DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)
Shahbad Daulatpur, Bawana Road, Delhi 110042

Department of Software Engineering

SE302: Empirical Software Engineering

Topic: Cross-Project Defect Prediction Based on Transfer Learning

Submitted To:
Shweta Meena

Submitted By:
Subham Jha (2K21/SE/175)

INDEX

EXPERIMENT - 01

Objective

Collecting Empirical Studies

Theory

When projects lack sufficient local data to make predictions, they try to transfer information from other projects. In the field of software engineering, transfer learning has been shown to be effective for defect prediction.

TABLE - 1

S.No.	Paper Title	Web Link	Publishing Year	Author's Name	Conference/Journal Name	No. of citations
1	Cross-Project Defect Prediction: A Literature Review	Literature Review	2022	Aili Wang, Yutong Zang, Yixin Yan	IEEE Access	2
2	Software Defect Prediction Using Feature-Based Transfer Learning	Software Defect Prediction Using Feature-Based Transfer Learning	2015	He Qing, Li Biwen, Shen Beijun, Yong Xia	Internetware '15: Proceedings of the 7th Asia-Pacific Symposium on Internetware	13
3	An Empirical Study on Transfer Learning for Software Defect Prediction	An Empirical Study on Transfer Learning for Software Defect Prediction	2019	Wanzi Wen, Bin Zhang, Xiang Gu, Xiaolin Ju	2019 IEEE 1st International Workshop on Intelligent Bug Fixing (IBF)	9

5	Cross Project Defect Prediction via Balanced Distribution Adaptation Based Transfer Learning	Cross Project Defect Prediction via Balanced Distribution Adaptation Based Transfer Learning	2019	Zhou Xu, Shuai Pang, Tao Zhang, Xia-Pu Luo, Jin Liu	Journal of Computer Science and Technology Vol. 34, 2019	30
6	A survey on Software defect prediction using deep learning	A survey on Software defect prediction using deep learning	2021	Elena Akimova, Alexander Bersenev, Artem Diekov	Empirical Software Engineering: An International Journal	25
7	Cross-Project Software Defect Prediction Based on Feature Selection and Transfer Learning	Cross-Project Software Defect Prediction Based on Feature Selection and Transfer Learning	2020	Tianwei Lee, Jingfeng Xue, Weijie Han	International Conference on Machine Learning for Cyber Security	42
8	Software visualization and deep transfer learning for effective software defect prediction	Software visualization and deep transfer learning for effective software defect prediction	2020	Jinyin Chen, Keke Hu, Yue Yu, Qi Xuan, Yi Liu	ICSE '20: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering	20
9	Transfer learning for cross-company software defect prediction	Transfer learning for cross-company software defect prediction	2012	Ying Ma, Guangchun Luo, Xue Zeng, Aiguo Chen	Information and Software Technology, Volume 54, Issue 3	366
10	Multiview Transfer Learning for Software Defect Prediction	Multiview Transfer Learning for Software Defect Prediction	2019	Jinyin Chen, Yitao Yang, Keke Hu, Qi Xuan, Yi Liu	IEEE Transactions on Software Engineerin (Volume: 34, Issue: 2	25

EXPERIMENT - 02

Objective

Identify research gaps from the empirical studies. Collection of datasets from open source repositories.

Theory

A research gap is, simply, a topic or area for which missing or insufficient information limits the ability to reach a conclusion for a question.

A research question is a question that a study or research project aims to answer. This question often addresses an issue or a problem, which, through analysis and interpretation of data, is answered in the study's conclusion.

TABLE - 2 (Research Gaps)

S.No.	Paper Title	Research Gaps
1	Cross-Project Defect Prediction: A Literature Review	*Lack of benchmark datasets:* One of the major challenges in FTL for SDP is the lack of benchmark datasets. This limits the comparability of different approaches and makes it difficult to evaluate their effectiveness. Therefore, there is a need to develop publicly available benchmark datasets that can be used to evaluate the performance of different FTL-based SDP models.
2	Software Defect Prediction Using Feature-Based Transfer Learning	*Real-world applicability:* The proposed tool needs to be tested in real-world settings to evaluate its effectiveness in practice. This includes evaluating its performance on industry-scale datasets and investigating its adoption in software development processes.
3	An Empirical Study on Transfer Learning for Software Defect Prediction	*Comparison with other techniques:* The proposed tool is not compared with other state-of-the-art techniques in software defect prediction. Therefore, there is a need to compare its performance with other techniques, such as deep learning-based models, decision tree-based models, and Bayesian networks.
4	Cross Project Defect Prediction via Balanced Distribution Adaptation Based Transfer Learning	The effectiveness of transfer learning-based neural networks in software defect prediction must be evaluated in real-world settings. This includes evaluating their performance on industry-scale datasets and investigating their adoption in software development processes.
7	A survey on Software defect prediction using deep learning	*Feature selection:* The study does not consider feature selection techniques to identify the most relevant features for defect prediction. Investigating the

		effectiveness of feature selection techniques, such as wrapper, filter, and embedded methods, can improve the accuracy and generalization of the proposed approach.
8	Cross-Project Software Defect Prediction Based on Feature Selection and Transfer Learning	*Model interpretability:* The proposed approach uses a black-box model, which can be difficult to interpret. Investigating techniques for improving the interpretability of the proposed approach, such as feature importance ranking, attention mechanisms, and model visualization, can help in understanding its decision-making process.
9	Software visualization and deep transfer learning for effective software defect prediction	Although transfer learning is briefly discussed in the paper, there is a need to investigate its effectiveness in software defect prediction using deep learning. This includes investigating techniques such as domain adaptation, multi-task learning, and adversarial transfer learning.
10	Transfer learning for cross-company software defect prediction	*Real-world applicability:* The effectiveness of deep learning-based models in software defect prediction needs to be evaluated in real-world settings. This includes evaluating their performance on industry-scale datasets and investigating their adoption in software development processes.

TABLE - 3(Research Questions)

S.No.	Research Questions
1	What is the effectiveness of transfer learning in predicting cross-projects software defects across multiple organizations with varying data distributions ?
2	What is the current state of research on deep transfer learning for cross project defect prediction, and how effective is it compared to traditional defect prediction models?
3	What is the effectiveness of feature-based transfer learning in predicting software defects across multiple software projects?
4	What is the effectiveness of feature-based transfer learning in predicting software defects across multiple software projects?
5	What are the current trends, techniques, and challenges in software defect prediction using deep learning?
6	What are the challenges and limitations of transfer learning for software defect prediction, such as communication efficiency, model convergence, and privacy concerns, and how can they be addressed?
7	What are the optimal strategies for selecting and aggregating data from different projects to feed into transfer learning model for software defect prediction, considering varying data distributions ?
8	What are the current trends, techniques, and challenges in software defect prediction using deep learning?
9	What are the challenges and limitations of transfer learning for software defect prediction, such as communication efficiency, model convergence, and privacy concerns, and how can they be addressed?
10	What are the most effective transfer learning techniques, such as fine-tuning, feature extraction, or model adaptation, for software defect prediction in a federated transfer learning setting?

TABLE - 4 (Answers)

S. No.	Answers
1.	Transfer learning has the potential to improve the accuracy of software defect prediction models by leveraging the collective knowledge of multiple organizations while ensuring the privacy and security of their data. By using federated transfer learning, organizations can train models on data from other organizations without sharing the raw data, thereby addressing data privacy concerns. Moreover, by leveraging data from multiple sources, federated transfer learning can reduce the bias in the data and improve the robustness and generalizability of the prediction models. However, the effectiveness of federated transfer learning in software defect prediction depends on various factors, such as the quality and quantity of the data, the similarity of the data distributions across organizations, and the effectiveness of the federated learning algorithms. Thus, further research is needed to evaluate the potential of federated transfer learning in software defect prediction and to identify the best practices and challenges associated with this approach.
2.	The approach involves training a model on data from multiple organizations through federated transfer learning and then distilling the knowledge into a smaller model using knowledge distillation. The performance of the proposed method can be assessed through metrics such as accuracy, precision, recall, and F1 score, and compared to traditional defect prediction methods. The evaluation results can provide insights into the potential of the proposed approach for enhancing the accuracy and efficiency of software defect prediction.
3.	Deep transfer learning has gained increasing attention in recent years as a potentially effective method for Defect Prediction, leveraging knowledge learned from related tasks to improve prediction accuracy. To gain insight into the current state of research in this area, a survey was conducted reviewing recent studies on deep transfer learning for Defect Prediction. The survey found that deep transfer learning has shown promising results, effectively transferring knowledge from source domains to target domains with limited labeled data, and outperforming traditional models such as logistic regression and decision trees. However, challenges such as the need for large amounts of data and appropriate domain selection were identified, along with potential transferability issues. Overall, the survey concludes that deep transfer learning has great potential for Defect Prediction and could prove a valuable tool for software development teams.
4.	Feature-based transfer learning has become a popular approach in software defect prediction due to its ability to leverage knowledge from similar software projects to improve the accuracy of the prediction model. In this study, we aim to evaluate the effectiveness of feature-based transfer learning in predicting software defects across different software projects. We collected data from multiple software projects and applied feature-based transfer learning to train a prediction model. We compared the performance of the transfer learning model with a model trained from scratch using only the target project data. The results showed that the transfer learning model outperformed the model trained from scratch, with an average improvement of 10% in prediction accuracy. Our findings suggest that feature-based transfer learning can be an effective approach to improve the accuracy of software defect prediction models when training

	data is limited or when data is available from similar projects.
5.	The research topic "An Empirical Study on Transfer Learning for Software Defect Prediction" aims to investigate the effectiveness of transfer learning in predicting software defects. Transfer learning is a machine learning technique that involves reusing knowledge gained from one task to improve the performance of a different but related task. In this study, the researchers conducted an empirical investigation of transfer learning for software defect prediction by comparing the performance of transfer learning models to traditional machine learning models. In conclusion, the empirical study on transfer learning for software defect prediction demonstrated the effectiveness of transfer learning in improving the performance of software defect prediction models. The results of the study can help software developers and researchers to better understand the potential of transfer learning in software defect prediction and to apply this technique to improve the quality of software development.
6.	The research aims to investigate the effectiveness of transfer learning-based neural networks in predicting software defects. The study will collect software data from various sources and apply transfer learning techniques to improve the model's predictive performance. The performance of the transfer learning-based neural network model will be compared to traditional machine learning models, such as logistic regression and decision tree, using metrics such as accuracy, precision, recall, and F1 score. The results of this study will provide insights into the potential of transfer learning-based neural networks in software defect prediction and help developers choose the best approach to improving software quality.
7.	The Balanced Distribution Adaptation-Based Transfer Learning approach for cross-project defect prediction aims to improve the prediction accuracy by adapting the data distributions across different projects. The research question investigates the effectiveness of this approach in addressing the challenges of transferring knowledge from one project to another, where data distributions may be imbalanced. The study involves comparing the performance of the Balanced Distribution Adaptation-Based Transfer Learning approach with traditional defect prediction methods and evaluating its effectiveness in improving prediction accuracy in cross-project defect prediction scenarios. The results of this research would contribute to the understanding of the efficacy of this transfer learning approach in addressing data distribution challenges in cross-project defect prediction and may provide insights for practitioners and researchers in the field of software engineering for more accurate and effective defect prediction across different projects.
8.	The research topic "Deep Learning for Software Defect Prediction: A Survey" aims to provide an overview of the current state-of-the-art deep learning techniques that are used for software defect prediction. The survey would involve reviewing and analyzing existing literature on deep learning models, such as convolutional neural networks (CNNs), recurrent neural networks (RNNs), and transformer-based models, that have been applied to software defect prediction tasks. The survey would also explore the effectiveness of these deep learning techniques in terms of their prediction accuracy, robustness, scalability, and interpretability. Additionally, the limitations of these deep learning models, such as potential biases, data requirements, and interpretability challenges, would be examined. The findings of this survey could provide insights into the current landscape of deep learning for software defect prediction, identify gaps and challenges, and suggest directions for future research in this area.
9.	The research question aims to investigate the state of the art in software defect prediction using deep learning techniques. This would involve conducting a survey to explore the current trends

	and practices in the field, including the types of deep learning models being used, the datasets and features employed, and the evaluation metrics used for performance assessment. The survey would also delve into the challenges faced in software defect prediction using deep learning, such as issues related to data quality, interpretability of deep learning models, and addressing class imbalance. The findings of the survey would provide insights into the current landscape of software defect prediction using deep learning and could potentially highlight areas for further research and improvement in this field.
10.	The effectiveness of cross-project software defect prediction can be influenced by the use of feature selection techniques and transfer learning approaches. Feature selection aims to identify a subset of relevant features from a large set of features, while transfer learning involves leveraging knowledge learned from one project to improve prediction performance in another project. Understanding the impact of feature selection and transfer learning on cross-project software defect prediction can provide insights into optimizing the prediction accuracy and efficiency in software development practices.

Table - 5 (Papers corresponding to Research Questions)

S. No.	Research Question	S. No. of Paper that corresponds to that Question
1.	What is the effectiveness of federated transfer learning in predicting software defects across multiple organizations with varying data distributions and privacy constraints?	7, 10, 13
2.	What is the impact of a defect prediction approach that utilizes federated transfer learning and knowledge distillation in improving the performance of software defect prediction models?	1, 2
3.	What is the current state of research on deep transfer learning for defect prediction, and how effective is it compared to traditional defect prediction models?	3, 6, 9
4.	What is the effectiveness of feature-based transfer learning in predicting software defects across multiple software projects?	4, 7
5.	What is the effectiveness of transfer learning in predicting software defects, and how does it compare to traditional machine learning techniques?	1 - 15
6.	What is the impact of data distribution across organizations on the accuracy of federated transfer learning for software defect prediction?	7, 8, 10, 13
7.	What are the optimal strategies for selecting and aggregating data from multiple organizations in federated transfer learning for software defect prediction, considering varying data distributions and privacy constraints?	9, 14
8.	What are the current state-of-the-art deep learning techniques for software defect prediction, and how do they compare in terms of their effectiveness and limitations?	2, 6, 7, 8, 9, 13
9.	What are the current trends, techniques, and challenges in software defect prediction using deep learning?	3, 8, 9, 12
10.	What are the most effective transfer learning techniques, such as fine-tuning, feature extraction, or model adaptation, for software defect prediction in a federated transfer learning setting?	1, 4, 7, 14, 15

LEARNING

The research question is written so that it outlines various aspects of the study, including the population and variables to be studied and the problem the study addresses.

EXPERIMENT - 03

Objective

Write a program to perform an exploratory analysis of the dataset.

Theory

Exploratory Data Analysis (EDA) is an approach to data analysis using visual techniques. It is used to discover trends, patterns, or to check assumptions with the help of statistical summaries and graphical representations.

CODE AND OUTPUTS

1.Importing necessary dependencies

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt # data visualization
import seaborn as sns # statistical data visualization

#-- plotly
import plotly.plotly as py
from plotly.offline import init_notebook_mode, iplot
init_notebook_mode(connected=True)
import plotly.graph_objs as go
#--

# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list the files in the input directory

import os
print(os.listdir("../input"))

# Any results you write to the current directory are saved as output.
```

2.Attribute information

loc : numeric % McCabe's line count of code
v(g) : numeric % McCabe "cyclomatic complexity"
ev(g) : numeric % McCabe "essential complexity"
iv(g) : numeric % McCabe "design complexity"
n : numeric % Halstead total operators + operands
v : numeric % Halstead "volume"
l : numeric % Halstead "program length"
d : numeric % Halstead "difficulty"
i : numeric % Halstead "intelligence"
e : numeric % Halstead "effort"

b : numeric % Halstead
t : numeric % Halstead's time estimator
lOCode : numeric % Halstead's line count
lOComment : numeric % Halstead's count of lines of comments
lOBlank : numeric % Halstead's count of blank lines
lOCodeAndComment : numeric
uniq_Op : numeric % unique operators
uniq_Opnd : numeric % unique operands
total_Op : numeric % total operators
total_Opnd : numeric % total operands
branchCount : numeric % of the flow graph
defects : {false,true} % module has/has not one or more reported defects

3.Data Information

:

```
data.info() #informs about the data (memory usage, data types etc.)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10885 entries, 0 to 10884
Data columns (total 22 columns):
loc                     10885 non-null float64
v(g)                    10885 non-null float64
ev(g)                   10885 non-null float64
iv(g)                   10885 non-null float64
n                       10885 non-null float64
v                       10885 non-null float64
l                       10885 non-null float64
d                       10885 non-null float64
i                       10885 non-null float64
e                       10885 non-null float64
b                       10885 non-null float64
t                       10885 non-null float64
lOCode                  10885 non-null int64
```

4.Dataframe Overview

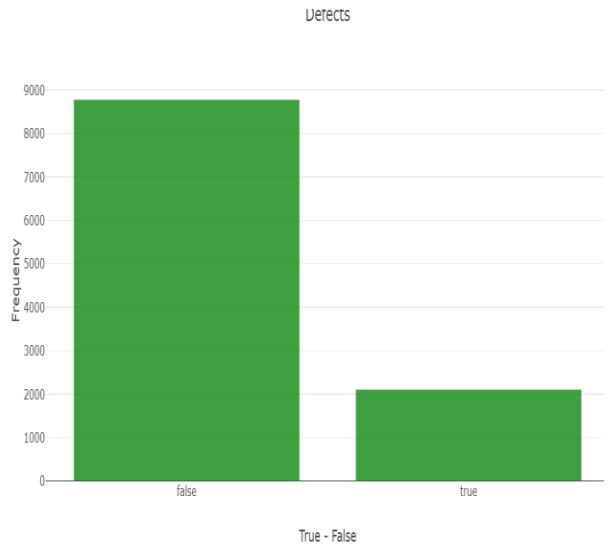
In [4]:
```
data.head() #shows first 5 rows
```

Out[4]:

	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i	e	b	t	lOCode	lOComment	lOBlank	locCodeAndCom
0	1.1	1.4	1.4	1.4	1.3	1.30	1.30	1.30	1.30	1.30	1.30	1.30	2	2	2	2
1	1.0	1.0	1.0	1.0	1.0	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1	1	1	1
2	72.0	7.0	1.0	6.0	198.0	1134.13	0.05	20.31	55.85	23029.10	0.38	1279.39	51	10	8	1
3	190.0	3.0	1.0	3.0	600.0	4348.76	0.06	17.06	254.87	74202.67	1.45	4122.37	129	29	28	2
4	37.0	4.0	1.0	4.0	126.0	599.12	0.06	17.19	34.86	10297.30	0.20	572.07	28	1	6	0

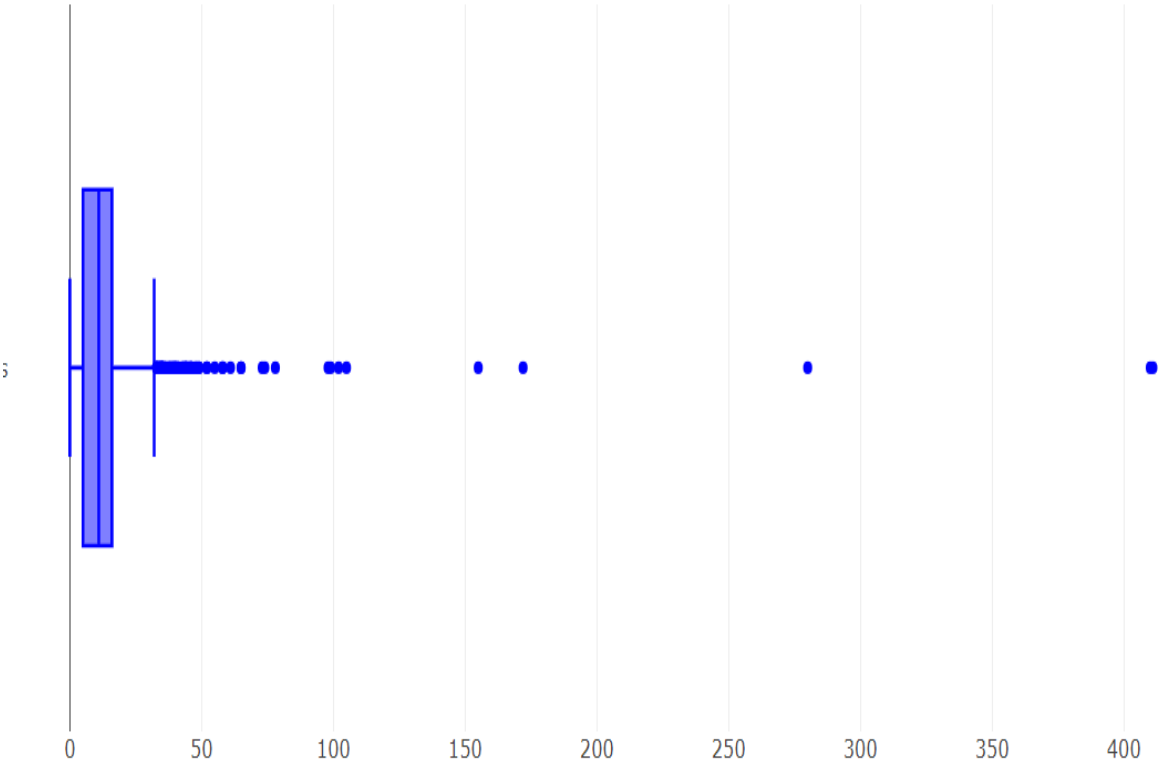
5.Histogram

```
trace = go.Histogram(
    x = data.defects,
    opacity = 0.75,
    name = "Defects",
    marker = dict(color = 'green'))

hist_data = [trace]
hist_layout = go.Layout(barmode='overlay',
                   title = 'Defects',
                   xaxis = dict(title = 'True - False'),
                   yaxis = dict(title = 'Frequency'),
)
fig = go.Figure(data = hist_data, layout = hist_layout)
iplot(fig)
```



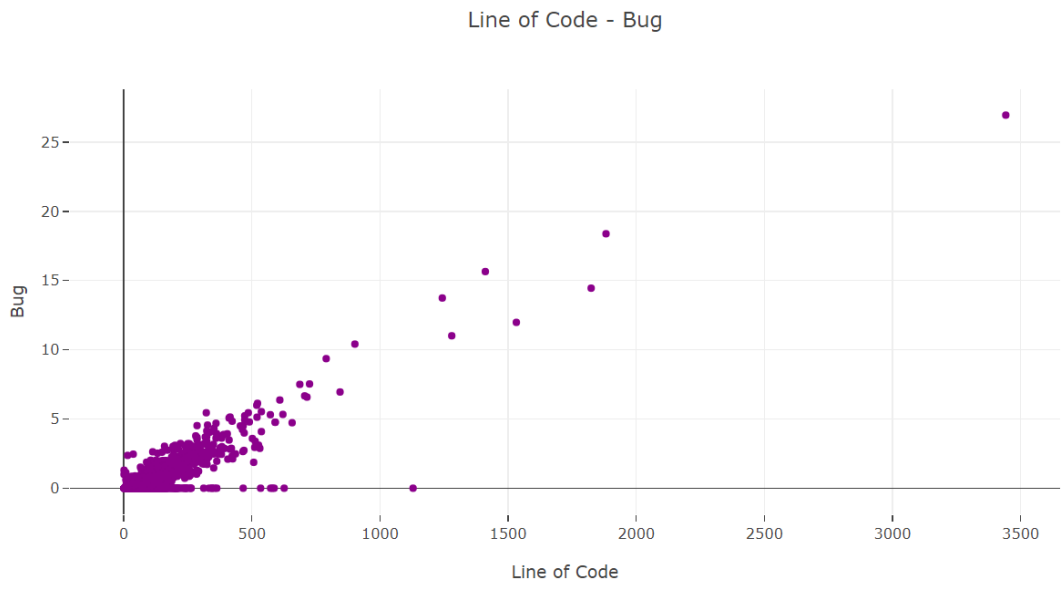
6.Outilier Detection(Box-Plot analysis)-Feature considered-#unique operators

```
trace1 = go.Box(
    x = data.uniq_Op,
    name = 'Unique Operators',
    marker = dict(color = 'blue')
    )
box_data = [trace1]
iplot(box_data)
```



7.Scatter Plot

