 0.098\nsamples = 79\nvalue = [78, 1]'),
 Text(0.33532934131736525, 0.39285714285714285, 'x[50] <= 0.006\nentropy = 1.0\nsamples = 2\nvalue = [1, 1]'),
 Text(0.32335329341317365, 0.32142857142857145, 'entropy = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.3473053892215569, 0.32142857142857145, 'entropy = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.3592814371257485, 0.39285714285714285, 'entropy = 0.0\nsamples = 77\nvalue = [77, 0]'),
 Text(0.3473053892215569, 0.6071428571428571, 'entropy = 0.0\nsamples = 4\nvalue = [0, 4]'),
 Text(0.437125748502994, 0.75, 'x[43] <= 2.021\nentropy = 0.759\nsamples = 123\nvalue = [27, 96]'),
 Text(0.40718562874251496, 0.6785714285714286, 'x[18] <= 84.5\nentropy = 0.459\nsamples = 93\nvalue = [9, 84]')
,
 Text(0.39520958083832336, 0.6071428571428571, 'x[11] <= 6.413\nentropy = 0.353\nsamples = 90\nvalue = [6, 84]'
),
 Text(0.38323353293413176, 0.5357142857142857, 'x[21] <= 0.126\nentropy = 0.684\nsamples = 33\nvalue = [6, 27]'
),
 Text(0.3712574850299401, 0.4642857142857143, 'entropy = 0.0\nsamples = 21\nvalue = [0, 21]'),
 Text(0.39520958083832336, 0.4642857142857143, 'x[36] <= 0.002\nentropy = 1.0\nsamples = 12\nvalue = [6, 6]'),
 Text(0.38323353293413176, 0.39285714285714285, 'x[52] <= 3.5\nentropy = 0.811\nsamples = 8\nvalue = [6, 2]'),
 Text(0.3712574850299401, 0.32142857142857145, 'entropy = 0.0\nsamples = 6\nvalue = [6, 0]'),
 Text(0.39520958083832336, 0.32142857142857145, 'entropy = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(0.40718562874251496, 0.39285714285714285, 'entropy = 0.0\nsamples = 4\nvalue = [0, 4]'),
 Text(0.40718562874251496, 0.5357142857142857, 'entropy = 0.0\nsamples = 57\nvalue = [0, 57]'),
 Text(0.41916167664670656, 0.6071428571428571, 'entropy = 0.0\nsamples = 3\nvalue = [3, 0]'),
 Text(0.46706586826347307, 0.6785714285714286, 'x[27] <= 95.376\nentropy = 0.971\nsamples = 30\nvalue = [18, 12
]'),
 Text(0.4431137724550898, 0.6071428571428571, 'x[41] <= 17.5\nentropy = 0.485\nsamples = 19\nvalue = [17, 2]'),
 Text(0.4311377245508982, 0.5357142857142857, 'entropy = 0.0\nsamples = 17\nvalue = [17, 0]'),
 Text(0.4550898203592814, 0.5357142857142857, 'entropy = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(0.49101796407185627, 0.6071428571428571, 'x[42] <= 1.53\nentropy = 0.439\nsamples = 11\nvalue = [1, 10]')
,
 Text(0.47904191616766467, 0.5357142857142857, 'entropy = 0.0\nsamples = 10\nvalue = [0, 10]'),
 Text(0.5029940119760479, 0.5357142857142857, 'entropy = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.749625748502994, 0.8928571428571429, 'x[55] <= 0.016\nentropy = 0.704\nsamples = 580\nvalue = [469, 111
]'),
 Text(0.6279940119760479, 0.8214285714285714, 'x[50] <= 0.027\nentropy = 0.922\nsamples = 279\nvalue = [185, 94
]'),
 Text(0.5389221556886228, 0.75, 'x[8] <= 0.001\nentropy = 0.857\nsamples = 32\nvalue = [9, 23]'),
 Text(0.5269461077844312, 0.6785714285714286, 'x[2] <= 0.0\nentropy = 1.0\nsamples = 18\nvalue = [9, 9]'),
 Text(0.5149700598802395, 0.6071428571428571, 'entropy = 0.0\nsamples = 7\nvalue = [0, 7]'),
 Text(0.5389221556886228, 0.6071428571428571, 'x[38] <= 0.005\nentropy = 0.684\nsamples = 11\nvalue = [9, 2]'),

```
 Text(0.5269461077844312, 0.5357142857142857, 'entropy = 0.0\nsamples = 9\nvalue = [9, 0]'),
 Text(0.5508982035928144, 0.5357142857142857, 'entropy = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(0.5508982035928144, 0.6785714285714286, 'entropy = 0.0\nsamples = 14\nvalue = [0, 14]'),
 Text(0.7170658682634731, 0.75, 'x[29] <= 4.785\nentropy = 0.865\nsamples = 247\nvalue = [176, 71]'),
 Text(0.6616766467065869, 0.6785714285714286, 'x[53] <= 0.012\nentropy = 0.791\nsamples = 223\nvalue = [170, 53
]'),
 Text(0.6497005988023952, 0.6071428571428571, 'x[7] <= 0.031\nentropy = 0.754\nsamples = 217\nvalue = [170, 47]
'),
 Text(0.5748502994011976, 0.5357142857142857, 'x[24] <= 2.965\nentropy = 0.973\nsamples = 57\nvalue = [34, 23]'
),
 Text(0.562874251497006, 0.4642857142857143, 'entropy = 0.0\nsamples = 16\nvalue = [16, 0]'),
 Text(0.5868263473053892, 0.4642857142857143, 'x[27] <= 39.963\nentropy = 0.989\nsamples = 41\nvalue = [18, 23]
'),
 Text(0.562874251497006, 0.39285714285714285, 'x[56] <= 10.431\nentropy = 0.61\nsamples = 20\nvalue = [3, 17]')
,
 Text(0.5508982035928144, 0.32142857142857145, 'x[36] <= 0.008\nentropy = 0.971\nsamples = 5\nvalue = [3, 2]'),
 Text(0.5389221556886228, 0.25, 'entropy = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(0.562874251497006, 0.25, 'entropy = 0.0\nsamples = 3\nvalue = [3, 0]'),
 Text(0.5748502994011976, 0.32142857142857145, 'entropy = 0.0\nsamples = 15\nvalue = [0, 15]'),
 Text(0.6107784431137725, 0.39285714285714285, 'x[46] <= 1.97\nentropy = 0.863\nsamples = 21\nvalue = [15, 6]')
,
 Text(0.5988023952095808, 0.32142857142857145, 'x[15] <= 0.028\nentropy = 0.523\nsamples = 17\nvalue = [15, 2]'
),
 Text(0.5868263473053892, 0.25, 'x[14] <= 1.52\nentropy = 0.918\nsamples = 3\nvalue = [1, 2]'),
 Text(0.5748502994011976, 0.17857142857142858, 'entropy = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.5988023952095808, 0.17857142857142858, 'entropy = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(0.6107784431137725, 0.25, 'entropy = 0.0\nsamples = 14\nvalue = [14, 0]'),
 Text(0.6227544910179641, 0.32142857142857145, 'entropy = 0.0\nsamples = 4\nvalue = [0, 4]'),
 Text(0.7245508982035929, 0.5357142857142857, 'x[1] <= 19.5\nentropy = 0.61\nsamples = 160\nvalue = [136, 24]')
,
 Text(0.7125748502994012, 0.4642857142857143, 'x[60] <= 0.297\nentropy = 0.568\nsamples = 157\nvalue = [136, 21
]'),
 Text(0.6706586826347305, 0.39285714285714285, 'x[39] <= 1.05\nentropy = 0.48\nsamples = 145\nvalue = [130, 15]
'),
 Text(0.6467065868263473, 0.32142857142857145, 'x[39] <= 0.937\nentropy = 0.857\nsamples = 32\nvalue = [23, 9]'
),
 Text(0.6347305389221557, 0.25, 'entropy = 0.0\nsamples = 16\nvalue = [16, 0]'),
 Text(0.6586826347305389, 0.25, 'x[7] <= 0.035\nentropy = 0.989\nsamples = 16\nvalue = [7, 9]'),
 Text(0.6467065868263473, 0.17857142857142858, 'entropy = 0.0\nsamples = 4\nvalue = [4, 0]'),
 Text(0.6706586826347305, 0.17857142857142858, 'x[55] <= 0.012\nentropy = 0.811\nsamples = 12\nvalue = [3, 9]')
,
 Text(0.6586826347305389, 0.10714285714285714, 'x[6] <= 4.456\nentropy = 0.469\nsamples = 10\nvalue = [1, 9]'),
 Text(0.6467065868263473, 0.03571428571428571, 'entropy = 0.0\nsamples = 9\nvalue = [0, 9]'),
 Text(0.6706586826347305, 0.03571428571428571, 'entropy = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.6826347305389222, 0.10714285714285714, 'entropy = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(0.6946107784431138, 0.32142857142857145, 'x[49] <= 4.076\nentropy = 0.299\nsamples = 113\nvalue = [107, 6
]'),
 Text(0.6826347305389222, 0.25, 'entropy = 0.0\nsamples = 85\nvalue = [85, 0]'),
 Text(0.7065868263473054, 0.25, 'x[17] <= 0.794\nentropy = 0.75\nsamples = 28\nvalue = [22, 6]'),
 Text(0.6946107784431138, 0.17857142857142858, 'entropy = 0.0\nsamples = 16\nvalue = [16, 0]'),
 Text(0.718562874251497, 0.17857142857142858, 'x[4] <= 17.748\nentropy = 1.0\nsamples = 12\nvalue = [6, 6]'),
 Text(0.7065868263473054, 0.10714285714285714, 'entropy = 0.0\nsamples = 5\nvalue = [0, 5]'),
 Text(0.7305389221556886, 0.10714285714285714, 'x[21] <= 1.105\nentropy = 0.592\nsamples = 7\nvalue = [6, 1]'),
 Text(0.718562874251497, 0.03571428571428571, 'entropy = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.7425149700598802, 0.03571428571428571, 'entropy = 0.0\nsamples = 6\nvalue = [6, 0]'),
 Text(0.7544910179640718, 0.39285714285714285, 'x[28] <= 20.561\nentropy = 1.0\nsamples = 12\nvalue = [6, 6]'),
 Text(0.7425149700598802, 0.32142857142857145, 'x[6] <= 4.185\nentropy = 0.592\nsamples = 7\nvalue = [1, 6]'),
 Text(0.7305389221556886, 0.25, 'entropy = 0.0\nsamples = 6\nvalue = [0, 6]'),
 Text(0.7544910179640718, 0.25, 'entropy = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.7664670658682635, 0.32142857142857145, 'entropy = 0.0\nsamples = 5\nvalue = [5, 0]'),
 Text(0.7365269461077845, 0.4642857142857143, 'entropy = 0.0\nsamples = 3\nvalue = [0, 3]'),
 Text(0.6736526946107785, 0.6071428571428571, 'entropy = 0.0\nsamples = 6\nvalue = [0, 6]'),
 Text(0.7724550898203593, 0.6785714285714286, 'x[60] <= 0.17\nentropy = 0.811\nsamples = 24\nvalue = [6, 18]'),
 Text(0.7604790419161677, 0.6071428571428571, 'x[53] <= 0.002\nentropy = 0.811\nsamples = 8\nvalue = [6, 2]'),
 Text(0.7485029940119761, 0.5357142857142857, 'entropy = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(0.7724550898203593, 0.5357142857142857, 'entropy = 0.0\nsamples = 6\nvalue = [6, 0]'),
 Text(0.7844311377245509, 0.6071428571428571, 'entropy = 0.0\nsamples = 16\nvalue = [0, 16]'),
 Text(0.8712574850299402, 0.8214285714285714, 'x[7] <= 0.051\nentropy = 0.313\nsamples = 301\nvalue = [284, 17]
'),
 Text(0.8323353293413174, 0.75, 'x[4] <= 24.243\nentropy = 0.877\nsamples = 27\nvalue = [19, 8]'),
 Text(0.8203592814371258, 0.6785714285714286, 'x[0] <= 432.28\nentropy = 0.738\nsamples = 24\nvalue = [19, 5]')
,
 Text(0.8083832335329342, 0.6071428571428571, 'entropy = 0.0\nsamples = 13\nvalue = [13, 0]'),
 Text(0.8323353293413174, 0.6071428571428571, 'x[0] <= 768.91\nentropy = 0.994\nsamples = 11\nvalue = [6, 5]'),
 Text(0.8203592814371258, 0.5357142857142857, 'entropy = 0.0\nsamples = 5\nvalue = [0, 5]'),
 Text(0.844311377245509, 0.5357142857142857, 'entropy = 0.0\nsamples = 6\nvalue = [6, 0]'),
 Text(0.844311377245509, 0.6785714285714286, 'entropy = 0.0\nsamples = 3\nvalue = [0, 3]'),
 Text(0.9101796407185628, 0.75, 'x[41] <= 14.016\nentropy = 0.208\nsamples = 274\nvalue = [265, 9]'),
 Text(0.8682634730538922, 0.6785714285714286, 'x[57] <= 8.696\nentropy = 0.787\nsamples = 17\nvalue = [13, 4]')
,
 Text(0.8562874251497006, 0.6071428571428571, 'entropy = 0.0\nsamples = 12\nvalue = [12, 0]'),
 Text(0.8802395209580839, 0.6071428571428571, 'x[21] <= 4.501\nentropy = 0.722\nsamples = 5\nvalue = [1, 4]'),
 Text(0.8682634730538922, 0.5357142857142857, 'entropy = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.8922155688622755, 0.5357142857142857, 'entropy = 0.0\nsamples = 4\nvalue = [0, 4]'),
 Text(0.9520958083832335, 0.6785714285714286, 'x[51] <= 0.143\nentropy = 0.138\nsamples = 257\nvalue = [252, 5]
'),
 Text(0.9281437125748503, 0.6071428571428571, 'x[44] <= 2.0\nentropy = 1.0\nsamples = 4\nvalue = [2, 2]'),
 Text(0.9161676646706587, 0.5357142857142857, 'entropy = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(0.9401197604790419, 0.5357142857142857, 'entropy = 0.0\nsamples = 2\nvalue = [2, 0]'),
```

```
 Text(0.9760479041916168, 0.6071428571428571, 'x[43] <= 4.089\nentropy = 0.093\nsamples = 253\nvalue = [250, 3]
'),
 Text(0.9640718562874252, 0.5357142857142857, 'x[49] <= 9.09\nentropy = 0.353\nsamples = 45\nvalue = [42, 3]'),
 Text(0.9520958083832335, 0.4642857142857143, 'entropy = 0.0\nsamples = 41\nvalue = [41, 0]'),
 Text(0.9760479041916168, 0.4642857142857143, 'x[57] <= 49.0\nentropy = 0.811\nsamples = 4\nvalue = [1, 3]'),
 Text(0.9640718562874252, 0.39285714285714285, 'entropy = 0.0\nsamples = 3\nvalue = [0, 3]'),
 Text(0.9880239520958084, 0.39285714285714285, 'entropy = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.9880239520958084, 0.5357142857142857, 'entropy = 0.0\nsamples = 208\nvalue = [208, 0]')]
```



SVM

In [38]:
```
from sklearn.svm import SVC
svm_model = SVC()
```

In [39]:
```
svm_model.fit(X_resampled, y_resampled)
```

Out[39]:
```
▾ SVC
SVC()
```

In [40]:
```
svm_model = SVC()
svm_model.fit(X_train, y_train)
svm_pred = svm_model.predict(X_test)
```

In [41]:
```
svm_model.score(X_test, y_test)
```

Out[41]:
```
0.795
```

In [42]:
```
svm_model.predict(X_test)
```

Out[42]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1])
```

In [43]:
```
#Tune parameters
#1. Regularization (C)
svm_model_C = SVC(C=1)
svm_model_C.fit(X_resampled, y_resampled)
svm_model_C.score(X_test, y_test)
```

Out[43]:
```
0.81
```

In [44]:
```
svm_model_C = SVC(C=10)
svm_model_C.fit(X_resampled, y_resampled)
svm_model_C.score(X_test, y_test)
```

Out[44]:
```
0.805
```

NAIVE

In [45]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
```

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report, accuracy_score
```

In [46]:
```
from sklearn.naive_bayes import GaussianNB

# Assuming X contains numerical features
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a Gaussian Naive Bayes classifier
nb_model = GaussianNB()

# Train the classifier
nb_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = nb_model.predict(X_test)

# Print classification report and accuracy
print("Classification Report:")
print(classification_report(y_test, y_pred))

print("Accuracy:", accuracy_score(y_test, y_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.66      0.47      0.55        45
           1       0.86      0.93      0.89       155

    accuracy                           0.82       200
   macro avg       0.76      0.70      0.72       200
weighted avg       0.81      0.82      0.81       200

Accuracy: 0.825
```

In [47]:
```
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)
nb_pred = nb_model.predict(X_test)
```

GENETIC ALGORITHM BASED ON ARTIFICAL NEURAL NETWORK (GA-ANN)

In [48]:
```
pip install deap
```

```
Collecting deap
  Downloading deap-1.4.1-cp310-cp310-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_17_x86_64.manylinux2014
_x86_64.whl (135 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 135.4/135.4 kB 1.8 MB/s eta 0:00:00
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from deap) (1.23.5)
Installing collected packages: deap
Successfully installed deap-1.4.1
```

In [49]:
```
pip install deap keras
```

```
Requirement already satisfied: deap in /usr/local/lib/python3.10/dist-packages (1.4.1)
Requirement already satisfied: keras in /usr/local/lib/python3.10/dist-packages (2.15.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from deap) (1.23.5)
```

In [50]:
```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from deap import base, creator, tools, algorithms
import matplotlib.pyplot as plt

# 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)

# Define genetic algorithm parameters
population_size = 10
num_generations = 5
mutation_rate = 0.1

# Function to create a simple feedforward neural network
def create_neural_network():
    model = keras.Sequential([
        keras.layers.Input(shape=(X_train.shape[1],)),
        keras.layers.Dense(64, activation='relu'),
        keras.layers.Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model

# Function to initialize a random population of neural networks
def initialize_population(population_size):
    return [create_neural_network() for _ in range(population_size)]

# Function to evaluate the fitness of each neural network in the population
```

```python
def evaluate_population(population, X_train, y_train, X_test, y_test):
    fitness_scores = []
    for model in population:
        model.fit(X_train, y_train, epochs=5, verbose=0)
        predictions = (model.predict(X_test) > 0.5).astype(int).flatten()
        accuracy = accuracy_score(y_test, predictions)
        fitness_scores.append(accuracy)
    return fitness_scores

# Function for tournament selection
def tournament_selection(fitness_scores, tournament_size):
    selected_indices = []
    for _ in range(len(fitness_scores)):
        tournament_indices = np.random.choice(len(fitness_scores), tournament_size, replace=False)
        winner_index = max(tournament_indices, key=lambda i: fitness_scores[i])
        selected_indices.append(winner_index)
    return selected_indices

# Function for one-point crossover
def crossover(parent1, parent2):
    child1 = create_neural_network()
    child2 = create_neural_network()

    for layer in range(len(child1.layers)):
        crossover_point = np.random.randint(0, 2)
        if crossover_point == 0:
            child1.layers[layer].set_weights(parent1.layers[layer].get_weights())
            child2.layers[layer].set_weights(parent2.layers[layer].get_weights())
        else:
            child1.layers[layer].set_weights(parent2.layers[layer].get_weights())
            child2.layers[layer].set_weights(parent1.layers[layer].get_weights())

    return child1, child2

# Function for mutation
def mutate(model, mutation_rate):
    for layer in range(len(model.layers)):
        if np.random.rand() < mutation_rate:
            new_weights = [w + np.random.normal(0, 0.1, w.shape) for w in model.layers[layer].get_weights()]
            model.layers[layer].set_weights(new_weights)
    return model

# Main genetic algorithm loop
population = initialize_population(population_size)

# Lists to store evolution data
gen_numbers = []
max_accuracies = []

for generation in range(num_generations):
    fitness_scores = evaluate_population(population, X_train, y_train, X_test, y_test)

    # Select parents using tournament selection
    selected_indices = tournament_selection(fitness_scores, tournament_size=2)