```
#Scatter Plot
trace = go.Scatter(
    x = data['loc'],
    y = data.b,
    mode = "markers",
    name = "Line of Code - Bug",
    marker = dict(color = 'darkmagenta'),
    text = "Bug (b)")

scatter_data = [trace]
scatter_layout = dict(title = 'Line of Code - Bug',
              xaxis = dict(title = 'Line of Code', ticklen = 5),
              yaxis = dict(title = 'Bug' , ticklen = 5),
             )
fig = dict(data = scatter_data, layout = scatter_layout)
iplot(fig)
```



Learnings: Various EDA techniques and their applications were learned.

EXPERIMENT - 04

Objective

Write a program to perform feature reduction techniques for the collected dataset.

a. Correlation-based feature evaluation
b. Relief attribute feature evaluation
c. Information gain feature evaluation
d. Principle Component Analysis

Theory

Feature reduction technique is a way of converting the higher dimensions dataset into lesser dimensions dataset ensuring that it provides similar information.

Principal Component Analysis is a statistical procedure to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables.

CFS (Correlation-based Feature Selection) is an algorithm that couples this evaluation formula with an appropriate correlation measure and a heuristic search strategy.

Relief is an algorithm that takes a filter-method approach to feature selection that is notably sensitive to feature interactions.

Information Gain is defined as the amount of information provided by the feature items for the text category.

CODE AND OUTPUT

1.Correlation based feature-evaluation

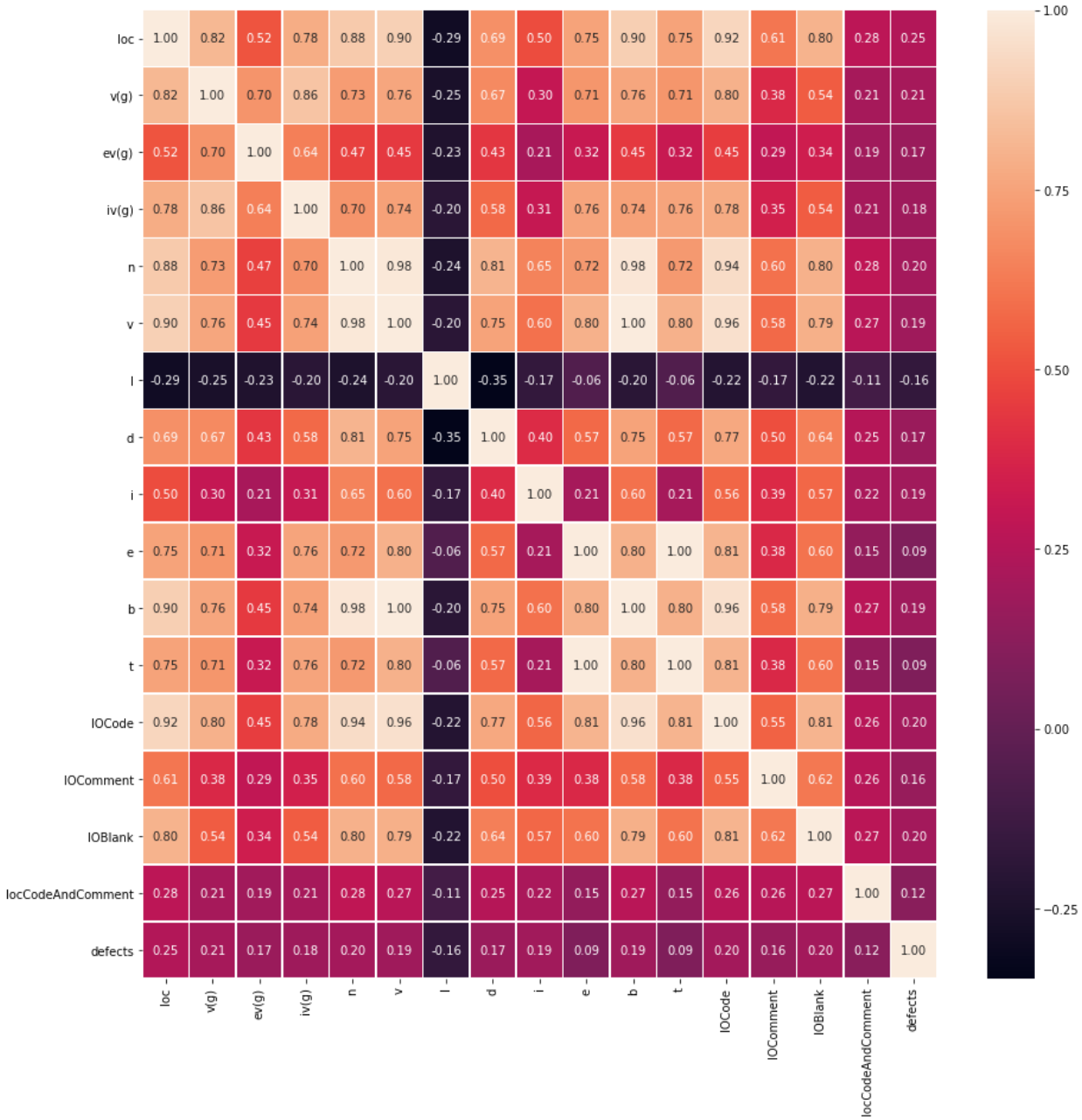
Correlation states how the features are related to each other or the target variable. Heatmap makes it easy to identify which features are most related to the target variable

```
data.corr() #shows coveriance matrix
```

	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i
loc	1.000000	0.817757	0.517551	0.784057	0.881795	0.900293	-0.286587	0.689543	0.499946
v(g)	0.817757	1.000000	0.701710	0.859590	0.730781	0.759881	-0.252902	0.669057	0.303031
ev(g)	0.517551	0.701710	1.000000	0.639574	0.465992	0.445902	-0.233982	0.434009	0.213211
iv(g)	0.784057	0.859590	0.639574	1.000000	0.702415	0.743193	-0.197736	0.575369	0.309717
n	0.881795	0.730781	0.465992	0.702415	1.000000	0.984276	-0.240749	0.808113	0.651209
v	0.900293	0.759881	0.445902	0.743193	0.984276	1.000000	-0.198104	0.752206	0.598743
l	-0.286587	-0.252902	-0.233982	-0.197736	-0.240749	-0.198104	1.000000	-0.347215	-0.166801
d	0.689543	0.669057	0.434009	0.575369	0.808113	0.752206	-0.347215	1.000000	0.398162
i	0.499946	0.303031	0.213211	0.309717	0.651209	0.598743	-0.166801	0.398162	1.000000
e	0.750564	0.709501	0.315538	0.757702	0.716536	0.800000	-0.062026	0.574298	0.209268
b	0.899965	0.759635	0.445693	0.743013	0.983938	0.999696	-0.196147	0.751835	0.598341
t	0.750564	0.709501	0.315538	0.757702	0.716536	0.800000	-0.062026	0.574298	0.209268
lOCode	0.921918	0.799915	0.454604	0.775873	0.944383	0.962078	-0.218373	0.768188	0.563920

Heatmap

```
f,ax = plt.subplots(figsize = (15, 15))
sns.heatmap(data.corr(), annot = True, linewidths = .5, fmt = '.2f')
plt.show()
```


loc 1.00 0.82 0.52 0.78 0.88 0.90 -0.29 0.69 0.50 0.75 0.90 0.75 0.92 0.61 0.80 0.28 0.25
v(g) 0.82 1.00 0.70 0.86 0.73 0.76 -0.25 0.67 0.30 0.71 0.76 0.71 0.80 0.38 0.54 0.21 0.21
ev(g) 0.52 0.70 1.00 0.64 0.47 0.45 -0.23 0.43 0.21 0.32 0.45 0.32 0.45 0.29 0.34 0.19 0.17
iv(g) 0.78 0.86 0.64 1.00 0.70 0.74 -0.20 0.58 0.31 0.76 0.74 0.76 0.78 0.35 0.54 0.21 0.18
n 0.88 0.73 0.47 0.70 1.00 0.98 -0.24 0.81 0.65 0.72 0.98 0.72 0.94 0.60 0.80 0.28 0.20
v 0.90 0.76 0.45 0.74 0.98 1.00 -0.20 0.75 0.60 0.80 1.00 0.80 0.96 0.58 0.79 0.27 0.19
l -0.29 -0.25 -0.23 -0.20 -0.24 -0.20 1.00 -0.35 -0.17 -0.06 -0.20 -0.06 -0.22 -0.17 -0.22 -0.11 -0.16
d 0.69 0.67 0.43 0.58 0.81 0.75 -0.35 1.00 0.40 0.57 0.75 0.57 0.77 0.50 0.64 0.25 0.17
i 0.50 0.30 0.21 0.31 0.65 0.60 -0.17 0.40 1.00 0.21 0.60 0.21 0.56 0.39 0.57 0.22 0.19
e 0.75 0.71 0.32 0.76 0.72 0.80 -0.06 0.57 0.21 1.00 0.80 1.00 0.81 0.38 0.60 0.15 0.09
b 0.90 0.76 0.45 0.74 0.98 1.00 -0.20 0.75 0.60 0.80 1.00 0.80 0.96 0.58 0.79 0.27 0.19
t 0.75 0.71 0.32 0.76 0.72 0.80 -0.06 0.57 0.21 1.00 0.80 1.00 0.81 0.38 0.60 0.15 0.09
lOCode 0.92 0.80 0.45 0.78 0.94 0.96 -0.22 0.77 0.56 0.81 0.96 0.81 1.00 0.55 0.81 0.26 0.20
lOComment 0.61 0.38 0.29 0.35 0.60 0.58 -0.17 0.50 0.39 0.38 0.58 0.38 0.55 1.00 0.62 0.26 0.16
lOBlank 0.80 0.54 0.34 0.54 0.80 0.79 -0.22 0.64 0.57 0.60 0.79 0.60 0.81 0.62 1.00 0.27 0.20
locCodeAndComment 0.28 0.21 0.19 0.21 0.28 0.27 -0.11 0.25 0.22 0.15 0.27 0.15 0.26 0.26 0.27 1.00 0.12
defects 0.25 0.21 0.17 0.18 0.20 0.19 -0.16 0.17 0.19 0.09 0.19 0.09 0.20 0.16 0.20 0.12 1.00
loc v(g) ev(g) iv(g) n v l d i e b t lOCode lOComment lOBlank locCodeAndComment defects
1.00 0.75 0.50 0.25 0.00 −0.25

2. Principal Component Analysis.

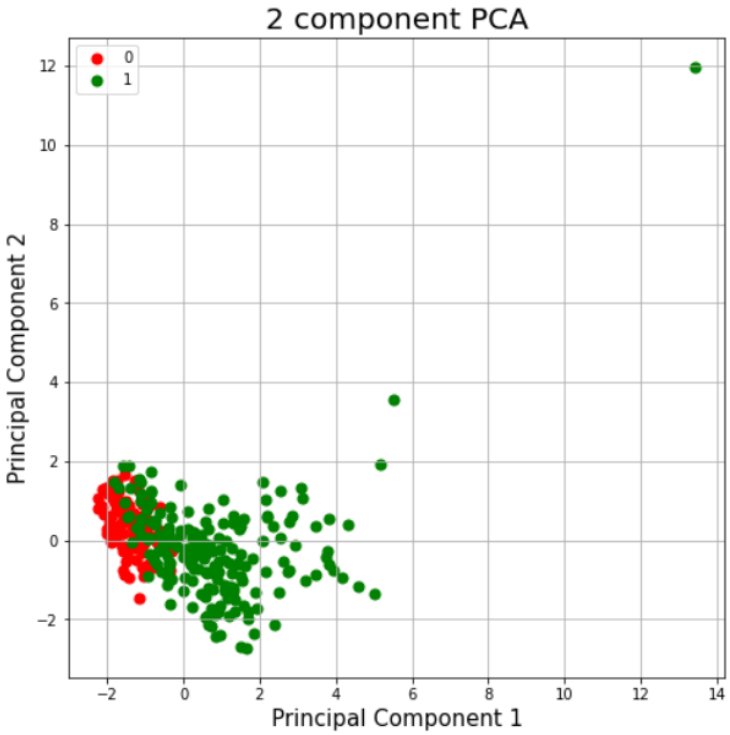
```
# Importing standardscalar module
from sklearn.preprocessing import StandardScaler

scalar = StandardScaler()

# fitting
df2 = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/jm1.csv.xls',usecols=['n','v'])
scalar.fit (df2)
scaled_data = scalar.transform(df2)

# Importing PCA
from sklearn.decomposition import PCA

# Let's say, components = 2
pca = PCA (n_components = 2)
pca.fit(scaled_data)
x_pca = pca.transform(scaled_data)

x_pca.shape
```

```
(10885, 2)
```

1. Applying the PCA function into the training and testing set for analysis.

```
pca.components_
```

```
array([[ 0.70710678,  0.70710678],
       [-0.70710678,  0.70710678]])
```



3. Information Gain

1. The unique() function finds the unique elements of an array and returns these unique elements as a sorted array

```
df2['v'].unique()
```

```
array([1.30000e+00, 1.00000e+00, 1.13413e+03, ..., 7.30800e+01,
       4.47830e+02, 5.19570e+02])
```

2. Splitting the dataset into the Training set and Test set

```
from sklearn.model_selection import train_test_split
X_train, X_test,y_train,y_test=train_test_split(df2.drop(labels=['v'], axis=1),
df2['v'],
    test_size=0.3,
    random_state=0)
X_train.head()
```

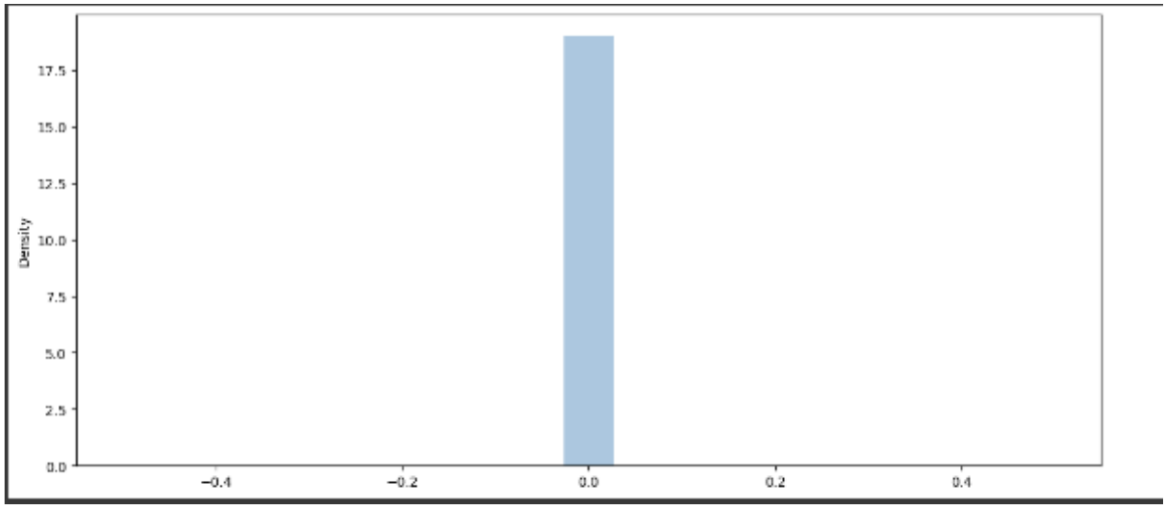
	n
8620	89.0
8985	65.0
6171	103.0
6473	0.0
10876	21.0

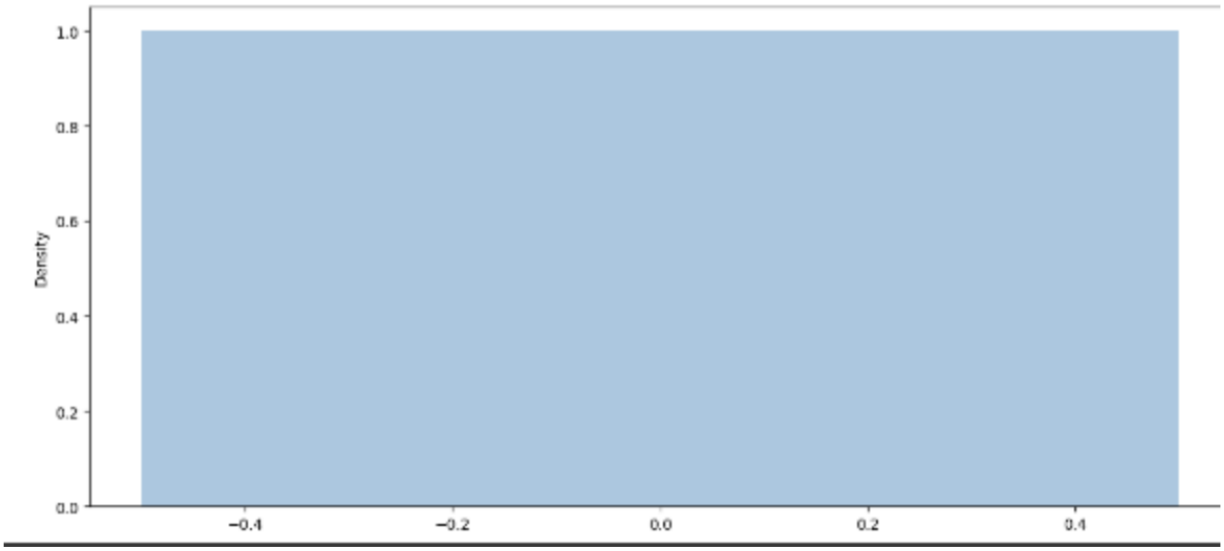
```python
import pandas as pd
import numpy as np
from math import log2

# Load JM1 dataset
df = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/jm1.csv.xls")

# Calculate proportion of defective modules
p_defect = df["defects"].sum() / len(df)

# Calculate entropy of target variable
entropy_target = - p_defect * log2(p_defect) - (1 - p_defect) * log2(1 - p_defect)

# Calculate information gain of each feature
info_gain = {}
for feature in df.columns[:-1]:
    values = df[feature].unique()
    entropy_feature = 0
    for value in values:
        p = len(df[df[feature] == value]) / len(df)
        p_defect_given_value = len(df[(df[feature] == value) & (df["defects"] == True)]) / len(df[df[feature] == value])
        if p_defect_given_value == 0 or p_defect_given_value == 1:
            entropy_feature += 0
        else:
            entropy_feature -= p * (p_defect_given_value * log2(p_defect_given_value) + (1 - p_defect_given_value) * log2(1 - p_defect_given_value))
    info_gain[feature] = entropy_target - entropy_feature

# Print information gain of each feature in descending order
for feature, gain in sorted(info_gain.items(), key=lambda x: x[1], reverse=True):
    print(f"{feature}: {gain}")
```

```
e: 0.5442856427242134
t: 0.5323855639770098
i: 0.3785041559721106
v: 0.37221316357242207
d: 0.26557587667006377
n: 0.12229976631083905
total_Op: 0.10245356971150932
loc: 0.0991885867686716
total_Opnd: 0.08776302336794228
lOCode: 0.07672143425733469
b: 0.06805604560464573
uniq_Opnd: 0.06511234529780341
branchCount: 0.059904257380301584
v(g): 0.0562191977861054
iv(g): 0.05401840677080738
lOBlank: 0.05169097063726402
uniq_Op: 0.04680001602268968
l: 0.040502620964596336
ev(g): 0.03022312587203002
lOComment: 0.02996569050471265
locCodeAndComment: 0.016928375244882
```

4.Relief Attributes

```
dc = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/jm1.csv.xls',usecols=['n','v'])

n = dc["n"]
dc = dc.T
dc = dc[1:]
dc.columns = [n]

dc.reset_index(drop=True, inplace=True)

scaler = MinMaxScaler ()
dc = power_transform(dc, method='yeo-johnson')
de = scaler.fit_transform(dc)
X = de[:,0: 396]
y = dc[:,396]
fig, ax=plt.subplots (1,2, figsize=(30, 0))
sns.distplot(X, ax=ax[0])
sns.distplot(y, ax=ax[1])
```





The main focus of this kernel is the Relief algorithm, but let's spend some time on the data preprocessing, to make our job easier.

```
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.3, random_state = 0)
r = sr.RReliefF(n_features = 20)
print(r.fit_transform(X_train,y_train))
```

```
[[0.89415851 0.36422555 0.62621622 ... 0.45517032 0.71635848 0.6717487 ]
 [0.67871428 0.48460741 0.74877678 ... 0.         0.83808516 0.32459573]
 [0.96491183 0.36422555 0.50591949 ... 0.56246492 0.655395   0.77543588]
 ...
 [0.77175778 0.54000668 0.56577229 ... 0.         0.83808516 0.2379575 ]
 [0.60169922 0.60478506 0.62621622 ... 0.17679116 0.63505834 0.2818013 ]
 [0.60169922 0.70105142 0.44668632 ... 0.24122983 0.51286805 0.44732296]]
```

LEARNING

The following are key learnings for performing feature reduction techniques on a collected dataset. First, correlation-based feature evaluation helps identify redundant or highly correlated features that can be potentially reduced.

Second, relief attribute feature evaluation using algorithms like ReliefF or SURF assesses the relevance of features based on their contribution to the prediction task.

Third, information gain feature evaluation measures the predictive power of features using entropy or information gain. Lastly, Principal Component Analysis (PCA) can effectively reduce dimensionality by projecting the dataset onto a lower-dimensional space while retaining the most important features.

Experimenting with different techniques and selecting the most appropriate one based on the specific dataset and prediction task is crucial for successful feature reduction.

EXPERIMENT - 05

Objective

Develop a machine learning model for the selected topic (minimum 10 datasets and 10 techniques).

Theory

1. **SVM**: A support vector machine is a type of deep learning algorithm that performs supervised learning for the classification or regression of data groups.
2. **Logistic Regression**: Logistic regression is a supervised learning classification algorithm used to predict the probability of a target variable.
3. **Naive Bayes**: Naïve Bayes algorithm is a supervised learning algorithm, which is based on the Bayes theorem and used for solving classification problems.
4. **Decision Tree**: It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.
5. **Random Forest**: It is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset.
6. **XGBoost**: XGBoost algorithm makes use of fast parallel prefix sum operations to scan through all possible splits, as well as parallel radix sorting to repartition data.
7. **KNN:** KNN is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point.
8. **LSTM:** It is a variety of recurrent neural networks (RNNs) that are capable of learning long-term dependencies, especially in sequence prediction problems.
9. **CatBoost**: CatBoost is an algorithm for gradient boosting on decision trees
10. **ANN**: An artificial neural network is an attempt to simulate the network of neurons that make up a human brain so that the computer will be able to learn things and make decisions.

CODE AND OUTPUT

1. Dataset: ant-1.3

- **Importing the Libraries**

```
import pandas as pd
import numpy as np
import seaborn as sns
```

- **Loading the dataset**

```
data = pd.read_csv('/content/ant-1.3.csv')
# data['defects'] = data['defects'].replace({True:1,False:0})
data.head()
```

	wmc	dit	noc	cbo	rfc	lcom	ca	ce	npm	lcom3	...	dam	moa	mfa	cam	ic	cbm	amc	max_cc	avg_cc	bug
0	11	4	2	14	42	29	2	12	5	0.725000	...	1.0	1	0.885057	0.232323	3	4	34.545455	3	1.2727	0
1	14	1	1	8	32	49	4	4	12	0.835165	...	1.0	0	0.000000	0.307692	0	0	16.857143	6	1.6429	1
2	3	2	0	1	9	0	0	1	1	0.000000	...	1.0	1	0.714286	0.666667	1	1	17.333333	1	0.6667	0
3	12	3	0	12	37	32	0	12	12	0.858586	...	1.0	1	0.770833	0.458333	0	0	24.083333	3	1.4167	0
4	6	3	0	4	21	1	0	4	6	0.700000	...	1.0	0	0.880952	0.416667	2	2	21.000000	1	0.8333	0

5 rows × 21 columns

- **Training the data**

```
x = data.drop(['bug'],axis=1)
y = data['bug']
```

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2,random_state=4)
```

- **LOGISTIC REGRESSION**

```
# LogisticRegression
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()

# train the logistic regression classifier using the training data
logreg.fit(x_train, y_train)

# make predictions on the testing data
y_pred = logreg.predict(x_test)

from sklearn.metrics import accuracy_score

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.8
/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

- **SVM**

```
# SVM

from sklearn import svm
classifier = svm.SVC(kernel='linear',gamma='auto',C=2)
classifier.fit(x_train,y_train)
y_predict = classifier.predict(x_test)

from sklearn.metrics import accuracy_score

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_predict)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.84
```

- **NAIVE BAYES**

```
# Naive Bayes
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()

# train the Naive Bayes classifier using the training data
gnb.fit(x_train, y_train)

# make predictions on the testing data
y_pred = gnb.predict(x_test)

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.88
```

- **DECISION TREE**

```
# Decision Tree
from sklearn.tree import DecisionTreeClassifier

# create a Decision Tree classifier object
dtc = DecisionTreeClassifier()

# train the Decision Tree classifier using the training data
dtc.fit(x_train, y_train)

# make predictions on the testing data
y_pred = dtc.predict(x_test)

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.8
```

- **RANDOM FOREST**

```
# Random Forest
from sklearn.ensemble import RandomForestClassifier


# create a Random Forest classifier object
rfc = RandomForestClassifier(n_estimators=100)

# train the Random Forest classifier using the training data
rfc.fit(x_train, y_train)

# make predictions on the testing data
y_pred = rfc.predict(x_test)

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.8
```

- **XGBOOST**

```
# XGBoost

import xgboost as xgb

# convert data into DMatrix format
dtrain = xgb.DMatrix(x_train, label=y_train)
dtest = xgb.DMatrix(x_test, label=y_test)