    # Create new generation using crossover and mutation
    new_population = []
    for i in range(0, len(selected_indices), 2):
        parent1 = population[selected_indices[i]]
        parent2 = population[selected_indices[i + 1]]
        child1, child2 = crossover(parent1, parent2)
        child1 = mutate(child1, mutation_rate)
        child2 = mutate(child2, mutation_rate)
        new_population.extend([child1, child2])

    population = new_population

    # Collect evolution data
    best_model = max(population, key=lambda model: accuracy_score(y_test, (model.predict(X_test) > 0.5).astype(
    max_accuracy = accuracy_score(y_test, (best_model.predict(X_test) > 0.5).astype(int).flatten())

    gen_numbers.append(generation)
    max_accuracies.append(max_accuracy)
```

```
7/7 [==============================] - 0s 3ms/step
7/7 [==============================] - 0s 5ms/step
7/7 [==============================] - 0s 5ms/step
7/7 [==============================] - 0s 5ms/step
7/7 [==============================] - 0s 5ms/step
7/7 [==============================] - 0s 3ms/step
7/7 [==============================] - 0s 8ms/step
7/7 [==============================] - 0s 7ms/step
7/7 [==============================] - 0s 6ms/step
7/7 [==============================] - 0s 4ms/step
7/7 [==============================] - 0s 4ms/step
7/7 [==============================] - 0s 6ms/step
7/7 [==============================] - 0s 3ms/step
7/7 [==============================] - 0s 4ms/step
7/7 [==============================] - 0s 13ms/step
```

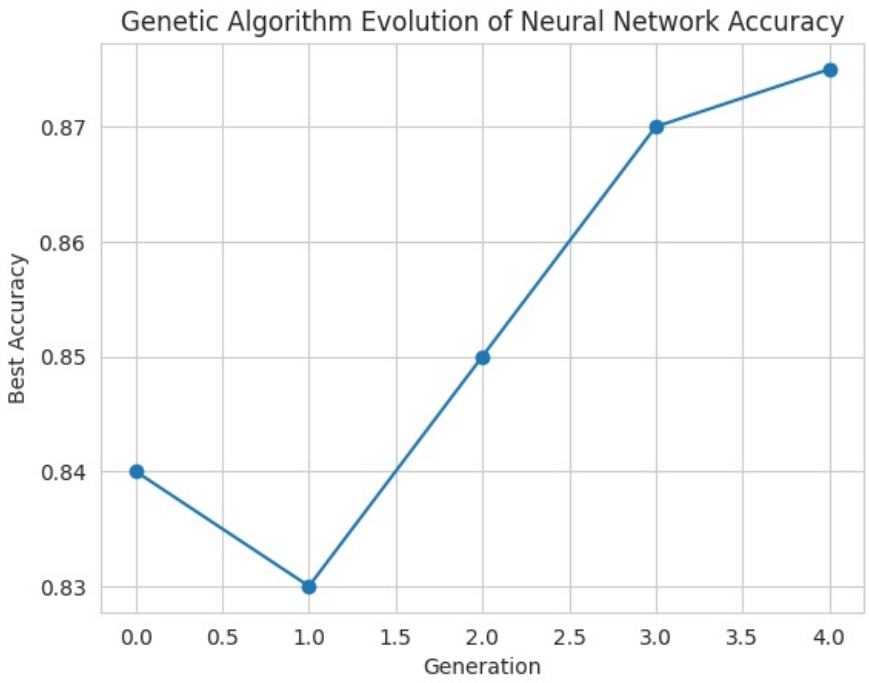
```
7/7 [==============================] - 0s 4ms/step
7/7 [==============================] - 0s 3ms/step
7/7 [==============================] - 0s 4ms/step
7/7 [==============================] - 0s 4ms/step
7/7 [==============================] - 0s 4ms/step
7/7 [==============================] - 0s 5ms/step
7/7 [==============================] - 0s 6ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 3ms/step
7/7 [==============================] - 0s 3ms/step
7/7 [==============================] - 0s 4ms/step
7/7 [==============================] - 0s 4ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 4ms/step
7/7 [==============================] - 0s 4ms/step
7/7 [==============================] - 0s 4ms/step
7/7 [==============================] - 0s 3ms/step
7/7 [==============================] - 0s 3ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 4ms/step
7/7 [==============================] - 0s 3ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 3ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 3ms/step
7/7 [==============================] - 0s 3ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 3ms/step
7/7 [==============================] - 0s 3ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 4ms/step
7/7 [==============================] - 0s 3ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 3ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 4ms/step
7/7 [==============================] - 0s 3ms/step
7/7 [==============================] - 0s 3ms/step
7/7 [==============================] - 0s 3ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
```

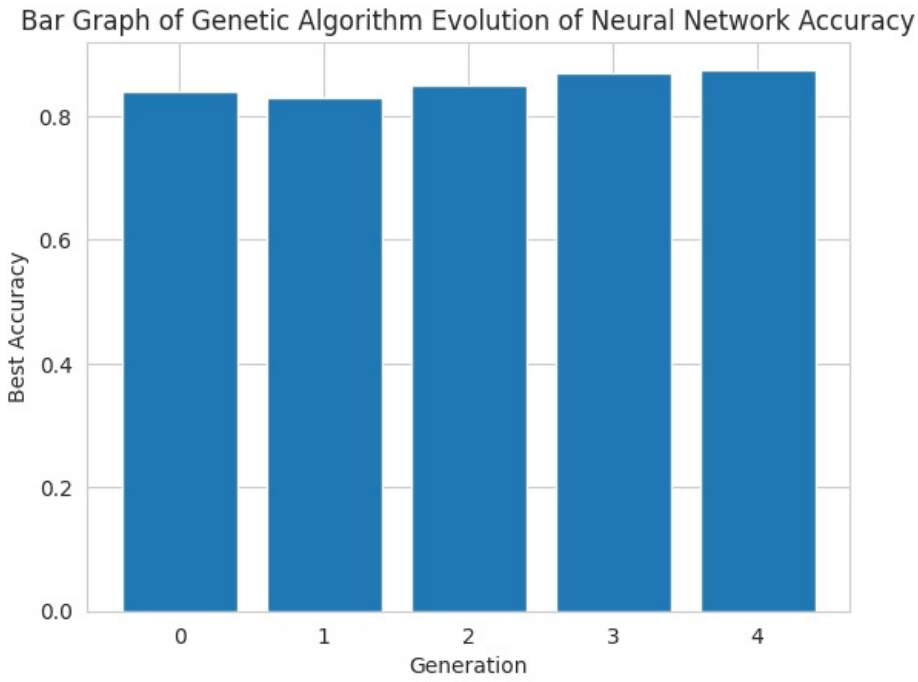
```
7/7 [==============================] - 0s 2ms/step
```

In [51]:
```
# Select the best neural network from the final population
best_model = max(population, key=lambda model: accuracy_score(y_test, (model.predict(X_test) > 0.5).astype(int)
print("Best Neural Network Accuracy:", accuracy_score(y_test, (best_model.predict(X_test) > 0.5).astype(int).fl
```

```
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 3ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 7ms/step
Best Neural Network Accuracy: 0.875
```

In [52]:
```
# Plot the evolution of accuracy
plt.plot(gen_numbers, max_accuracies, marker='o')
plt.xlabel('Generation')
plt.ylabel('Best Accuracy')
plt.title('Genetic Algorithm Evolution of Neural Network Accuracy')
plt.show()

# Plot bar graph for accuracy
plt.bar(range(len(max_accuracies)), max_accuracies)
plt.xlabel('Generation')
plt.ylabel('Best Accuracy')
plt.title('Bar Graph of Genetic Algorithm Evolution of Neural Network Accuracy')
plt.show()
```



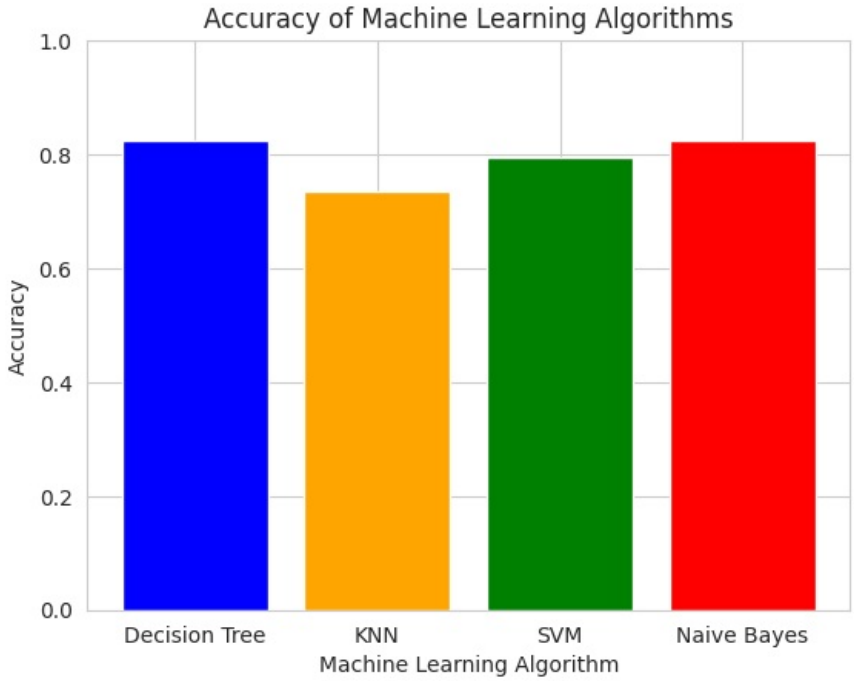


EVALUATING OTHER ALGORITHMS

In [53]:

```
# Evaluate models and store results

models = {'Decision Tree': dt_pred,
          'KNN': knn_pred,
          'SVM': svm_pred,
          'Naive Bayes': nb_pred}

model_results = {}
for name, pred in models.items():
    acc = accuracy_score(y_test, pred)
    model_results[name] = acc
    cm = confusion_matrix(y_test, pred)
    print(f"Model: {name}")
    print(f"Accuracy: {acc:.4f}")
    print(f"Confusion Matrix:\n{cm}\n")

# Plot bar graph
plt.bar(model_results.keys(), model_results.values(), color=['blue', 'orange', 'green', 'red', 'purple'])
plt.xlabel('Machine Learning Algorithm')
plt.ylabel('Accuracy')
plt.title('Accuracy of Machine Learning Algorithms')
plt.ylim([0, 1])  # Set the y-axis limit to better visualize differences
plt.show()
```

```
Model: Decision Tree
Accuracy: 0.8250
Confusion Matrix:
[[ 28  17]
 [ 18 137]]

Model: KNN
Accuracy: 0.7350
Confusion Matrix:
[[ 29  16]
 [ 37 118]]

Model: SVM
Accuracy: 0.7950
Confusion Matrix:
[[  8  37]
 [  4 151]]

Model: Naive Bayes
Accuracy: 0.8250
Confusion Matrix:
[[ 21  24]
 [ 11 144]]
```



IMPORTING DATASETS AND LIBRARIES

In [1]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
```

In [2]:
```
df = pd.read_csv('/content/csv_result-EQ.csv')
df.head()
```

Out[2]:

	id	ck_oo_numberOfPrivateMethods	LDHH_lcom	LDHH_fanIn	numberOfNonTrivialBugsFoundUntil:	WCHU_numberOfPublicAttributes	WCHU
0	1	3	0.002547	0.002555	4	0.00	
1	2	37	0.008643	0.004756	71	0.00	
2	3	3	0.001479	0.009143	5	1.01	
3	4	10	0.005642	0.005395	38	0.00	
4	5	1	0.001350	0.000000	1	1.01	

5 rows × 63 columns

In [3]:
```
import matplotlib.pyplot as plt
%matplotlib inline

import seaborn as sns
sns.set_style('whitegrid')

from tqdm import tqdm
```

EDA

- import some dependencies to plot
- use plotly to visualization
  - label classification
    - count and plot(visualization)
  - value visualization
    - use historgram to visualization attribution
    - relationship
      - covariance
      - heatmap
  - scatter

In [4]:
```
# import some dependencies for plotting

from plotly.offline import iplot
import plotly.graph_objs as go
```

In [5]:
```
# check data
def show_info(data, is_matrix_transpose=False):
    # basic shape
    print('data shape is: {}   sample number {}   attribute number {}\n'.format(df.shape, df.shape[0], df.shape
    # attribute(key)
    print('data columns number {}  \nall columns: {}\n'.format(len(df.columns) ,df.columns))
    # value's null
    print('data all attribute count null:\n', df.isna().sum())
    # data value analysis and data demo
    if is_matrix_transpose:
        print('data value analysis: ', df.describe().T)
        print('data demo without matrix transpose: ', df.head().T)
    else:
        print('data value analysis: ', df.describe())
        print('data demo without matrix transpose: ', df.head())

show_info(df)
```

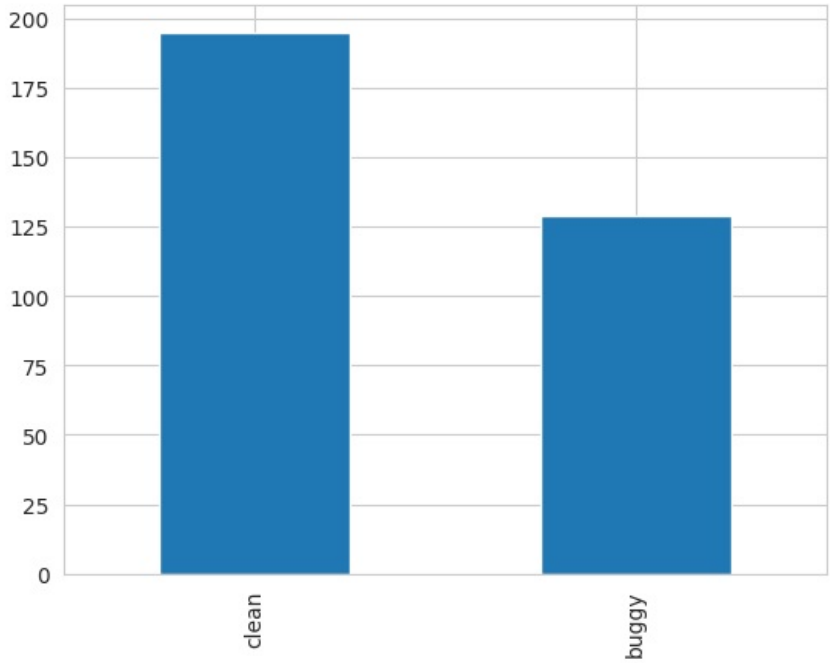
```
data shape is: (324, 63)   sample number 324   attribute number 63

data columns number 63
all columns: Index(['id', 'ck_oo_numberOfPrivateMethods', 'LDHH_lcom', 'LDHH_fanIn',
       'numberOfNonTrivialBugsFoundUntil:', 'WCHU_numberOfPublicAttributes',
       'WCHU_numberOfAttributes', 'CvsWEntropy', 'LDHH_numberOfPublicMethods',
       'WCHU_fanIn', 'LDHH_numberOfPrivateAttributes', 'CvsEntropy',
       'LDHH_numberOfPublicAttributes', 'WCHU_numberOfPrivateMethods',
       'WCHU_numberOfMethods', 'ck_oo_numberOfPublicAttributes', 'ck_oo_noc',
       'numberOfCriticalBugsFoundUntil:', 'ck_oo_wmc',
       'LDHH_numberOfPrivateMethods', 'WCHU_numberOfPrivateAttributes',
       'CvsLogEntropy', 'WCHU_noc', 'LDHH_numberOfAttributesInherited',
       'WCHU_wmc', 'ck_oo_fanOut', 'ck_oo_numberOfLinesOfCode',
```

```
       'ck_oo_numberOfAttributesInherited', 'ck_oo_numberOfMethods',
       'ck_oo_dit', 'ck_oo_fanIn', 'LDHH_noc', 'WCHU_dit', 'ck_oo_lcom',
       'WCHU_numberOfAttributesInherited', 'ck_oo_rfc', 'LDHH_wmc',
       'LDHH_numberOfAttributes', 'LDHH_numberOfLinesOfCode', 'WCHU_fanOut',
       'WCHU_lcom', 'ck_oo_cbo', 'WCHU_rfc', 'ck_oo_numberOfAttributes',
       'numberOfHighPriorityBugsFoundUntil:',
       'ck_oo_numberOfPrivateAttributes', 'numberOfMajorBugsFoundUntil:',
       'WCHU_numberOfPublicMethods', 'LDHH_dit', 'WCHU_cbo', 'CvsLinEntropy',
       'WCHU_numberOfMethodsInherited', 'numberOfBugsFoundUntil:',
       'LDHH_fanOut', 'LDHH_numberOfMethodsInherited', 'LDHH_rfc',
       'ck_oo_numberOfMethodsInherited', 'ck_oo_numberOfPublicMethods',
       'LDHH_cbo', 'WCHU_numberOfLinesOfCode', 'CvsExpEntropy',
       'LDHH_numberOfMethods', 'class'],
      dtype='object')

data all attribute count null:
 id                                 0
ck_oo_numberOfPrivateMethods        0
LDHH_lcom                           0
LDHH_fanIn                          0
numberOfNonTrivialBugsFoundUntil:   0
                                   ..
LDHH_cbo                            0
WCHU_numberOfLinesOfCode            0
CvsExpEntropy                       0
LDHH_numberOfMethods                0
class                               0
Length: 63, dtype: int64
data value analysis:                id  ck_oo_numberOfPrivateMethods   LDHH_lcom  LDHH_fanIn  \
count  324.00000                   324.000000  324.000000  324.000000
mean   162.50000                     2.083333    0.001212    0.001558
std     93.67497                     5.848646    0.003069    0.003452
min      1.00000                     0.000000    0.000000    0.000000
25%     81.75000                     0.000000    0.000000    0.000000
50%    162.50000                     0.000000    0.000000    0.000000
75%    243.25000                     1.000000    0.000851    0.001306
max    324.00000                    49.000000    0.024636    0.023417

       numberOfNonTrivialBugsFoundUntil:  WCHU_numberOfPublicAttributes  \
count                         324.000000                     324.000000
mean                            4.299383                       0.261574
std                             7.906567                       0.949242
min                             0.000000                       0.000000
25%                             1.000000                       0.000000
50%                             2.000000                       0.000000
75%                             4.000000                       0.000000
max                            71.000000                       7.740000

       WCHU_numberOfAttributes  CvsWEntropy  LDHH_numberOfPublicMethods  \
count               324.000000   324.000000                   324.000000
mean                  0.900432     0.154263                     0.000553
std                   2.017897     0.310351                     0.001573
min                   0.000000     0.000000                     0.000000
25%                   0.000000     0.000000                     0.000000
50%                   0.000000     0.034743                     0.000000
75%                   1.010000     0.141907                     0.000389
max                  16.250000     2.670750                     0.015215

       WCHU_fanIn  ...  numberOfBugsFoundUntil:  LDHH_fanOut  \
count  324.000000  ...               324.000000   324.000000
mean     0.887840  ...                 4.586420     0.002645
std      1.812751  ...                 8.452852     0.005133
min      0.000000  ...                 0.000000     0.000000
25%      0.000000  ...                 1.000000     0.000000
50%      0.000000  ...                 2.000000     0.000000
75%      1.010000  ...                 4.000000     0.003065
max     11.140000  ...                78.000000     0.037312

       LDHH_numberOfMethodsInherited    LDHH_rfc  \
count                     324.000000  324.000000
mean                        0.001492    0.003581
std                         0.002422    0.007835
min                         0.000000    0.000000
25%                         0.000000    0.000000
50%                         0.001486    0.000000
75%                         0.001829    0.003101
max                         0.019677    0.060301

       ck_oo_numberOfMethodsInherited  ck_oo_numberOfPublicMethods  \
count                      324.000000                   324.000000
mean                        14.722222                     5.743827
std                         14.142902                     7.186963
min                          0.000000                     0.000000
25%                          7.000000                     2.000000
50%                          9.000000                     4.000000
75%                         17.000000                     7.000000
max                        111.000000                    54.000000

         LDHH_cbo  WCHU_numberOfLinesOfCode  CvsExpEntropy  \
```

```
count  324.000000                324.000000     324.000000
mean     0.003652                  2.929660       0.034247
std      0.006038                  6.268788       0.053297
min      0.000000                  0.000000       0.000000
25%      0.000000                  0.000000       0.000000
50%      0.000571                  0.000000       0.002459
75%      0.004918                  3.227500       0.057904
max      0.039637                 48.680000       0.328692

       LDHH_numberOfMethods
count            324.000000
mean               0.001365
std                0.003409
min                0.000000
25%                0.000000
50%                0.000000
75%                0.000864
max                0.027806

[8 rows x 62 columns]
data demo without matrix transpose:     id  ck_oo_numberOfPrivateMethods  LDHH_lcom  LDHH_fanIn  \
0   1                             3   0.002547    0.002555
1   2                            37   0.008643    0.004756
2   3                             3   0.001479    0.009143
3   4                            10   0.005642    0.005395
4   5                             1   0.001350    0.000000

   numberOfNonTrivialBugsFoundUntil:  WCHU_numberOfPublicAttributes  \
0                                  4                          0.00
1                                 71                          0.00
2                                  5                          1.01
3                                 38                          0.00
4                                  1                          1.01

   WCHU_numberOfAttributes  CvsWEntropy  LDHH_numberOfPublicMethods  \
0                     3.04     0.393707                    0.003049
1                    14.37     2.093750                    0.001481
2                     3.08     0.484675                    0.000000
3                     1.06     0.811584                    0.000876
4                     1.01     0.031940                    0.000876

   WCHU_fanIn  ...  LDHH_fanOut  LDHH_numberOfMethodsInherited  LDHH_rfc  \
0        1.01  ...     0.005627                       0.000000  0.004406
1        2.02  ...     0.018761                       0.001486  0.060301
2        6.17  ...     0.003117                       0.001486  0.002325
3        6.07  ...     0.020376                       0.002338  0.030608
4        0.00  ...     0.000566                       0.003017  0.001492

   ck_oo_numberOfMethodsInherited  ck_oo_numberOfPublicMethods  LDHH_cbo  \
0                               8                            8  0.008431
1                               7                            7  0.021602
2                               7                            2  0.011859
3                              17                           10  0.020478
4                              22                            7  0.000652

   WCHU_numberOfLinesOfCode  CvsExpEntropy  LDHH_numberOfMethods  class
0                      3.50       0.103594              0.003611  buggy
1                     43.12       0.328692              0.009906  buggy
2                      4.68       0.125841              0.001655  clean
3                     24.06       0.170416              0.007000  buggy
4                      2.13       0.055912              0.001572  buggy

[5 rows x 63 columns]
```

In [6]:
```
# label classification
df['class'].value_counts().plot.bar()
```

Out[6]:
```
<Axes: >
```



Class Label

In [7]:
```
from sklearn.preprocessing import LabelEncoder
# finding the count of different labels
df['class'].value_counts()
```