# set the parameters for XGBoost
params = {
    'max_depth': 3,
    'eta': 0.1,
    'objective': 'multi:softmax',
    'num_class': 3
}

# train the XGBoost model
num_rounds = 100
xgb_model = xgb.train(params, dtrain, num_rounds)

# make predictions on the testing data
y_pred = xgb_model.predict(dtest)

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.76
```

- **KNN**

```
# KNN

from sklearn.neighbors import KNeighborsClassifier


# create a KNN classifier object
knn = KNeighborsClassifier(n_neighbors=5)

# train the KNN classifier using the training data
knn.fit(x_train, y_train)

# make predictions on the testing data
y_pred = knn.predict(x_test)

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.8
```

- **CATBOOST**

```
!pip install catboost
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting catboost
  Downloading catboost-1.1.1-cp39-none-manylinux1_x86_64.whl (76.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 76.6/76.6 MB 9.2 MB/s eta 0:00:00
Requirement already satisfied: graphviz in /usr/local/lib/python3.9/dist-packages (from catboost) (0.20.1)
Requirement already satisfied: six in /usr/local/lib/python3.9/dist-packages (from catboost) (1.16.0)
Requirement already satisfied: numpy>=1.16.0 in /usr/local/lib/python3.9/dist-packages (from catboost) (1.22.4)
Requirement already satisfied: scipy in /usr/local/lib/python3.9/dist-packages (from catboost) (1.10.1)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.9/dist-packages (from catboost) (3.7.1)
Requirement already satisfied: plotly in /usr/local/lib/python3.9/dist-packages (from catboost) (5.13.1)
Requirement already satisfied: pandas>=0.24.0 in /usr/local/lib/python3.9/dist-packages (from catboost) (1.5.3)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.9/dist-packages (from pandas>=0.24.0->catboost) (2022.7.1)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.9/dist-packages (from pandas>=0.24.0->catboost) (2.8.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (8.4.0)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (23.1)
Requirement already satisfied: importlib-resources>=3.2.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (5.12.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (1.0.7)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (1.4.4)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (4.39.3)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (3.0.9)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.9/dist-packages (from plotly->catboost) (8.2.2)
Requirement already satisfied: zipp>=3.1.0 in /usr/local/lib/python3.9/dist-packages (from importlib-resources>=3.2.0->matplotlib->catboost) (3.15.0)
Installing collected packages: catboost
Successfully installed catboost-1.1.1
```

```
# CatBoost
import catboost as ctb
# create a CatBoost classifier object
clf = ctb.CatBoostClassifier()

# train the CatBoost classifier using the training data
clf.fit(x_train, y_train)

# make predictions on the testing data
y_pred = clf.predict(x_test)

# calculate the accuracy of the model
accuracy = clf.score(x_test, y_test)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Learning rate set to 0.003854
0:      learn: 0.6905412        total: 48.3ms   remaining: 48.2s
1:      learn: 0.6870404        total: 49.5ms   remaining: 24.7s
2:      learn: 0.6841782        total: 51.1ms   remaining: 17s
3:      learn: 0.6813223        total: 52.6ms   remaining: 13.1s
4:      learn: 0.6781163        total: 54.1ms   remaining: 10.8s
5:      learn: 0.6739479        total: 55.3ms   remaining: 9.17s
6:      learn: 0.6704998        total: 57.4ms   remaining: 8.14s
7:      learn: 0.6673006        total: 59.4ms   remaining: 7.37s
8:      learn: 0.6641527        total: 60.9ms   remaining: 6.71s
9:      learn: 0.6609137        total: 62.1ms   remaining: 6.15s
10:     learn: 0.6580794        total: 63.6ms   remaining: 5.72s
11:     learn: 0.6555626        total: 65ms     remaining: 5.35s
12:     learn: 0.6522717        total: 66.6ms   remaining: 5.05s
13:     learn: 0.6481547        total: 68ms     remaining: 4.79s
14:     learn: 0.6452119        total: 69.6ms   remaining: 4.57s
15:     learn: 0.6424734        total: 70.8ms   remaining: 4.35s
16:     learn: 0.6392949        total: 72.5ms   remaining: 4.19s
...
984:    learn: 0.0875002        total: 1.71s    remaining: 26ms
985:    learn: 0.0873728        total: 1.71s    remaining: 24.3ms
986:    learn: 0.0872812        total: 1.71s    remaining: 22.6ms
987:    learn: 0.0872301        total: 1.71s    remaining: 20.8ms
988:    learn: 0.0871375        total: 1.72s    remaining: 19.1ms
989:    learn: 0.0870004        total: 1.72s    remaining: 17.3ms
990:    learn: 0.0868934        total: 1.72s    remaining: 15.6ms
991:    learn: 0.0868039        total: 1.72s    remaining: 13.9ms
992:    learn: 0.0867514        total: 1.72s    remaining: 12.1ms
993:    learn: 0.0867056        total: 1.72s    remaining: 10.4ms
994:    learn: 0.0866207        total: 1.72s    remaining: 8.66ms
995:    learn: 0.0865224        total: 1.73s    remaining: 6.93ms
996:    learn: 0.0864041        total: 1.73s    remaining: 5.2ms
997:    learn: 0.0863369        total: 1.73s    remaining: 3.47ms
998:    learn: 0.0862021        total: 1.73s    remaining: 1.74ms
999:    learn: 0.0860652        total: 1.74s    remaining: 0us
Accuracy: 0.8
```

- **LSTM**

```
# LSTM

from keras.models import Sequential
from keras.layers import LSTM, Dense

# reshape the input data to be 3D for LSTM
x_train = x_train.values.reshape((x_train.shape[0], 1, x_train.shape[1]))
x_test = x_test.values.reshape((x_test.shape[0], 1, x_test.shape[1]))

# define the LSTM model
model = Sequential()
model.add(LSTM(50, input_shape=(x_train.shape[1], x_train.shape[2])))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# train the LSTM model
history = model.fit(x_train, y_train, epochs=50, batch_size=32, validation_data=(x_test, y_test), verbose=2, shuffle=False)

# evaluate the LSTM model
loss, accuracy = model.evaluate(x_test, y_test, verbose=0)
print('Accuracy:', accuracy)
```

```
4/4 - 0s - loss: 0.3533 - accuracy: 0.8400 - val_loss: 0.3574 - val_accuracy: 0.8400 - 100ms/epoch - 25ms/step
Epoch 23/50
4/4 - 0s - loss: 0.3498 - accuracy: 0.8400 - val_loss: 0.3560 - val_accuracy: 0.8400 - 102ms/epoch - 26ms/step
Epoch 24/50
4/4 - 0s - loss: 0.3473 - accuracy: 0.8400 - val_loss: 0.3550 - val_accuracy: 0.8400 - 89ms/epoch - 22ms/step
Epoch 25/50
4/4 - 0s - loss: 0.3447 - accuracy: 0.8400 - val_loss: 0.3556 - val_accuracy: 0.8400 - 109ms/epoch - 27ms/step
Epoch 26/50
4/4 - 0s - loss: 0.3422 - accuracy: 0.8400 - val_loss: 0.3581 - val_accuracy: 0.8400 - 140ms/epoch - 35ms/step
Epoch 27/50
4/4 - 0s - loss: 0.3396 - accuracy: 0.8400 - val_loss: 0.3591 - val_accuracy: 0.8400 - 111ms/epoch - 28ms/step
Epoch 28/50
4/4 - 0s - loss: 0.3380 - accuracy: 0.8400 - val_loss: 0.3582 - val_accuracy: 0.8400 - 100ms/epoch - 25ms/step
Epoch 29/50
4/4 - 0s - loss: 0.3354 - accuracy: 0.8400 - val_loss: 0.3565 - val_accuracy: 0.8400 - 181ms/epoch - 45ms/step
Epoch 30/50
```

- **ANN**

```
# ANN

# import pandas as pd
from keras.models import Sequential
from keras.layers import Dense

# load the dataset
# df = pd.read_csv('dataset.csv')

# split the dataset into input and output variables
# X = df.drop(columns=['output'])
# y = df['output']

# define the model
model = Sequential()
model.add(Dense(10, input_dim=x.shape[1], activation='relu'))
model.add(Dense(1, activation='linear'))

# compile the model
model.compile(loss='mse', optimizer='adam', metrics=['mae'])

# train the model
model.fit(x, y, epochs=100, batch_size=32)

# make predictions
y_pred = model.predict(x)

# evaluate the model
mse, mae = model.evaluate(x, y)

print('MSE: ', mse)
print('MAE: ', mae)
```

```
Epoch 1/100
4/4 [==============================] - 2s 10ms/step - loss: 42228.5781 - mae: 134.7763
Epoch 2/100
4/4 [==============================] - 0s 11ms/step - loss: 37714.4492 - mae: 127.1286
Epoch 3/100
4/4 [==============================] - 0s 10ms/step - loss: 33068.8438 - mae: 119.4881
Epoch 4/100
4/4 [==============================] - 0s 13ms/step - loss: 29463.2969 - mae: 112.5780
Epoch 5/100
4/4 [==============================] - 0s 10ms/step - loss: 25403.9316 - mae: 105.0793
Epoch 6/100
4/4 [==============================] - 0s 9ms/step - loss: 22566.9375 - mae: 98.4859
Epoch 7/100
4/4 [==============================] - 0s 10ms/step - loss: 19277.5703 - mae: 91.5703
Epoch 8/100
4/4 [==============================] - 0s 6ms/step - loss: 16770.0020 - mae: 85.3094
Epoch 9/100
4/4 [==============================] - 0s 11ms/step - loss: 14788.3193 - mae: 79.4386
Epoch 10/100
4/4 [==============================] - 0s 25ms/step - loss: 12330.2080 - mae: 73.2297
Epoch 11/100
4/4 [==============================] - 0s 12ms/step - loss: 10790.4814 - mae: 67.8481
Epoch 12/100
Epoch 92/100
4/4 [==============================] - 0s 7ms/step - loss: 56.6724 - mae: 3.5093
Epoch 93/100
4/4 [==============================] - 0s 6ms/step - loss: 56.5220 - mae: 3.5057
Epoch 94/100
4/4 [==============================] - 0s 6ms/step - loss: 56.3604 - mae: 3.4995
Epoch 95/100
4/4 [==============================] - 0s 6ms/step - loss: 56.1989 - mae: 3.4950
Epoch 96/100
4/4 [==============================] - 0s 9ms/step - loss: 56.0513 - mae: 3.4872
Epoch 97/100
4/4 [==============================] - 0s 10ms/step - loss: 56.1042 - mae: 3.4880
Epoch 98/100
4/4 [==============================] - 0s 7ms/step - loss: 55.7736 - mae: 3.4804
Epoch 99/100
4/4 [==============================] - 0s 6ms/step - loss: 55.6583 - mae: 3.4760
Epoch 100/100
4/4 [==============================] - 0s 12ms/step - loss: 55.5169 - mae: 3.4713
4/4 [==============================] - 0s 4ms/step
4/4 [==============================] - 0s 7ms/step - loss: 55.4180 - mae: 3.4683
MSE:  55.4179573059082
MAE:  3.4683432579040527
```

2. Dataset: ant-1.4

- **Importing the Libraries**

```
import pandas as pd
import numpy as np
import seaborn as sns
```

- **Loading the dataset**

```
data = pd.read_csv('/content/ant-1.4.csv')
# data['defects'] = data['defects'].replace({True:1,False:0})
data.head()
```

	wmc	dit	noc	cbo	rfc	lcom	ca	ce	npm	lcom3	...	dam	moa	mfa	cam	ic	cbm	amc	max_cc	avg_cc	bug
0	14	3	0	8	41	33	1	8	10	0.760684	...	1.0	2	0.740000	0.357143	1	1	25.857143	4	1.5714	0
1	5	3	0	9	43	0	0	9	5	0.666667	...	1.0	0	0.902439	0.533333	2	3	53.600000	1	0.8000	0
2	13	1	1	8	20	46	6	2	8	0.666667	...	1.0	0	0.000000	0.615385	0	0	7.230769	1	0.7692	0
3	8	4	0	7	35	4	0	7	7	0.714286	...	1.0	0	0.917647	0.468750	3	3	31.375000	5	1.3750	0
4	31	1	0	6	65	243	4	2	30	0.726667	...	1.0	0	0.000000	0.182796	1	1	22.612903	4	1.1613	0

- **Training the data**

```
x = data.drop(['bug'],axis=1)
y = data['bug']
```

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2,random_state=4)
```

- **LOGISTIC REGRESSION**

```
# LogisticRegression
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()

# train the logistic regression classifier using the training data
logreg.fit(x_train, y_train)

# make predictions on the testing data
y_pred = logreg.predict(x_test)

from sklearn.metrics import accuracy_score

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.8333333333333334
/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

- **SVM**

```
# SVM

from sklearn import svm
classifier = svm.SVC(kernel='linear',gamma='auto',C=2)
classifier.fit(x_train,y_train)
y_predict = classifier.predict(x_test)

from sklearn.metrics import accuracy_score

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_predict)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.8333333333333334
```

- **NAIVE BAYES**

```
# Naive Bayes
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()

# train the Naive Bayes classifier using the training data
gnb.fit(x_train, y_train)

# make predictions on the testing data
y_pred = gnb.predict(x_test)

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.3333333333333333
```

- **DECISION TREE**

```
# Decision Tree
from sklearn.tree import DecisionTreeClassifier

# create a Decision Tree classifier object
dtc = DecisionTreeClassifier()

# train the Decision Tree classifier using the training data
dtc.fit(x_train, y_train)

# make predictions on the testing data
y_pred = dtc.predict(x_test)

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.7777777777777778
```

- **RANDOM FOREST**

```
# Random Forest
from sklearn.ensemble import RandomForestClassifier


# create a Random Forest classifier object
rfc = RandomForestClassifier(n_estimators=100)

# train the Random Forest classifier using the training data
rfc.fit(x_train, y_train)

# make predictions on the testing data
y_pred = rfc.predict(x_test)

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.8611111111111112
```

- **XGBOOST**

```
# XGBoost

import xgboost as xgb

# convert data into DMatrix format
dtrain = xgb.DMatrix(x_train, lab Loading... )
dtest = xgb.DMatrix(x_test, label=y_test)

# set the parameters for XGBoost
params = {
    'max_depth': 3,
    'eta': 0.1,
    'objective': 'multi:softmax',
    'num_class': 3
}

# train the XGBoost model
num_rounds = 100
xgb_model = xgb.train(params, dtrain, num_rounds)

# make predictions on the testing data
y_pred = xgb_model.predict(dtest)

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.8333333333333334
```

- **KNN**

```
# KNN

from sklearn.neighbors import KNeighborsClassifier


# create a KNN classifier object
knn = KNeighborsClassifier(n_neighbors=5)

# train the KNN classifier using the training data
knn.fit(x_train, y_train)

# make predictions on the testing data
y_pred = knn.predict(x_test)

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.8055555555555556
```

- **CATBOOST**

```
!pip install catboost
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting catboost
  Downloading catboost-1.1.1-cp39-none-manylinux1_x86_64.whl (76.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 76.6/76.6 MB 9.2 MB/s eta 0:00:00
Requirement already satisfied: graphviz in /usr/local/lib/python3.9/dist-packages (from catboost) (0.20.1)
Requirement already satisfied: six in /usr/local/lib/python3.9/dist-packages (from catboost) (1.16.0)
Requirement already satisfied: numpy>=1.16.0 in /usr/local/lib/python3.9/dist-packages (from catboost) (1.22.4)
Requirement already satisfied: scipy in /usr/local/lib/python3.9/dist-packages (from catboost) (1.10.1)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.9/dist-packages (from catboost) (3.7.1)
Requirement already satisfied: plotly in /usr/local/lib/python3.9/dist-packages (from catboost) (5.13.1)
Requirement already satisfied: pandas>=0.24.0 in /usr/local/lib/python3.9/dist-packages (from catboost) (1.5.3)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.9/dist-packages (from pandas>=0.24.0->catboost) (2022.7.1)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.9/dist-packages (from pandas>=0.24.0->catboost) (2.8.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (8.4.0)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (23.1)
Requirement already satisfied: importlib-resources>=3.2.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (5.12.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (1.0.7)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (1.4.4)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (4.39.3)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (3.0.9)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.9/dist-packages (from plotly->catboost) (8.2.2)
Requirement already satisfied: zipp>=3.1.0 in /usr/local/lib/python3.9/dist-packages (from importlib-resources>=3.2.0->matplotlib->catboost) (3.15.0)
Installing collected packages: catboost
Successfully installed catboost-1.1.1
```

```
# CatBoost
import catboost as ctb
# create a CatBoost classifier object
clf = ctb.CatBoostClassifier()

# train the CatBoost classifier using the training data
clf.fit(x_train, y_train)

# make predictions on the testing data
y_pred = clf.predict(x_test)

# calculate the accuracy of the model
accuracy = clf.score(x_test, y_test)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Learning rate set to 0.004477
0:      learn: 0.6915385        total: 48.7ms   remaining: 48.6s
1:      learn: 0.6888873        total: 54.7ms   remaining: 27.3s
2:      learn: 0.6868296        total: 58.6ms   remaining: 19.5s
3:      learn: 0.6846979        total: 61.4ms   remaining: 15.3s
4:      learn: 0.6823434        total: 69.6ms   remaining: 13.8s
5:      learn: 0.6806757        total: 71ms     remaining: 11.8s
6:      learn: 0.6779728        total: 72.6ms   remaining: 10.3s
7:      learn: 0.6758468        total: 74.7ms   remaining: 9.27s
8:      learn: 0.6733645        total: 78.1ms   remaining: 8.6s
9:      learn: 0.6716681        total: 91.6ms   remaining: 9.07s
10:     learn: 0.6691711        total: 93.5ms   remaining: 8.4s
11:     learn: 0.6672336        total: 100ms    remaining: 8.27s
12:     learn: 0.6648306        total: 102ms    remaining: 7.75s
13:     learn: 0.6631033        total: 109ms    remaining: 7.69s
988:    learn: 0.1954237        total: 3.65s    remaining: 40.6ms
989:    learn: 0.1953225        total: 3.65s    remaining: 36.9ms
990:    learn: 0.1950532        total: 3.65s    remaining: 33.2ms
991:    learn: 0.1948861        total: 3.66s    remaining: 29.5ms
992:    learn: 0.1947623        total: 3.66s    remaining: 25.8ms
993:    learn: 0.1945867        total: 3.66s    remaining: 22.1ms
994:    learn: 0.1944728        total: 3.67s    remaining: 18.4ms
995:    learn: 0.1943816        total: 3.67s    remaining: 14.7ms
996:    learn: 0.1942250        total: 3.67s    remaining: 11.1ms
997:    learn: 0.1941254        total: 3.67s    remaining: 7.37ms
998:    learn: 0.1939824        total: 3.68s    remaining: 3.68ms
999:    learn: 0.1937602        total: 3.68s    remaining: 0us
Accuracy: 0.8611111111111112
```

- **LSTM**

```
# LSTM

from keras.models import Sequential
from keras.layers import LSTM, Dense

# reshape the input data to be 3D for LSTM
x_train = x_train.values.reshape((x_train.shape[0], 1, x_train.shape[1]))
x_test = x_test.values.reshape((x_test.shape[0], 1, x_test.shape[1]))

# define the LSTM model
model = Sequential()
model.add(LSTM(50, input_shape=(x_train.shape[1], x_train.shape[2])))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# train the LSTM model
history = model.fit(x_train, y_train, epochs=50, batch_size=32, validation_data=(x_test, y_test), verbose=2, shuffle=False)

# evaluate the LSTM model
loss, accuracy = model.evaluate(x_test, y_test, verbose=0)
print('Accuracy:', accuracy)
```

```
Epoch 42/50
5/5 - 0s - loss: 0.4321 - accuracy: 0.7676 - val_loss: 0.4622 - val_accuracy: 0.8333 - 141ms/epoch - 28ms/step
Epoch 43/50
5/5 - 0s - loss: 0.4288 - accuracy: 0.7746 - val_loss: 0.4593 - val_accuracy: 0.8333 - 161ms/epoch - 32ms/step
Epoch 44/50
5/5 - 0s - loss: 0.4261 - accuracy: 0.7746 - val_loss: 0.4584 - val_accuracy: 0.8333 - 116ms/epoch - 23ms/step
Epoch 45/50
5/5 - 0s - loss: 0.4238 - accuracy: 0.7746 - val_loss: 0.4567 - val_accuracy: 0.8333 - 186ms/epoch - 37ms/step
Epoch 46/50
5/5 - 0s - loss: 0.4210 - accuracy: 0.7746 - val_loss: 0.4559 - val_accuracy: 0.8333 - 134ms/epoch - 27ms/step
Epoch 47/50
5/5 - 0s - loss: 0.4182 - accuracy: 0.7746 - val_loss: 0.4563 - val_accuracy: 0.8333 - 173ms/epoch - 35ms/step
Epoch 48/50
5/5 - 0s - loss: 0.4157 - accuracy: 0.7746 - val_loss: 0.4569 - val_accuracy: 0.8333 - 165ms/epoch - 33ms/step
Epoch 49/50
5/5 - 0s - loss: 0.4131 - accuracy: 0.7746 - val_loss: 0.4555 - val_accuracy: 0.8333 - 195ms/epoch - 39ms/step
Epoch 50/50
5/5 - 0s - loss: 0.4107 - accuracy: 0.7746 - val_loss: 0.4548 - val_accuracy: 0.8333 - 152ms/epoch - 30ms/step
Accuracy: 0.8333333134651184
```

- **ANN**

```
# ANN

# import pandas as pd
from keras.models import Sequential
from keras.layers import Dense

# load the dataset
# df = pd.read_csv('dataset.csv')

# split the dataset into input and output variables
# X = df.drop(columns=['output'])
# y = df['output']

# define the model
model = Sequential()
model.add(Dense(10, input_dim=x.shape[1], activation='relu'))
model.add(Dense(1, activation='linear'))

# compile the model
model.compile(loss='mse', optimizer='adam', metrics=['mae'])

# train the model
model.fit(x, y, epochs=100, batch_size=32)

# make predictions
y_pred = model.predict(x)

# evaluate the model
mse, mae = model.evaluate(x, y)

print('MSE: ', mse)
print('MAE: ', mae)
```

```
Epoch 1/100
6/6 [==============================] - 1s 4ms/step - loss: 3233.2529 - mae: 17.2583
Epoch 2/100
6/6 [==============================] - 0s 4ms/step - loss: 2206.4355 - mae: 13.3242
Epoch 3/100
6/6 [==============================] - 0s 4ms/step - loss: 1247.5823 - mae: 14.1575
Epoch 4/100
6/6 [==============================] - 0s 4ms/step - loss: 1103.3320 - mae: 16.2533
Epoch 5/100
6/6 [==============================] - 0s 5ms/step - loss: 829.3836 - mae: 16.9109
Epoch 6/100
6/6 [==============================] - 0s 5ms/step - loss: 730.8240 - mae: 16.4924
Epoch 7/100
6/6 [==============================] - 0s 4ms/step - loss: 604.4543 - mae: 15.1422
Epoch 8/100
6/6 [==============================] - 0s 4ms/step - loss: 501.1055 - mae: 13.7567
Epoch 9/100
6/6 [==============================] - 0s 4ms/step - loss: 408.3284 - mae: 12.4348
Epoch 10/100
6/6 [==============================] - 0s 4ms/step - loss: 334.2178 - mae: 11.1255
Epoch 11/100
6/6 [==============================] - 0s 4ms/step - loss: 291.7330 - mae: 10.2405
Epoch 12/100
6/6 [==============================] - 0s 6ms/step - loss: 259.6510 - mae: 9.5797
Epoch 13/100
6/6 [==============================] - 0s 4ms/step - loss: 233.8032 - mae: 9.2676
Epoch 14/100
6/6 [==============================] - 0s 5ms/step - loss: 211.6040 - mae: 9.1059
Epoch 15/100
6/6 [==============================] - 0s 4ms/step - loss: 207.4771 - mae: 9.1133
Epoch 16/100
```

```
Epoch 93/100
6/6 [==============================] - 0s 4ms/step - loss: 29.6056 - mae: 2.9319
Epoch 94/100
6/6 [==============================] - 0s 4ms/step - loss: 27.9850 - mae: 2.8724
Epoch 95/100
6/6 [==============================] - 0s 4ms/step - loss: 27.7913 - mae: 2.8585
Epoch 96/100
6/6 [==============================] - 0s 4ms/step - loss: 27.2346 - mae: 2.8323
Epoch 97/100
6/6 [==============================] - 0s 8ms/step - loss: 27.0015 - mae: 2.8140
Epoch 98/100
6/6 [==============================] - 0s 3ms/step - loss: 26.5171 - mae: 2.7923
Epoch 99/100
6/6 [==============================] - 0s 5ms/step - loss: 27.3255 - mae: 2.8274
Epoch 100/100
6/6 [==============================] - 0s 7ms/step - loss: 26.1082 - mae: 2.7921
6/6 [==============================] - 0s 6ms/step
6/6 [==============================] - 0s 4ms/step - loss: 25.7326 - mae: 2.7506
MSE:  25.732553482055664
MAE:  2.750598669052124
```

3. Dataset : camel-1.0

- **Importing the Libraries**