Out[7]:
```
clean    195
buggy    129
Name: class, dtype: int64
```

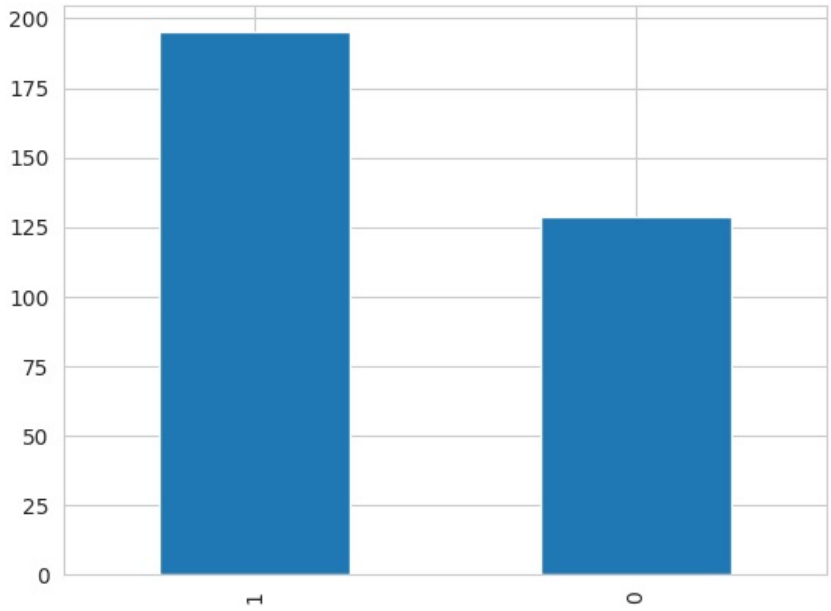
In [8]:
```
# load the Label Encoder function
label_encode = LabelEncoder()
```

In [9]:
```
labels = label_encode.fit_transform(df["class"])
```

In [10]:
```
# appending the labels to the DataFrame
df["class"] = labels
```

In [11]:
```
# label classification
df['class'].value_counts().plot.bar()
```

Out[11]:
```
<Axes: >
```



In [12]:
```
df.corr()
```

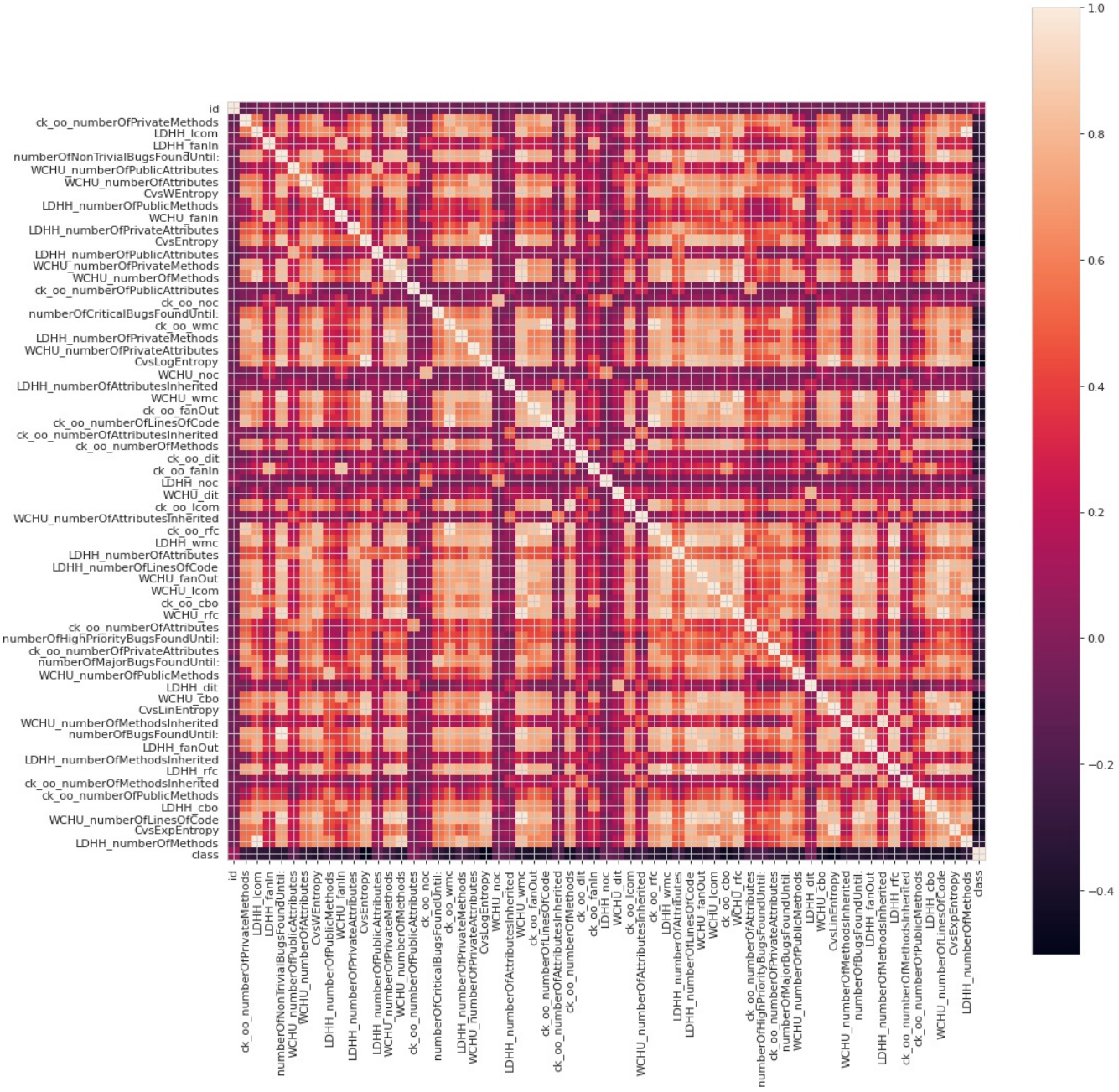
Out[12]:

	id	ck_oo_numberOfPrivateMethods	LDHH_lcom	LDHH_fanIn	numberOfNonTrivialBugsFoundUntil:
id	1.000000	-0.100516	-0.048358	0.013203	-0.061909
ck_oo_numberOfPrivateMethods	-0.100516	1.000000	0.569849	0.055677	0.715696
LDHH_lcom	-0.048358	0.569849	1.000000	0.272784	0.734496
LDHH_fanIn	0.013203	0.055677	0.272784	1.000000	0.301897
numberOfNonTrivialBugsFoundUntil:	-0.061909	0.715696	0.734496	0.301897	1.000000
...	...	...	...	...	...
LDHH_cbo	-0.030178	0.495247	0.689506	0.663645	0.709185
WCHU_numberOfLinesOfCode	-0.065483	0.718291	0.823965	0.278659	0.934818
CvsExpEntropy	-0.038868	0.553600	0.575086	0.301717	0.711977
LDHH_numberOfMethods	-0.049463	0.570174	0.998046	0.274576	0.738909
class	0.099331	-0.262637	-0.312516	-0.318088	-0.342137

63 rows × 63 columns

In [13]:
```
# plot corr
def plotCorrelationMatrix(df, graphWidth):
    df = df.dropna('columns') # drop columns with NaN
    df = df[[col for col in df if df[col].nunique() > 1]] # keep columns where there are more than 1 unique val
    if df.shape[1] < 2:
        print(f'No correlation plots shown: The number of non-NaN or constant columns ({df.shape[1]}) is less t
        return
    corr = df.corr()
    plt.figure(num=None, figsize=(graphWidth, graphWidth), dpi=80, facecolor='w', edgecolor='k')
    corrMat = plt.matshow(corr, fignum = 1)
    plt.xticks(range(len(corr.columns)), corr.columns, rotation=90)
    plt.yticks(range(len(corr.columns)), corr.columns)
    plt.gca().xaxis.tick_bottom()
    plt.colorbar(corrMat)
    plt.show()

plotCorrelationMatrix(df, 15)
```

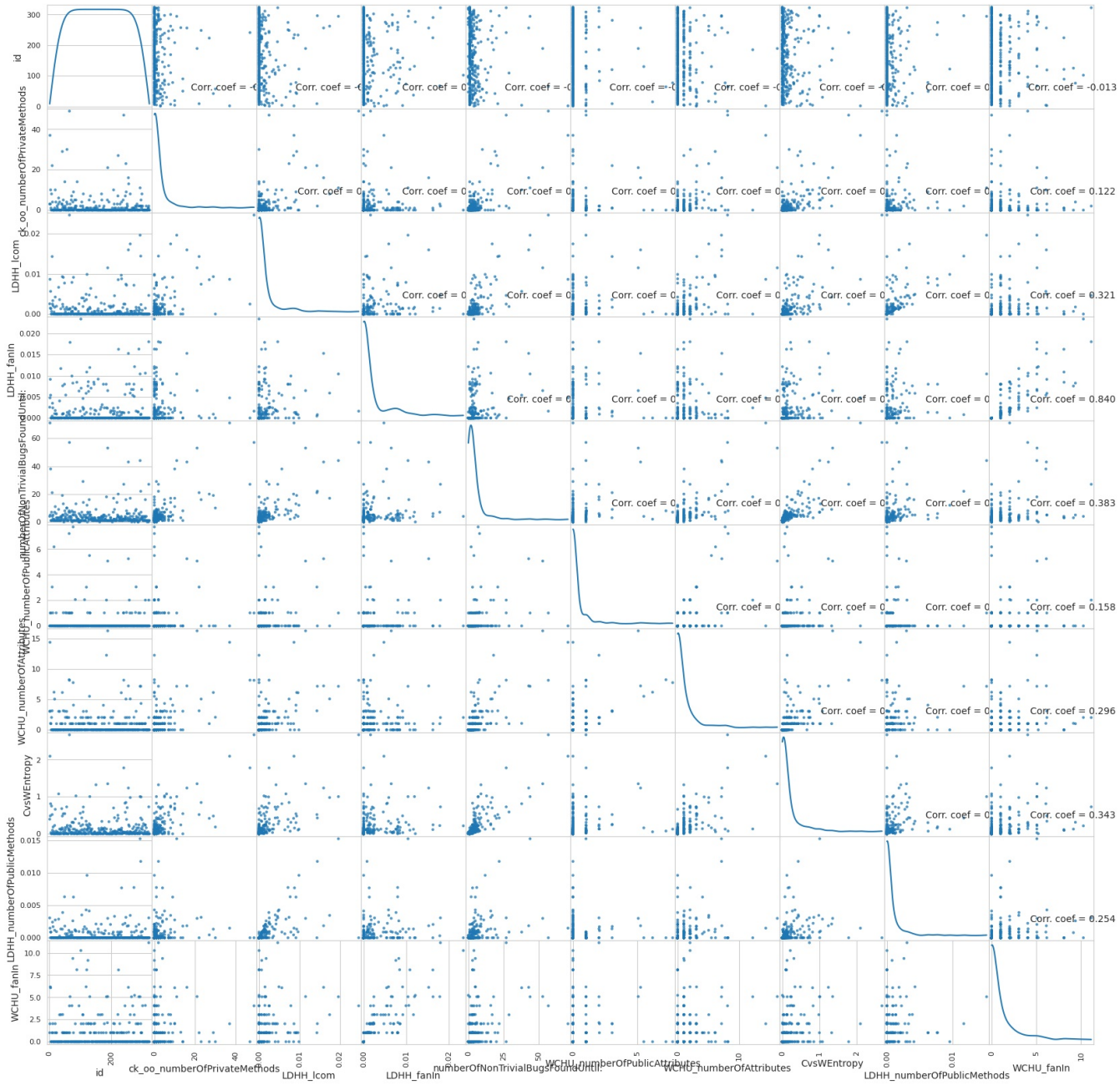
```
<ipython-input-13-4abd0cdf933f>:3: FutureWarning:

In a future version of pandas all arguments of DataFrame.dropna will be keyword-only.
```



In [14]:
```
# Scatter and density plots
def plotScatterMatrix(df, plotSize, textSize):
    df = df.select_dtypes(include =[np.number]) # keep only numerical columns
    # Remove rows and columns that would lead to df being singular
    df = df.dropna('columns')
    df = df[[col for col in df if df[col].nunique() > 1]] # keep columns where there are more than 1 unique val
    columnNames = list(df)
    if len(columnNames) > 10: # reduce the number of columns for matrix inversion of kernel density plots
        columnNames = columnNames[:10]
    df = df[columnNames]
    ax = pd.plotting.scatter_matrix(df, alpha=0.75, figsize=[plotSize, plotSize], diagonal='kde')
    corrs = df.corr().values
    for i, j in zip(*plt.np.triu_indices_from(ax, k = 1)):
        ax[i, j].annotate('Corr. coef = %.3f' % corrs[i, j], (0.8, 0.2), xycoords='axes fraction', ha='center',
    plt.suptitle('Scatter and Density Plot')
    plt.show()

plotScatterMatrix(df, 20, 10)
```

```
<ipython-input-14-6ec3fc198971>:5: FutureWarning:

In a future version of pandas all arguments of DataFrame.dropna will be keyword-only.
```

Scatter and Density Plot



NORMALIZATION

In [15]:
```
from sklearn import preprocessing
def change(X):
    length = X.shape[0]
    d=pd.Series(np.ones((length)))
    for i in range(0, length, 73):
        d[i]=0
    return d
def a(X):
    length = X.shape[0]
    d=pd.Series(np.zeros((length)))
    for i in range(0, length, 20):
        d[i]=1
    return d
```

In [16]:
```
def change(X):
    length = X.shape[0]
    d=pd.Series(np.ones((length)))
    for i in range(0, length, 73):
```

```
        d[i]=0
    return d
def a(X):
    length = X.shape[0]
    d=pd.Series(np.zeros((length)))
    for i in range(0, length, 20):
        d[i]=1
    return d
```

SMOTE-TOMEK

In [17]:
```
pip install -U imbalanced-learn
```

```
Requirement already satisfied: imbalanced-learn in /usr/local/lib/python3.10/dist-packages (0.10.1)
Collecting imbalanced-learn
  Downloading imbalanced_learn-0.11.0-py3-none-any.whl (235 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 235.6/235.6 kB 1.8 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn)
(1.23.5)
Requirement already satisfied: scipy>=1.5.0 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn)
(1.11.4)
Requirement already satisfied: scikit-learn>=1.0.2 in /usr/local/lib/python3.10/dist-packages (from imbalanced-
learn) (1.2.2)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn)
(1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from imbalanced
-learn) (3.2.0)
Installing collected packages: imbalanced-learn
  Attempting uninstall: imbalanced-learn
    Found existing installation: imbalanced-learn 0.10.1
    Uninstalling imbalanced-learn-0.10.1:
      Successfully uninstalled imbalanced-learn-0.10.1
Successfully installed imbalanced-learn-0.11.0
```

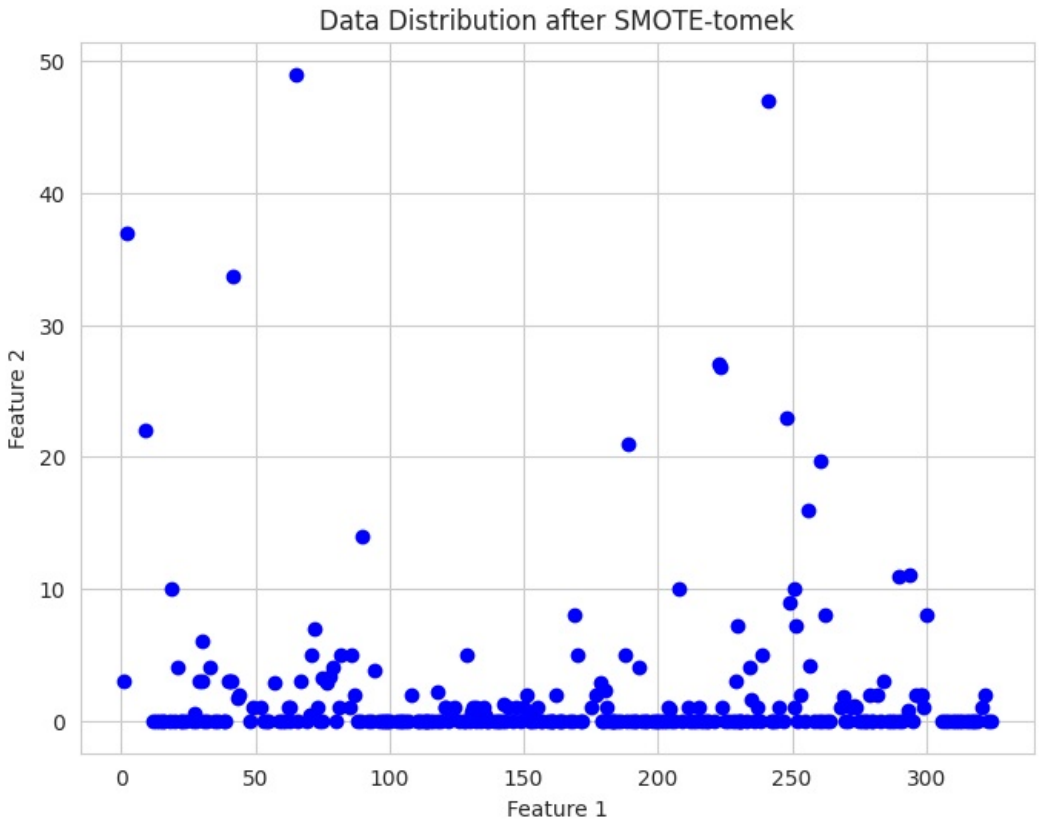
In [18]:
```
pip install imblearn
```

```
Collecting imblearn
  Downloading imblearn-0.0-py2.py3-none-any.whl (1.9 kB)
Requirement already satisfied: imbalanced-learn in /usr/local/lib/python3.10/dist-packages (from imblearn) (0.1
1.0)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn-
>imblearn) (1.23.5)
Requirement already satisfied: scipy>=1.5.0 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn->
imblearn) (1.11.4)
Requirement already satisfied: scikit-learn>=1.0.2 in /usr/local/lib/python3.10/dist-packages (from imbalanced-
learn->imblearn) (1.2.2)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn-
>imblearn) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from imbalanced
-learn->imblearn) (3.2.0)
Installing collected packages: imblearn
Successfully installed imblearn-0.0
```

In [19]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from imblearn.combine import SMOTETomek
from collections import Counter


# Assuming the last column is the target variable
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values

# 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)

# Apply SMOTE-Tomek only on the training data
smt = SMOTETomek(random_state=42)
X_resampled, y_resampled = smt.fit_resample(X_train, y_train)
```

In [20]:
```
from imblearn.combine import SMOTETomek

# Visualize the data distribution after applying SMOTE
plt.figure(figsize=(8, 6))
plt.scatter(X_resampled[:, 0], X_resampled[:, 1], c='blue', cmap=plt.cm.Paired, marker='o')
plt.title("Data Distribution after SMOTE-tomek")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```

```
<ipython-input-20-996aeb56b06b>:5: UserWarning:

No data for colormapping provided via 'c'. Parameters 'cmap' will be ignored
```



KNN

In [21]:
```
pip install sklearn
```

```
Collecting sklearn
  Downloading sklearn-0.0.post12.tar.gz (2.6 kB)
  error: subprocess-exited-with-error

  × python setup.py egg_info did not run successfully.
  │ exit code: 1
  └─> See above for output.

  note: This error originates from a subprocess, and is likely not a problem with pip.
  Preparing metadata (setup.py) ... error
error: metadata-generation-failed

× Encountered error while generating package metadata.
└─> See above for output.

note: This is an issue with the package mentioned above, not pip.
hint: See above for details.
```

In [22]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score
```

In [23]:
```
import math
math.sqrt(len(y_test))
```

Out[23]:
```
8.06225774829855
```

In [24]:
```
#Define the model
knn_model = KNeighborsClassifier(n_neighbors=8, p=2, metric='euclidean')
#fit model
knn_model.fit(X_resampled, y_resampled)
```

Out[24]:
```
KNeighborsClassifier(metric='euclidean', n_neighbors=8)
```

In [25]:
```
X_test.shape
```

Out[25]:
```
(65, 62)
```

In [26]:
```
# Assuming the last column is the target variable
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values


# 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)

# Apply SMOTE-Tomek only on the training data
smt = SMOTETomek(random_state=42)
X_resampled, y_resampled = smt.fit_resample(X_train, y_train)
```

In [27]:
```
# Predict the test set results
knn_pred = knn_model.predict(X_test)
y_pred = knn_model.predict(X_test)
y_pred
```

Out[27]:
```
array([0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0,
       0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1,
       0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0])
```

In [28]:
```
# Evaluate Model
cm = confusion_matrix(y_test, y_pred)
print (cm)
```

```
[[21  7]
 [22 15]]
```

In [29]:
```
print(accuracy_score(y_test, y_pred))
```

```
0.5538461538461539
```

DECISION TREE

In [30]:
```
#import the necessary packages
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
from sklearn import tree
import matplotlib.pyplot as plt
```

In [31]:
```
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values

# 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)
```

In [32]:
```
feature_names = X_train
dt_model = DecisionTreeClassifier(criterion='entropy',random_state=42)
dt_model.fit(X_train, y_train)
```

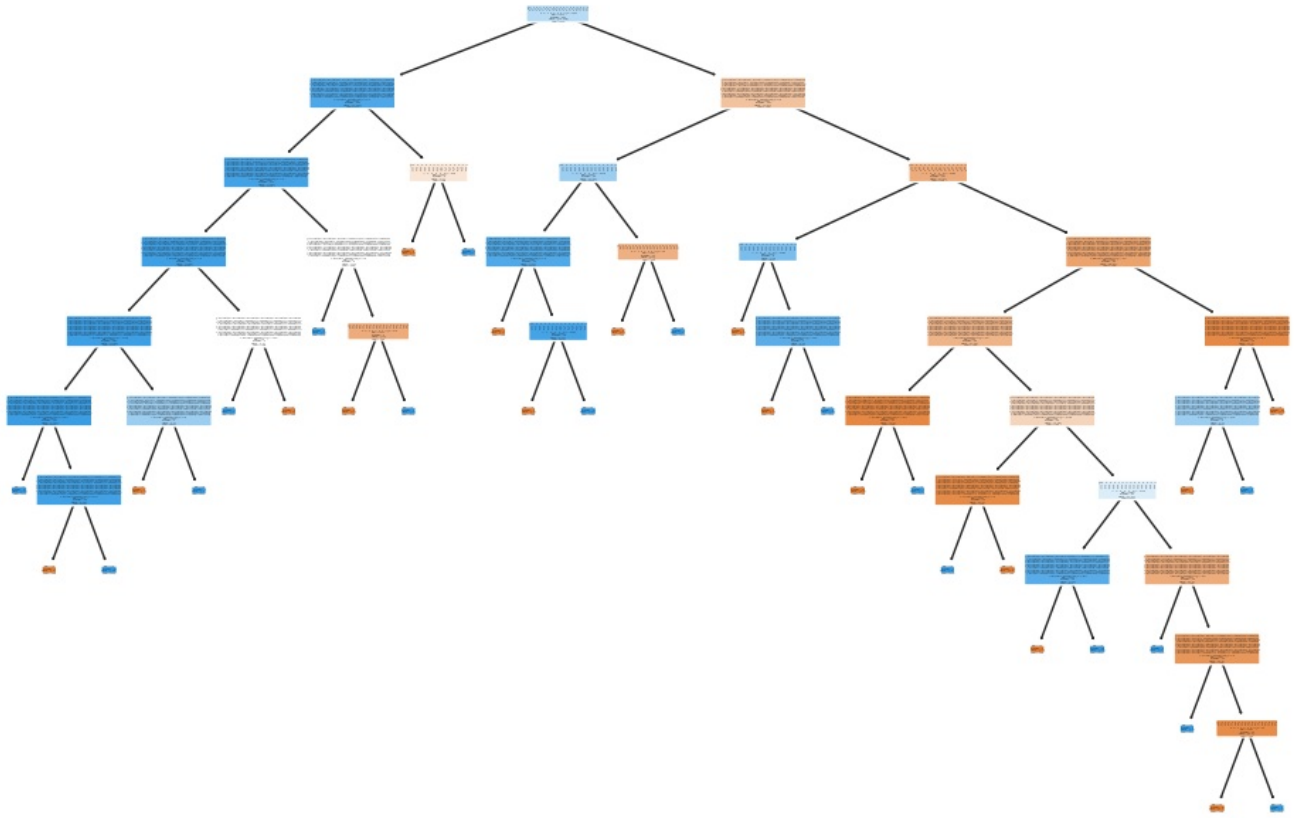
Out[32]:
```
DecisionTreeClassifier(criterion='entropy', random_state=42)
```

In [33]:
```
# Assuming the last column is the target variable
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values

# 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)

# Apply SMOTE-Tomek only on the training data
smt = SMOTETomek(random_state=42)
X_resampled, y_resampled = smt.fit_resample(X_train, y_train)
```

In [34]:
```
dt_model = DecisionTreeClassifier()
dt_model.fit(X_train, y_train)
dt_pred = dt_model.predict(X_test)
```

In [35]:
```
# Make predictions on the test set
y_pred = dt_model.predict(X_test)

# Calculate and print the accuracy of the classifier on the test set
accuracy = accuracy_score(y_test, y_test)
print(f"Accuracy: {accuracy:.2f}")

# Visualize the decision tree
plt.figure(figsize=(12, 8))
tree.plot_tree(dt_model, feature_names=feature_names, class_names=['True','False'], filled=True, rounded=True)
plt.show()
```

```
Accuracy: 1.00
```



In [36]:
```
#function to perform training with entropy
dt_model = DecisionTreeClassifier(criterion='entropy')
dt_model= dt_model.fit(X_resampled, y_resampled)
```

In [37]:
```
y_pred_en = dt_model.predict(X_test)
y_pred_en
```

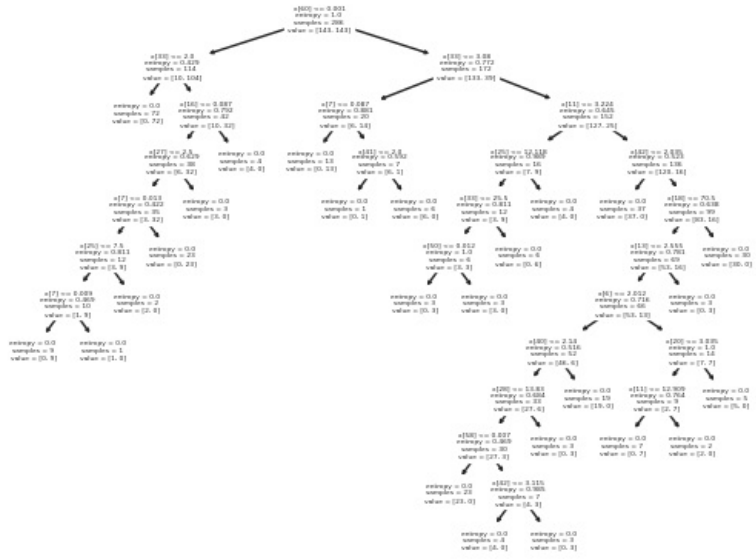
Out[37]:
```
array([1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1,
       1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0,
       1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0])
```

In [38]:
```
print ("Accuracy is "),metrics.accuracy_score(y_test,y_pred_en)*100
```

Out[38]:
```
Accuracy is
(None, 66.15384615384615)
```

In [39]:
```
from sklearn import tree
tree.plot_tree(dt_model)
```

Out[39]:
```
[Text(0.42045454545454547, 0.9583333333333334, 'x[60] <= 0.001\nentropy = 1.0\nsamples = 286\nvalue = [143, 143
]'),
 Text(0.22727272727272727, 0.875, 'x[33] <= 2.0\nentropy = 0.429\nsamples = 114\nvalue = [10, 104]'),
 Text(0.18181818181818182, 0.7916666666666666, 'entropy = 0.0\nsamples = 72\nvalue = [0, 72]'),
 Text(0.2727272727272727, 0.7916666666666666, 'x[16] <= 0.087\nentropy = 0.792\nsamples = 42\nvalue = [10, 32]'
),
 Text(0.22727272727272727, 0.7083333333333334, 'x[27] <= 2.5\nentropy = 0.629\nsamples = 38\nvalue = [6, 32]'),
 Text(0.18181818181818182, 0.625, 'x[7] <= 0.013\nentropy = 0.422\nsamples = 35\nvalue = [3, 32]'),
 Text(0.13636363636363635, 0.5416666666666666, 'x[25] <= 7.5\nentropy = 0.811\nsamples = 12\nvalue = [3, 9]'),
 Text(0.09090909090909091, 0.4583333333333333, 'x[7] <= 0.009\nentropy = 0.469\nsamples = 10\nvalue = [1, 9]'),
 Text(0.045454545454545456, 0.375, 'entropy = 0.0\nsamples = 9\nvalue = [0, 9]'),
 Text(0.13636363636363635, 0.375, 'entropy = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.18181818181818182, 0.4583333333333333, 'entropy = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(0.22727272727272727, 0.5416666666666666, 'entropy = 0.0\nsamples = 23\nvalue = [0, 23]'),
 Text(0.2727272727272727, 0.625, 'entropy = 0.0\nsamples = 3\nvalue = [3, 0]'),
 Text(0.3181818181818182, 0.7083333333333334, 'entropy = 0.0\nsamples = 4\nvalue = [4, 0]'),
 Text(0.6136363636363636, 0.875, 'x[33] <= 3.08\nentropy = 0.772\nsamples = 172\nvalue = [133, 39]'),
 Text(0.45454545454545453, 0.7916666666666666, 'x[7] <= 0.087\nentropy = 0.881\nsamples = 20\nvalue = [6, 14]')
,
 Text(0.4090909090909091, 0.7083333333333334, 'entropy = 0.0\nsamples = 13\nvalue = [0, 13]'),
 Text(0.5, 0.7083333333333334, 'x[41] <= 2.0\nentropy = 0.592\nsamples = 7\nvalue = [6, 1]'),
 Text(0.45454545454545453, 0.625, 'entropy = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.5454545454545454, 0.625, 'entropy = 0.0\nsamples = 6\nvalue = [6, 0]'),
 Text(0.7727272727272727, 0.7916666666666666, 'x[11] <= 3.224\nentropy = 0.645\nsamples = 152\nvalue = [127, 25
]'),
 Text(0.6818181818181818, 0.7083333333333334, 'x[25] <= 12.118\nentropy = 0.989\nsamples = 16\nvalue = [7, 9]')
,
 Text(0.6363636363636364, 0.625, 'x[33] <= 25.5\nentropy = 0.811\nsamples = 12\nvalue = [3, 9]'),
 Text(0.5909090909090909, 0.5416666666666666, 'x[50] <= 0.012\nentropy = 1.0\nsamples = 6\nvalue = [3, 3]'),
 Text(0.5454545454545454, 0.4583333333333333, 'entropy = 0.0\nsamples = 3\nvalue = [0, 3]'),
 Text(0.6363636363636364, 0.4583333333333333, 'entropy = 0.0\nsamples = 3\nvalue = [3, 0]'),
 Text(0.6818181818181818, 0.5416666666666666, 'entropy = 0.0\nsamples = 6\nvalue = [0, 6]'),
 Text(0.7272727272727273, 0.625, 'entropy = 0.0\nsamples = 4\nvalue = [4, 0]'),
 Text(0.8636363636363636, 0.7083333333333334, 'x[42] <= 2.035\nentropy = 0.523\nsamples = 136\nvalue = [120, 16
]'),
 Text(0.8181818181818182, 0.625, 'entropy = 0.0\nsamples = 37\nvalue = [37, 0]'),
 Text(0.9090909090909091, 0.625, 'x[18] <= 70.5\nentropy = 0.638\nsamples = 99\nvalue = [83, 16]'),
 Text(0.8636363636363636, 0.5416666666666666, 'x[13] <= 2.555\nentropy = 0.781\nsamples = 69\nvalue = [53, 16]'
),
 Text(0.8181818181818182, 0.4583333333333333, 'x[6] <= 2.012\nentropy = 0.716\nsamples = 66\nvalue = [53, 13]')
,
 Text(0.7727272727272727, 0.375, 'x[40] <= 2.14\nentropy = 0.516\nsamples = 52\nvalue = [46, 6]'),
 Text(0.6818181818181818, 0.2916666666666667, 'x[28] <= 13.83\nentropy = 0.684\nsamples = 33\nvalue = [27, 6]')
,
 Text(0.6363636363636364, 0.20833333333333334, 'x[58] <= 0.007\nentropy = 0.469\nsamples = 30\nvalue = [27, 3]'
),
 Text(0.5909090909090909, 0.125, 'entropy = 0.0\nsamples = 23\nvalue = [23, 0]'),
 Text(0.6818181818181818, 0.125, 'x[42] <= 3.115\nentropy = 0.985\nsamples = 7\nvalue = [4, 3]'),
 Text(0.6363636363636364, 0.041666666666666664, 'entropy = 0.0\nsamples = 4\nvalue = [4, 0]'),
 Text(0.7272727272727273, 0.041666666666666664, 'entropy = 0.0\nsamples = 3\nvalue = [0, 3]'),
 Text(0.7727272727272727, 0.20833333333333334, 'entropy = 0.0\nsamples = 3\nvalue = [0, 3]'),
 Text(0.8636363636363636, 0.2916666666666667, 'entropy = 0.0\nsamples = 19\nvalue = [19, 0]'),
 Text(0.9090909090909091, 0.375, 'x[20] <= 3.035\nentropy = 1.0\nsamples = 14\nvalue = [7, 7]'),
 Text(0.8636363636363636, 0.2916666666666667, 'x[11] <= 12.909\nentropy = 0.764\nsamples = 9\nvalue = [2, 7]'),
 Text(0.8181818181818182, 0.20833333333333334, 'entropy = 0.0\nsamples = 7\nvalue = [0, 7]'),
 Text(0.9090909090909091, 0.20833333333333334, 'entropy = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(0.9545454545454546, 0.2916666666666667, 'entropy = 0.0\nsamples = 5\nvalue = [5, 0]'),
 Text(0.9090909090909091, 0.4583333333333333, 'entropy = 0.0\nsamples = 3\nvalue = [0, 3]'),
 Text(0.9545454545454546, 0.5416666666666666, 'entropy = 0.0\nsamples = 30\nvalue = [30, 0]')]
```



SVM

In [40]:
```
from sklearn.svm import SVC
svm_model = SVC()
```

In [41]:
```
svm_model.fit(X_resampled, y_resampled)
```

Out[41]:
```
SVC()
```

In [42]:
```
svm_model = SVC()
svm_model.fit(X_train, y_train)
svm_pred = svm_model.predict(X_test)
```

In [43]:
```
svm_model.score(X_test, y_test)
```

Out[43]:
```
0.6
```

In [44]:
```
svm_model.predict(X_test)
```

Out[44]:
```
array([0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1])
```

In [45]:
```
#Tune parameters
#1. Regularization (C)
svm_model_C = SVC(C=1)
svm_model_C.fit(X_resampled, y_resampled)
svm_model_C.score(X_test, y_test)
```

Out[45]:
```
0.6307692307692307
```

In [46]:
```
svm_model_C = SVC(C=10)
svm_model_C.fit(X_resampled, y_resampled)
svm_model_C.score(X_test, y_test)
```

Out[46]:
```
0.6307692307692307
```

NAIVE

In [47]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report, accuracy_score
```

In [48]:
```
from sklearn.naive_bayes import GaussianNB

# Assuming X contains numerical features
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a Gaussian Naive Bayes classifier
nb_model = GaussianNB()

# Train the classifier
nb_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = nb_model.predict(X_test)

# Print classification report and accuracy
print("Classification Report:")
print(classification_report(y_test, y_pred))

print("Accuracy:", accuracy_score(y_test, y_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.75      0.43      0.55        28
           1       0.67      0.89      0.77        37

    accuracy                           0.69        65
   macro avg       0.71      0.66      0.66        65
weighted avg       0.71      0.69      0.67        65

Accuracy: 0.6923076923076923
```

In [49]:
```
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)
nb_pred = nb_model.predict(X_test)
```

GENETIC ALGORITHM BASED ON ARTIFICAL NEURAL NETWORK (GA-ANN)

In [50]:
```
pip install deap
```

```
Collecting deap
  Downloading deap-1.4.1-cp310-cp310-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_17_x86_64.manylinux2014
_x86_64.whl (135 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 135.4/135.4 kB 2.8 MB/s eta 0:00:00
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from deap) (1.23.5)
Installing collected packages: deap
Successfully installed deap-1.4.1
```

In [51]:

```python
import numpy as np
import tensorflow as tf
from tensorflow import keras
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from deap import base, creator, tools, algorithms
import matplotlib.pyplot as plt


# 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)

# Define genetic algorithm parameters
population_size = 10
num_generations = 5
mutation_rate = 0.1

# Function to create a simple feedforward neural network
def create_neural_network():
    model = keras.Sequential([
        keras.layers.Input(shape=(X_train.shape[1],)),
        keras.layers.Dense(64, activation='relu'),
        keras.layers.Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model

# Function to initialize a random population of neural networks
def initialize_population(population_size):
    return [create_neural_network() for _ in range(population_size)]

# Function to evaluate the fitness of each neural network in the population
def evaluate_population(population, X_train, y_train, X_test, y_test):
    fitness_scores = []
    for model in population:
        model.fit(X_train, y_train, epochs=5, verbose=0)
        predictions = (model.predict(X_test) > 0.5).astype(int).flatten()
        accuracy = accuracy_score(y_test, predictions)
        fitness_scores.append(accuracy)
    return fitness_scores

# Function for tournament selection
def tournament_selection(fitness_scores, tournament_size):
    selected_indices = []
    for _ in range(len(fitness_scores)):
        tournament_indices = np.random.choice(len(fitness_scores), tournament_size, replace=False)
        winner_index = max(tournament_indices, key=lambda i: fitness_scores[i])
        selected_indices.append(winner_index)
    return selected_indices

# Function for one-point crossover
def crossover(parent1, parent2):
    child1 = create_neural_network()
    child2 = create_neural_network()

    for layer in range(len(child1.layers)):
        crossover_point = np.random.randint(0, 2)
        if crossover_point == 0:
            child1.layers[layer].set_weights(parent1.layers[layer].get_weights())
            child2.layers[layer].set_weights(parent2.layers[layer].get_weights())
        else:
            child1.layers[layer].set_weights(parent2.layers[layer].get_weights())
            child2.layers[layer].set_weights(parent1.layers[layer].get_weights())

    return child1, child2

# Function for mutation
def mutate(model, mutation_rate):
    for layer in range(len(model.layers)):
        if np.random.rand() < mutation_rate:
            new_weights = [w + np.random.normal(0, 0.1, w.shape) for w in model.layers[layer].get_weights()]
            model.layers[layer].set_weights(new_weights)
    return model

# Main genetic algorithm loop
population = initialize_population(population_size)

# Lists to store evolution data
gen_numbers = []
max_accuracies = []
```

```python
for generation in range(num_generations):
    fitness_scores = evaluate_population(population, X_train, y_train, X_test, y_test)

    # Select parents using tournament selection
    selected_indices = tournament_selection(fitness_scores, tournament_size=2)

    # Create new generation using crossover and mutation
    new_population = []
    for i in range(0, len(selected_indices), 2):
        parent1 = population[selected_indices[i]]
        parent2 = population[selected_indices[i + 1]]
        child1, child2 = crossover(parent1, parent2)
        child1 = mutate(child1, mutation_rate)
        child2 = mutate(child2, mutation_rate)
        new_population.extend([child1, child2])

    population = new_population

    # Collect evolution data
    best_model = max(population, key=lambda model: accuracy_score(y_test, (model.predict(X_test) > 0.5).astype(
    max_accuracy = accuracy_score(y_test, (best_model.predict(X_test) > 0.5).astype(int).flatten())

    gen_numbers.append(generation)
    max_accuracies.append(max_accuracy)
```

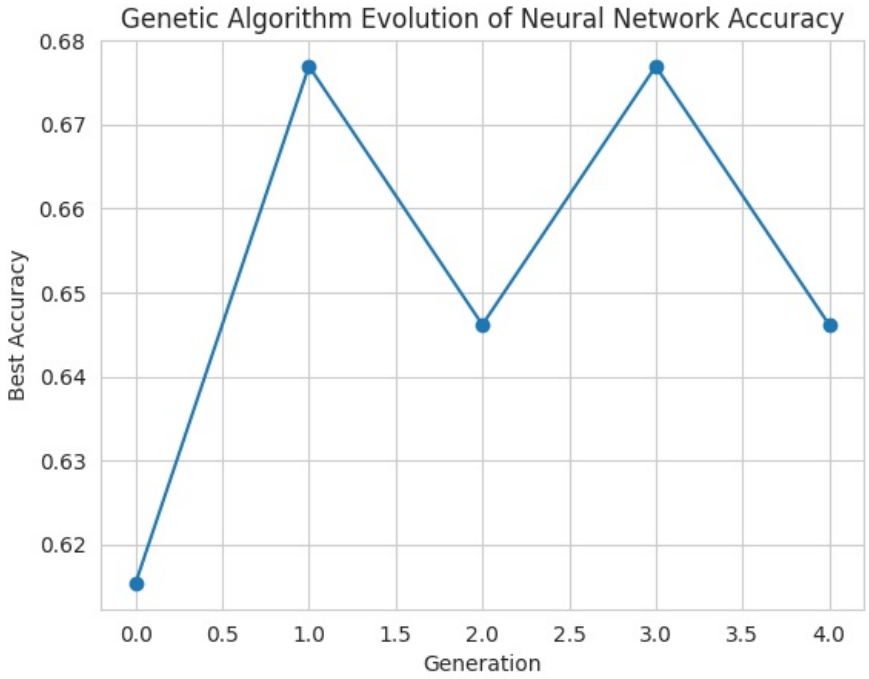
```
3/3 [==============================] - 0s 5ms/step
3/3 [==============================] - 0s 5ms/step
3/3 [==============================] - 0s 9ms/step
3/3 [==============================] - 0s 6ms/step
WARNING:tensorflow:5 out of the last 13 calls to <function Model.make_predict_function.<locals>.predict_functio
n at 0x79a8b4faa4d0> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings
could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3)
passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2
), @tf.function has reduce_retracing=True option that can avoid unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/
tf/function for  more details.
3/3 [==============================] - 0s 9ms/step
WARNING:tensorflow:5 out of the last 13 calls to <function Model.make_predict_function.<locals>.predict_functio
n at 0x79a8b4fabb50> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings
could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3)
passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2
), @tf.function has reduce_retracing=True option that can avoid unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/
tf/function for  more details.
3/3 [==============================] - 0s 7ms/step
3/3 [==============================] - 0s 8ms/step
3/3 [==============================] - 0s 13ms/step
3/3 [==============================] - 0s 5ms/step
3/3 [==============================] - 0s 5ms/step
3/3 [==============================] - 0s 8ms/step
3/3 [==============================] - 0s 8ms/step
3/3 [==============================] - 0s 7ms/step
3/3 [==============================] - 0s 7ms/step
3/3 [==============================] - 0s 8ms/step
3/3 [==============================] - 0s 4ms/step
3/3 [==============================] - 0s 4ms/step
3/3 [==============================] - 0s 4ms/step
3/3 [==============================] - 0s 4ms/step
3/3 [==============================] - 0s 4ms/step
3/3 [==============================] - 0s 5ms/step
3/3 [==============================] - 0s 9ms/step
3/3 [==============================] - 0s 7ms/step
3/3 [==============================] - 0s 4ms/step
3/3 [==============================] - 0s 8ms/step
3/3 [==============================] - 0s 6ms/step
3/3 [==============================] - 0s 3ms/step
3/3 [==============================] - 0s 4ms/step
3/3 [==============================] - 0s 4ms/step
3/3 [==============================] - 0s 4ms/step
3/3 [==============================] - 0s 5ms/step
3/3 [==============================] - 0s 4ms/step
3/3 [==============================] - 0s 5ms/step
3/3 [==============================] - 0s 5ms/step
3/3 [==============================] - 0s 4ms/step
3/3 [==============================] - 0s 4ms/step
3/3 [==============================] - 0s 5ms/step
3/3 [==============================] - 0s 4ms/step
3/3 [==============================] - 0s 5ms/step
3/3 [==============================] - 0s 5ms/step
3/3 [==============================] - 0s 5ms/step
3/3 [==============================] - 0s 3ms/step
3/3 [==============================] - 0s 4ms/step
3/3 [==============================] - 0s 7ms/step
3/3 [==============================] - 0s 4ms/step
3/3 [==============================] - 0s 5ms/step
3/3 [==============================] - 0s 4ms/step
3/3 [==============================] - 0s 4ms/step
3/3 [==============================] - 0s 4ms/step
```