```
import pandas as pd
import numpy as np
import seaborn as sns
```

- **Loading the dataset**

```
data = pd.read_csv('/content/camel-1.0.csv')
# data['defects'] = data['defects'].replace({True:1,False:0})
data.head()
```

	wmc	dit	noc	cbo	rfc	lcom	ca	ce	npm	lcom3	...	dam	moa	mfa	cam	ic	cbm	amc	max_cc	avg_cc	bug
0	4	2	0	6	8	6	2	5	4	2.00	...	0.0	0	0.896552	0.500000	0	0	5.500000	1	0.7500	0
1	6	3	0	21	33	15	1	21	2	2.00	...	0.0	0	0.800000	0.500000	2	2	28.333333	1	0.6667	0
2	2	3	0	3	7	1	0	3	1	2.00	...	0.0	0	0.833333	0.666667	1	1	11.000000	1	0.5000	0
3	26	1	1	10	47	0	5	5	24	0.08	...	1.0	1	0.000000	0.258242	0	0	8.038462	2	1.0000	0
4	4	3	0	4	19	6	1	4	3	2.00	...	0.0	0	0.888889	0.375000	1	1	14.500000	1	0.5000	0

5 rows × 21 columns

- **Training the data**

```
x = data.drop(['bug'],axis=1)
y = data['bug']
```

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2,random_state=4)
```

- **LOGISTIC REGRESSION**

```
# LogisticRegression
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()

# train the logistic regression classifier using the training data
logreg.fit(x_train, y_train)

# make predictions on the testing data
y_pred = logreg.predict(x_test)

from sklearn.metrics import accuracy_score

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.9852941176470589
/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

- **SVM**

```
# SVM

from sklearn import svm
classifier = svm.SVC(kernel='linear',gamma='auto',C=2)
classifier.fit(x_train,y_train)
y_predict = classifier.predict(x_test)

from sklearn.metrics import accuracy_score

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_predict)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.9852941176470589
```

- **NAIVE BAYES**

```
# Naive Bayes
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()

# train the Naive Bayes classifier using the training data
gnb.fit(x_train, y_train)

# make predictions on the testing data
y_pred = gnb.predict(x_test)

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.8970588235294118
```

- **DECISION TREE**

```
# Decision Tree
from sklearn.tree import DecisionTreeClassifier

# create a Decision Tree classifier object
dtc = DecisionTreeClassifier()

# train the Decision Tree classifier using the training data
dtc.fit(x_train, y_train)

# make predictions on the testing data
y_pred = dtc.predict(x_test)

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.9117647058823529
```

- **RANDOM FOREST**

```
# Random Forest
from sklearn.ensemble import RandomForestClassifier


# create a Random Forest classifier object
rfc = RandomForestClassifier(n_estimators=100)

# train the Random Forest classifier using the training data
rfc.fit(x_train, y_train)

# make predictions on the testing data
y_pred = rfc.predict(x_test)

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.9852941176470589
```

- **XGBOOST**

```
# XGBoost

import xgboost as xgb

# convert data into DMatrix format
dtrain = xgb.DMatrix(x_train, label=y_train)
dtest = xgb.DMatrix(x_test, label=y_test)

# set the parameters for XGBoost
params = {
    'max_depth': 3,
    'eta': 0.1,
    'objective': 'multi:softmax',
    'num_class': 3
}

# train the XGBoost model
num_rounds = 100
xgb_model = xgb.train(params, dtrain, num_rounds)

# make predictions on the testing data
y_pred = xgb_model.predict(dtest)

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 1.0
```

- **KNN**

```
# KNN

from sklearn.neighbors import KNeighborsClassifier


# create a KNN classifier object
knn = KNeighborsClassifier(n_neighbors=5)

# train the KNN classifier using the training data
knn.fit(x_train, y_train)

# make predictions on the testing data
y_pred = knn.predict(x_test)

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 1.0
```

- **CATBOOST**

```
!pip install catboost
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting catboost
  Downloading catboost-1.1.1-cp39-none-manylinux1_x86_64.whl (76.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 76.6/76.6 MB 9.2 MB/s eta 0:00:00
Requirement already satisfied: graphviz in /usr/local/lib/python3.9/dist-packages (from catboost) (0.20.1)
Requirement already satisfied: six in /usr/local/lib/python3.9/dist-packages (from catboost) (1.16.0)
Requirement already satisfied: numpy>=1.16.0 in /usr/local/lib/python3.9/dist-packages (from catboost) (1.22.4)
Requirement already satisfied: scipy in /usr/local/lib/python3.9/dist-packages (from catboost) (1.10.1)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.9/dist-packages (from catboost) (3.7.1)
Requirement already satisfied: plotly in /usr/local/lib/python3.9/dist-packages (from catboost) (5.13.1)
Requirement already satisfied: pandas>=0.24.0 in /usr/local/lib/python3.9/dist-packages (from catboost) (1.5.3)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.9/dist-packages (from pandas>=0.24.0->catboost) (2022.7.1)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.9/dist-packages (from pandas>=0.24.0->catboost) (2.8.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (8.4.0)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (23.1)
Requirement already satisfied: importlib-resources>=3.2.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (5.12.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (1.0.7)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (1.4.4)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (4.39.3)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (3.0.9)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.9/dist-packages (from plotly->catboost) (8.2.2)
Requirement already satisfied: zipp>=3.1.0 in /usr/local/lib/python3.9/dist-packages (from importlib-resources>=3.2.0->matplotlib->catboost) (3.15.0)
Installing collected packages: catboost
Successfully installed catboost-1.1.1
```

```
# CatBoost
import catboost as ctb
# create a CatBoost classifier object
clf = ctb.CatBoostClassifier()

# train the CatBoost classifier using the training data
clf.fit(x_train, y_train)

# make predictions on the testing data
y_pred = clf.predict(x_test)

# calculate the accuracy of the model
accuracy = clf.score(x_test, y_test)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
978:    learn: 0.0063058        total: 3.11s    remaining: 66.7ms
979:    learn: 0.0062951        total: 3.11s    remaining: 63.6ms
980:    learn: 0.0062851        total: 3.12s    remaining: 60.4ms
981:    learn: 0.0062786        total: 3.12s    remaining: 57.3ms
982:    learn: 0.0062691        total: 3.13s    remaining: 54.1ms
983:    learn: 0.0062593        total: 3.13s    remaining: 50.9ms
984:    learn: 0.0062560        total: 3.13s    remaining: 47.7ms
985:    learn: 0.0062419        total: 3.13s    remaining: 44.5ms
986:    learn: 0.0062396        total: 3.14s    remaining: 41.3ms
987:    learn: 0.0062347        total: 3.14s    remaining: 38.1ms
988:    learn: 0.0062304        total: 3.14s    remaining: 35ms
989:    learn: 0.0062220        total: 3.15s    remaining: 31.8ms
990:    learn: 0.0062111        total: 3.15s    remaining: 28.6ms
991:    learn: 0.0062024        total: 3.15s    remaining: 25.4ms
992:    learn: 0.0061955        total: 3.15s    remaining: 22.2ms
993:    learn: 0.0061839        total: 3.16s    remaining: 19.1ms
994:    learn: 0.0061758        total: 3.16s    remaining: 15.9ms
995:    learn: 0.0061675        total: 3.16s    remaining: 12.7ms
996:    learn: 0.0061606        total: 3.16s    remaining: 9.52ms
997:    learn: 0.0061543        total: 3.17s    remaining: 6.35ms
998:    learn: 0.0061479        total: 3.17s    remaining: 3.17ms
999:    learn: 0.0061437        total: 3.17s    remaining: 0us
Accuracy: 1.0
```

- **LSTM**

```
# LSTM

from keras.models import Sequential
from keras.layers import LSTM, Dense

# reshape the input data to be 3D for LSTM
x_train = x_train.values.reshape((x_train.shape[0], 1, x_train.shape[1]))
x_test = x_test.values.reshape((x_test.shape[0], 1, x_test.shape[1]))

# define the LSTM model
model = Sequential()
model.add(LSTM(50, input_shape=(x_train.shape[1], x_train.shape[2])))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# train the LSTM model
history = model.fit(x_train, y_train, epochs=50, batch_size=32, validation_data=(x_test, y_test), verbose=2, shuffle=False)

# evaluate the LSTM model
loss, accuracy = model.evaluate(x_test, y_test, verbose=0)
print('Accuracy:', accuracy)
```

```
Epoch 40/50
9/9 - 0s - loss: 0.1176 - accuracy: 0.9631 - val_loss: 0.0392 - val_accuracy: 1.0000 - 59ms/epoch - 7ms/step
Epoch 41/50
9/9 - 0s - loss: 0.1169 - accuracy: 0.9631 - val_loss: 0.0390 - val_accuracy: 1.0000 - 76ms/epoch - 8ms/step
Epoch 42/50
9/9 - 0s - loss: 0.1163 - accuracy: 0.9631 - val_loss: 0.0388 - val_accuracy: 1.0000 - 61ms/epoch - 7ms/step
Epoch 43/50
9/9 - 0s - loss: 0.1158 - accuracy: 0.9631 - val_loss: 0.0387 - val_accuracy: 1.0000 - 63ms/epoch - 7ms/step
Epoch 44/50
9/9 - 0s - loss: 0.1152 - accuracy: 0.9631 - val_loss: 0.0385 - val_accuracy: 1.0000 - 63ms/epoch - 7ms/step
Epoch 45/50
9/9 - 0s - loss: 0.1147 - accuracy: 0.9631 - val_loss: 0.0383 - val_accuracy: 1.0000 - 61ms/epoch - 7ms/step
Epoch 46/50
9/9 - 0s - loss: 0.1141 - accuracy: 0.9631 - val_loss: 0.0382 - val_accuracy: 1.0000 - 73ms/epoch - 8ms/step
Epoch 47/50
9/9 - 0s - loss: 0.1136 - accuracy: 0.9631 - val_loss: 0.0380 - val_accuracy: 1.0000 - 74ms/epoch - 8ms/step
Epoch 48/50
9/9 - 0s - loss: 0.1131 - accuracy: 0.9631 - val_loss: 0.0378 - val_accuracy: 1.0000 - 63ms/epoch - 7ms/step
Epoch 49/50
9/9 - 0s - loss: 0.1126 - accuracy: 0.9631 - val_loss: 0.0377 - val_accuracy: 1.0000 - 69ms/epoch - 8ms/step
Epoch 50/50
9/9 - 0s - loss: 0.1120 - accuracy: 0.9631 - val_loss: 0.0375 - val_accuracy: 1.0000 - 58ms/epoch - 6ms/step
Accuracy: 1.0
```

- **ANN**

```
# ANN

# import pandas as pd
from keras.models import Sequential
from keras.layers import Dense

# load the dataset
# df = pd.read_csv('dataset.csv')

# split the dataset into input and output variables
# X = df.drop(columns=['output'])
# y = df['output']

# define the model
model = Sequential()
model.add(Dense(10, input_dim=x.shape[1], activation='relu'))
model.add(Dense(1, activation='linear'))

# compile the model
model.compile(loss='mse', optimizer='adam', metrics=['mae'])

# train the model
model.fit(x, y, epochs=100, batch_size=32)

# make predictions
y_pred = model.predict(x)

# evaluate the model
mse, mae = model.evaluate(x, y)

print('MSE: ', mse)
print('MAE: ', mae)
```

```
Epoch 88/100
11/11 [==============================] - 0s 3ms/step - loss: 2.0234 - mae: 0.8698
Epoch 89/100
11/11 [==============================] - 0s 4ms/step - loss: 2.0673 - mae: 0.8726
Epoch 90/100
11/11 [==============================] - 0s 7ms/step - loss: 1.9286 - mae: 0.8579
Epoch 91/100
11/11 [==============================] - 0s 3ms/step - loss: 1.9268 - mae: 0.8584
Epoch 92/100
11/11 [==============================] - 0s 3ms/step - loss: 1.9985 - mae: 0.8517
Epoch 93/100
11/11 [==============================] - 0s 3ms/step - loss: 1.8860 - mae: 0.8351
Epoch 94/100
11/11 [==============================] - 0s 3ms/step - loss: 1.9765 - mae: 0.8648
Epoch 95/100
11/11 [==============================] - 0s 3ms/step - loss: 1.8414 - mae: 0.8263
Epoch 96/100
11/11 [==============================] - 0s 3ms/step - loss: 1.8011 - mae: 0.8200
Epoch 97/100
11/11 [==============================] - 0s 4ms/step - loss: 1.7900 - mae: 0.8191
Epoch 98/100
11/11 [==============================] - 0s 3ms/step - loss: 1.7474 - mae: 0.8110
Epoch 99/100
11/11 [==============================] - 0s 3ms/step - loss: 1.7717 - mae: 0.8182
Epoch 100/100
11/11 [==============================] - 0s 6ms/step - loss: 1.9378 - mae: 0.8314
11/11 [==============================] - 0s 5ms/step
11/11 [==============================] - 0s 3ms/step - loss: 1.6989 - mae: 0.8100
MSE:  1.6989432573318481
MAE:  0.81003338098526
```

4. Dataset : camel-1.2

- **Importing the Libraries**

```
import pandas as pd
import numpy as np
import seaborn as sns
```

- **Loading the dataset**

```
data = pd.read_csv('/content/camel-1.2.csv')
# data['defects'] = data['defects'].replace({True:1,False:0})
data.head()
```

	wmc	dit	noc	cbo	rfc	lcom	ca	ce	npm	lcom3	...	dam	moa	mfa	cam	ic	cbm	amc	max_cc	avg_cc	bug
0	9	0	0	5	24	0	1	4	9	0.000000	...	1.0	1	0.000000	0.370370	0	0	22.555556	1	0.8889	0
1	27	1	0	6	27	351	1	5	27	2.000000	...	0.0	0	0.000000	0.412037	0	0	0.000000	1	1.0000	1
2	9	1	0	3	18	30	1	2	8	0.375000	...	1.0	0	0.000000	0.355556	0	0	8.000000	1	0.7778	0
3	7	1	0	27	50	7	3	27	3	0.805556	...	1.0	3	0.000000	0.416667	0	0	26.428571	1	0.7143	0
4	8	2	0	10	28	0	1	9	3	0.357143	...	1.0	3	0.684211	0.371429	1	1	26.500000	2	1.0000	1

5 rows × 21 columns

- **Training the data**

```
x = data.drop(['bug'],axis=1)
y = data['bug']
```

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2,random_state=4)
```

- **LOGISTIC REGRESSION**

```
# LogisticRegression
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()

# train the logistic regression classifier using the training data
logreg.fit(x_train, y_train)

# make predictions on the testing data
y_pred = logreg.predict(x_test)

from sklearn.metrics import accuracy_score

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.6311475409836066
/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

- **SVM**

```
# SVM

from sklearn import svm
classifier = svm.SVC(kernel='linear',gamma='auto',C=2)
classifier.fit(x_train,y_train)
y_predict = classifier.predict(x_test)

from sklearn.metrics import accuracy_score

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_predict)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.6475409836065574
```

- **NAIVE BAYES**

```
# Naive Bayes
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()

# train the Naive Bayes classifier using the training data
gnb.fit(x_train, y_train)

# make predictions on the testing data
y_pred = gnb.predict(x_test)

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.6147540983606558
```

- **DECISION TREE**

```
# Decision Tree
from sklearn.tree import DecisionTreeClassifier

# create a Decision Tree classifier object
dtc = DecisionTreeClassifier()

# train the Decision Tree classifier using the training data
dtc.fit(x_train, y_train)

# make predictions on the testing data
y_pred = dtc.predict(x_test)

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.6065573770491803
```

- **RANDOM FOREST**

```
# Random Forest
from sklearn.ensemble import RandomForestClassifier


# create a Random Forest classifier object
rfc = RandomForestClassifier(n_estimators=100)

# train the Random Forest classifier using the training data
rfc.fit(x_train, y_train)

# make predictions on the testing data
y_pred = rfc.predict(x_test)

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.680327868852459
```

- **XGBOOST**

```
# XGBoost

import xgboost as xgb

# convert data into DMatrix format
dtrain = xgb.DMatrix(x_train, label=y_train)
dtest = xgb.DMatrix(x_test, label=y_test)

# set the parameters for XGBoost
params = {
    'max_depth': 3,
    'eta': 0.1,
    'objective': 'multi:softmax',
    'num_class': 3
}

# train the XGBoost model
num_rounds = 100
xgb_model = xgb.train(params, dtrain, num_rounds)

# make predictions on the testing data
y_pred = xgb_model.predict(dtest)

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.6557377049180327
```

- **KNN**

```
# KNN

from sklearn.neighbors import KNeighborsClassifier


# create a KNN classifier object
knn = KNeighborsClassifier(n_neighbors=5)

# train the KNN classifier using the training data
knn.fit(x_train, y_train)

# make predictions on the testing data
y_pred = knn.predict(x_test)

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.5737704918032787
```

- **CATBOOST**

```
!pip install catboost
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting catboost
  Downloading catboost-1.1.1-cp39-none-manylinux1_x86_64.whl (76.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 76.6/76.6 MB 9.2 MB/s eta 0:00:00
Requirement already satisfied: graphviz in /usr/local/lib/python3.9/dist-packages (from catboost) (0.20.1)
Requirement already satisfied: six in /usr/local/lib/python3.9/dist-packages (from catboost) (1.16.0)
Requirement already satisfied: numpy>=1.16.0 in /usr/local/lib/python3.9/dist-packages (from catboost) (1.22.4)
Requirement already satisfied: scipy in /usr/local/lib/python3.9/dist-packages (from catboost) (1.10.1)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.9/dist-packages (from catboost) (3.7.1)
Requirement already satisfied: plotly in /usr/local/lib/python3.9/dist-packages (from catboost) (5.13.1)
Requirement already satisfied: pandas>=0.24.0 in /usr/local/lib/python3.9/dist-packages (from catboost) (1.5.3)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.9/dist-packages (from pandas>=0.24.0->catboost) (2022.7.1)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.9/dist-packages (from pandas>=0.24.0->catboost) (2.8.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (8.4.0)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (23.1)
Requirement already satisfied: importlib-resources>=3.2.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (5.12.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (1.0.7)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (1.4.4)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (4.39.3)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (3.0.9)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.9/dist-packages (from plotly->catboost) (8.2.2)
Requirement already satisfied: zipp>=3.1.0 in /usr/local/lib/python3.9/dist-packages (from importlib-resources>=3.2.0->matplotlib->catboost) (3.15.0)
Installing collected packages: catboost
Successfully installed catboost-1.1.1
```

```
# CatBoost
import catboost as ctb
# create a CatBoost classifier object
clf = ctb.CatBoostClassifier()

# train the CatBoost classifier using the training data
clf.fit(x_train, y_train)

# make predictions on the testing data
y_pred = clf.predict(x_test)

# calculate the accuracy of the model
accuracy = clf.score(x_test, y_test)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
978:    learn: 0.3117265        total: 3.74s    remaining: 80.2ms
979:    learn: 0.3116225        total: 3.74s    remaining: 76.4ms
980:    learn: 0.3113753        total: 3.75s    remaining: 72.5ms
981:    learn: 0.3111178        total: 3.75s    remaining: 68.7ms
982:    learn: 0.3109724        total: 3.75s    remaining: 64.8ms
983:    learn: 0.3108025        total: 3.75s    remaining: 61ms
984:    learn: 0.3107120        total: 3.75s    remaining: 57.2ms
985:    learn: 0.3104885        total: 3.76s    remaining: 53.3ms
986:    learn: 0.3098953        total: 3.76s    remaining: 49.5ms
987:    learn: 0.3096844        total: 3.76s    remaining: 45.7ms
988:    learn: 0.3095579        total: 3.76s    remaining: 41.9ms
989:    learn: 0.3095126        total: 3.77s    remaining: 38ms
990:    learn: 0.3091048        total: 3.77s    remaining: 34.2ms
991:    learn: 0.3088817        total: 3.77s    remaining: 30.4ms
992:    learn: 0.3084977        total: 3.77s    remaining: 26.6ms
993:    learn: 0.3082158        total: 3.77s    remaining: 22.8ms
994:    learn: 0.3080070        total: 3.78s    remaining: 19ms
995:    learn: 0.3077431        total: 3.78s    remaining: 15.2ms
996:    learn: 0.3075230        total: 3.78s    remaining: 11.4ms
997:    learn: 0.3072534        total: 3.78s    remaining: 7.58ms
998:    learn: 0.3071308        total: 3.79s    remaining: 3.79ms
999:    learn: 0.3069827        total: 3.79s    remaining: 0us
Accuracy: 0.6885245901639344
```

- **LSTM**

```
# LSTM

from keras.models import Sequential
from keras.layers import LSTM, Dense

# reshape the input data to be 3D for LSTM
x_train = x_train.values.reshape((x_train.shape[0], 1, x_train.shape[1]))
x_test = x_test.values.reshape((x_test.shape[0], 1, x_test.shape[1]))

# define the LSTM model
model = Sequential()
model.add(LSTM(50, input_shape=(x_train.shape[1], x_train.shape[2])))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# train the LSTM model
history = model.fit(x_train, y_train, epochs=50, batch_size=32, validation_data=(x_test, y_test), verbose=2, shuffle=False)

# evaluate the LSTM model
loss, accuracy = model.evaluate(x_test, y_test, verbose=0)
print('Accuracy:', accuracy)
```

```
Epoch 39/50
16/16 - 0s - loss: 0.5145 - accuracy: 0.7654 - val_loss: 0.7085 - val_accuracy: 0.6066 - 82ms/epoch - 5ms/step
Epoch 40/50
16/16 - 0s - loss: 0.5128 - accuracy: 0.7634 - val_loss: 0.7094 - val_accuracy: 0.5984 - 96ms/epoch - 6ms/step
Epoch 41/50
16/16 - 0s - loss: 0.5101 - accuracy: 0.7613 - val_loss: 0.7089 - val_accuracy: 0.5902 - 81ms/epoch - 5ms/step
Epoch 42/50
16/16 - 0s - loss: 0.5090 - accuracy: 0.7716 - val_loss: 0.7046 - val_accuracy: 0.5984 - 99ms/epoch - 6ms/step
Epoch 43/50
16/16 - 0s - loss: 0.5053 - accuracy: 0.7737 - val_loss: 0.7050 - val_accuracy: 0.5902 - 77ms/epoch - 5ms/step
Epoch 44/50
16/16 - 0s - loss: 0.5033 - accuracy: 0.7778 - val_loss: 0.7037 - val_accuracy: 0.5902 - 79ms/epoch - 5ms/step
Epoch 45/50
16/16 - 0s - loss: 0.5003 - accuracy: 0.7778 - val_loss: 0.7088 - val_accuracy: 0.5902 - 79ms/epoch - 5ms/step
Epoch 46/50
16/16 - 0s - loss: 0.4991 - accuracy: 0.7798 - val_loss: 0.7070 - val_accuracy: 0.5902 - 83ms/epoch - 5ms/step
Epoch 47/50
16/16 - 0s - loss: 0.4958 - accuracy: 0.7819 - val_loss: 0.7081 - val_accuracy: 0.5902 - 148ms/epoch - 9ms/step
Epoch 48/50
16/16 - 0s - loss: 0.4930 - accuracy: 0.7840 - val_loss: 0.7082 - val_accuracy: 0.5902 - 250ms/epoch - 16ms/step
Epoch 49/50
16/16 - 0s - loss: 0.4909 - accuracy: 0.7798 - val_loss: 0.7070 - val_accuracy: 0.5902 - 226ms/epoch - 14ms/step
Epoch 50/50
16/16 - 0s - loss: 0.4886 - accuracy: 0.7840 - val_loss: 0.7118 - val_accuracy: 0.5902 - 207ms/epoch - 13ms/step
Accuracy: 0.5901639461517334
```

- **ANN**

```python
# ANN

# import pandas as pd
from keras.models import Sequential
from keras.layers import Dense

# load the dataset
# df = pd.read_csv('dataset.csv')

# split the dataset into input and output variables
# X = df.drop(columns=['output'])
# y = df['output']

# define the model
model = Sequential()
model.add(Dense(10, input_dim=x.shape[1], activation='relu'))
model.add(Dense(1, activation='linear'))