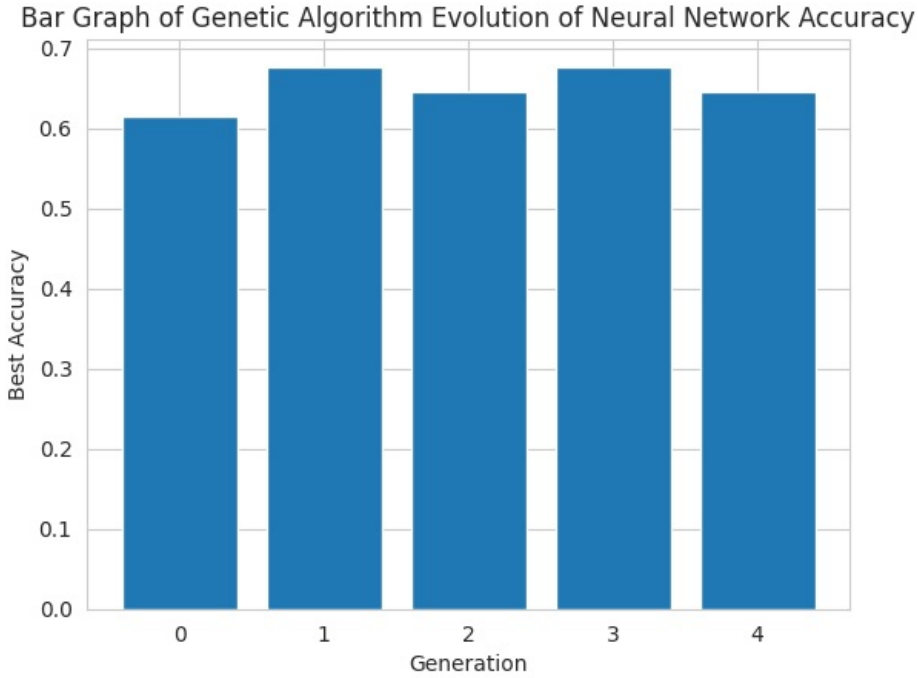
```
3/3 [==============================] - 0s 4ms/step
3/3 [==============================] - 0s 4ms/step
3/3 [==============================] - 0s 5ms/step
3/3 [==============================] - 0s 4ms/step
3/3 [==============================] - 0s 4ms/step
3/3 [==============================] - 0s 8ms/step
3/3 [==============================] - 0s 3ms/step
3/3 [==============================] - 0s 5ms/step
3/3 [==============================] - 0s 5ms/step
3/3 [==============================] - 0s 4ms/step
3/3 [==============================] - 0s 5ms/step
3/3 [==============================] - 0s 4ms/step
3/3 [==============================] - 0s 4ms/step
3/3 [==============================] - 0s 4ms/step
3/3 [==============================] - 0s 4ms/step
3/3 [==============================] - 0s 4ms/step
3/3 [==============================] - 0s 4ms/step
3/3 [==============================] - 0s 6ms/step
3/3 [==============================] - 0s 4ms/step
3/3 [==============================] - 0s 6ms/step
3/3 [==============================] - 0s 5ms/step
3/3 [==============================] - 0s 5ms/step
3/3 [==============================] - 0s 4ms/step
3/3 [==============================] - 0s 4ms/step
3/3 [==============================] - 0s 5ms/step
3/3 [==============================] - 0s 5ms/step
3/3 [==============================] - 0s 4ms/step
3/3 [==============================] - 0s 5ms/step
3/3 [==============================] - 0s 4ms/step
3/3 [==============================] - 0s 5ms/step
3/3 [==============================] - 0s 5ms/step
3/3 [==============================] - 0s 4ms/step
3/3 [==============================] - 0s 4ms/step
3/3 [==============================] - 0s 4ms/step
3/3 [==============================] - 0s 5ms/step
3/3 [==============================] - 0s 5ms/step
3/3 [==============================] - 0s 4ms/step
3/3 [==============================] - 0s 7ms/step
3/3 [==============================] - 0s 6ms/step
3/3 [==============================] - 0s 5ms/step
3/3 [==============================] - 0s 4ms/step
3/3 [==============================] - 0s 4ms/step
3/3 [==============================] - 0s 4ms/step
3/3 [==============================] - 0s 5ms/step
3/3 [==============================] - 0s 5ms/step
3/3 [==============================] - 0s 6ms/step
3/3 [==============================] - 0s 4ms/step
3/3 [==============================] - 0s 4ms/step
3/3 [==============================] - 0s 5ms/step
3/3 [==============================] - 0s 5ms/step
3/3 [==============================] - 0s 5ms/step
3/3 [==============================] - 0s 5ms/step
3/3 [==============================] - 0s 5ms/step
3/3 [==============================] - 0s 4ms/step
3/3 [==============================] - 0s 5ms/step
3/3 [==============================] - 0s 4ms/step
```

In [52]:
```python
# Select the best neural network from the final population
best_model = max(population, key=lambda model: accuracy_score(y_test, (model.predict(X_test) > 0.5).astype(int)
print("Best Neural Network Accuracy:", accuracy_score(y_test, (best_model.predict(X_test) > 0.5).astype(int).fl
```

```
3/3 [==============================] - 0s 4ms/step
3/3 [==============================] - 0s 4ms/step
3/3 [==============================] - 0s 5ms/step
3/3 [==============================] - 0s 4ms/step
3/3 [==============================] - 0s 4ms/step
3/3 [==============================] - 0s 4ms/step
3/3 [==============================] - 0s 4ms/step
3/3 [==============================] - 0s 5ms/step
3/3 [==============================] - 0s 4ms/step
3/3 [==============================] - 0s 4ms/step
3/3 [==============================] - 0s 4ms/step
Best Neural Network Accuracy: 0.6461538461538462
```

In [53]:
```python
# Plot the evolution of accuracy
plt.plot(gen_numbers, max_accuracies, marker='o')
plt.xlabel('Generation')
plt.ylabel('Best Accuracy')
plt.title('Genetic Algorithm Evolution of Neural Network Accuracy')
plt.show()

# Plot bar graph for accuracy
plt.bar(range(len(max_accuracies)), max_accuracies)
plt.xlabel('Generation')
plt.ylabel('Best Accuracy')
plt.title('Bar Graph of Genetic Algorithm Evolution of Neural Network Accuracy')
plt.show()
```



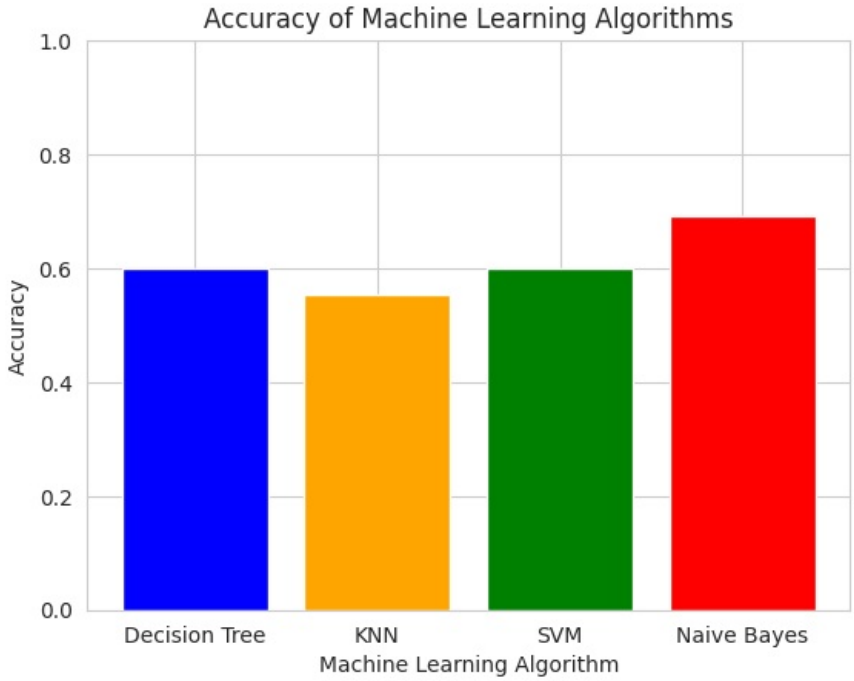


EVALUATING OTHER ALGORITHMS

In [54]:

```
# Evaluate models and store results

models = {'Decision Tree': dt_pred,
          'KNN': knn_pred,
          'SVM': svm_pred,
          'Naive Bayes': nb_pred}

model_results = {}
for name, pred in models.items():
    acc = accuracy_score(y_test, pred)
    model_results[name] = acc
    cm = confusion_matrix(y_test, pred)
    print(f"Model: {name}")
    print(f"Accuracy: {acc:.4f}")
    print(f"Confusion Matrix:\n{cm}\n")

# Plot bar graph
plt.bar(model_results.keys(), model_results.values(), color=['blue', 'orange', 'green', 'red', 'purple'])
plt.xlabel('Machine Learning Algorithm')
plt.ylabel('Accuracy')
plt.title('Accuracy of Machine Learning Algorithms')
plt.ylim([0, 1])  # Set the y-axis limit to better visualize differences
plt.show()
```

```
Model: Decision Tree
Accuracy: 0.6000
Confusion Matrix:
[[12 16]
 [10 27]]

Model: KNN
Accuracy: 0.5538
Confusion Matrix:
[[21  7]
 [22 15]]

Model: SVM
Accuracy: 0.6000
Confusion Matrix:
[[10 18]
 [ 8 29]]

Model: Naive Bayes
Accuracy: 0.6923
Confusion Matrix:
[[12 16]
 [ 4 33]]
```



IMPORTING DATASETS AND LIBRARIES

In [3]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
```

In [4]:
```
# load data
data_churn = pd.read_csv('/content/churn.csv', sep='\s;\s*', engine='python')
data_entry = pd.read_csv('/content/ent.csv', sep='\s;\s*', engine='python')
data_bugs = pd.read_csv('/content/bug-metrics.csv', sep='\s;\s*', engine='python')
data_change = pd.read_csv('/content/change-metrics.csv', sep='\s;\s*', engine='python')
data_complexity_change = pd.read_csv('/content/complexity-code-change.csv', sep='\s;\s*', engine='python')
data_single_version_ck_oo = pd.read_csv('/content/single-version-ck-oo.csv', sep='\s;\s*', engine='python')

data = data_churn.merge(data_entry, how='left')\
        .merge(data_bugs, how='left')\
        .merge(data_change, how='left')\
        .merge(data_complexity_change, how='left')\
        .merge(data_single_version_ck_oo, how='left')


def remove_unnamed_cols(data):
    return data.loc[:, ~data.columns.str.contains('^Unnamed')]


# merge data
data = data_churn.merge(data_entry, how='left')\
        .merge(data_bugs, how='left')\
        .merge(data_change, how='left')\
        .merge(data_complexity_change, how='left')\
        .merge(data_single_version_ck_oo, how='left')

#remove unnamed columns
data = remove_unnamed_cols(data)

# add defect column
data['defect'] = data['bugs'] > 0

data
```

```
<ipython-input-4-b35cac13740c>:9: UserWarning: You are merging on int and float columns where the float values
are not equal to their int representation.
  data = data_churn.merge(data_entry, how='left')\
<ipython-input-4-b35cac13740c>:21: UserWarning: You are merging on int and float columns where the float values
are not equal to their int representation.
  data = data_churn.merge(data_entry, how='left')\
```

Out[4]:

	classname	cbo	dit	fanIn	fanOut	lcom	noc	numberOfAttributes	numberOfAttributesInherited	numberO
0	org::eclipse::jdt::internal::core::search::ind...	0	0	0	0	0	0	0	0	
1	org::eclipse::jdt::internal::compiler::codegen...	0	0	0	0	0	0	0	0	
2	org::eclipse::jdt::internal::compiler::ast::AS...	14	2	12	4	19	0	37	128	
3	org::eclipse::jdt::internal::compiler::lookup:...	3	2	0	3	4	0	0	412	
4	org::eclipse::jdt::internal::eval::CodeSnippet...	0	2	0	0	117	0	1	108	
...	...	...	...	...	...	...	...	...	...	
5366	org::eclipse::pde::internal::ui::editor::text:...	0	0	0	0	0	0	0	0	
5367	org::eclipse::pde::internal::core::builders::E...	0	0	0	0	0	0	0	0	
5368	org::eclipse::pde::internal::core::builders::D...	2	0	0	2	0	0	0	0	
5369	org::eclipse::pde::internal::ui::templates::id...	0	0	0	0	0	0	0	0	
5370	org::eclipse::pde::internal::core::plugin::Abb...	0	0	0	0	0	0	0	0	

5371 rows × 49 columns

In [5]:
```
import matplotlib.pyplot as plt
%matplotlib inline

import seaborn as sns
sns.set_style('whitegrid')

from tqdm import tqdm
```

EDA

- import some dependencies to plot
- use plotly to visualization
  - label classification

- count and plot(visualization)
- value visualization
  - use historgram to visualization attribution
  - relationship
    - covariance
    - heatmap
- scatter

In [6]:
```
# import some dependencies for plotting

from plotly.offline import iplot
import plotly.graph_objs as go
```

In [7]:
```
# check data
def show_info(data, is_matrix_transpose=False):
    # basic shape
    print('data shape is: {}   sample number {}   attribute number {}\n'.format(data.shape, data.shape[0], data
    # attribute(key)
    print('data columns number {}  \nall columns: {}\n'.format(len(data.columns) ,data.columns))
    # value's null
    print('data all attribute count null:\n', data.isna().sum())
    # data value analysis and data demo
    if is_matrix_transpose:
        print('data value analysis: ', data.describe().T)
        print('data demo without matrix transpose: ', data.head().T)
    else:
        print('data value analysis: ', data.describe())
        print('data demo without matrix transpose: ', data.head())

show_info(data)
```

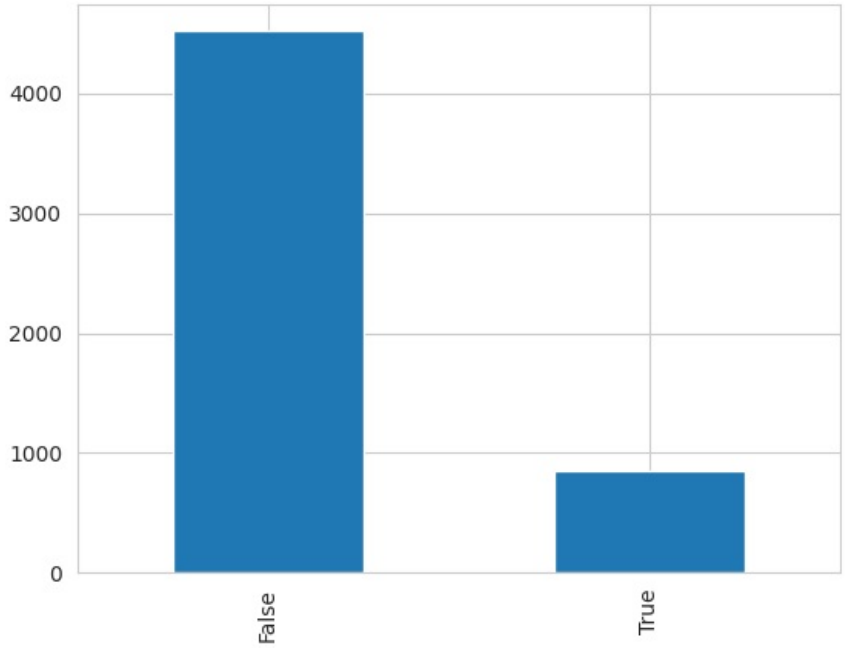
```
data shape is: (5371, 49)   sample number 5371   attribute number 49

data columns number 49
all columns: Index(['classname', 'cbo', 'dit', 'fanIn', 'fanOut', 'lcom', 'noc',
       'numberOfAttributes', 'numberOfAttributesInherited',
       'numberOfLinesOfCode', 'numberOfMethods', 'numberOfMethodsInherited',
       'numberOfPrivateAttributes', 'numberOfPrivateMethods',
       'numberOfPublicAttributes', 'numberOfPublicMethods', 'rfc', 'wmc',
       'bugs', 'nonTrivialBugs', 'majorBugs', 'criticalBugs',
       'highPriorityBugs', 'numberOfBugsFoundUntil:',
       'numberOfNonTrivialBugsFoundUntil:', 'numberOfMajorBugsFoundUntil:',
       'numberOfCriticalBugsFoundUntil:',
       'numberOfHighPriorityBugsFoundUntil:', 'numberOfVersionsUntil:',
       'numberOfFixesUntil:', 'numberOfRefactoringsUntil:',
       'numberOfAuthorsUntil:', 'linesAddedUntil:', 'maxLinesAddedUntil:',
       'avgLinesAddedUntil:', 'linesRemovedUntil:', 'maxLinesRemovedUntil:',
       'avgLinesRemovedUntil:', 'codeChurnUntil:', 'maxCodeChurnUntil:',
       'avgCodeChurnUntil:', 'ageWithRespectTo:', 'weightedAgeWithRespectTo:',
       'CvsEntropy', 'CvsWEntropy', 'CvsLinEntropy', 'CvsLogEntropy',
       'CvsExpEntropy', 'defect'],
      dtype='object')

data all attribute count null:
 classname                              0
cbo                                     0
dit                                     0
fanIn                                   0
fanOut                                  0
lcom                                    0
noc                                     0
numberOfAttributes                      0
numberOfAttributesInherited             0
numberOfLinesOfCode                     0
numberOfMethods                         0
numberOfMethodsInherited                0
numberOfPrivateAttributes               0
numberOfPrivateMethods                  0
numberOfPublicAttributes                0
numberOfPublicMethods                   0
rfc                                     0
wmc                                     0
bugs                                    0
nonTrivialBugs                          0
majorBugs                               0
criticalBugs                            0
highPriorityBugs                        0
numberOfBugsFoundUntil:                 0
numberOfNonTrivialBugsFoundUntil:       0
numberOfMajorBugsFoundUntil:            0
numberOfCriticalBugsFoundUntil:         0
numberOfHighPriorityBugsFoundUntil:     0
numberOfVersionsUntil:                  0
numberOfFixesUntil:                     0
numberOfRefactoringsUntil:              0
```