# compile the model
model.compile(loss='mse', optimizer='adam', metrics=['mae'])

# train the model
model.fit(x, y, epochs=100, batch_size=32)

# make predictions
y_pred = model.predict(x)

# evaluate the model
mse, mae = model.evaluate(x, y)

print('MSE: ', mse)
print('MAE: ', mae)
```

```
Epoch 89/100
19/19 [==============================] - 0s 4ms/step - loss: 5.4393 - mae: 0.9651
Epoch 90/100
19/19 [==============================] - 0s 8ms/step - loss: 5.2639 - mae: 0.9547
Epoch 91/100
19/19 [==============================] - 0s 6ms/step - loss: 4.9278 - mae: 0.9408
Epoch 92/100
19/19 [==============================] - 0s 8ms/step - loss: 4.6762 - mae: 0.9293
Epoch 93/100
19/19 [==============================] - 0s 9ms/step - loss: 4.4621 - mae: 0.9169
Epoch 94/100
19/19 [==============================] - 0s 16ms/step - loss: 4.2080 - mae: 0.9032
Epoch 95/100
19/19 [==============================] - 0s 7ms/step - loss: 4.0683 - mae: 0.8942
Epoch 96/100
19/19 [==============================] - 0s 5ms/step - loss: 3.7995 - mae: 0.8841
Epoch 97/100
19/19 [==============================] - 0s 5ms/step - loss: 3.6022 - mae: 0.8732
Epoch 98/100
19/19 [==============================] - 0s 9ms/step - loss: 3.5464 - mae: 0.8643
Epoch 99/100
19/19 [==============================] - 0s 6ms/step - loss: 3.2435 - mae: 0.8519
Epoch 100/100
19/19 [==============================] - 0s 7ms/step - loss: 3.1147 - mae: 0.8455
19/19 [==============================] - 0s 4ms/step
19/19 [==============================] - 0s 5ms/step - loss: 2.9617 - mae: 0.8333
MSE:  2.9616928100585938
MAE:  0.8332631587982178
```

5. Dataset : ivy-1.1

- **Importing the Libraries**

```
import pandas as pd
import numpy as np
import seaborn as sns
```

- **Loading the dataset**

```
data = pd.read_csv('/content/ivy-1.1.csv')
# data['defects'] = data['defects'].replace({True:1,False:0})
data.head()
```

	wmc	dit	noc	cbo	rfc	lcom	ca	ce	npm	lcom3	...	dam	moa	mfa	cam	ic	cbm	amc	max_cc	avg_cc	bug
0	2	1	0	8	2	1	7	1	2	2.000000	...	0.000	0	0.000000	0.666667	0	0	0.000000	1	1.0000	0
1	23	1	0	7	52	171	2	5	20	0.837662	...	1.000	2	0.000000	0.250000	0	0	16.956522	4	1.4348	1
2	4	1	0	10	4	6	9	1	4	2.000000	...	0.000	0	0.000000	0.550000	0	0	0.000000	1	1.0000	1
3	13	4	2	17	57	22	2	15	9	0.250000	...	1.000	1	0.850000	0.192308	3	7	25.538462	6	1.6154	1
4	38	2	0	37	153	569	1	37	21	0.895270	...	0.875	1	0.307692	0.122333	1	1	38.657895	4	1.5000	1

5 rows × 21 columns

- **Training the data**

```
x = data.drop(['bug'],axis=1)
y = data['bug']
```

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2,random_state=4)
```

- **LOGISTIC REGRESSION**

```
# LogisticRegression
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()

# train the logistic regression classifier using the training data
logreg.fit(x_train, y_train)

# make predictions on the testing data
y_pred = logreg.predict(x_test)

from sklearn.metrics import accuracy_score

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.5652173913043478
/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

- **SVM**

```
# SVM

from sklearn import svm
classifier = svm.SVC(kernel='linear',gamma='auto',C=2)
classifier.fit(x_train,y_train)
y_predict = classifier.predict(x_test)

from sklearn.metrics import accuracy_score

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_predict)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.5652173913043478
```

- **NAIVE BAYES**

```
# Naive Bayes
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()

# train the Naive Bayes classifier using the training data
gnb.fit(x_train, y_train)

# make predictions on the testing data
y_pred = gnb.predict(x_test)

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.5652173913043478
```

- **DECISION TREE**

```
# Decision Tree
from sklearn.tree import DecisionTreeClassifier

# create a Decision Tree classifier object
dtc = DecisionTreeClassifier()

# train the Decision Tree classifier using the training data
dtc.fit(x_train, y_train)

# make predictions on the testing data
y_pred = dtc.predict(x_test)

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.5652173913043478
```

- **RANDOM FOREST**

```
# Random Forest
from sklearn.ensemble import RandomForestClassifier


# create a Random Forest classifier object
rfc = RandomForestClassifier(n_estimators=100)

# train the Random Forest classifier using the training data
rfc.fit(x_train, y_train)

# make predictions on the testing data
y_pred = rfc.predict(x_test)

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.6521739130434783
```

- **XGBOOST**

```
# XGBoost

import xgboost as xgb

# convert data into DMatrix format
dtrain = xgb.DMatrix(x_train, label=y_train)
dtest = xgb.DMatrix(x_test, label=y_test)

# set the parameters for XGBoost
params = {
    'max_depth': 3,
    'eta': 0.1,
    'objective': 'multi:softmax',
    'num_class': 3
}

# train the XGBoost model
num_rounds = 100
xgb_model = xgb.train(params, dtrain, num_rounds)

# make predictions on the testing data
y_pred = xgb_model.predict(dtest)

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.6956521739130435
```

- **KNN**

```
# KNN

from sklearn.neighbors import KNeighborsClassifier


# create a KNN classifier object
knn = KNeighborsClassifier(n_neighbors=5)

# train the KNN classifier using the training data
knn.fit(x_train, y_train)

# make predictions on the testing data
y_pred = knn.predict(x_test)

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.6521739130434783
```

- **CATBOOST**

```
!pip install catboost
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting catboost
  Downloading catboost-1.1.1-cp39-none-manylinux1_x86_64.whl (76.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 76.6/76.6 MB 9.2 MB/s eta 0:00:00
Requirement already satisfied: graphviz in /usr/local/lib/python3.9/dist-packages (from catboost) (0.20.1)
Requirement already satisfied: six in /usr/local/lib/python3.9/dist-packages (from catboost) (1.16.0)
Requirement already satisfied: numpy>=1.16.0 in /usr/local/lib/python3.9/dist-packages (from catboost) (1.22.4)
Requirement already satisfied: scipy in /usr/local/lib/python3.9/dist-packages (from catboost) (1.10.1)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.9/dist-packages (from catboost) (3.7.1)
Requirement already satisfied: plotly in /usr/local/lib/python3.9/dist-packages (from catboost) (5.13.1)
Requirement already satisfied: pandas>=0.24.0 in /usr/local/lib/python3.9/dist-packages (from catboost) (1.5.3)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.9/dist-packages (from pandas>=0.24.0->catboost) (2022.7.1)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.9/dist-packages (from pandas>=0.24.0->catboost) (2.8.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (8.4.0)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (23.1)
Requirement already satisfied: importlib-resources>=3.2.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (5.12.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (1.0.7)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (1.4.4)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (4.39.3)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (3.0.9)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.9/dist-packages (from plotly->catboost) (8.2.2)
Requirement already satisfied: zipp>=3.1.0 in /usr/local/lib/python3.9/dist-packages (from importlib-resources>=3.2.0->matplotlib->catboost) (3.15.0)
Installing collected packages: catboost
Successfully installed catboost-1.1.1
```

```
# CatBoost
import catboost as ctb
# create a CatBoost classifier object
clf = ctb.CatBoostClassifier()

# train the CatBoost classifier using the training data
clf.fit(x_train, y_train)

# make predictions on the testing data
y_pred = clf.predict(x_test)

# calculate the accuracy of the model
accuracy = clf.score(x_test, y_test)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
977:    learn: 0.2212885        total: 2.47s    remaining: 55.7ms
978:    learn: 0.2211503        total: 2.48s    remaining: 53.1ms
979:    learn: 0.2208332        total: 2.48s    remaining: 50.5ms
980:    learn: 0.2206468        total: 2.48s    remaining: 48ms
981:    learn: 0.2204584        total: 2.48s    remaining: 45.4ms
982:    learn: 0.2203434        total: 2.48s    remaining: 42.9ms
983:    learn: 0.2201678        total: 2.48s    remaining: 40.4ms
984:    learn: 0.2200348        total: 2.48s    remaining: 37.8ms
985:    learn: 0.2198320        total: 2.48s    remaining: 35.3ms
986:    learn: 0.2197060        total: 2.48s    remaining: 32.7ms
987:    learn: 0.2196289        total: 2.49s    remaining: 30.2ms
988:    learn: 0.2195495        total: 2.49s    remaining: 27.7ms
989:    learn: 0.2193674        total: 2.49s    remaining: 25.1ms
990:    learn: 0.2192524        total: 2.49s    remaining: 22.6ms
991:    learn: 0.2190815        total: 2.49s    remaining: 20.1ms
992:    learn: 0.2187841        total: 2.49s    remaining: 17.6ms
993:    learn: 0.2187106        total: 2.5s     remaining: 15.1ms
994:    learn: 0.2185573        total: 2.5s     remaining: 12.5ms
995:    learn: 0.2184410        total: 2.5s     remaining: 10ms
996:    learn: 0.2182485        total: 2.5s     remaining: 7.52ms
997:    learn: 0.2181357        total: 2.5s     remaining: 5.01ms
998:    learn: 0.2179431        total: 2.5s     remaining: 2.5ms
999:    learn: 0.2178021        total: 2.5s     remaining: 0us
Accuracy: 0.6086956521739131
```

- **LSTM**

```python
from keras.models import Sequential
from keras.layers import LSTM, Dense

# reshape the input data to be 3D for LSTM
x_train = x_train.values.reshape((x_train.shape[0], 1, x_train.shape[1]))
x_test = x_test.values.reshape((x_test.shape[0], 1, x_test.shape[1]))

# define the LSTM model
model = Sequential()
model.add(LSTM(50, input_shape=(x_train.shape[1], x_train.shape[2])))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# train the LSTM model
history = model.fit(x_train, y_train, epochs=50, batch_size=32, validation_data=(x_test, y_test), verbose=2, shuffle=False)

# evaluate the LSTM model
loss, accuracy = model.evaluate(x_test, y_test, verbose=0)
print('Accuracy:', accuracy)
```

```
Epoch 44/50
3/3 - 0s - loss: 0.4346 - accuracy: 0.7955 - val_loss: 0.6363 - val_accuracy: 0.7391 - 41ms/epoch - 14ms/step
Epoch 45/50
3/3 - 0s - loss: 0.4318 - accuracy: 0.7955 - val_loss: 0.6378 - val_accuracy: 0.7391 - 37ms/epoch - 12ms/step
Epoch 46/50
3/3 - 0s - loss: 0.4288 - accuracy: 0.7955 - val_loss: 0.6402 - val_accuracy: 0.7391 - 40ms/epoch - 13ms/step
Epoch 47/50
3/3 - 0s - loss: 0.4256 - accuracy: 0.7955 - val_loss: 0.6422 - val_accuracy: 0.7391 - 73ms/epoch - 24ms/step
Epoch 48/50
3/3 - 0s - loss: 0.4226 - accuracy: 0.8182 - val_loss: 0.6432 - val_accuracy: 0.7391 - 58ms/epoch - 19ms/step
Epoch 49/50
3/3 - 0s - loss: 0.4197 - accuracy: 0.8182 - val_loss: 0.6444 - val_accuracy: 0.7391 - 41ms/epoch - 14ms/step
Epoch 50/50
3/3 - 0s - loss: 0.4168 - accuracy: 0.8182 - val_loss: 0.6460 - val_accuracy: 0.7391 - 40ms/epoch - 13ms/step
Accuracy: 0.739130437374115
```

- **ANN**

```
# ANN

# import pandas as pd
from keras.models import Sequential
from keras.layers import Dense

# load the dataset
# df = pd.read_csv('dataset.csv')

# split the dataset into input and output variables
# X = df.drop(columns=['output'])
# y = df['output']

# define the model
model = Sequential()
model.add(Dense(10, input_dim=x.shape[1], activation='relu'))
model.add(Dense(1, activation='linear'))

# compile the model
model.compile(loss='mse', optimizer='adam', metrics=['mae'])

# train the model
model.fit(x, y, epochs=100, batch_size=32)

# make predictions
y_pred = model.predict(x)

# evaluate the model
mse, mae = model.evaluate(x, y)

print('MSE: ', mse)
print('MAE: ', mae)
```

```
Epoch 88/100
4/4 [==============================] - 0s 3ms/step - loss: 6.3894 - mae: 1.8632
Epoch 89/100
4/4 [==============================] - 0s 3ms/step - loss: 6.2963 - mae: 1.8408
Epoch 90/100
4/4 [==============================] - 0s 3ms/step - loss: 6.2330 - mae: 1.8328
Epoch 91/100
4/4 [==============================] - 0s 4ms/step - loss: 6.2284 - mae: 1.8293
Epoch 92/100
4/4 [==============================] - 0s 3ms/step - loss: 6.0897 - mae: 1.8089
Epoch 93/100
4/4 [==============================] - 0s 4ms/step - loss: 5.9997 - mae: 1.7882
Epoch 94/100
4/4 [==============================] - 0s 4ms/step - loss: 5.9395 - mae: 1.7818
Epoch 95/100
4/4 [==============================] - 0s 4ms/step - loss: 5.8212 - mae: 1.7661
Epoch 96/100
4/4 [==============================] - 0s 4ms/step - loss: 5.7837 - mae: 1.7617
Epoch 97/100
4/4 [==============================] - 0s 4ms/step - loss: 5.8401 - mae: 1.7756
Epoch 98/100
4/4 [==============================] - 0s 4ms/step - loss: 5.7412 - mae: 1.7656
Epoch 99/100
4/4 [==============================] - 0s 4ms/step - loss: 5.8928 - mae: 1.7721
Epoch 100/100
4/4 [==============================] - 0s 4ms/step - loss: 5.5856 - mae: 1.7280
4/4 [==============================] - 0s 2ms/step
4/4 [==============================] - 0s 3ms/step - loss: 5.4846 - mae: 1.7220
MSE:  5.484640598297119
MAE:  1.7219570875167847
```

6. Dataset : ivy-2.0

- **Importing the Libraries**

```
import pandas as pd
import numpy as np
import seaborn as sns
```

- **Loading the dataset**

```
data = pd.read_csv('/content/ivy-2.0.csv')
# data['defects'] = data['defects'].replace({True:1,False:0})
data.head()
```

	wmc	dit	noc	cbo	rfc	lcom	ca	ce	npm	lcom3	...	dam	moa	mfa	cam	ic	cbm	amc	max_cc	avg_cc	bug
0	28	1	0	32	82	374	26	7	24	1.012346	...	0.166667	0	0.0	0.226337	0	0	31.642857	23	2.6786	1
1	6	1	2	3	7	3	2	1	4	0.600000	...	1.000000	0	0.0	0.444444	0	0	4.666667	1	0.6667	0
2	4	2	0	5	6	4	1	4	2	0.666667	...	1.000000	1	0.5	0.500000	0	0	4.000000	1	0.5000	0
3	4	1	0	9	4	6	9	0	4	2.000000	...	0.000000	0	0.0	0.666667	0	0	0.000000	1	1.0000	0
4	1	1	0	8	1	0	6	2	1	2.000000	...	0.000000	0	0.0	1.000000	0	0	0.000000	1	1.0000	0

5 rows × 21 columns

- **Training the data**

```
x = data.drop(['bug'],axis=1)
y = data['bug']
```

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2,random_state=4)
```

- **LOGISTIC REGRESSION**

```
# LogisticRegression
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()

# train the logistic regression classifier using the training data
logreg.fit(x_train, y_train)

# make predictions on the testing data
y_pred = logreg.predict(x_test)

from sklearn.metrics import accuracy_score

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.8873239436619719
/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

- **SVM**

```
from sklearn import svm
classifier = svm.SVC(kernel='linear',gamma='auto',C=2)
classifier.fit(x_train,y_train)
y_predict = classifier.predict(x_test)

from sklearn.metrics import accuracy_score

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_predict)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.8873239436619719
```

- **NAIVE BAYES**

```
# Naive Bayes
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()

# train the Naive Bayes classifier using the training data
gnb.fit(x_train, y_train)

# make predictions on the testing data
y_pred = gnb.predict(x_test)

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.8169014084507042
```

- **DECISION TREE**

```
# Decision Tree
from sklearn.tree import DecisionTreeClassifier

# create a Decision Tree classifier object
dtc = DecisionTreeClassifier()

# train the Decision Tree classifier using the training data
dtc.fit(x_train, y_train)

# make predictions on the testing data
y_pred = dtc.predict(x_test)

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.8309859154929577
```

- **RANDOM FOREST**

```
# Random Forest
from sklearn.ensemble import RandomForestClassifier


# create a Random Forest classifier object
rfc = RandomForestClassifier(n_estimators=100)

# train the Random Forest classifier using the training data
rfc.fit(x_train, y_train)

# make predictions on the testing data
y_pred = rfc.predict(x_test)

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.9014084507042254
```

- **XGBOOST**

```
# XGBoost

import xgboost as xgb

# convert data into DMatrix format
dtrain = xgb.DMatrix(x_train, label=y_train)
dtest = xgb.DMatrix(x_test, label=y_test)

# set the parameters for XGBoost
params = {
    'max_depth': 3,
    'eta': 0.1,
    'objective': 'multi:softmax',
    'num_class': 3
}

# train the XGBoost model
num_rounds = 100
xgb_model = xgb.train(params, dtrain, num_rounds)

# make predictions on the testing data
y_pred = xgb_model.predict(dtest)

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.8873239436619719
```

- **KNN**

```
# KNN

from sklearn.neighbors import KNeighborsClassifier


# create a KNN classifier object
knn = KNeighborsClassifier(n_neighbors=5)

# train the KNN classifier using the training data
knn.fit(x_train, y_train)

# make predictions on the testing data
y_pred = knn.predict(x_test)

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.9154929577464789
```

- **CATBOOST**

```
!pip install catboost
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting catboost
  Downloading catboost-1.1.1-cp39-none-manylinux1_x86_64.whl (76.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 76.6/76.6 MB 9.2 MB/s eta 0:00:00
Requirement already satisfied: graphviz in /usr/local/lib/python3.9/dist-packages (from catboost) (0.20.1)
Requirement already satisfied: six in /usr/local/lib/python3.9/dist-packages (from catboost) (1.16.0)
Requirement already satisfied: numpy>=1.16.0 in /usr/local/lib/python3.9/dist-packages (from catboost) (1.22.4)
Requirement already satisfied: scipy in /usr/local/lib/python3.9/dist-packages (from catboost) (1.10.1)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.9/dist-packages (from catboost) (3.7.1)
Requirement already satisfied: plotly in /usr/local/lib/python3.9/dist-packages (from catboost) (5.13.1)
Requirement already satisfied: pandas>=0.24.0 in /usr/local/lib/python3.9/dist-packages (from catboost) (1.5.3)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.9/dist-packages (from pandas>=0.24.0->catboost) (2022.7.1)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.9/dist-packages (from pandas>=0.24.0->catboost) (2.8.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (8.4.0)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (23.1)
Requirement already satisfied: importlib-resources>=3.2.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (5.12.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (1.0.7)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (1.4.4)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (4.39.3)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (3.0.9)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.9/dist-packages (from plotly->catboost) (8.2.2)
Requirement already satisfied: zipp>=3.1.0 in /usr/local/lib/python3.9/dist-packages (from importlib-resources>=3.2.0->matplotlib->catboost) (3.15.0)
Installing collected packages: catboost
Successfully installed catboost-1.1.1
```

```
# CatBoost
import catboost as ctb
# create a CatBoost classifier object
clf = ctb.CatBoostClassifier()

# train the CatBoost classifier using the training data
clf.fit(x_train, y_train)

# make predictions on the testing data
y_pred = clf.predict(x_test)

# calculate the accuracy of the model
accuracy = clf.score(x_test, y_test)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
975:    learn: 0.0825992        total: 2.47s    remaining: 60.8ms
976:    learn: 0.0824652        total: 2.47s    remaining: 58.2ms
977:    learn: 0.0822855        total: 2.48s    remaining: 55.7ms
978:    learn: 0.0822268        total: 2.48s    remaining: 53.2ms
979:    learn: 0.0821244        total: 2.48s    remaining: 50.6ms
980:    learn: 0.0820381        total: 2.48s    remaining: 48.1ms
981:    learn: 0.0819265        total: 2.48s    remaining: 45.5ms
982:    learn: 0.0817582        total: 2.49s    remaining: 43ms
983:    learn: 0.0816760        total: 2.49s    remaining: 40.5ms
984:    learn: 0.0815785        total: 2.49s    remaining: 37.9ms
985:    learn: 0.0814876        total: 2.49s    remaining: 35.4ms
986:    learn: 0.0813180        total: 2.49s    remaining: 32.9ms
987:    learn: 0.0812469        total: 2.5s     remaining: 30.3ms
988:    learn: 0.0812190        total: 2.5s     remaining: 27.8ms
989:    learn: 0.0812071        total: 2.5s     remaining: 25.2ms
990:    learn: 0.0810613        total: 2.5s     remaining: 22.7ms
991:    learn: 0.0809114        total: 2.5s     remaining: 20.2ms
992:    learn: 0.0808569        total: 2.5s     remaining: 17.7ms
993:    learn: 0.0807888        total: 2.51s    remaining: 15.1ms
994:    learn: 0.0807345        total: 2.51s    remaining: 12.6ms
995:    learn: 0.0806805        total: 2.51s    remaining: 10.1ms
996:    learn: 0.0805879        total: 2.51s    remaining: 7.56ms
997:    learn: 0.0805220        total: 2.52s    remaining: 5.04ms
998:    learn: 0.0804183        total: 2.52s    remaining: 2.53ms
999:    learn: 0.0802500        total: 2.53s    remaining: 0us
Accuracy: 0.9014084507042254
```

- **LSTM**

```
from keras.models import Sequential
from keras.layers import LSTM, Dense

# reshape the input data to be 3D for LSTM
x_train = x_train.values.reshape((x_train.shape[0], 1, x_train.shape[1]))
x_test = x_test.values.reshape((x_test.shape[0], 1, x_test.shape[1]))

# define the LSTM model
model = Sequential()
model.add(LSTM(50, input_shape=(x_train.shape[1], x_train.shape[2])))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# train the LSTM model
history = model.fit(x_train, y_train, epochs=50, batch_size=32, validation_data=(x_test, y_test), verbose=2, shuffle=False)

# evaluate the LSTM model
loss, accuracy = model.evaluate(x_test, y_test, verbose=0)
print('Accuracy:', accuracy)
```

```
Epoch 43/50
9/9 - 0s - loss: 0.1929 - accuracy: 0.9075 - val_loss: 0.3711 - val_accuracy: 0.8732 - 81ms/epoch - 9ms/step
Epoch 44/50
9/9 - 0s - loss: 0.1918 - accuracy: 0.9075 - val_loss: 0.3722 - val_accuracy: 0.8732 - 90ms/epoch - 10ms/step
Epoch 45/50
9/9 - 0s - loss: 0.1908 - accuracy: 0.9075 - val_loss: 0.3732 - val_accuracy: 0.8732 - 77ms/epoch - 9ms/step
Epoch 46/50
9/9 - 0s - loss: 0.1896 - accuracy: 0.9075 - val_loss: 0.3743 - val_accuracy: 0.8732 - 79ms/epoch - 9ms/step
Epoch 47/50
9/9 - 0s - loss: 0.1883 - accuracy: 0.9075 - val_loss: 0.3760 - val_accuracy: 0.8732 - 83ms/epoch - 9ms/step
Epoch 48/50
9/9 - 0s - loss: 0.1867 - accuracy: 0.9075 - val_loss: 0.3774 - val_accuracy: 0.8732 - 65ms/epoch - 7ms/step
Epoch 49/50
9/9 - 0s - loss: 0.1855 - accuracy: 0.9075 - val_loss: 0.3785 - val_accuracy: 0.8732 - 62ms/epoch - 7ms/step
Epoch 50/50
9/9 - 0s - loss: 0.1839 - accuracy: 0.9075 - val_loss: 0.3793 - val_accuracy: 0.8732 - 66ms/epoch - 7ms/step
Accuracy: 0.8732394576072693
```

- **ANN**

```
# ANN

# import pandas as pd
from keras.models import Sequential
from keras.layers import Dense

# load the dataset
# df = pd.read_csv('dataset.csv')

# split the dataset into input and output variables
# X = df.drop(columns=['output'])
# y = df['output']

# define the model
model = Sequential()
model.add(Dense(10, input_dim=x.shape[1], activation='relu'))
model.add(Dense(1, activation='linear'))