```
numberOfAuthorsUntil:                 0
linesAddedUntil:                      0
maxLinesAddedUntil:                   0
avgLinesAddedUntil:                   0
linesRemovedUntil:                    0
maxLinesRemovedUntil:                 0
avgLinesRemovedUntil:                 0
codeChurnUntil:                       0
maxCodeChurnUntil:                    0
avgCodeChurnUntil:                    0
ageWithRespectTo:                     0
weightedAgeWithRespectTo:             0
CvsEntropy                            0
CvsWEntropy                           0
CvsLinEntropy                         0
CvsLogEntropy                         0
CvsExpEntropy                         0
defect                                0
dtype: int64
data value analysis:                 cbo          dit        fanIn       fanOut          lcom  \
count  5371.000000  5371.000000  5371.000000  5371.000000   5371.000000
mean      3.967231     0.150624     2.097002     2.112270     53.681251
std      11.160771     0.499375     9.950945     4.415507    649.639293
min       0.000000     0.000000     0.000000     0.000000      0.000000
25%       0.000000     0.000000     0.000000     0.000000      0.000000
50%       1.000000     0.000000     0.000000     0.000000      0.000000
75%       4.000000     0.000000     1.000000     2.000000      9.000000
max     358.000000     4.000000   358.000000    60.000000  35248.000000

               noc  numberOfAttributes  numberOfAttributesInherited  \
count  5371.000000         5371.000000                  5371.000000
mean      0.180786            1.617017                     8.803389
std       1.212402           21.462467                    39.270372
min       0.000000            0.000000                     0.000000
25%       0.000000            0.000000                     0.000000
50%       0.000000            0.000000                     0.000000
75%       0.000000            1.000000                     0.000000
max      41.000000         1518.000000                   567.000000

       numberOfLinesOfCode  numberOfMethods  ...  codeChurnUntil:  \
count          5371.000000      5371.000000  ...      5371.000000
mean             37.164960         1.799292  ...        78.297338
std             147.193913         5.893746  ...       364.664466
min               0.000000         0.000000  ...     -1745.000000
25%               0.000000         0.000000  ...         0.000000
50%               2.000000         0.000000  ...         4.000000
75%              21.000000         1.000000  ...        45.000000
max            4997.000000       241.000000  ...     10624.000000

       maxCodeChurnUntil:  avgCodeChurnUntil:  ageWithRespectTo:  \
count         5371.000000         5371.000000        5371.000000
mean            41.116366            3.064683         150.921026
std            109.179820           10.560036         112.614942
min              0.000000         -133.800000           0.000000
25%              1.000000            0.000000          51.857100
50%              9.000000            0.600000         125.857000
75%             38.000000            3.398275         227.429000
max           2768.000000          312.625000         381.000000

       weightedAgeWithRespectTo:   CvsEntropy  CvsWEntropy  CvsLinEntropy  \
count                5371.000000  5371.000000  5371.000000    5371.000000
mean                   45.221031     7.022268     0.050255       0.065986
std                    46.554970     7.068468     0.154135       0.073452
min                     0.000000     0.000000     0.000000       0.000000
25%                     8.145750     1.847030     0.001875       0.009619
50%                    32.458100     5.076520     0.007752       0.027211
75%                    67.783600     9.459490     0.034558       0.140346
max                   314.776000    58.151800     4.490230       0.329150

       CvsLogEntropy  CvsExpEntropy
count    5371.000000    5371.000000
mean        2.801887       0.116143
std         4.048816       0.131436
min         0.000000       0.000000
25%         0.095785       0.005732
50%         0.209384       0.051280
75%         8.254325       0.205382
max        10.250000       0.727558

[8 rows x 47 columns]
data demo without matrix transpose:                                                 classname  cbo  dit  fanIn  fan
Out  \
0  org::eclipse::jdt::internal::core::search::ind...    0    0      0       0
1  org::eclipse::jdt::internal::compiler::codegen...    0    0      0       0
2  org::eclipse::jdt::internal::compiler::ast::AS...   14    2     12       4
3  org::eclipse::jdt::internal::compiler::lookup:...    3    2      0       3
4  org::eclipse::jdt::internal::eval::CodeSnippet...    0    2      0       0

   lcom  noc  numberOfAttributes  numberOfAttributesInherited  \
```

```
0     0    0                  0                          0
1     0    0                  0                          0
2    19    0                 37                        128
3     4    0                  0                        412
4   117    0                  1                        108

   numberOfLinesOfCode  ...  maxCodeChurnUntil:  avgCodeChurnUntil:  \
0                  6.0  ...                15.0            0.646154
1                  0.0  ...                10.0            5.000000
2                261.0  ...                66.0            4.941670
3                 25.0  ...                23.0            0.535714
4                 26.0  ...                18.0           -0.118280

   ageWithRespectTo:  weightedAgeWithRespectTo:  CvsEntropy  CvsWEntropy  \
0            350.571                    63.0609    10.91310     0.008302
1            117.714                     0.0000     1.10349     0.000657
2            238.429                   104.2040    37.86060     0.228509
3            367.000                   112.6780     5.86013     0.009105
4            367.000                    54.9702    13.59600     0.016005

   CvsLinEntropy  CvsLogEntropy  CvsExpEntropy  defect
0       0.014767       0.253257       0.001125   False
1       0.001886       0.027116       0.000351   False
2       0.106180       1.028400       0.210412    True
3       0.010113       0.143680       0.003485   False
4       0.025751       0.338602       0.021378   False

[5 rows x 49 columns]
```

In [8]:
```
# label classification
data['defect'].value_counts().plot.bar()
```

Out[8]:
```
<Axes: >
```



In [9]:
```
data.corr()
```

```
<ipython-input-9-c44ded798807>:1: FutureWarning:

The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to Fals
e. Select only valid columns or specify the value of numeric_only to silence this warning.
```

Out[9]:

	cbo	dit	fanIn	fanOut	lcom	noc	numberOfAttributes	numberOfAttributesInh
cbo	1.000000	0.035233	0.933357	0.557145	0.166317	0.188841	0.056084	0.0
dit	0.035233	1.000000	0.000507	0.103724	0.031666	-0.006230	0.011430	0.6
fanIn	0.933357	0.000507	1.000000	0.234934	0.119034	0.210581	0.038859	0.0
fanOut	0.557145	0.103724	0.234934	1.000000	0.202514	0.022332	0.069263	0.0
lcom	0.166317	0.031666	0.119034	0.202514	1.000000	0.025038	0.036231	0.0
noc	0.188841	-0.006230	0.210581	0.022332	0.025038	1.000000	0.002983	-0.0
numberOfAttributes	0.056084	0.011430	0.038859	0.069263	0.036231	0.002983	1.000000	0.0
numberOfAttributesInherited	0.017822	0.692571	0.018481	0.018450	0.063001	-0.005546	0.001374	1.0
numberOfLinesOfCode	0.347392	0.108146	0.191911	0.542502	0.537017	0.036516	0.085556	0.1
numberOfMethods	0.418696	0.103156	0.271077	0.554598	0.754292	0.075599	0.079413	0.0
numberOfMethodsInherited	0.055780	0.233807	0.017359	0.114704	0.035535	0.001429	0.003391	0.1
numberOfPrivateAttributes	0.340916	0.070425	0.209819	0.497231	0.177215	0.048724	0.122716	-0.0
numberOfPrivateMethods	0.379922	0.041502	0.199071	0.619381	0.295200	0.038332	0.087651	-0.0
numberOfPublicAttributes	0.000408	0.001704	0.000743	0.001123	0.011344	-0.002381	0.985504	0.0
numberOfPublicMethods	0.335698	0.080402	0.262147	0.338633	0.766038	0.066696	0.047681	0.0
rfc	0.422978	0.094115	0.219660	0.682330	0.530881	0.034212	0.086722	0.0
wmc	0.344814	0.118015	0.194204	0.527233	0.548440	0.031937	0.087191	0.1
bugs	0.217341	0.076156	0.140568	0.313515	0.218869	0.013024	0.481754	0.1
nonTrivialBugs	0.093990	-0.003199	0.062564	0.130183	0.073738	0.008666	0.130975	-0.0
majorBugs	0.079043	0.058270	0.041096	0.137806	0.211652	-0.004456	0.030287	0.0
criticalBugs	0.045649	0.042776	0.012000	0.109222	0.206611	-0.009117	0.014480	0.0
highPriorityBugs	0.064968	0.013069	0.026693	0.127966	0.018157	0.001635	0.025871	-0.0
numberOfBugsFoundUntil:	0.342341	0.159303	0.203068	0.511610	0.301225	0.030047	0.317469	0.1
numberOfNonTrivialBugsFoundUntil:	0.317629	0.180740	0.195629	0.462568	0.326316	0.028751	0.264851	0.2
numberOfMajorBugsFoundUntil:	0.254306	0.175574	0.158304	0.372457	0.250872	0.008051	0.117267	0.2
numberOfCriticalBugsFoundUntil:	0.232717	0.210163	0.160405	0.307110	0.216808	0.002230	0.044354	0.2
numberOfHighPriorityBugsFoundUntil:	0.275976	0.043522	0.169078	0.386306	0.050563	0.028196	0.033111	-0.0
numberOfVersionsUntil:	0.317963	0.245258	0.200326	0.447922	0.287269	0.025574	0.235910	0.3
numberOfFixesUntil:	0.134348	0.165657	0.082243	0.196288	0.228945	0.003833	0.077554	0.2
numberOfRefactoringsUntil:	0.193990	-0.016906	0.119147	0.272406	0.053741	0.013614	0.199977	-0.0
numberOfAuthorsUntil:	0.249877	0.256617	0.180307	0.273451	0.112535	0.088373	0.075545	0.3
linesAddedUntil:	0.175596	0.140585	0.107211	0.260100	0.287070	0.008852	0.078876	0.1
maxLinesAddedUntil:	0.197073	0.136946	0.119729	0.281886	0.325285	0.010219	0.056340	0.2
avgLinesAddedUntil:	0.111594	0.048668	0.055650	0.179416	0.146953	0.011648	0.024550	0.0
linesRemovedUntil:	0.153719	0.146614	0.092217	0.232789	0.261654	0.003889	0.063589	0.2
maxLinesRemovedUntil:	0.196175	0.144483	0.118634	0.280448	0.347073	0.008853	0.067496	0.2
avgLinesRemovedUntil:	0.105836	0.072461	0.051297	0.174624	0.169710	0.006945	0.022662	0.0
codeChurnUntil:	0.211679	0.070892	0.136096	0.292217	0.302724	0.026836	0.117947	0.1
maxCodeChurnUntil:	0.228438	0.090138	0.126307	0.350502	0.221036	0.026679	0.067623	0.0
avgCodeChurnUntil:	0.064187	-0.021707	0.034808	0.094761	0.027202	0.014450	0.015295	-0.0
ageWithRespectTo:	0.144091	0.262915	0.111637	0.137886	0.063641	0.065618	0.030947	0.3
weightedAgeWithRespectTo:	0.208346	0.143669	0.173898	0.170510	0.077373	0.111496	0.041086	0.1
CvsEntropy	0.404418	0.248731	0.229857	0.596501	0.205404	0.063902	0.182797	0.2
CvsWEntropy	0.349864	0.073899	0.232140	0.473875	0.305561	0.051269	0.132816	0.0
CvsLinEntropy	0.214707	0.092957	0.128918	0.303978	0.085694	0.023873	0.076798	0.0
CvsLogEntropy	0.097432	0.065730	0.054238	0.147454	0.033506	0.001952	0.020745	0.0
CvsExpEntropy	0.255337	0.072452	0.157843	0.352024	0.096066	0.031716	0.093553	0.0
defect	0.186688	0.070919	0.111516	0.272658	0.118456	0.008736	0.086060	0.0

48 rows × 48 columns

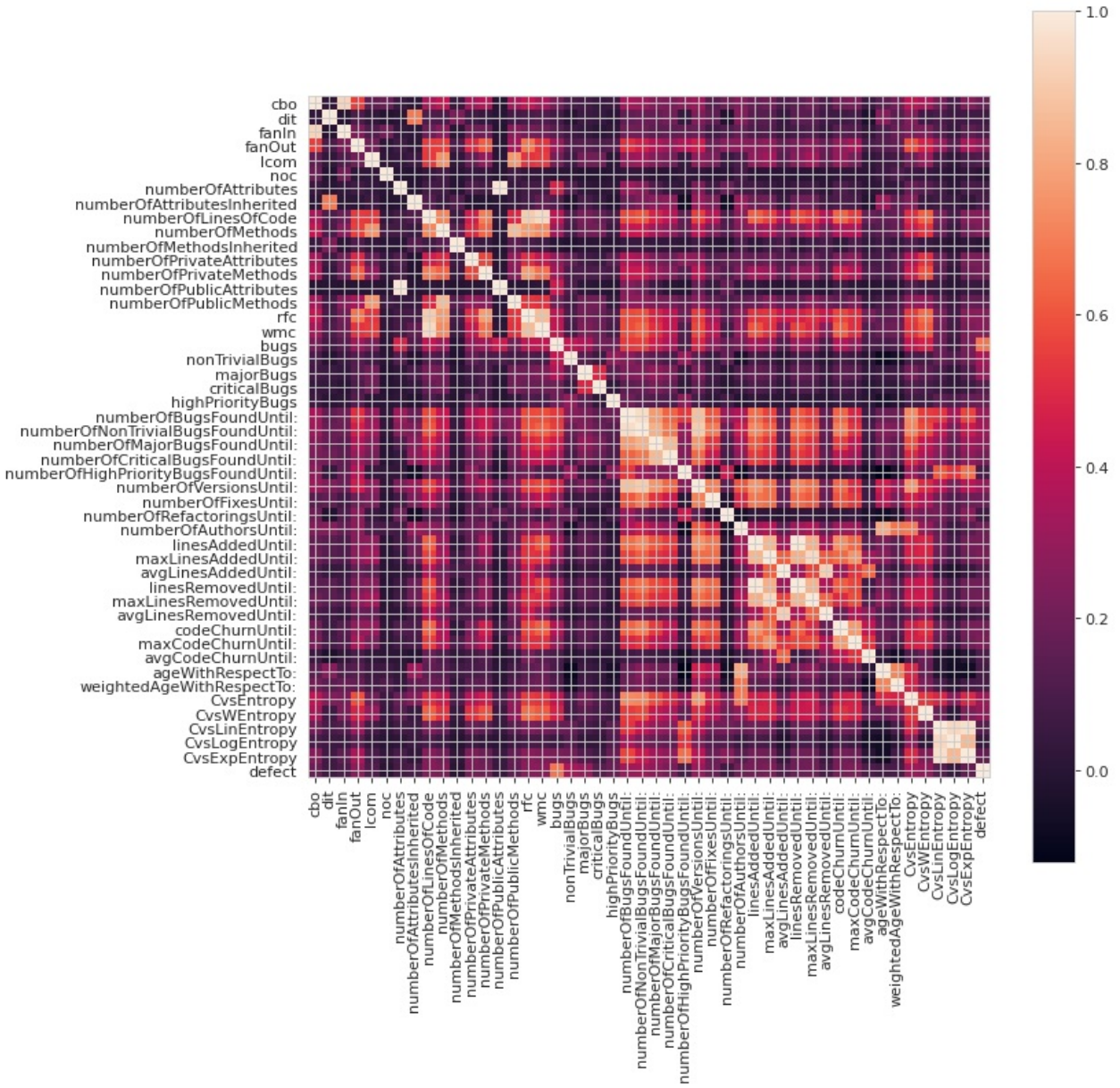
In [10]:
```
# plot columns distribution
def plotPerColumnDistribution(df, nGraphShown, nGraphPerRow):
    nunique = df.nunique()
    df = df[[col for col in df if nunique[col] > 1 and nunique[col] < 50]] # For displaying purposes, pick colu
```

```
    nRow, nCol = df.shape
    columnNames = list(df)
    nGraphRow = (nCol + nGraphPerRow - 1) / nGraphPerRow
    plt.figure(num = None, figsize = (6 * nGraphPerRow, 8 * nGraphRow), dpi = 80, facecolor = 'w', edgecolor =
    for i in range(min(nCol, nGraphShown)):
        plt.subplot(nGraphRow, nGraphPerRow, i + 1)
        columnDf = df.iloc[:, i]
        if (not np.issubdtype(type(columnDf.iloc[0]), np.number)):
            valueCounts = columnDf.value_counts()
            valueCounts.plot.bar()
        else:
            columnDf.hist()
        plt.ylabel('counts')
        plt.xticks(rotation = 90)
        plt.title(f'{columnNames[i]} (column {i})')
    plt.tight_layout(pad = 1.0, w_pad = 1.0, h_pad = 1.0)
    plt.show()
```

In [11]:

```
# plot corr
def plotCorrelationMatrix(df, graphWidth):
    df = df.dropna('columns') # drop columns with NaN
    df = df[[col for col in df if df[col].nunique() > 1]] # keep columns where there are more than 1 unique val
    if df.shape[1] < 2:
        print(f'No correlation plots shown: The number of non-NaN or constant columns ({df.shape[1]}) is less t
        return
    corr = df.corr()
    plt.figure(num=None, figsize=(graphWidth, graphWidth), dpi=80, facecolor='w', edgecolor='k')
    corrMat = plt.matshow(corr, fignum = 1)
    plt.xticks(range(len(corr.columns)), corr.columns, rotation=90)
    plt.yticks(range(len(corr.columns)), corr.columns)
    plt.gca().xaxis.tick_bottom()
    plt.colorbar(corrMat)
    plt.show()

plotCorrelationMatrix(data, 10)
```

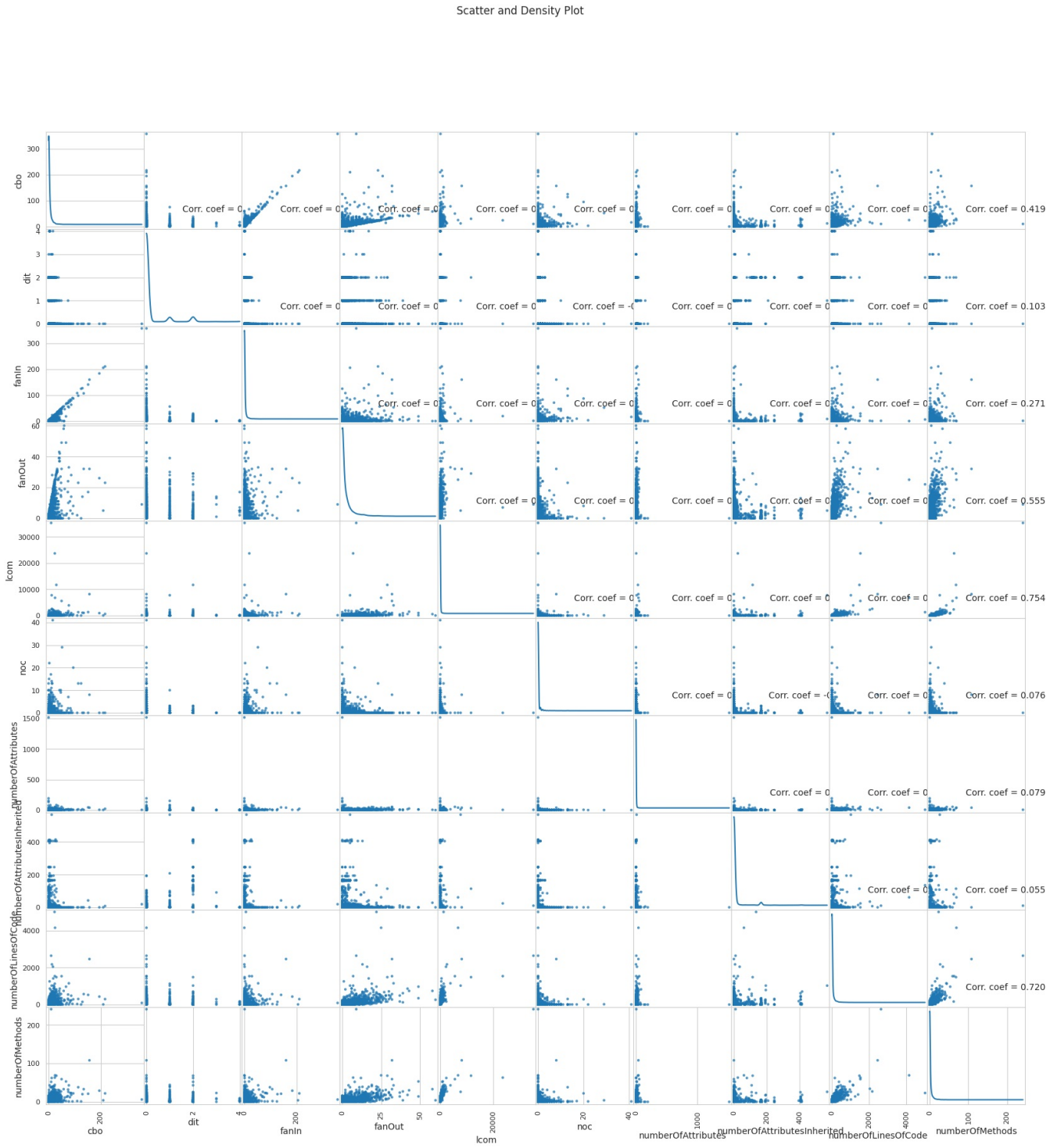
```
<ipython-input-11-c862859e6656>:3: FutureWarning:

In a future version of pandas all arguments of DataFrame.dropna will be keyword-only.