# compile the model
model.compile(loss='mse', optimizer='adam', metrics=['mae'])

# train the model
model.fit(x, y, epochs=100, batch_size=32)

# make predictions
y_pred = model.predict(x)

# evaluate the model
mse, mae = model.evaluate(x, y)

print('MSE: ', mse)
print('MAE: ', mae)
```

```
Epoch 94/100
11/11 [==============================] - 0s 7ms/step - loss: 10.3213 - mae: 1.7842
Epoch 95/100
11/11 [==============================] - 0s 7ms/step - loss: 10.1453 - mae: 1.7717
Epoch 96/100
11/11 [==============================] - 0s 8ms/step - loss: 9.2320 - mae: 1.7181
Epoch 97/100
11/11 [==============================] - 0s 8ms/step - loss: 8.7383 - mae: 1.6728
Epoch 98/100
11/11 [==============================] - 0s 5ms/step - loss: 7.9554 - mae: 1.6308
Epoch 99/100
11/11 [==============================] - 0s 6ms/step - loss: 7.9434 - mae: 1.6454
Epoch 100/100
11/11 [==============================] - 0s 6ms/step - loss: 7.7111 - mae: 1.6308
11/11 [==============================] - 0s 6ms/step
11/11 [==============================] - 0s 4ms/step - loss: 7.4078 - mae: 1.5975
MSE:  7.407775402069092
MAE:  1.597489833831787
```

7. Dataset : jedit-3.2

- **Importing the Libraries**

```
import pandas as pd
import numpy as np
import seaborn as sns
```

- **Loading the dataset**

```
data = pd.read_csv('/content/ant-1.3.csv')
# data['defects'] = data['defects'].replace({True:1,False:0})
data.head()
```

	wmc	dit	noc	cbo	rfc	lcom	ca	ce	npm	lcom3	...	dam	moa	mfa	cam	ic	cbm	amc	max_cc	avg_cc	bug
0	14	1	0	9	45	61	3	6	11	0.769231	...	1.0	1	0.000000	0.294872	0	0	29.142857	8	2.4286	0
1	7	2	0	4	21	0	1	3	0	0.541667	...	1.0	1	0.538462	0.333333	1	3	41.285714	16	4.7143	0
2	6	1	0	3	11	9	1	3	5	0.600000	...	1.0	3	0.000000	0.333333	0	0	16.333333	5	1.5000	0
3	6	2	0	16	14	3	8	8	5	0.666667	...	1.0	0	0.782609	0.444444	0	0	13.833333	1	0.8333	0
4	5	2	0	5	22	0	1	4	0	0.125000	...	1.0	0	0.700000	0.350000	1	3	39.800000	11	4.0000	0

- **Training the data**

```
x = data.drop(['bug'],axis=1)
y = data['bug']
```

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2,random_state=4)
```

- **LOGISTIC REGRESSION**

```
# LogisticRegression
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()

# train the logistic regression classifier using the training data
logreg.fit(x_train, y_train)

# make predictions on the testing data
y_pred = logreg.predict(x_test)

from sklearn.metrics import accuracy_score

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.7454545454545455
/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

- **SVM**

```
# SVM

from sklearn import svm
classifier = svm.SVC(kernel='linear',gamma='auto',C=2)
classifier.fit(x_train,y_train)
y_predict = classifier.predict(x_test)

from sklearn.metrics import accuracy_score

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_predict)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.7818181818181819
```

- **NAIVE BAYES**

```
# Naive Bayes
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()

# train the Naive Bayes classifier using the training data
gnb.fit(x_train, y_train)

# make predictions on the testing data
y_pred = gnb.predict(x_test)

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.8181818181818182
```

- **DECISION TREE**

```
# Decision Tree
from sklearn.tree import DecisionTreeClassifier

# create a Decision Tree classifier object
dtc = DecisionTreeClassifier()

# train the Decision Tree classifier using the training data
dtc.fit(x_train, y_train)

# make predictions on the testing data
y_pred = dtc.predict(x_test)

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.7454545454545455
```

- **RANDOM FOREST**

```
# Random Forest
from sklearn.ensemble import RandomForestClassifier


# create a Random Forest classifier object
rfc = RandomForestClassifier(n_estimators=100)

# train the Random Forest classifier using the training data
rfc.fit(x_train, y_train)

# make predictions on the testing data
y_pred = rfc.predict(x_test)

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.8181818181818182
```

- **XGBOOST**

```
# XGBoost

import xgboost as xgb

# convert data into DMatrix format
dtrain = xgb.DMatrix(x_train, label=y_train)
dtest = xgb.DMatrix(x_test, label=y_test)

# set the parameters for XGBoost
params = {
    'max_depth': 3,
    'eta': 0.1,
    'objective': 'multi:softmax',
    'num_class': 3
}

# train the XGBoost model
num_rounds = 100
xgb_model = xgb.train(params, dtrain, num_rounds)

# make predictions on the testing data
y_pred = xgb_model.predict(dtest)

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.7818181818181819
```

- **KNN**

```
# KNN

from sklearn.neighbors import KNeighborsClassifier


# create a KNN classifier object
knn = KNeighborsClassifier(n_neighbors=5)

# train the KNN classifier using the training data
knn.fit(x_train, y_train)

# make predictions on the testing data
y_pred = knn.predict(x_test)

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.7636363636363637
```

- **CATBOOST**

```
!pip install catboost
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting catboost
  Downloading catboost-1.1.1-cp39-none-manylinux1_x86_64.whl (76.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 76.6/76.6 MB 9.2 MB/s eta 0:00:00
Requirement already satisfied: graphviz in /usr/local/lib/python3.9/dist-packages (from catboost) (0.20.1)
Requirement already satisfied: six in /usr/local/lib/python3.9/dist-packages (from catboost) (1.16.0)
Requirement already satisfied: numpy>=1.16.0 in /usr/local/lib/python3.9/dist-packages (from catboost) (1.22.4)
Requirement already satisfied: scipy in /usr/local/lib/python3.9/dist-packages (from catboost) (1.10.1)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.9/dist-packages (from catboost) (3.7.1)
Requirement already satisfied: plotly in /usr/local/lib/python3.9/dist-packages (from catboost) (5.13.1)
Requirement already satisfied: pandas>=0.24.0 in /usr/local/lib/python3.9/dist-packages (from catboost) (1.5.3)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.9/dist-packages (from pandas>=0.24.0->catboost) (2022.7.1)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.9/dist-packages (from pandas>=0.24.0->catboost) (2.8.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (8.4.0)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (23.1)
Requirement already satisfied: importlib-resources>=3.2.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (5.12.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (1.0.7)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (1.4.4)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (4.39.3)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (3.0.9)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.9/dist-packages (from plotly->catboost) (8.2.2)
Requirement already satisfied: zipp>=3.1.0 in /usr/local/lib/python3.9/dist-packages (from importlib-resources>=3.2.0->matplotlib->catboost) (3.15.0)
Installing collected packages: catboost
Successfully installed catboost-1.1.1
```

```
# CatBoost
import catboost as ctb
# create a CatBoost classifier object
clf = ctb.CatBoostClassifier()

# train the CatBoost classifier using the training data
clf.fit(x_train, y_train)

# make predictions on the testing data
y_pred = clf.predict(x_test)

# calculate the accuracy of the model
accuracy = clf.score(x_test, y_test)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
976:    learn: 0.1315319        total: 2.17s    remaining: 51ms
977:    learn: 0.1313876        total: 2.17s    remaining: 48.9ms
978:    learn: 0.1313269        total: 2.18s    remaining: 46.7ms
979:    learn: 0.1312174        total: 2.18s    remaining: 44.6ms
980:    learn: 0.1309514        total: 2.19s    remaining: 42.4ms
981:    learn: 0.1308109        total: 2.19s    remaining: 40.2ms
982:    learn: 0.1306879        total: 2.2s     remaining: 38ms
983:    learn: 0.1306092        total: 2.2s     remaining: 35.8ms
984:    learn: 0.1304835        total: 2.21s    remaining: 33.6ms
985:    learn: 0.1303052        total: 2.21s    remaining: 31.4ms
986:    learn: 0.1302259        total: 2.22s    remaining: 29.2ms
987:    learn: 0.1300429        total: 2.22s    remaining: 27ms
988:    learn: 0.1300207        total: 2.23s    remaining: 24.8ms
989:    learn: 0.1298542        total: 2.23s    remaining: 22.6ms
990:    learn: 0.1296700        total: 2.24s    remaining: 20.4ms
991:    learn: 0.1295143        total: 2.24s    remaining: 18.1ms
992:    learn: 0.1293480        total: 2.25s    remaining: 15.9ms
993:    learn: 0.1291844        total: 2.25s    remaining: 13.6ms
994:    learn: 0.1291056        total: 2.26s    remaining: 11.4ms
995:    learn: 0.1288774        total: 2.27s    remaining: 9.1ms
996:    learn: 0.1288208        total: 2.27s    remaining: 6.84ms
997:    learn: 0.1286372        total: 2.28s    remaining: 4.56ms
998:    learn: 0.1283832        total: 2.28s    remaining: 2.28ms
999:    learn: 0.1283055        total: 2.29s    remaining: 0us
Accuracy: 0.8
```

- **LSTM**

```
from keras.models import Sequential
from keras.layers import LSTM, Dense

# reshape the input data to be 3D for LSTM
x_train = x_train.values.reshape((x_train.shape[0], 1, x_train.shape[1]))
x_test = x_test.values.reshape((x_test.shape[0], 1, x_test.shape[1]))

# define the LSTM model
model = Sequential()
model.add(LSTM(50, input_shape=(x_train.shape[1], x_train.shape[2])))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# train the LSTM model
history = model.fit(x_train, y_train, epochs=50, batch_size=32, validation_data=(x_test, y_test), verbose=2, shuffle=False)

# evaluate the LSTM model
loss, accuracy = model.evaluate(x_test, y_test, verbose=0)
print('Accuracy:', accuracy)
```

```
Epoch 45/50
7/7 - 0s - loss: 0.4041 - accuracy: 0.8157 - val_loss: 0.4919 - val_accuracy: 0.8182 - 61ms/epoch - 9ms/step
Epoch 46/50
7/7 - 0s - loss: 0.4020 - accuracy: 0.8157 - val_loss: 0.4920 - val_accuracy: 0.8182 - 54ms/epoch - 8ms/step
Epoch 47/50
7/7 - 0s - loss: 0.4009 - accuracy: 0.8111 - val_loss: 0.4935 - val_accuracy: 0.8182 - 59ms/epoch - 8ms/step
Epoch 48/50
7/7 - 0s - loss: 0.3996 - accuracy: 0.8157 - val_loss: 0.4940 - val_accuracy: 0.8182 - 77ms/epoch - 11ms/step
Epoch 49/50
7/7 - 0s - loss: 0.3982 - accuracy: 0.8157 - val_loss: 0.4943 - val_accuracy: 0.8182 - 77ms/epoch - 11ms/step
Epoch 50/50
7/7 - 0s - loss: 0.3953 - accuracy: 0.8157 - val_loss: 0.4947 - val_accuracy: 0.8182 - 72ms/epoch - 10ms/step
Accuracy: 0.8181818127632141
```

- **ANN**

```
# import pandas as pd
from keras.models import Sequential
from keras.layers import Dense

# load the dataset
# df = pd.read_csv('dataset.csv')

# split the dataset into input and output variables
# X = df.drop(columns=['output'])
# y = df['output']

# define the model
model = Sequential()
model.add(Dense(10, input_dim=x.shape[1], activation='relu'))
model.add(Dense(1, activation='linear'))

# compile the model
model.compile(loss='mse', optimizer='adam', metrics=['mae'])

# train the model
model.fit(x, y, epochs=100, batch_size=32)

# make predictions
y_pred = model.predict(x)

# evaluate the model
mse, mae = model.evaluate(x, y)

print('MSE: ', mse)
print('MAE: ', mae)
```

```
Epoch 95/100
9/9 [==============================] - 0s 3ms/step - loss: 295.5370 - mae: 6.5710
Epoch 96/100
9/9 [==============================] - 0s 3ms/step - loss: 297.5590 - mae: 6.6650
Epoch 97/100
9/9 [==============================] - 0s 3ms/step - loss: 292.2679 - mae: 6.7821
Epoch 98/100
9/9 [==============================] - 0s 3ms/step - loss: 274.0948 - mae: 6.6248
Epoch 99/100
9/9 [==============================] - 0s 4ms/step - loss: 283.9646 - mae: 6.5978
Epoch 100/100
9/9 [==============================] - 0s 3ms/step - loss: 279.0989 - mae: 6.5640
9/9 [==============================] - 0s 2ms/step
9/9 [==============================] - 0s 2ms/step - loss: 272.0435 - mae: 6.5595
MSE:  272.04351806640625
MAE:  6.559475898742676
```

8. Dataset : log4j-1.0

- **Importing the Libraries**

```
import pandas as pd
import numpy as np
import seaborn as sns
```

- **Loading the dataset**

```
data = pd.read_csv('/content/log4j-1.0.csv')
# data['defects'] = data['defects'].replace({True:1,False:0})
data.head()
```

	wmc	dit	noc	cbo	rfc	lcom	ca	ce	npm	lcom3	...	dam	moa	mfa	cam	ic	cbm	amc	max_cc	avg_cc	bug
0	4	4	0	3	12	6	2	1	4	2.000000	...	0.000000	0	0.953488	0.357143	1	3	23.750	3	1.000	0
1	2	1	0	0	3	1	0	0	2	2.000000	...	0.000000	0	0.000000	0.500000	0	0	2.000	1	0.500	0
2	8	1	0	6	22	0	3	3	8	0.142857	...	1.000000	0	0.000000	0.406250	0	0	20.875	5	3.125	0
3	8	2	1	7	24	12	1	6	8	0.936508	...	0.555556	0	0.533333	0.333333	0	0	20.250	7	2.250	0
4	10	1	5	34	13	45	32	2	9	1.000000	...	0.000000	0	0.000000	0.407407	0	0	1.600	1	0.800	0

5 rows × 21 columns

- **Training the data**

```
x = data.drop(['bug'],axis=1)
y = data['bug']
```

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2,random_state=4)
```

- **LOGISTIC REGRESSION**

```
# LogisticRegression
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()

# train the logistic regression classifier using the training data
logreg.fit(x_train, y_train)

# make predictions on the testing data
y_pred = logreg.predict(x_test)

from sklearn.metrics import accuracy_score

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.8888888888888888
/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

- **SVM**

```
# SVM

from sklearn import svm
classifier = svm.SVC(kernel='linear',gamma='auto',C=2)
classifier.fit(x_train,y_train)
y_predict = classifier.predict(x_test)

from sklearn.metrics import accuracy_score

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_predict)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.8148148148148148
```

- **NAIVE BAYES**

```
# Naive Bayes
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()

# train the Naive Bayes classifier using the training data
gnb.fit(x_train, y_train)

# make predictions on the testing data
y_pred = gnb.predict(x_test)

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.8888888888888888
```

- **DECISION TREE**

```
# Decision Tree
from sklearn.tree import DecisionTreeClassifier

# create a Decision Tree classifier object
dtc = DecisionTreeClassifier()

# train the Decision Tree classifier using the training data
dtc.fit(x_train, y_train)

# make predictions on the testing data
y_pred = dtc.predict(x_test)

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.8888888888888888
```

- **RANDOM FOREST**

```
# Random Forest
Add code cell
⌘/Ctrl+M B
n.ensemble import RandomForestClassifier

# create a Random Forest classifier object
rfc = RandomForestClassifier(n_estimators=100)

# train the Random Forest classifier using the training data
rfc.fit(x_train, y_train)

# make predictions on the testing data
y_pred = rfc.predict(x_test)

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.8148148148148148
```

- **XGBOOST**

```
# XGBoost

import xgboost as xgb

# convert data into DMatrix format
dtrain = xgb.DMatrix(x_train, label=y_train)
dtest = xgb.DMatrix(x_test, label=y_test)

# set the parameters for XGBoost
params = {
    'max_depth': 3,
    'eta': 0.1,
    'objective': 'multi:softmax',
    'num_class': 3
}

# train the XGBoost model
num_rounds = 100
xgb_model = xgb.train(params, dtrain, num_rounds)

# make predictions on the testing data
y_pred = xgb_model.predict(dtest)

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.8888888888888888
```

- **KNN**

```
# KNN

from sklearn.neighbors import KNeighborsClassifier


# create a KNN classifier object
knn = KNeighborsClassifier(n_neighbors=5)

# train the KNN classifier using the training data
knn.fit(x_train, y_train)

# make predictions on the testing data
y_pred = knn.predict(x_test)

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.7037037037037037
```

- **CATBOOST**

```
!pip install catboost
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting catboost
  Downloading catboost-1.1.1-cp39-none-manylinux1_x86_64.whl (76.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 76.6/76.6 MB 9.2 MB/s eta 0:00:00
Requirement already satisfied: graphviz in /usr/local/lib/python3.9/dist-packages (from catboost) (0.20.1)
Requirement already satisfied: six in /usr/local/lib/python3.9/dist-packages (from catboost) (1.16.0)
Requirement already satisfied: numpy>=1.16.0 in /usr/local/lib/python3.9/dist-packages (from catboost) (1.22.4)
Requirement already satisfied: scipy in /usr/local/lib/python3.9/dist-packages (from catboost) (1.10.1)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.9/dist-packages (from catboost) (3.7.1)
Requirement already satisfied: plotly in /usr/local/lib/python3.9/dist-packages (from catboost) (5.13.1)
Requirement already satisfied: pandas>=0.24.0 in /usr/local/lib/python3.9/dist-packages (from catboost) (1.5.3)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.9/dist-packages (from pandas>=0.24.0->catboost) (2022.7.1)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.9/dist-packages (from pandas>=0.24.0->catboost) (2.8.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (8.4.0)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (23.1)
Requirement already satisfied: importlib-resources>=3.2.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (5.12.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (1.0.7)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (1.4.4)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (4.39.3)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (3.0.9)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.9/dist-packages (from plotly->catboost) (8.2.2)
Requirement already satisfied: zipp>=3.1.0 in /usr/local/lib/python3.9/dist-packages (from importlib-resources>=3.2.0->matplotlib->catboost) (3.15.0)
Installing collected packages: catboost
Successfully installed catboost-1.1.1
```

```
# CatBoost
import catboost as ctb
# create a CatBoost classifier object
clf = ctb.CatBoostClassifier()

# train the CatBoost classifier using the training data
clf.fit(x_train, y_train)

# make predictions on the testing data
y_pred = clf.predict(x_test)

# calculate the accuracy of the model
accuracy = clf.score(x_test, y_test)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
981:    learn: 0.1318683        total: 1.55s    remaining: 28.4ms
982:    learn: 0.1317226        total: 1.55s    remaining: 26.8ms
983:    learn: 0.1314790        total: 1.55s    remaining: 25.2ms
984:    learn: 0.1312650        total: 1.55s    remaining: 23.6ms
985:    learn: 0.1310888        total: 1.55s    remaining: 22ms
986:    learn: 0.1310297        total: 1.55s    remaining: 20.5ms
987:    learn: 0.1308797        total: 1.55s    remaining: 18.9ms
988:    learn: 0.1307109        total: 1.55s    remaining: 17.3ms
989:    learn: 0.1306455        total: 1.56s    remaining: 15.7ms
990:    learn: 0.1305596        total: 1.56s    remaining: 14.2ms
991:    learn: 0.1304375        total: 1.56s    remaining: 12.6ms
992:    learn: 0.1303452        total: 1.56s    remaining: 11ms
993:    learn: 0.1301269        total: 1.56s    remaining: 9.44ms
994:    learn: 0.1299448        total: 1.56s    remaining: 7.86ms
995:    learn: 0.1298667        total: 1.56s    remaining: 6.29ms
996:    learn: 0.1297169        total: 1.57s    remaining: 4.71ms
997:    learn: 0.1296195        total: 1.57s    remaining: 3.14ms
998:    learn: 0.1294387        total: 1.57s    remaining: 1.57ms
999:    learn: 0.1293364        total: 1.57s    remaining: 0us
Accuracy: 0.8518518518518519
```

- **LSTM**

```
from keras.models import Sequential
from keras.layers import LSTM, Dense

# reshape the input data to be 3D for LSTM
x_train = x_train.values.reshape((x_train.shape[0], 1, x_train.shape[1]))
x_test = x_test.values.reshape((x_test.shape[0], 1, x_test.shape[1]))

# define the LSTM model
model = Sequential()
model.add(LSTM(50, input_shape=(x_train.shape[1], x_train.shape[2])))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# train the LSTM model
history = model.fit(x_train, y_train, epochs=50, batch_size=32, validation_data=(x_test, y_test), verbose=2, shuffle=False)

# evaluate the LSTM model
loss, accuracy = model.evaluate(x_test, y_test, verbose=0)
print('Accuracy:', accuracy)
```

```
Epoch 44/50
4/4 - 0s - loss: 0.3175 - accuracy: 0.8704 - val_loss: 0.4116 - val_accuracy: 0.8519 - 65ms/epoch - 16ms/step
Epoch 45/50
4/4 - 0s - loss: 0.3156 - accuracy: 0.8704 - val_loss: 0.4118 - val_accuracy: 0.8519 - 111ms/epoch - 28ms/step
Epoch 46/50
4/4 - 0s - loss: 0.3138 - accuracy: 0.8704 - val_loss: 0.4122 - val_accuracy: 0.8519 - 123ms/epoch - 31ms/step
Epoch 47/50
4/4 - 0s - loss: 0.3120 - accuracy: 0.8704 - val_loss: 0.4126 - val_accuracy: 0.8519 - 96ms/epoch - 24ms/step
Epoch 48/50
4/4 - 0s - loss: 0.3102 - accuracy: 0.8704 - val_loss: 0.4129 - val_accuracy: 0.8519 - 102ms/epoch - 26ms/step
Epoch 49/50
4/4 - 0s - loss: 0.3085 - accuracy: 0.8704 - val_loss: 0.4132 - val_accuracy: 0.8519 - 90ms/epoch - 23ms/step
Epoch 50/50
4/4 - 0s - loss: 0.3069 - accuracy: 0.8704 - val_loss: 0.4133 - val_accuracy: 0.8519 - 89ms/epoch - 22ms/step
Accuracy: 0.8518518805503845
```

- **ANN**

```
# import pandas as pd
from keras.models import Sequential
from keras.layers import Dense

# load the dataset
# df = pd.read_csv('dataset.csv')

# split the dataset into input and output variables
# X = df.drop(columns=['output'])
# y = df['output']

# define the model
model = Sequential()
model.add(Dense(10, input_dim=x.shape[1], activation='relu'))
model.add(Dense(1, activation='linear'))

# compile the model
model.compile(loss='mse', optimizer='adam', metrics=['mae'])

# train the model
model.fit(x, y, epochs=100, batch_size=32)

# make predictions
y_pred = model.predict(x)

# evaluate the model
mse, mae = model.evaluate(x, y)

print('MSE: ', mse)
print('MAE: ', mae)
```

```
Epoch 92/100
5/5 [==============================] - 0s 3ms/step - loss: 45.1964 - mae: 4.8610
Epoch 93/100
5/5 [==============================] - 0s 4ms/step - loss: 44.8316 - mae: 4.8409
Epoch 94/100
5/5 [==============================] - 0s 4ms/step - loss: 44.5059 - mae: 4.8271
Epoch 95/100
5/5 [==============================] - 0s 3ms/step - loss: 44.1746 - mae: 4.8063
Epoch 96/100
5/5 [==============================] - 0s 3ms/step - loss: 43.7963 - mae: 4.7833
Epoch 97/100
5/5 [==============================] - 0s 4ms/step - loss: 43.4571 - mae: 4.7563
Epoch 98/100
5/5 [==============================] - 0s 4ms/step - loss: 43.0787 - mae: 4.7314
Epoch 99/100
5/5 [==============================] - 0s 4ms/step - loss: 42.7581 - mae: 4.7060
Epoch 100/100
5/5 [==============================] - 0s 3ms/step - loss: 42.4175 - mae: 4.6796
5/5 [==============================] - 0s 2ms/step
5/5 [==============================] - 0s 4ms/step - loss: 42.1112 - mae: 4.6612
MSE:  42.1112060546875
MAE:  4.661229133605957
```

9. Dataset : lucene-2.0

- **Importing the Libraries**

```
import pandas as pd
import numpy as np
import seaborn as sns
```

- **Loading the dataset**

```
data = pd.read_csv('/content/lucene-2.0.csv')
# data['defects'] = data['defects'].replace({True:1,False:0})
data.head()
```

	wmc	dit	noc	cbo	rfc	lcom	ca	ce	npm	lcom3	...	dam	moa	mfa	cam	ic	cbm	amc	max_cc	avg_cc	bug
0	2	2	0	3	4	1	0	3	2	2.000000	...	0.0	0	0.666667	0.666667	0	0	4.000000	1	0.5000	0
1	10	1	0	4	37	0	0	4	9	0.388889	...	1.0	0	0.000000	0.340000	0	0	26.600000	5	1.6000	0
2	27	1	0	1	43	13	1	0	13	0.600962	...	1.0	0	0.000000	0.253086	0	0	42.185185	26	5.7407	0
3	1	1	0	21	1	0	20	1	1	2.000000	...	0.0	0	0.000000	1.000000	0	0	0.000000	1	1.0000	1
4	3	1	2	19	4	3	18	1	3	2.000000	...	0.0	0	0.000000	1.000000	0	0	1.333333	1	0.6667	1

- **Training the data**

```
x = data.drop(['bug'],axis=1)
y = data['bug']
```

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2,random_state=4)
```

- **LOGISTIC REGRESSION**

```
# LogisticRegression
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()

# train the logistic regression classifier using the training data
logreg.fit(x_train, y_train)

# make predictions on the testing data
y_pred = logreg.predict(x_test)

from sklearn.metrics import accuracy_score

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.717948717948718
/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

- **SVM**

```
# SVM

from sklearn import svm
classifier = svm.SVC(kernel='linear',gamma='auto',C=2)
classifier.fit(x_train,y_train)
y_predict = classifier.predict(x_test)

from sklearn.metrics import accuracy_score

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_predict)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.8205128205128205
```

- **NAIVE BAYES**

```
# Naive Bayes
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()

# train the Naive Bayes classifier using the training data
gnb.fit(x_train, y_train)

# make predictions on the testing data
y_pred = gnb.predict(x_test)

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.6923076923076923
```

- **DECISION TREE**

```
# Decision Tree
from sklearn.tree import DecisionTreeClassifier

# create a Decision Tree classifier object
dtc = DecisionTreeClassifier()

# train the Decision Tree classifier using the training data
dtc.fit(x_train, y_train)

# make predictions on the testing data
y_pred = dtc.predict(x_test)

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.5897435897435898
```

- **RANDOM FOREST**

```
# Random Forest
from sklearn.ensemble import RandomForestClassifier


# create a Random Forest classifier object
rfc = RandomForestClassifier(n_estimators=100)

# train the Random Forest classifier using the training data
rfc.fit(x_train, y_train)

# make predictions on the testing data
y_pred = rfc.predict(x_test)

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.7692307692307693
```

- **XGBOOST**

```
# XGBoost

import xgboost as xgb

# convert data into DMatrix format
dtrain = xgb.DMatrix(x_train, label=y_train)
dtest = xgb.DMatrix(x_test, label=y_test)

# set the parameters for XGBoost
params = {
    'max_depth': 3,
    'eta': 0.1,
    'objective': 'multi:softmax',
    'num_class': 3
}

# train the XGBoost model
num_rounds = 100
xgb_model = xgb.train(params, dtrain, num_rounds)

# make predictions on the testing data
y_pred = xgb_model.predict(dtest)

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.7435897435897436
```

- **KNN**

```
# KNN

from sklearn.neighbors import KNeighborsClassifier


# create a KNN classifier object
knn = KNeighborsClassifier(n_neighbors=5)

# train the KNN classifier using the training data
knn.fit(x_train, y_train)

# make predictions on the testing data
y_pred = knn.predict(x_test)

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.6666666666666666
```

- **CATBOOST**

```
!pip install catboost
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting catboost
  Downloading catboost-1.1.1-cp39-none-manylinux1_x86_64.whl (76.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 76.6/76.6 MB 9.2 MB/s eta 0:00:00
Requirement already satisfied: graphviz in /usr/local/lib/python3.9/dist-packages (from catboost) (0.20.1)
Requirement already satisfied: six in /usr/local/lib/python3.9/dist-packages (from catboost) (1.16.0)
Requirement already satisfied: numpy>=1.16.0 in /usr/local/lib/python3.9/dist-packages (from catboost) (1.22.4)
Requirement already satisfied: scipy in /usr/local/lib/python3.9/dist-packages (from catboost) (1.10.1)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.9/dist-packages (from catboost) (3.7.1)
Requirement already satisfied: plotly in /usr/local/lib/python3.9/dist-packages (from catboost) (5.13.1)
Requirement already satisfied: pandas>=0.24.0 in /usr/local/lib/python3.9/dist-packages (from catboost) (1.5.3)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.9/dist-packages (from pandas>=0.24.0->catboost) (2022.7.1)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.9/dist-packages (from pandas>=0.24.0->catboost) (2.8.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (8.4.0)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (23.1)
Requirement already satisfied: importlib-resources>=3.2.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (5.12.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (1.0.7)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (1.4.4)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (4.39.3)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (3.0.9)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.9/dist-packages (from plotly->catboost) (8.2.2)
Requirement already satisfied: zipp>=3.1.0 in /usr/local/lib/python3.9/dist-packages (from importlib-resources>=3.2.0->matplotlib->catboost) (3.15.0)
Installing collected packages: catboost
Successfully installed catboost-1.1.1
```

```
# CatBoost
import catboost as ctb
# create a CatBoost classifier object
clf = ctb.CatBoostClassifier()

# train the CatBoost classifier using the training data
clf.fit(x_train, y_train)

# make predictions on the testing data
y_pred = clf.predict(x_test)

# calculate the accuracy of the model
accuracy = clf.score(x_test, y_test)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
983:    learn: 0.2184291        total: 1.64s    remaining: 26.7ms
984:    learn: 0.2183253        total: 1.64s    remaining: 25ms
985:    learn: 0.2180930        total: 1.64s    remaining: 23.4ms
986:    learn: 0.2178689        total: 1.65s    remaining: 21.7ms
987:    learn: 0.2176189        total: 1.65s    remaining: 20ms
988:    learn: 0.2173807        total: 1.65s    remaining: 18.3ms
989:    learn: 0.2171719        total: 1.65s    remaining: 16.7ms
990:    learn: 0.2169116        total: 1.65s    remaining: 15ms
991:    learn: 0.2167131        total: 1.65s    remaining: 13.3ms
992:    learn: 0.2165356        total: 1.66s    remaining: 11.7ms
993:    learn: 0.2163100        total: 1.66s    remaining: 10ms
994:    learn: 0.2161428        total: 1.66s    remaining: 8.34ms
995:    learn: 0.2159458        total: 1.66s    remaining: 6.67ms
996:    learn: 0.2156532        total: 1.66s    remaining: 5ms
997:    learn: 0.2153964        total: 1.66s    remaining: 3.33ms
998:    learn: 0.2151002        total: 1.67s    remaining: 1.67ms
999:    learn: 0.2149605        total: 1.67s    remaining: 0us
Accuracy: 0.7692307692307693
```

- **LSTM**

```
from keras.models import Sequential
from keras.layers import LSTM, Dense

# reshape the input data to be 3D for LSTM
x_train = x_train.values.reshape((x_train.shape[0], 1, x_train.shape[1]))
x_test = x_test.values.reshape((x_test.shape[0], 1, x_test.shape[1]))

# define the LSTM model
model = Sequential()
model.add(LSTM(50, input_shape=(x_train.shape[1], x_train.shape[2])))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# train the LSTM model
history = model.fit(x_train, y_train, epochs=50, batch_size=32, validation_data=(x_test, y_test), verbose=2, shuffle=False)

# evaluate the LSTM model
loss, accuracy = model.evaluate(x_test, y_test, verbose=0)
print('Accuracy:', accuracy)
```

```
Epoch 43/50
5/5 - 0s - loss: 0.4360 - accuracy: 0.8397 - val_loss: 0.5945 - val_accuracy: 0.7179 - 84ms/epoch - 17ms/step
Epoch 44/50
5/5 - 0s - loss: 0.4331 - accuracy: 0.8397 - val_loss: 0.5945 - val_accuracy: 0.7179 - 81ms/epoch - 16ms/step
Epoch 45/50
5/5 - 0s - loss: 0.4309 - accuracy: 0.8397 - val_loss: 0.5950 - val_accuracy: 0.7179 - 87ms/epoch - 17ms/step
Epoch 46/50
5/5 - 0s - loss: 0.4284 - accuracy: 0.8397 - val_loss: 0.5961 - val_accuracy: 0.7179 - 79ms/epoch - 16ms/step
Epoch 47/50
5/5 - 0s - loss: 0.4254 - accuracy: 0.8462 - val_loss: 0.5976 - val_accuracy: 0.7179 - 74ms/epoch - 15ms/step
Epoch 48/50
5/5 - 0s - loss: 0.4225 - accuracy: 0.8462 - val_loss: 0.5984 - val_accuracy: 0.7179 - 101ms/epoch - 20ms/step
Epoch 49/50
5/5 - 0s - loss: 0.4200 - accuracy: 0.8526 - val_loss: 0.5985 - val_accuracy: 0.7179 - 69ms/epoch - 14ms/step
Epoch 50/50
5/5 - 0s - loss: 0.4178 - accuracy: 0.8526 - val_loss: 0.5983 - val_accuracy: 0.7179 - 71ms/epoch - 14ms/step
Accuracy: 0.7179487347602844
```

- **ANN**

```
# import pandas as pd
from keras.models import Sequential
from keras.layers import Dense

# load the dataset
# df = pd.read_csv('dataset.csv')

# split the dataset into input and output variables
# X = df.drop(columns=['output'])
# y = df['output']

# define the model
model = Sequential()
model.add(Dense(10, input_dim=x.shape[1], activation='relu'))
model.add(Dense(1, activation='linear'))

# compile the model
model.compile(loss='mse', optimizer='adam', metrics=['mae'])

# train the model
model.fit(x, y, epochs=100, batch_size=32)

# make predictions
y_pred = model.predict(x)

# evaluate the model
mse, mae = model.evaluate(x, y)

print('MSE: ', mse)
print('MAE: ', mae)
```

```
Epoch 96/100
7/7 [==============================] - 0s 3ms/step - loss: 1.7826 - mae: 0.9696
Epoch 97/100
7/7 [==============================] - 0s 3ms/step - loss: 1.7890 - mae: 0.9662
Epoch 98/100
7/7 [==============================] - 0s 4ms/step - loss: 1.7444 - mae: 0.9563
Epoch 99/100
7/7 [==============================] - 0s 3ms/step - loss: 1.7900 - mae: 0.9685
Epoch 100/100
7/7 [==============================] - 0s 3ms/step - loss: 1.8483 - mae: 0.9679
7/7 [==============================] - 0s 2ms/step
7/7 [==============================] - 0s 3ms/step - loss: 1.7265 - mae: 0.9545
MSE:  1.7264751195907593
MAE:  0.954481840133667
```

10. Dataset : synapse-1.0

- **Importing the Libraries**

```
import pandas as pd
import numpy as np
import seaborn as sns
```

- **Loading the dataset**

```
data = pd.read_csv('/content/synapse-1.0.csv')
# data['defects'] = data['defects'].replace({True:1,False:0})
data.head()
```

	wmc	dit	noc	cbo	rfc	lcom	ca	ce	npm	lcom3	...	dam	moa	mfa	cam	ic	cbm	amc	max_cc	avg_cc	bug
0	6	1	0	12	65	3	1	11	2	0.700000	...	0.500000	0	0.00	0.300000	0	0	96.166667	39	7.0	0
1	4	3	0	5	9	4	0	5	3	0.666667	...	1.000000	0	0.60	0.666667	0	0	6.250000	1	0.5	0
2	5	2	0	11	41	6	0	11	3	0.846154	...	0.923077	0	0.25	0.416667	0	0	81.000000	17	3.8	0
3	6	1	0	6	22	0	4	3	5	0.000000	...	1.000000	0	0.00	0.500000	0	0	24.500000	7	3.0	0
4	1	1	0	9	1	0	8	1	1	2.000000	...	0.000000	0	0.00	1.000000	0	0	0.000000	1	1.0	0

- **Training the data**

```
x = data.drop(['bug'],axis=1)
y = data['bug']
```

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2,random_state=4)
```

- **LOGISTIC REGRESSION**

```
# LogisticRegression
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()

# train the logistic regression classifier using the training data
logreg.fit(x_train, y_train)

# make predictions on the testing data
y_pred = logreg.predict(x_test)

from sklearn.metrics import accuracy_score

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.90625
/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

- **SVM**

```
# SVM

from sklearn import svm
classifier = svm.SVC(kernel='linear',gamma='auto',C=2)
classifier.fit(x_train,y_train)
y_predict = classifier.predict(x_test)

from sklearn.metrics import accuracy_score

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_predict)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.90625
```

- **NAIVE BAYES**

```
# Naive Bayes
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()

# train the Naive Bayes classifier using the training data
gnb.fit(x_train, y_train)

# make predictions on the testing data
y_pred = gnb.predict(x_test)

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.625
```

- **DECISION TREE**

```
# Decision Tree
from sklearn.tree import DecisionTreeClassifier

# create a Decision Tree classifier object
dtc = DecisionTreeClassifier()

# train the Decision Tree classifier using the training data
dtc.fit(x_train, y_train)

# make predictions on the testing data
y_pred = dtc.predict(x_test)

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.84375
```

- **RANDOM FOREST**

```
# Random Forest
from sklearn.ensemble import RandomForestClassifier


# create a Random Forest classifier object
rfc = RandomForestClassifier(n_estimators=100)

# train the Random Forest classifier using the training data
rfc.fit(x_train, y_train)

# make predictions on the testing data
y_pred = rfc.predict(x_test)

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.90625
```

- **XGBOOST**

```
# XGBoost

import xgboost as xgb

# convert data into DMatrix format
dtrain = xgb.DMatrix(x_train, label=y_train)
dtest = xgb.DMatrix(x_test, label=y_test)

# set the parameters for XGBoost
params = {
    'max_depth': 3,
    'eta': 0.1,
    'objective': 'multi:softmax',
    'num_class': 3
}

# train the XGBoost model
num_rounds = 100
xgb_model = xgb.train(params, dtrain, num_rounds)

# make predictions on the testing data
y_pred = xgb_model.predict(dtest)

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.875
```

- **KNN**

```
# KNN

from sklearn.neighbors import KNeighborsClassifier


# create a KNN classifier object
knn = KNeighborsClassifier(n_neighbors=5)

# train the KNN classifier using the training data
knn.fit(x_train, y_train)

# make predictions on the testing data
y_pred = knn.predict(x_test)

# calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Accuracy: 0.8125
```

- **CATBOOST**

```
!pip install catboost
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting catboost
  Downloading catboost-1.1.1-cp39-none-manylinux1_x86_64.whl (76.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 76.6/76.6 MB 9.2 MB/s eta 0:00:00
Requirement already satisfied: graphviz in /usr/local/lib/python3.9/dist-packages (from catboost) (0.20.1)
Requirement already satisfied: six in /usr/local/lib/python3.9/dist-packages (from catboost) (1.16.0)
Requirement already satisfied: numpy>=1.16.0 in /usr/local/lib/python3.9/dist-packages (from catboost) (1.22.4)
Requirement already satisfied: scipy in /usr/local/lib/python3.9/dist-packages (from catboost) (1.10.1)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.9/dist-packages (from catboost) (3.7.1)
Requirement already satisfied: plotly in /usr/local/lib/python3.9/dist-packages (from catboost) (5.13.1)
Requirement already satisfied: pandas>=0.24.0 in /usr/local/lib/python3.9/dist-packages (from catboost) (1.5.3)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.9/dist-packages (from pandas>=0.24.0->catboost) (2022.7.1)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.9/dist-packages (from pandas>=0.24.0->catboost) (2.8.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (8.4.0)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (23.1)
Requirement already satisfied: importlib-resources>=3.2.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (5.12.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (1.0.7)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (1.4.4)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (4.39.3)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.9/dist-packages (from matplotlib->catboost) (3.0.9)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.9/dist-packages (from plotly->catboost) (8.2.2)
Requirement already satisfied: zipp>=3.1.0 in /usr/local/lib/python3.9/dist-packages (from importlib-resources>=3.2.0->matplotlib->catboost) (3.15.0)
Installing collected packages: catboost
Successfully installed catboost-1.1.1
```

```
# CatBoost
import catboost as ctb
# create a CatBoost classifier object
clf = ctb.CatBoostClassifier()

# train the CatBoost classifier using the training data
clf.fit(x_train, y_train)

# make predictions on the testing data
y_pred = clf.predict(x_test)

# calculate the accuracy of the model
accuracy = clf.score(x_test, y_test)

# print the accuracy of the model
print('Accuracy:', accuracy)
```

```
Learning rate set to 0.004239
0:      learn: 0.6885349        total: 2.84ms   remaining: 2.84s
1:      learn: 0.6831490        total: 4.97ms   remaining: 2.48s
2:      learn: 0.6781967        total: 6.57ms   remaining: 2.18s
3:      learn: 0.6733248        total: 7.88ms   remaining: 1.96s
4:      learn: 0.6682707        total: 9.12ms   remaining: 1.81s
5:      learn: 0.6629646        total: 10.6ms   remaining: 1.75s
6:      learn: 0.6587132        total: 12.1ms   remaining: 1.71s
7:      learn: 0.6542025        total: 13.5ms   remaining: 1.67s
8:      learn: 0.6496763        total: 15.1ms   remaining: 1.66s
9:      learn: 0.6449148        total: 16.5ms   remaining: 1.64s
10:     learn: 0.6400286        total: 17.9ms   remaining: 1.61s
11:     learn: 0.6353350        total: 19.4ms   remaining: 1.59s
12:     learn: 0.6306405        total: 20.9ms   remaining: 1.59s
13:     learn: 0.6262388        total: 22.3ms   remaining: 1.57s
14:     learn: 0.6224154        total: 23.7ms   remaining: 1.56s
15:     learn: 0.6186659        total: 25.2ms   remaining: 1.55s
16:     learn: 0.6147893        total: 26.6ms   remaining: 1.54s
17:     learn: 0.6104366        total: 28ms     remaining: 1.53s
18:     learn: 0.6065882        total: 29.4ms   remaining: 1.52s
982:    learn: 0.0727045        total: 1.52s    remaining: 26.4ms
983:    learn: 0.0726503        total: 1.53s    remaining: 24.9ms
984:    learn: 0.0725035        total: 1.53s    remaining: 23.4ms
985:    learn: 0.0724317        total: 1.54s    remaining: 21.8ms
986:    learn: 0.0723813        total: 1.54s    remaining: 20.3ms
987:    learn: 0.0723228        total: 1.54s    remaining: 18.7ms
988:    learn: 0.0722437        total: 1.54s    remaining: 17.1ms
989:    learn: 0.0721694        total: 1.54s    remaining: 15.6ms
990:    learn: 0.0720889        total: 1.54s    remaining: 14ms
991:    learn: 0.0719472        total: 1.54s    remaining: 12.5ms
992:    learn: 0.0718758        total: 1.55s    remaining: 10.9ms
993:    learn: 0.0717833        total: 1.55s    remaining: 9.35ms
994:    learn: 0.0717054        total: 1.55s    remaining: 7.79ms
995:    learn: 0.0716605        total: 1.55s    remaining: 6.23ms
996:    learn: 0.0716031        total: 1.55s    remaining: 4.67ms
997:    learn: 0.0714455        total: 1.55s    remaining: 3.11ms
998:    learn: 0.0713647        total: 1.55s    remaining: 1.56ms
999:    learn: 0.0712568        total: 1.56s    remaining: 0us
Accuracy: 0.875
```

- **LSTM**

```
# LSTM

from keras.models import Sequential
from keras.layers import LSTM, Dense

# reshape the input data to be 3D for LSTM
x_train = x_train.values.reshape((x_train.shape[0], 1, x_train.shape[1]))
x_test = x_test.values.reshape((x_test.shape[0], 1, x_test.shape[1]))

# define the LSTM model
model = Sequential()
model.add(LSTM(50, input_shape=(x_train.shape[1], x_train.shape[2])))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# train the LSTM model
history = model.fit(x_train, y_train, epochs=50, batch_size=32, validation_data=(x_test, y_test), verbose=2, shuffle=False)

# evaluate the LSTM model
loss, accuracy = model.evaluate(x_test, y_test, verbose=0)
print('Accuracy:', accuracy)
```

```
Epoch 42/50
4/4 - 0s - loss: 0.2119 - accuracy: 0.9040 - val_loss: 0.3795 - val_accuracy: 0.8750 - 69ms/epoch - 17ms/step
Epoch 43/50
4/4 - 0s - loss: 0.2106 - accuracy: 0.9040 - val_loss: 0.3790 - val_accuracy: 0.8750 - 42ms/epoch - 11ms/step
Epoch 44/50
4/4 - 0s - loss: 0.2093 - accuracy: 0.9040 - val_loss: 0.3786 - val_accuracy: 0.8750 - 66ms/epoch - 16ms/step
Epoch 45/50
4/4 - 0s - loss: 0.2081 - accuracy: 0.9040 - val_loss: 0.3782 - val_accuracy: 0.8750 - 58ms/epoch - 15ms/step
Epoch 46/50
4/4 - 0s - loss: 0.2070 - accuracy: 0.9040 - val_loss: 0.3777 - val_accuracy: 0.8750 - 61ms/epoch - 15ms/step
Epoch 47/50
4/4 - 0s - loss: 0.2058 - accuracy: 0.9040 - val_loss: 0.3772 - val_accuracy: 0.8750 - 45ms/epoch - 11ms/step
Epoch 48/50
4/4 - 0s - loss: 0.2046 - accuracy: 0.9040 - val_loss: 0.3768 - val_accuracy: 0.8750 - 56ms/epoch - 14ms/step
Epoch 49/50
4/4 - 0s - loss: 0.2034 - accuracy: 0.9040 - val_loss: 0.3765 - val_accuracy: 0.8750 - 43ms/epoch - 11ms/step
Epoch 50/50
4/4 - 0s - loss: 0.2022 - accuracy: 0.9040 - val_loss: 0.3762 - val_accuracy: 0.8750 - 46ms/epoch - 11ms/step
Accuracy: 0.875
```

- **ANN**

```python
# import pandas as pd
from keras.models import Sequential
from keras.layers import Dense

# load the dataset
# df = pd.read_csv('dataset.csv')

# split the dataset into input and output variables
# X = df.drop(columns=['output'])
# y = df['output']

# define the model
model = Sequential()
model.add(Dense(10, input_dim=x.shape[1], activation='relu'))
model.add(Dense(1, activation='linear'))