<ipython-input-11-c862859e6656>:8: FutureWarning:

The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to Fals
e. Select only valid columns or specify the value of numeric_only to silence this warning.
```



In [12]:

```
# Scatter and density plots
def plotScatterMatrix(df, plotSize, textSize):
    df = df.select_dtypes(include =[np.number]) # keep only numerical columns
    # Remove rows and columns that would lead to df being singular
    df = df.dropna('columns')
    df = df[[col for col in df if df[col].nunique() > 1]] # keep columns where there are more than 1 unique val
    columnNames = list(df)
    if len(columnNames) > 10: # reduce the number of columns for matrix inversion of kernel density plots
        columnNames = columnNames[:10]
    df = df[columnNames]
    ax = pd.plotting.scatter_matrix(df, alpha=0.75, figsize=[plotSize, plotSize], diagonal='kde')
    corrs = df.corr().values
    for i, j in zip(*plt.np.triu_indices_from(ax, k = 1)):
        ax[i, j].annotate('Corr. coef = %.3f' % corrs[i, j], (0.8, 0.2), xycoords='axes fraction', ha='center',
    plt.suptitle('Scatter and Density Plot')
    plt.show()

plotScatterMatrix(data, 20, 10)
```

```
<ipython-input-12-94136f4651fb>:5: FutureWarning:

In a future version of pandas all arguments of DataFrame.dropna will be keyword-only.
```



Data cleaning

- fillna
- remove outliars (by use boxplot to visualization)

In [13]:
```
data['bugs']
trace1 = go.Box(x=data['cbo'])
box_data = [trace1]
iplot(box_data)
```

NORMALIZATION

In [14]:
```
from sklearn import preprocessing
```

In [15]:
```
def change(X):
    length = X.shape[0]
    d=pd.Series(np.ones((length)))
    for i in range(0, length, 73):
        d[i]=0
    return d
def a(X):
    length = X.shape[0]
    d=pd.Series(np.zeros((length)))
    for i in range(0, length, 20):
        d[i]=1
    return d
```

In [16]:
```
def change(X):
    length = X.shape[0]
    d=pd.Series(np.ones((length)))
    for i in range(0, length, 73):
        d[i]=0
    return d
def a(X):
    length = X.shape[0]
    d=pd.Series(np.zeros((length)))
    for i in range(0, length, 20):
        d[i]=1
    return d
```

In [17]:
```
X = data[data.columns.difference(['defect', 'classname'])]
d=change(X)
a=change(X)
X['bugs'] = (X['bugs']+a)*d
y = data['defect']
preprocessing.scale(X)
show_info(X)
```

```
data shape is: (5371, 47)   sample number 5371   attribute number 47

data columns number 47
all columns: Index(['CvsEntropy', 'CvsExpEntropy', 'CvsLinEntropy', 'CvsLogEntropy',
       'CvsWEntropy', 'ageWithRespectTo:', 'avgCodeChurnUntil:',
       'avgLinesAddedUntil:', 'avgLinesRemovedUntil:', 'bugs', 'cbo',
       'codeChurnUntil:', 'criticalBugs', 'dit', 'fanIn', 'fanOut',
       'highPriorityBugs', 'lcom', 'linesAddedUntil:', 'linesRemovedUntil:',
       'majorBugs', 'maxCodeChurnUntil:', 'maxLinesAddedUntil:',
       'maxLinesRemovedUntil:', 'noc', 'nonTrivialBugs', 'numberOfAttributes',
       'numberOfAttributesInherited', 'numberOfAuthorsUntil:',
       'numberOfBugsFoundUntil:', 'numberOfCriticalBugsFoundUntil:',
       'numberOfFixesUntil:', 'numberOfHighPriorityBugsFoundUntil:',
       'numberOfLinesOfCode', 'numberOfMajorBugsFoundUntil:',
       'numberOfMethods', 'numberOfMethodsInherited',
```

```
       'numberOfNonTrivialBugsFoundUntil:', 'numberOfPrivateAttributes',
       'numberOfPrivateMethods', 'numberOfPublicAttributes',
       'numberOfPublicMethods', 'numberOfRefactoringsUntil:',
       'numberOfVersionsUntil:', 'rfc', 'weightedAgeWithRespectTo:', 'wmc'],
      dtype='object')

data all attribute count null:
 CvsEntropy                            0
CvsExpEntropy                          0
CvsLinEntropy                          0
CvsLogEntropy                          0
CvsWEntropy                            0
ageWithRespectTo:                      0
avgCodeChurnUntil:                     0
avgLinesAddedUntil:                    0
avgLinesRemovedUntil:                  0
bugs                                   0
cbo                                    0
codeChurnUntil:                        0
criticalBugs                           0
dit                                    0
fanIn                                  0
fanOut                                 0
highPriorityBugs                       0
lcom                                   0
linesAddedUntil:                       0
linesRemovedUntil:                     0
majorBugs                              0
maxCodeChurnUntil:                     0
maxLinesAddedUntil:                    0
maxLinesRemovedUntil:                  0
noc                                    0
nonTrivialBugs                         0
numberOfAttributes                     0
numberOfAttributesInherited            0
numberOfAuthorsUntil:                  0
numberOfBugsFoundUntil:                0
numberOfCriticalBugsFoundUntil:        0
numberOfFixesUntil:                    0
numberOfHighPriorityBugsFoundUntil:    0
numberOfLinesOfCode                    0
numberOfMajorBugsFoundUntil:           0
numberOfMethods                        0
numberOfMethodsInherited               0
numberOfNonTrivialBugsFoundUntil:      0
numberOfPrivateAttributes              0
numberOfPrivateMethods                 0
numberOfPublicAttributes               0
numberOfPublicMethods                  0
numberOfRefactoringsUntil:             0
numberOfVersionsUntil:                 0
rfc                                    0
weightedAgeWithRespectTo:              0
wmc                                    0
dtype: int64
data value analysis:          CvsEntropy  CvsExpEntropy  CvsLinEntropy  CvsLogEntropy  CvsWEntropy  \
count  5371.000000    5371.000000    5371.000000    5371.000000  5371.000000
mean      7.022268       0.116143       0.065986       2.801887     0.050255
std       7.068468       0.131436       0.073452       4.048816     0.154135
min       0.000000       0.000000       0.000000       0.000000     0.000000
25%       1.847030       0.005732       0.009619       0.095785     0.001875
50%       5.076520       0.051280       0.027211       0.209384     0.007752
75%       9.459490       0.205382       0.140346       8.254325     0.034558
max      58.151800       0.727558       0.329150      10.250000     4.490230

       ageWithRespectTo:  avgCodeChurnUntil:  avgLinesAddedUntil:  \
count        5371.000000         5371.000000          5371.000000
mean          150.921026            3.064683            10.740006
std           112.614942           10.560036            24.997371
min             0.000000         -133.800000             0.000000
25%            51.857100            0.000000             2.000000
50%           125.857000            0.600000             5.777780
75%           227.429000            3.398275            13.042900
max           381.000000          312.625000          1446.870000

       avgLinesRemovedUntil:         bugs  ...  \
count           5371.000000  5371.000000  ...
mean               7.675324     1.242227  ...
std               19.952796     0.889193  ...
min                0.000000     0.000000  ...
25%                1.303845     1.000000  ...
50%                3.900000     1.000000  ...
75%                9.027240     1.000000  ...
max             1134.250000    29.000000  ...

       numberOfNonTrivialBugsFoundUntil:  numberOfPrivateAttributes  \
count                        5371.000000                5371.000000
mean                           4.509961                   0.781046
std                           10.250986                   2.590000
```

```
min                            0.000000                  0.000000
25%                            1.000000                  0.000000
50%                            2.000000                  0.000000
75%                            4.000000                  0.000000
max                          200.000000                 48.000000

       numberOfPrivateMethods  numberOfPublicAttributes  \
count             5371.000000               5371.000000
mean                 0.613294                  0.612363
std                  2.550416                 21.136309
min                  0.000000                  0.000000
25%                  0.000000                  0.000000
50%                  0.000000                  0.000000
75%                  0.000000                  0.000000
max                 64.000000               1518.000000

       numberOfPublicMethods  numberOfRefactoringsUntil:  \
count            5371.000000                 5371.000000
mean                1.026066                    0.175386
std                 4.266590                    0.526891
min                 0.000000                    0.000000
25%                 0.000000                    0.000000
50%                 0.000000                    0.000000
75%                 1.000000                    0.000000
max               229.000000                    7.000000

       numberOfVersionsUntil:          rfc  weightedAgeWithRespectTo:  \
count             5371.000000  5371.000000                5371.000000
mean                17.535096    14.961832                  45.221031
std                 33.340856    52.043516                  46.554970
min                  0.000000     0.000000                   0.000000
25%                  3.000000     0.000000                   8.145750
50%                  7.000000     1.000000                  32.458100
75%                 18.000000     9.000000                  67.783600
max                709.000000  1295.000000                 314.776000

             wmc
count  5371.00000
mean      8.73450
std      34.86155
min       0.00000
25%       0.00000
50%       0.00000
75%       5.00000
max     984.00000

[8 rows x 47 columns]
data demo without matrix transpose:     CvsEntropy  CvsExpEntropy  CvsLinEntropy  CvsLogEntropy  CvsWEntropy  \
0    10.91310       0.001125       0.014767       0.253257     0.008302
1     1.10349       0.000351       0.001886       0.027116     0.000657
2    37.86060       0.210412       0.106180       1.028400     0.228509
3     5.86013       0.003485       0.010113       0.143680     0.009105
4    13.59600       0.021378       0.025751       0.338602     0.016005

   ageWithRespectTo:  avgCodeChurnUntil:  avgLinesAddedUntil:  \
0            350.571            0.646154              9.35385
1            117.714            5.000000              5.00000
2            238.429            4.941670             11.34170
3            367.000            0.535714              4.92857
4            367.000           -0.118280             20.10750

   avgLinesRemovedUntil:  bugs  ...  numberOfNonTrivialBugsFoundUntil:  \
0                8.70769   0.0  ...                                  2
1                0.00000   1.0  ...                                  0
2                6.40000   2.0  ...                                 48
3                4.39286   1.0  ...                                  3
4               20.22580   1.0  ...                                 13

   numberOfPrivateAttributes  numberOfPrivateMethods  \
0                          0                       0
1                          0                       0
2                          0                       0
3                          0                       0
4                          0                       0

   numberOfPublicAttributes  numberOfPublicMethods  \
0                       0.0                      0
1                       0.0                      0
2                       0.0                      1
3                       0.0                      1
4                       1.0                      0

   numberOfRefactoringsUntil:  numberOfVersionsUntil:   rfc  \
0                           0                      65   0.0
1                           0                       2   0.0
2                           0                     120  85.0
3                           0                      28   5.0
4                           0                      93   7.0
```

```
   weightedAgeWithRespectTo:   wmc
0                    63.0609   0.0
1                     0.0000   0.0
2                   104.2040  92.0
3                   112.6780   5.0
4                    54.9702   4.0

[5 rows x 47 columns]
```

```
<ipython-input-17-42061a1c27ef>:4: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

SMOTE-TOMEK

In [18]:
```
pip install -U imbalanced-learn
```

```
Requirement already satisfied: imbalanced-learn in /usr/local/lib/python3.10/dist-packages (0.10.1)
Collecting imbalanced-learn
  Downloading imbalanced_learn-0.11.0-py3-none-any.whl (235 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 235.6/235.6 kB 3.7 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (1.23.5)
Requirement already satisfied: scipy>=1.5.0 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (1.11.4)
Requirement already satisfied: scikit-learn>=1.0.2 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (1.2.2)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (3.2.0)
Installing collected packages: imbalanced-learn
  Attempting uninstall: imbalanced-learn
    Found existing installation: imbalanced-learn 0.10.1
    Uninstalling imbalanced-learn-0.10.1:
      Successfully uninstalled imbalanced-learn-0.10.1
Successfully installed imbalanced-learn-0.11.0
```

In [19]:
```
pip install imblearn
```

```
Collecting imblearn
  Downloading imblearn-0.0-py2.py3-none-any.whl (1.9 kB)
Requirement already satisfied: imbalanced-learn in /usr/local/lib/python3.10/dist-packages (from imblearn) (0.11.0)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn->imblearn) (1.23.5)
Requirement already satisfied: scipy>=1.5.0 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn->imblearn) (1.11.4)
Requirement already satisfied: scikit-learn>=1.0.2 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn->imblearn) (1.2.2)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn->imblearn) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn->imblearn) (3.2.0)
Installing collected packages: imblearn
Successfully installed imblearn-0.0
```

In [20]:
```
pip install seaborn
```

```
Requirement already satisfied: seaborn in /usr/local/lib/python3.10/dist-packages (0.12.2)
Requirement already satisfied: numpy!=1.24.0,>=1.17 in /usr/local/lib/python3.10/dist-packages (from seaborn) (
1.23.5)
Requirement already satisfied: pandas>=0.25 in /usr/local/lib/python3.10/dist-packages (from seaborn) (1.5.3)
Requirement already satisfied: matplotlib!=3.6.1,>=3.1 in /usr/local/lib/python3.10/dist-packages (from seaborn
) (3.7.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.
6.1,>=3.1->seaborn) (1.2.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,
>=3.1->seaborn) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3
.6.1,>=3.1->seaborn) (4.46.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3
.6.1,>=3.1->seaborn) (1.4.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6
.1,>=3.1->seaborn) (23.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1
,>=3.1->seaborn) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.
6.1,>=3.1->seaborn) (3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib
!=3.6.1,>=3.1->seaborn) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.25->seab
orn) (2023.3.post1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->
matplotlib!=3.6.1,>=3.1->seaborn) (1.16.0)
```

In [21]:
```
from sklearn.preprocessing import LabelEncoder
# Choose the column you want to convert (e.g., 'classname')
column_name = 'classname'

# Initialize LabelEncoder
label_encoder = LabelEncoder()

# Transform the selected column to integer labels
data[column_name] = label_encoder.fit_transform(data[column_name])

# Convert the integer labels to float
data[column_name] = data[column_name].astype(float)
```

In [22]:
```
# data prepare -- train test split

from sklearn.model_selection import train_test_split
# hyper-parameter
validation_size = 0.2
random_seed=8

X = data.iloc[:, 0:47]
y = data.iloc[:, 48]
X_train, X_val, y_train, y_val = train_test_split(
    X.values,
    y.values,
    test_size=validation_size,
    random_state=random_seed
    )
X_train.shape, X_val.shape, y_train.shape, y_val.shape
```

Out[22]:
```
((4296, 47), (1075, 47), (4296,), (1075,))
```

In [23]:
```
X_train
```

Out[23]:
```
array([[4.71800e+03, 0.00000e+00, 0.00000e+00, ..., 9.80191e-03,
        1.85353e-02, 1.36860e-01],
       [1.38000e+02, 2.00000e+00, 0.00000e+00, ..., 8.73984e-04,
        5.35961e-02, 1.70881e-01],
       [1.43200e+03, 1.10000e+01, 0.00000e+00, ..., 8.46606e-02,
        1.67331e-01, 8.53730e+00],
       ...,
       [3.06000e+03, 0.00000e+00, 0.00000e+00, ..., 0.00000e+00,
        0.00000e+00, 0.00000e+00],
       [5.88000e+02, 0.00000e+00, 0.00000e+00, ..., 1.81519e-03,
        6.45093e-03, 3.93921e-02],
       [4.74800e+03, 3.00000e+00, 0.00000e+00, ..., 1.47213e-01,
        6.42199e-02, 4.46990e-01]])
```

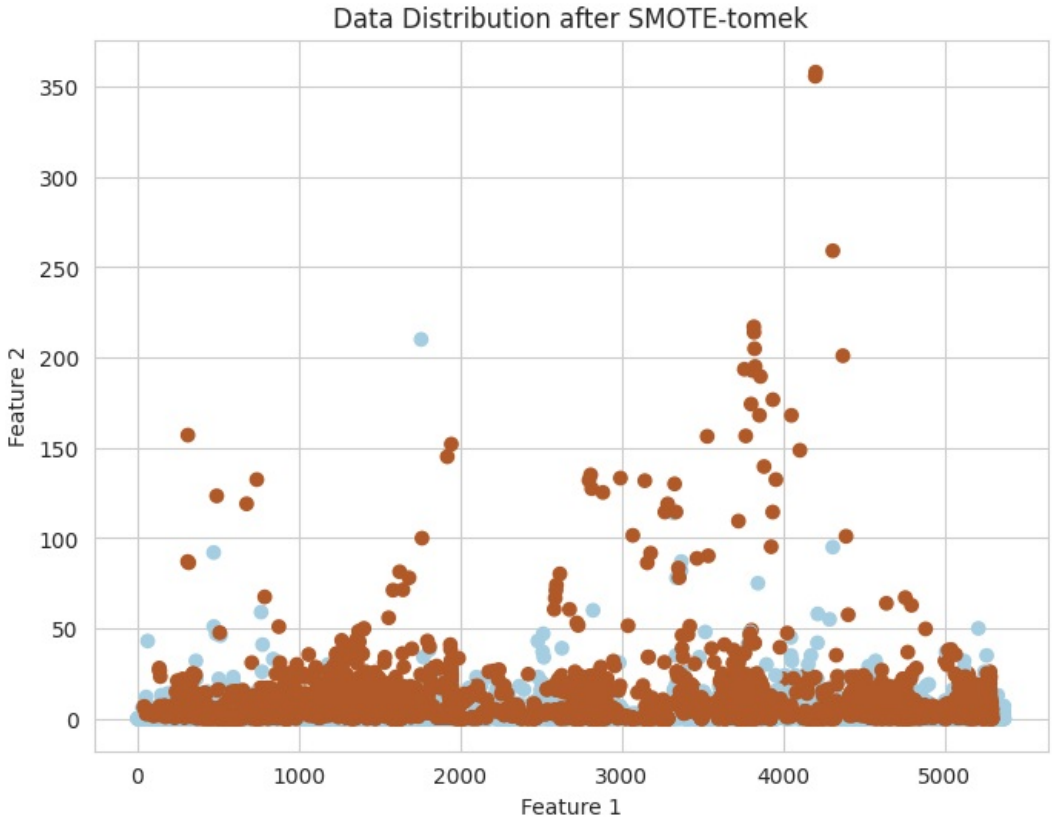
In [24]:
```
data
```

Out[24]:

	classname	cbo	dit	fanIn	fanOut	lcom	noc	numberOfAttributes	numberOfAttributesInherited	numberOfLinesOfCode	...	maxCodeCl
0	1562.0	0	0	0	0	0	0	0	0	6.0	...	
1	1172.0	0	0	0	0	0	0	0	0	0.0	...	
2	1043.0	14	2	12	4	19	0	37	128	261.0	...	
3	1237.0	3	2	0	3	4	0	0	412	25.0	...	
4	1694.0	0	2	0	0	117	0	1	108	26.0	...	
...	...	...	...	...	...	...	...	...	...	...	...	
5366	4673.0	0	0	0	0	0	0	0	0	0.0	...	
5367	3869.0	0	0	0	0	0	0	0	0	1.0	...	
5368	3866.0	2	0	0	2	0	0	0	0	1.0	...	
5369	4914.0	0	0	0	0	0	0	0	0	0.0	...	
5370	3949.0	0	0	0	0	0	0	0	0	0.0	...	

5371 rows × 49 columns

In [25]:
```
from imblearn.combine import SMOTETomek


smote_tomek = SMOTETomek(random_state=42)
X_train_resampled, y_train_resampled = smote_tomek.fit_resample(X_train, y_train)

# Visualize the data distribution after applying SMOTE
plt.figure(figsize=(8, 6))
plt.scatter(X_train_resampled[:, 0], X_train_resampled[:, 1], c=y_train_resampled, cmap=plt.cm.Paired, marker='
plt.title("Data Distribution after SMOTE-tomek")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```



KNN

In [26]:
```
pip install sklearn
```

```
Collecting sklearn
  Downloading sklearn-0.0.post12.tar.gz (2.6 kB)
  error: subprocess-exited-with-error

  × python setup.py egg_info did not run successfully.
  │ exit code: 1
  └─> See above for output.

  note: This error originates from a subprocess, and is likely not a problem with pip.
  Preparing metadata (setup.py) ... error
error: metadata-generation-failed

× Encountered error while generating package metadata.
└─> See above for output.

note: This is an issue with the package mentioned above, not pip.
hint: See above for details.
```

In [27]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score
```

In [28]:
```
validation_size = 0.2
random_seed=8

X = data.iloc[:, 0:47]
y = data.iloc[:, 48]
X_train, X_val, y_train, y_val = train_test_split(
    X.values,
    y.values,
    test_size=validation_size,
    random_state=random_seed
    )

#x=X_train_resampled[2:47]
sc_X = StandardScaler() #data between 1 to -1
X_train = sc_X.fit_transform(X_train_resampled)
X_test = sc_X.transform(X_val)
```

In [29]:
```
import math
math.sqrt(len(y_val))
```

Out[29]:
```
32.78719262151
```

In [30]:
```
#Define the model
knn_model = KNeighborsClassifier (n_neighbors=31, p=2, metric='euclidean')

#fit model
knn_model.fit(X_train_resampled, y_train_resampled)
```

Out[30]:
```
KNeighborsClassifier(metric='euclidean', n_neighbors=31)
```

In [31]:
```
# Predict the test set results
knn_pred = knn_model.predict(X_val)
y_pred = knn_model.predict(X_val)
y_pred
```

Out[31]:
```
array([False,  True,  True, ...,  True, False, False])
```

In [32]:
```
# Evaluate Model
cm = confusion_matrix(y_val, y_pred)
print (cm)
```

```
[[648 249]
 [ 66 112]]
```

In [33]:
```
print(accuracy_score(y_val, y_pred))
```

```
0.7069767441860465
```

In [34]:
```
knn_pred = knn_model.predict(X_test)
```

DECISION TREE

In [35]:
```
#import the necessary packages
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
```

```
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
from sklearn import tree
import matplotlib.pyplot as plt
```

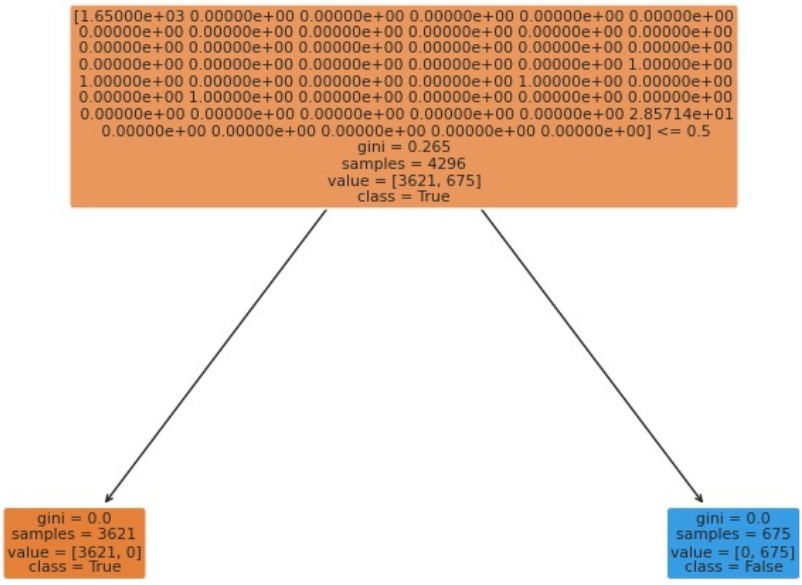
In [36]:
```
X = data.iloc[:, 0:47]
y = data.iloc[:, 48]
X_train, X_val, y_train, y_val = train_test_split(
    X.values,
    y.values,
    test_size=validation_size,
    random_state=random_seed
)

feature_names = X_train
dt_model = DecisionTreeClassifier(criterion='entropy',random_state=42)
dt_model.fit(X_train, y_train)
```

Out[36]:
```
DecisionTreeClassifier(criterion='entropy', random_state=42)
```

In [37]:
```
dt_model = DecisionTreeClassifier()
dt_model.fit(X_train, y_train)
dt_pred = dt_model.predict(X_val)
```

In [38]:
```
# Make predictions on the test set
y_pred = dt_model.predict(X_val)

# Calculate and print the accuracy of the classifier on the test set
accuracy = accuracy_score(y_val, y_pred)
print(f"Accuracy: {accuracy:.2f}")

# Visualize the decision tree
plt.figure(figsize=(12, 8))
tree.plot_tree(dt_model, feature_names=feature_names, class_names=['True','False'], filled=True, rounded=True)
plt.show()
```

```
Accuracy: 1.00
```



In [39]:
```
#function to perform training with entropy
dt_model = DecisionTreeClassifier(criterion='entropy')
dt_model= dt_model.fit(X_train_resampled, y_train_resampled)
```

In [40]:
```
y_pred_en = dt_model.predict(X_val)
y_pred_en
```

Out[40]:
```
array([False,  True, False, ..., False, False, False])
```

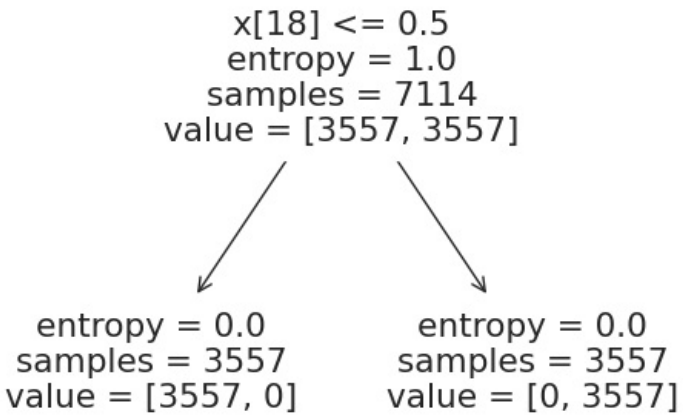
In [41]:
```
print ("Accuracy is "),metrics.accuracy_score(y_val,y_pred_en)*100
```

```
#print("accuracy is ", accuracy_score(y_test,y_pred_en)*100)
```

Out[41]:
```
Accuracy is
(None, 100.0)
```

In [42]:
```
from sklearn import tree
tree.plot_tree(dt_model)
```

Out[42]:
```
[Text(0.5, 0.75, 'x[18] <= 0.5\nentropy = 1.0\nsamples = 7114\nvalue = [3557, 3557]'),
 Text(0.25, 0.25, 'entropy = 0.0\nsamples = 3557\nvalue = [3557, 0]'),
 Text(0.75, 0.25, 'entropy = 0.0\nsamples = 3557\nvalue = [0, 3557]')]
```



SVM

In [43]:
```
from sklearn.svm import SVC
svm_model = SVC()
```

In [44]:
```
svm_model.fit(X_train_resampled, y_train_resampled)
```

Out[44]:
```
▾ SVC
SVC()
```

In [45]:
```
svm_model = SVC()
svm_model.fit(X_train, y_train)
svm_pred = svm_model.predict(X_val)
```

In [46]:
```
print("Accuracy:", accuracy_score(y_val, y_pred))
```

```
Accuracy: 1.0
```

In [47]:
```
svm_model.score(X_val, y_val)
```

Out[47]:
```
0.8520930232558139
```

In [48]:
```
svm_model.predict(X_val)
```

Out[48]:
```
array([False, False, False, ..., False, False, False])
```

In [49]:
```
#Tune parameters
#1. Regularization (C)
svm_model_C = SVC(C=1)
svm_model_C.fit(X_train_resampled, y_train_resampled)
svm_model_C.score(X_val, y_val)
```

Out[49]:
```
0.813953488372093
```

In [50]:
```
svm_model_C = SVC(C=10)
svm_model_C.fit(X_train_resampled, y_train_resampled)
svm_model_C.score(X_val, y_val)
```

Out[50]:
```
0.7879069767441861
```

In [50]:
```

```

NAIVE

In [51]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
```

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report, accuracy_score
```

In [52]:
```
from sklearn.naive_bayes import GaussianNB

# Assuming X contains numerical features
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a Gaussian Naive Bayes classifier
nb_model = GaussianNB()

# Train the classifier
nb_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = nb_model.predict(X_val)

# Print classification report and accuracy
print("Classification Report:")
print(classification_report(y_val, y_pred))

print("Accuracy:", accuracy_score(y_val, y_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

       False       1.00      0.96      0.98       897
        True       0.83      1.00      0.91       178

    accuracy                           0.97      1075
   macro avg       0.91      0.98      0.94      1075
weighted avg       0.97      0.97      0.97      1075

Accuracy: 0.9655813953488372
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning:

X does not have valid feature names, but GaussianNB was fitted with feature names
```

In [53]:
```
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)
nb_pred = nb_model.predict(X_val)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning:

X does not have valid feature names, but GaussianNB was fitted with feature names
```

GENETIC ALGORITHM BASED ON ARTIFICAL NEURAL NETWORK (GA-ANN)

In [54]:
```
pip install deap
```

```
Collecting deap
  Downloading deap-1.4.1-cp310-cp310-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_17_x86_64.manylinux2014
_x86_64.whl (135 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 135.4/135.4 kB 1.4 MB/s eta 0:00:00
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from deap) (1.23.5)
Installing collected packages: deap
Successfully installed deap-1.4.1
```

In [55]:
```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from deap import base, creator, tools, algorithms
import matplotlib.pyplot as plt


# 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)

# Define genetic algorithm parameters
population_size = 10
num_generations = 5
mutation_rate = 0.1

# Function to create a simple feedforward neural network
def create_neural_network():
    model = keras.Sequential([
        keras.layers.Input(shape=(X_train.shape[1],)),
        keras.layers.Dense(64, activation='relu'),
        keras.layers.Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model
```

```python
# Function to initialize a random population of neural networks
def initialize_population(population_size):
    return [create_neural_network() for _ in range(population_size)]

# Function to evaluate the fitness of each neural network in the population
def evaluate_population(population, X_train, y_train, X_test, y_test):
    fitness_scores = []
    for model in population:
        model.fit(X_train, y_train, epochs=5, verbose=0)
        predictions = (model.predict(X_test) > 0.5).astype(int).flatten()
        accuracy = accuracy_score(y_test, predictions)
        fitness_scores.append(accuracy)
    return fitness_scores

# Function for tournament selection
def tournament_selection(fitness_scores, tournament_size):
    selected_indices = []
    for _ in range(len(fitness_scores)):
        tournament_indices = np.random.choice(len(fitness_scores), tournament_size, replace=False)
        winner_index = max(tournament_indices, key=lambda i: fitness_scores[i])
        selected_indices.append(winner_index)
    return selected_indices

# Function for one-point crossover
def crossover(parent1, parent2):
    child1 = create_neural_network()
    child2 = create_neural_network()

    for layer in range(len(child1.layers)):
        crossover_point = np.random.randint(0, 2)
        if crossover_point == 0:
            child1.layers[layer].set_weights(parent1.layers[layer].get_weights())
            child2.layers[layer].set_weights(parent2.layers[layer].get_weights())
        else:
            child1.layers[layer].set_weights(parent2.layers[layer].get_weights())
            child2.layers[layer].set_weights(parent1.layers[layer].get_weights())

    return child1, child2

# Function for mutation
def mutate(model, mutation_rate):
    for layer in range(len(model.layers)):
        if np.random.rand() < mutation_rate:
            new_weights = [w + np.random.normal(0, 0.1, w.shape) for w in model.layers[layer].get_weights()]
            model.layers[layer].set_weights(new_weights)
    return model

# Main genetic algorithm loop
population = initialize_population(population_size)

# Lists to store evolution data
gen_numbers = []
max_accuracies = []

for generation in range(num_generations):
    fitness_scores = evaluate_population(population, X_train, y_train, X_test, y_test)

    # Select parents using tournament selection
    selected_indices = tournament_selection(fitness_scores, tournament_size=2)

    # Create new generation using crossover and mutation
    new_population = []
    for i in range(0, len(selected_indices), 2):
        parent1 = population[selected_indices[i]]
        parent2 = population[selected_indices[i + 1]]
        child1, child2 = crossover(parent1, parent2)
        child1 = mutate(child1, mutation_rate)
        child2 = mutate(child2, mutation_rate)
        new_population.extend([child1, child2])

    population = new_population

    # Collect evolution data
    best_model = max(population, key=lambda model: accuracy_score(y_test, (model.predict(X_test) > 0.5).astype(
    max_accuracy = accuracy_score(y_test, (best_model.predict(X_test) > 0.5).astype(int).flatten())

    gen_numbers.append(generation)
    max_accuracies.append(max_accuracy)
```

```
34/34 [==============================] - 0s 6ms/step
34/34 [==============================] - 0s 7ms/step
34/34 [==============================] - 1s 14ms/step
34/34 [==============================] - 0s 5ms/step
34/34 [==============================] - 0s 6ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 3ms/step
```

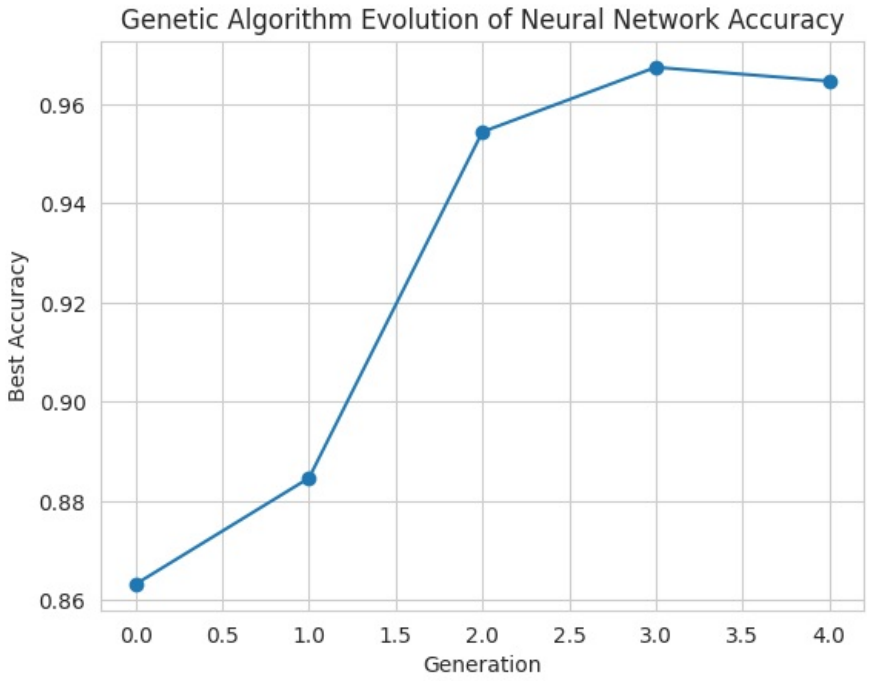
```
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 3ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 3ms/step
34/34 [==============================] - 0s 3ms/step
34/34 [==============================] - 0s 3ms/step
34/34 [==============================] - 0s 3ms/step
34/34 [==============================] - 0s 3ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 3ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 3ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 3ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 4ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 4ms/step
34/34 [==============================] - 0s 1ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 1ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 3ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 1ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
```

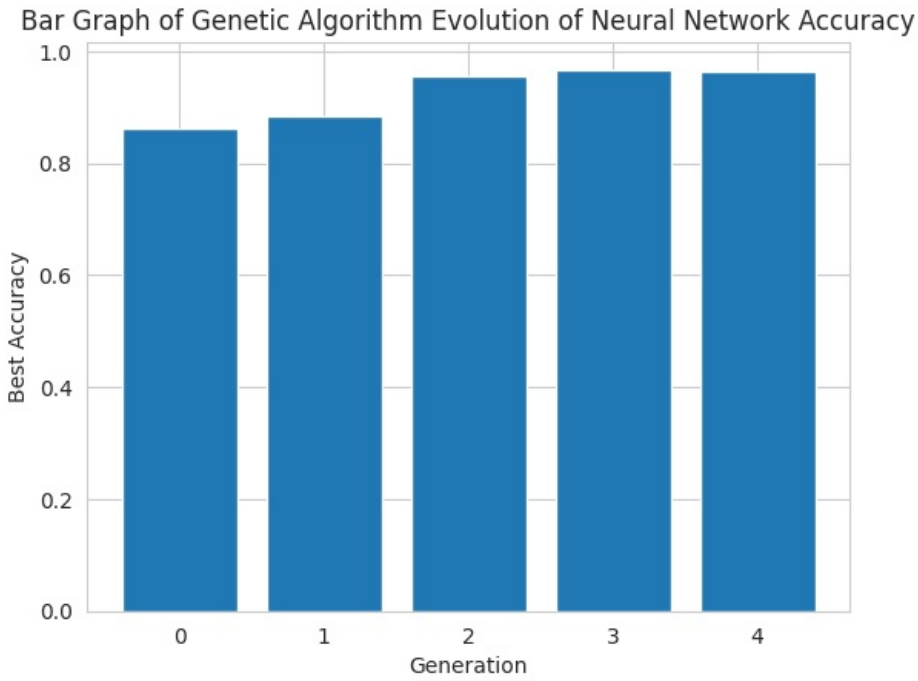
```
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
```

In [56]:
```python
# Select the best neural network from the final population
best_model = max(population, key=lambda model: accuracy_score(y_test, (model.predict(X_test) > 0.5).astype(int)
print("Best Neural Network Accuracy:", accuracy_score(y_test, (best_model.predict(X_test) > 0.5).astype(int).fl
```

```
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 3ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 3ms/step
34/34 [==============================] - 0s 4ms/step
34/34 [==============================] - 0s 2ms/step
34/34 [==============================] - 0s 3ms/step
Best Neural Network Accuracy: 0.9646511627906976
```

In [57]:
```python
# Plot the evolution of accuracy
plt.plot(gen_numbers, max_accuracies, marker='o')
plt.xlabel('Generation')
plt.ylabel('Best Accuracy')
plt.title('Genetic Algorithm Evolution of Neural Network Accuracy')
plt.show()

# Plot bar graph for accuracy
plt.bar(range(len(max_accuracies)), max_accuracies)
plt.xlabel('Generation')
plt.ylabel('Best Accuracy')
plt.title('Bar Graph of Genetic Algorithm Evolution of Neural Network Accuracy')
plt.show()
```



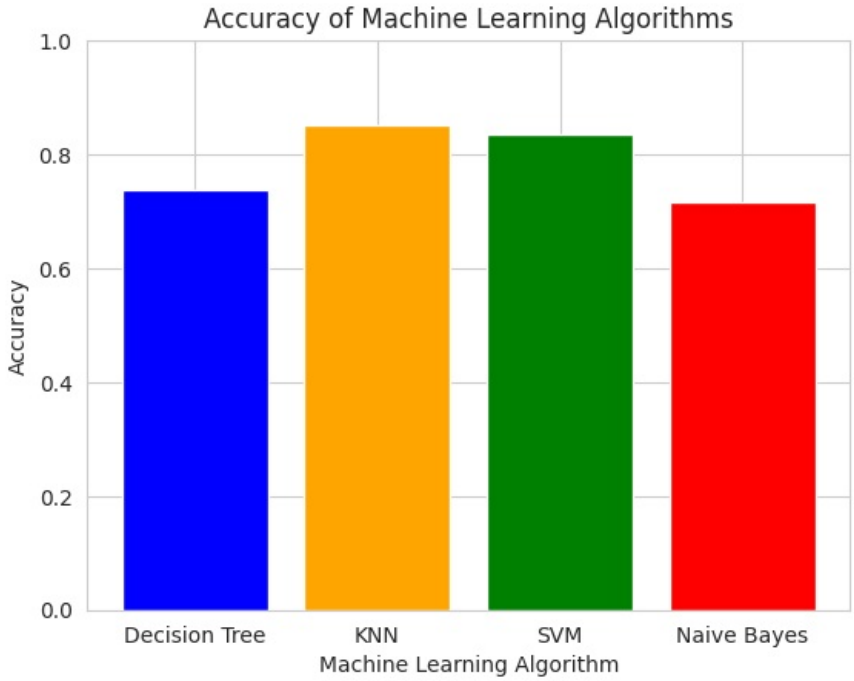


EVALUATING OTHER ALGORITHMS

In [58]:

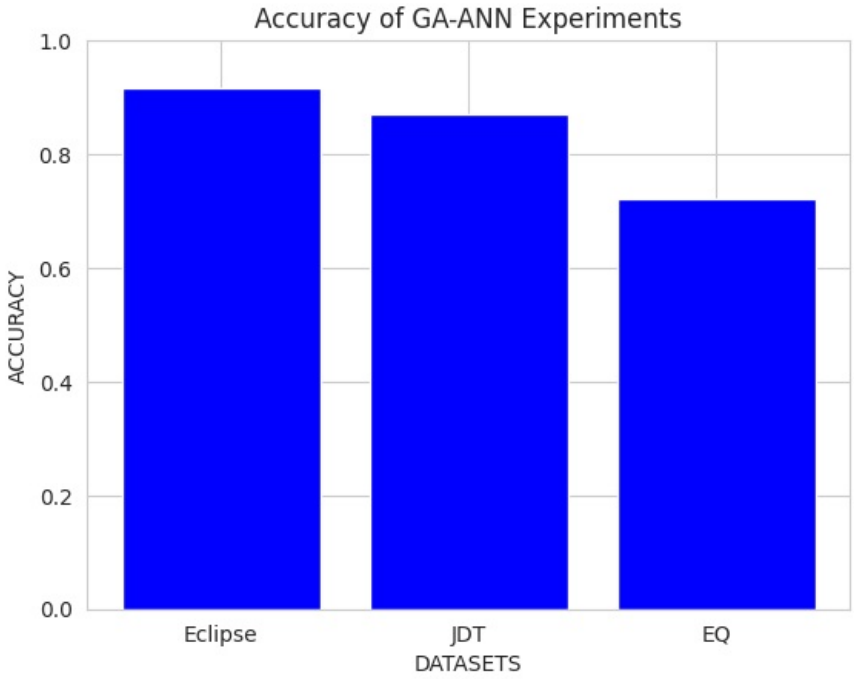
```
# Evaluate models and store results

models = {'Decision Tree': dt_pred,
          'KNN': knn_pred,
          'SVM': svm_pred,
          'Naive Bayes': nb_pred}

model_results = {}
for name, pred in models.items():
    acc = accuracy_score(y_test, pred)
    model_results[name] = acc
    cm = confusion_matrix(y_test, pred)
    print(f"Model: {name}")
    print(f"Accuracy: {acc:.4f}")
    print(f"Confusion Matrix:\n{cm}\n")

# Plot bar graph
plt.bar(model_results.keys(), model_results.values(), color=['blue', 'orange', 'green', 'red', 'purple'])
plt.xlabel('Machine Learning Algorithm')
plt.ylabel('Accuracy')
plt.title('Accuracy of Machine Learning Algorithms')
plt.ylim([0, 1])  # Set the y-axis limit to better visualize differences
plt.show()
```

```
Model: Decision Tree
Accuracy: 0.7386
Confusion Matrix:
[[766 150]
 [131  28]]

Model: KNN
Accuracy: 0.8521
Confusion Matrix:
[[916   0]
 [159   0]]

Model: SVM
Accuracy: 0.8344
Confusion Matrix:
[[892  24]
 [154   5]]

Model: Naive Bayes
Accuracy: 0.7153
Confusion Matrix:
[[735 181]
 [125  34]]
```



Accuracies compared to other datasets

In [59]:
```python
import matplotlib.pyplot as plt

# Accuracy values
accuracies = [0.916, 0.87, 0.723]

# Plotting the bar graph
plt.bar(range(len(accuracies)), accuracies, color='blue')
plt.xlabel('DATASETS')
plt.ylabel('ACCURACY')
plt.title('Accuracy of GA-ANN Experiments')
plt.xticks(range(len(accuracies)), ['Eclipse', 'JDT', 'EQ'])
plt.ylim(0, 1)  # Set the y-axis limit to better visualize differences
plt.show()
```


Accuracy of GA-ANN Experiments
1.0
0.8
0.6
0.4
0.2
0.0
ACCURACY
Eclipse
JDT
EQ
DATASETS

7) CONCLUSION

Hence the project has successfully implemented the research work of Mansi Gupta, Kumar Rajnish, Vandana Bhattacharjee in predicting Software Faults with imbalanced dataset using SMOTE-Tomek Link and Genetic Algorithm models. The project further compliments the research approach and agrees with results established in the paper.