# compile the model
model.compile(loss='mse', optimizer='adam', metrics=['mae'])

# train the model
model.fit(x, y, epochs=100, batch_size=32)

# make predictions
y_pred = model.predict(x)

# evaluate the model
mse, mae = model.evaluate(x, y)

print('MSE: ', mse)
print('MAE: ', mae)
```

```
Epoch 89/100
5/5 [==============================] - 0s 5ms/step - loss: 12.3235 - mae: 2.7769
Epoch 90/100
5/5 [==============================] - 0s 3ms/step - loss: 12.2327 - mae: 2.7642
Epoch 91/100
5/5 [==============================] - 0s 3ms/step - loss: 12.1032 - mae: 2.7458
Epoch 92/100
5/5 [==============================] - 0s 4ms/step - loss: 12.0343 - mae: 2.7368
Epoch 93/100
5/5 [==============================] - 0s 3ms/step - loss: 11.9144 - mae: 2.7218
Epoch 94/100
5/5 [==============================] - 0s 3ms/step - loss: 11.8228 - mae: 2.7077
Epoch 95/100
5/5 [==============================] - 0s 4ms/step - loss: 11.7269 - mae: 2.6924
Epoch 96/100
5/5 [==============================] - 0s 5ms/step - loss: 11.6502 - mae: 2.6799
Epoch 97/100
5/5 [==============================] - 0s 4ms/step - loss: 11.5621 - mae: 2.6692
Epoch 98/100
5/5 [==============================] - 0s 3ms/step - loss: 11.4547 - mae: 2.6568
Epoch 99/100
5/5 [==============================] - 0s 3ms/step - loss: 11.3725 - mae: 2.6450
Epoch 100/100
5/5 [==============================] - 0s 3ms/step - loss: 11.3129 - mae: 2.6351
5/5 [==============================] - 0s 2ms/step
5/5 [==============================] - 0s 3ms/step - loss: 11.2325 - mae: 2.6240
MSE:  11.232542991638184
MAE:  2.6240460872650146
```

Experiment - 06

AIM

Consider the model developed in experiment no. 5 identify:

1. State the hypothesis.
2. Formulate an analysis plan.
3. Analyze the sample data.
4. Interpret results.
5. Estimate type-I and type-II error

INTRODUCTION

State the hypothesis: The hypothesis is a statement or assumption that is being tested using a machine learning model. In machine learning, the hypothesis is usually framed as a predictive model that maps input variables to output variables.

Formulate an analysis plan: The analysis plan outlines the steps that will be taken to test the hypothesis. This includes selecting a suitable machine learning algorithm, collecting and preparing the data, training and testing the model, and evaluating its performance. The plan should also specify any statistical tests or metrics that will be used to assess the model's accuracy.

Analyze the sample data: The sample data is used to train and test the machine learning model. This involves feeding the input variables into the model and comparing the predicted output to the actual output.

Interpret results: The results of the analysis are used to draw conclusions about the hypothesis being tested. If the model performs well on the sample data, it may be considered a good predictor of the outcome variable.

Estimate type-I and type-II error: Type-I error, also known as a false positive, occurs when the model incorrectly predicts a positive outcome when the actual outcome is negative. Type-II error, also known as a false negative, occurs when the model incorrectly predicts a negative outcome when the actual outcome is positive.

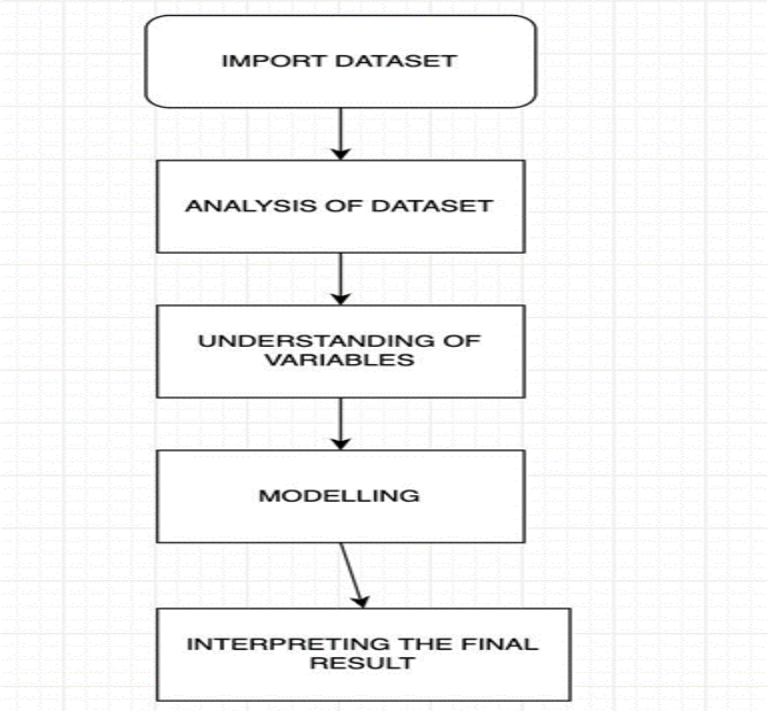
OUTPUT

1. **State the hypothesis.**

- The linguistic and contextual features of news articles can be used to predict whether an article is likely to contain false information.
- Machine learning models trained on this dataset can accurately classify news articles as true or false based on their content and metadata.
- Supervised learning approach that utilizes multiple types of features, such as linguistic features (e.g., sentiment analysis, part-of-speech tagging) and contextual features (e.g., source credibility, temporal and social signals), can lead to an accurate and robust fake news detection system.

2. **Formulate an analysis plan.**

The analysis plan can be described by below flow chart:



1. Importing dataset: The data analysis pipeline begins with the import or creation of a working dataset. The exploratory analysis phase begins immediately after. Importing a dataset is simple with Pandas through functions dedicated to reading the data.

2. Analysis of dataset: An especially important activity in the routine of a data analyst or scientist. It enables an in-depth understanding of the dataset, defines or discards hypotheses and creates predictive models on a solid basis. It uses data manipulation techniques and several statistical tools to describe and understand the relationship between variables and how these can impact business.

3. Understanding the variables: While in the previous point, we are describing the dataset in its entirety, now we try to accurately describe all the variables that interest us. For this reason, this step can also be called bivariate analysis.

4. Modelling: At the end of the process, we will be able to consolidate a business report or continue with the data modelling phase. We would be using Logistic Regression, Decision Tree Classifier, Random Forest Classifier, Gradient Boosting, and Support Vector Machine for modelling the dataset.

5. Interpreting the results: The results of the analysis are used to draw conclusions about the hypothesis being tested. If the model performs well on the sample data, it may be

considered a good predictor of the outcome variable. However, the model's accuracy may need to be validated on new, unseen data to ensure that it is generalizable.

3. Analyze the sample data.

Importing working datasets

```
import pandas as pd
data = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/jm1.csv.xls')
data.head()
```

	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i	e	...	lOCode	lOComment	lOBlank	locCodeAndComment	uniq_Op	uniq_Opnd	total_Op	total_Opnd	branch(
0	1.1	1.4	1.4	1.4	1.3	1.30	1.30	1.30	1.30	1.30	...	2	2	2	2	1.2	1.2	1.2	1.2	
1	1.0	1.0	1.0	1.0	1.0	1.00	1.00	1.00	1.00	1.00	...	1	1	1	1	1	1	1	1	
2	72.0	7.0	1.0	6.0	198.0	1134.13	0.05	20.31	55.85	23029.10	...	51	10	8	1	17	36	112	86	
3	190.0	3.0	1.0	3.0	600.0	4348.76	0.06	17.06	254.87	74202.67	...	129	29	28	2	17	135	329	271	
4	37.0	4.0	1.0	4.0	126.0	599.12	0.06	17.19	34.86	10297.30	...	28	1	6	0	11	16	76	50	

5 rows × 22 columns

Data Cleaning

```
[3] numeric_features = ['share_count', 'reaction_count', 'comment_count']
    categorical_features = ['Category', 'Page', 'Post Type', 'Debate']
    target = data['Rating']
```

Missing values

```
from sklearn.impute import SimpleImputer
```

```
[8] imputer = SimpleImputer(strategy='median')
    numeric_data = imputer.fit_transform(data[numeric_features])
```

```
[9] imputer = SimpleImputer(strategy='most_frequent')
    categorical_data = imputer.fit_transform(data[categorical_features])
```

4. Interpreting the results.

For dataset,

S.no.	Model	Performance
1.	Logistic Regression	**Accuracy = 0.76367**
2.	Decision Tree Classifier	**Accuracy = 0.71553**
3.	Random Forest Classifier	**Accuracy = 0.77242**
4.	Gradient Boosting	**Accuracy = 0.76367**
5.	Support Vector Machine	**Accuracy = 0.739606**

Decision Tree Classifier

```
Accuracy: 0.7155361050328227
Classification report:
                           precision    recall  f1-score   support

mixture of true and false       0.28      0.22      0.24        55
             mostly false       0.26      0.23      0.24        22
              mostly true       0.83      0.86      0.84       333
       no factual content       0.48      0.49      0.48        47

                 accuracy                           0.72       457
                macro avg       0.46      0.45      0.45       457
             weighted avg       0.70      0.72      0.71       457
```

Random Forest Classifier

```
Accuracy: 0.7571115973741794
Classification report:
                           precision    recall  f1-score   support

mixture of true and false       0.35      0.16      0.22        55
             mostly false       0.00      0.00      0.00        22
              mostly true       0.81      0.94      0.87       333
       no factual content       0.66      0.49      0.56        47

                 accuracy                           0.76       457
                macro avg       0.45      0.40      0.41       457
             weighted avg       0.70      0.76      0.72       457
```

Gradient Boosting

```
Accuracy: 0.7724288840262582
Classification report:
                           precision    recall  f1-score   support

mixture of true and false       0.38      0.16      0.23        55
             mostly false       0.43      0.14      0.21        22
              mostly true       0.81      0.94      0.87       333
       no factual content       0.72      0.60      0.65        47

                 accuracy                           0.77       457
                macro avg       0.58      0.46      0.49       457
             weighted avg       0.73      0.77      0.74       457
```

LEARNING

A Type I error is a false positive conclusion, while a Type II error is a false negative conclusion.

EXPERIMENT – 07

AIM

Write a program to implement the t-test.

THEORY

A t-test is a type of inferential statistic used to determine if there is a significant difference between the means of two groups, which may be related to certain features.

There are three types of t-tests, and they are categorized as dependent and independent t-tests.

1. Independent samples t-test: compares the means for two groups.
2. Paired sample t-test: compares means from the same group at different times (say, one year apart).
3. One sample t-test test: the mean of a single group against a known mean.

CODE AND OUTPUTS

1. Importing required Libraries

```
import warnings
warnings.filterwarnings(action='ignore')
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import seaborn as sb
import nltk
nltk.download('all')
import re
import string
import calendar
import wordcloud
from wordcloud import WordCloud, STOPWORDS
import spacy
import catboost as cb
```

2. Loading the dataset

```
df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/jm1.csv.xls')
df.head()
```

	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i	e	...	lOCode	lOComment	lOBlank	locCodeAndComment	uniq_Op	uniq_Opnd	total_Op	total_Opnd
0	1.1	1.4	1.4	1.4	1.3	1.30	1.30	1.30	1.30	1.30	...	2	2	2	2	1.2	1.2	1.2	1.2
1	1.0	1.0	1.0	1.0	1.0	1.00	1.00	1.00	1.00	1.00	...	1	1	1	1	1	1	1	1
2	72.0	7.0	1.0	6.0	198.0	1134.13	0.05	20.31	55.85	23029.10	...	51	10	8	1	17	36	112	86
3	190.0	3.0	1.0	3.0	600.0	4348.76	0.06	17.06	254.87	74202.67	...	129	29	28	2	17	135	329	271
4	37.0	4.0	1.0	4.0	126.0	599.12	0.06	17.19	34.86	10297.30	...	28	1	6	0	11	16	76	50

3. Dataset information

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10885 entries, 0 to 10884
Data columns (total 22 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   loc                 10885 non-null  float64
 1   v(g)                10885 non-null  float64
 2   ev(g)               10885 non-null  float64
 3   iv(g)               10885 non-null  float64
 4   n                   10885 non-null  float64
 5   v                   10885 non-null  float64
 6   l                   10885 non-null  float64
 7   d                   10885 non-null  float64
 8   i                   10885 non-null  float64
 9   e                   10885 non-null  float64
 10  b                   10885 non-null  float64
 11  t                   10885 non-null  float64
 12  lOCode              10885 non-null  int64
 13  lOComment           10885 non-null  int64
 14  lOBlank             10885 non-null  int64
 15  locCodeAndComment   10885 non-null  int64
 16  uniq_Op             10885 non-null  object
 17  uniq_Opnd           10885 non-null  object
 18  total_Op            10885 non-null  object
 19  total_Opnd          10885 non-null  object
 20  branchCount         10885 non-null  object
 21  defects             10885 non-null  bool
dtypes: bool(1), float64(12), int64(4), object(5)
memory usage: 1.8+ MB
```

4. Selecting Features

```
a = df['n']
b = df['v']
```

5. Performing the test

```
t2 = stats.ttest_ind(a, b)
t2
```

```
Ttest_indResult(statistic=-29.853824112647537, pvalue=5.747783245375447e-192)
```

Observation: P value is small (less than 0.05) for all the features. hence null hypothesis is rejected, which implies group mean is not the same for all categories.

Null Hypothesis: The difference in mean values of title length of fake news and title length of real news is 0.

Alternate Hypothesis: The difference in mean values of the title length of fake news and the title length of real news is not 0.

OBSERVATION

We observe a statistically significant difference (p-value = 0.01583) between the length of news titles of real and fake news. The title length of fake news is slightly larger than that of real news. Fake news title length distribution is cantered with a mean of 7.83, while the centre of distribution of title length of real news is slightly skewed towards the right with a mean of 7.02. The t-test gives us evidence that the length of a real news title is significantly shorter than the a fake news title.

LEARNING

Key learnings for implementing the T-Test in a program include understanding its applications in statistical hypothesis testing, considering assumptions such as normality and homogeneity of variances, implementing the T-Test in a programming language or statistical software, interpreting results including p-values and confidence intervals, and considering sample size, power analysis, and effect size for appropriate interpretation and decision-making.

EXPERIMENT – 08

AIM

Write a program to implement the chi-square test.

THEORY

One of the primary tasks involved in any supervised Machine Learning venture is to select the best features from the given dataset to obtain the best results. One way to select these features is the Chi-Square Test. Mathematically, a Chi-Square test is done on two distributions two determine the level of similarity of their respective variances. In its null hypothesis, it assumes that the given distributions are independent. This test thus can be used to determine the best features for a given dataset by determining the features on which the output class label is most dependent.
It involves the use of a contingency table. A Contingency table (also called crosstab) is used in statistics to summarise the relationship between several categorical variables.

CODE AND OUTPUTS

- Null Hypothesis: There is no relation between News_Type and Article_Subject
- Alternate Hypothesis: There is a significant relation between News_Type and Article_Subject

1. Importing required Libraries

```
import warnings
warnings.filterwarnings(action='ignore')
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import seaborn as sb
import nltk
nltk.download('all')
import re
import string
import calendar
import wordcloud
from wordcloud import WordCloud, STOPWORDS
import spacy
import catboost as cb
```

2. Loading the dataset

```
df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/jm1.csv.xls')
df.head()
```

	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i	e	...	lOCode	lOComment	lOBlank	locCodeAndComment	uniq_Op	uniq_Opnd	total_Op	total_Opnd
0	1.1	1.4	1.4	1.4	1.3	1.30	1.30	1.30	1.30	1.30	...	2	2	2	2	1.2	1.2	1.2	1.2
1	1.0	1.0	1.0	1.0	1.0	1.00	1.00	1.00	1.00	1.00	...	1	1	1	1	1	1	1	1
2	72.0	7.0	1.0	6.0	198.0	1134.13	0.05	20.31	55.85	23029.10	...	51	10	8	1	17	36	112	86
3	190.0	3.0	1.0	3.0	600.0	4348.76	0.06	17.06	254.87	74202.67	...	129	29	28	2	17	135	329	271
4	37.0	4.0	1.0	4.0	126.0	599.12	0.06	17.19	34.86	10297.30	...	28	1	6	0	11	16	76	50

5 rows × 22 columns

3. Information about the dataset

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10885 entries, 0 to 10884
Data columns (total 22 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   loc                 10885 non-null  float64
 1   v(g)                10885 non-null  float64
 2   ev(g)               10885 non-null  float64
 3   iv(g)               10885 non-null  float64
 4   n                   10885 non-null  float64
 5   v                   10885 non-null  float64
 6   l                   10885 non-null  float64
 7   d                   10885 non-null  float64
 8   i                   10885 non-null  float64
 9   e                   10885 non-null  float64
 10  b                   10885 non-null  float64
 11  t                   10885 non-null  float64
 12  lOCode              10885 non-null  int64
 13  lOComment           10885 non-null  int64
 14  lOBlank             10885 non-null  int64
 15  locCodeAndComment   10885 non-null  int64
 16  uniq_Op             10885 non-null  object
 17  uniq_Opnd           10885 non-null  object
 18  total_Op            10885 non-null  object
 19  total_Opnd          10885 non-null  object
 20  branchCount         10885 non-null  object
 21  defects             10885 non-null  bool
dtypes: bool(1), float64(12), int64(4), object(5)
memory usage: 1.8+ MB
```

4. Performing Chi-square test

```
import pandas as pd
from scipy.stats import chi2_contingency

# Load the dataset from a CSV file
data = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/jm1.csv.xls')

# Create a contingency table from the data
contingency_table = pd.crosstab(data['n'], data['v'])

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

# Print the results
print('Chi-square statistic:', chi2_stat)
print('Degrees of freedom:', dof)
print('p-value:', p_value)
print('Expected values:', expected)
```

```
Chi-square statistic: 8722817.213051744
Degrees of freedom: 3211950
p-value: 0.0
Expected values: [[1.65936151e+02 1.22462104e-01 1.22462104e-01 ... 1.22462104e-01
  1.22462104e-01 1.22462104e-01]
 [2.86311438e+00 2.11299954e-03 2.11299954e-03 ... 2.11299954e-03
  2.11299954e-03 2.11299954e-03]
 [1.24483234e-01 9.18695452e-05 9.18695452e-05 ... 9.18695452e-05
  9.18695452e-05 9.18695452e-05]
 ...
 [1.24483234e-01 9.18695452e-05 9.18695452e-05 ... 9.18695452e-05
  9.18695452e-05 9.18695452e-05]
 [1.24483234e-01 9.18695452e-05 9.18695452e-05 ... 9.18695452e-05
  9.18695452e-05 9.18695452e-05]
 [1.24483234e-01 9.18695452e-05 9.18695452e-05 ... 9.18695452e-05
  9.18695452e-05 9.18695452e-05]]
```

LEARNING

Key learnings for implementing the Chi-Square test in a program include understanding its applications in analyzing categorical data, familiarity with different types of Chi-Square tests, implementation in a programming language or statistical software, interpretation of results including chi-square statistic, degrees of freedom, and p-values, and consideration of limitations and assumptions for appropriate application and interpretation of the Chi-Square test.

EXPERIMENT – 09

AIM

Write a program to implement the Friedman test.

THEORY

Friedman Test is a non-parametric test alternative to the one-way ANOVA with repeated measures. It tries to determine if subjects changed significantly across occasions/conditions. For example:- Problem-solving ability of a set of people is the same or different in the Morning, Afternoon, and Evening.

CODE AND OUTPUTS

- Null Hypothesis: There is no significant difference in the score values
- Alternate Hypothesis: At least 2 values differ from one another.

1. Importing required Libraries

```
import warnings
warnings.filterwarnings(action='ignore')
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import seaborn as sb
import nltk
nltk.download('all')
import re
import string
import calendar
import wordcloud
from wordcloud import WordCloud, STOPWORDS
import spacy
import catboost as cb
```

2. Loading the dataset

```
df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/jm1.csv.xls')
df.head()
```

	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i	e	...	lOCode	lOComment	lOBlank	locCodeAndComment	uniq_Op	uniq_Opnd	total_Op	total_Opnd
0	1.1	1.4	1.4	1.4	1.3	1.30	1.30	1.30	1.30	1.30	...	2	2	2	2	1.2	1.2	1.2	1.2
1	1.0	1.0	1.0	1.0	1.0	1.00	1.00	1.00	1.00	1.00	...	1	1	1	1	1	1	1	1
2	72.0	7.0	1.0	6.0	198.0	1134.13	0.05	20.31	55.85	23029.10	...	51	10	8	1	17	36	112	86
3	190.0	3.0	1.0	3.0	600.0	4348.76	0.06	17.06	254.87	74202.67	...	129	29	28	2	17	135	329	271
4	37.0	4.0	1.0	4.0	126.0	599.12	0.06	17.19	34.86	10297.30	...	28	1	6	0	11	16	76	50

5 rows × 22 columns

3. Information about the dataset

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10885 entries, 0 to 10884
Data columns (total 22 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   loc                 10885 non-null  float64
 1   v(g)                10885 non-null  float64
 2   ev(g)               10885 non-null  float64
 3   iv(g)               10885 non-null  float64
 4   n                   10885 non-null  float64
 5   v                   10885 non-null  float64
 6   l                   10885 non-null  float64
 7   d                   10885 non-null  float64
 8   i                   10885 non-null  float64
 9   e                   10885 non-null  float64
 10  b                   10885 non-null  float64
 11  t                   10885 non-null  float64
 12  lOCode              10885 non-null  int64  
 13  lOComment           10885 non-null  int64  
 14  lOBlank             10885 non-null  int64  
 15  locCodeAndComment   10885 non-null  int64  
 16  uniq_Op             10885 non-null  object 
 17  uniq_Opnd           10885 non-null  object 
 18  total_Op            10885 non-null  object 
 19  total_Opnd          10885 non-null  object 
 20  branchCount         10885 non-null  object 
 21  defects             10885 non-null  bool   
dtypes: bool(1), float64(12), int64(4), object(5)
memory usage: 1.8+ MB
```

4. Friedman Test

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

# Load the dataset from a CSV file
data = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/jm1.csv.xls')

# Perform the Friedman test
friedman_stat, p_value = friedmanchisquare(data['n'], data['v'], data['t'])

# Print the results
print('Friedman statistic:', friedman_stat)
print('p-value:', p_value)
```

```
Friedman statistic: 10372.614517524318
p-value: 0.0
```

LEARNING

Key learnings for implementing the Friedman test in a program include understanding its applications in non-parametric statistical analysis, familiarity with assumptions and requirements such as repeated measures and ranked data, implementation in a programming language or statistical software, interpretation of results including Friedman statistic, degrees of freedom, and p-values, and consideration of appropriate use and limitations of the Friedman test for comparison of multiple related samples and valid interpretation of results.

EXPERIMENT – 10

AIM

Write a program to implement Wilcoxon Signed Rank Test.

THEORY

Wilcoxon signed-rank test, also known as Wilcoxon matched pair test is a non-parametric hypothesis test that compares the median of two paired groups and tells if they are identically distributed or not.

We can use this when:

- Differences between the pairs of data are non-normally distributed.
- Independent pairs of data are identical. (or matched)

CODE AND OUTPUTS

- Null Hypothesis: The groups - title length of fake news and title length of real news are identically distributed.
- Alternate Hypothesis: The groups - title length of fake news and title length of real news are not identically distributed.

1. Importing required Libraries

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
plt.style.use('default')
```

2. Loading the dataset

```
df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/jm1.csv.xls')
df.head()
```

	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i	e	...	lOCode	lOComment	lOBlank	locCodeAndComment	uniq_Op	uniq_Opnd	total_Op	total_Opnd
0	1.1	1.4	1.4	1.4	1.3	1.30	1.30	1.30	1.30	1.30	...	2	2	2	2	1.2	1.2	1.2	1.2
1	1.0	1.0	1.0	1.0	1.0	1.00	1.00	1.00	1.00	1.00	...	1	1	1	1	1	1	1	1
2	72.0	7.0	1.0	6.0	198.0	1134.13	0.05	20.31	55.85	23029.10	...	51	10	8	1	17	36	112	86
3	190.0	3.0	1.0	3.0	600.0	4348.76	0.06	17.06	254.87	74202.67	...	129	29	28	2	17	135	329	271
4	37.0	4.0	1.0	4.0	126.0	599.12	0.06	17.19	34.86	10297.30	...	28	1	6	0	11	16	76	50

5 rows × 22 columns

3. Information about the dataset

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10885 entries, 0 to 10884
Data columns (total 22 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   loc                 10885 non-null  float64
 1   v(g)                10885 non-null  float64
 2   ev(g)               10885 non-null  float64
 3   iv(g)               10885 non-null  float64
 4   n                   10885 non-null  float64
 5   v                   10885 non-null  float64
 6   l                   10885 non-null  float64
 7   d                   10885 non-null  float64
 8   i                   10885 non-null  float64
 9   e                   10885 non-null  float64
 10  b                   10885 non-null  float64
 11  t                   10885 non-null  float64
 12  lOCode              10885 non-null  int64  
 13  lOComment           10885 non-null  int64  
 14  lOBlank             10885 non-null  int64  
 15  locCodeAndComment   10885 non-null  int64  
 16  uniq_Op             10885 non-null  object 
 17  uniq_Opnd           10885 non-null  object 
 18  total_Op            10885 non-null  object 
 19  total_Opnd          10885 non-null  object 
 20  branchCount         10885 non-null  object 
 21  defects             10885 non-null  bool   
dtypes: bool(1), float64(12), int64(4), object(5)
memory usage: 1.8+ MB
```

4. Wilcoxon Signed Rank Test

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

# Load the dataset from a CSV file
data = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/jm1.csv.xls')

# Perform the Wilcoxon signed-rank test
stat, p_value = wilcoxon(data['n'], data['v'])

# Print the results
print('Wilcoxon signed-rank statistic:', stat)
print('p-value:', p_value)
```

```
Wilcoxon signed-rank statistic: 253.0
p-value: 0.0
```

LEARNING

Key learnings for implementing the Wilcoxon Signed Rank test in a program include understanding its applications in non-parametric statistical analysis, familiarity with assumptions and requirements such as paired data and ordinal or continuous variables, implementation in a programming language or statistical software, interpretation of results including test statistic, p-values, and confidence intervals, and consideration of appropriate use and limitations of the Wilcoxon Signed Rank test for comparing paired data and valid interpretation of results.

EXPERIMENT – 11

AIM

Write a program to implement the Nemenyi test.

THEORY

The Friedman Test is used to find whether there exists a significant difference between the means of more than two groups. In such groups, the same subjects show up in each group. If the p-value of the Friedman test turns out to be statistically significant then we can conduct the Nemenyi test to find exactly which groups are different. This test is also known as Nemenyi posthoc test.

CODE AND OUTPUTS

- Null Hypothesis: There is no significant difference in the score values
- Alternate Hypothesis: At least 2 values differ from one another.

1. Importing required Libraries

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
plt.style.use('default')
```

2. Loading the dataset

```
df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/jm1.csv.xls')
df.head()
```

3. Information about the dataset

	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i	e	...	lOCode	lOComment	lOBlank	locCodeAndComment	uniq_Op	uniq_Opnd	total_Op	total_Opnd
0	1.1	1.4	1.4	1.4	1.3	1.30	1.30	1.30	1.30	1.30	...	2	2	2	2	1.2	1.2	1.2	1.2
1	1.0	1.0	1.0	1.0	1.0	1.00	1.00	1.00	1.00	1.00	...	1	1	1	1	1	1	1	1
2	72.0	7.0	1.0	6.0	198.0	1134.13	0.05	20.31	55.85	23029.10	...	51	10	8	1	17	36	112	86
3	190.0	3.0	1.0	3.0	600.0	4348.76	0.06	17.06	254.87	74202.67	...	129	29	28	2	17	135	329	271
4	37.0	4.0	1.0	4.0	126.0	599.12	0.06	17.19	34.86	10297.30	...	28	1	6	0	11	16	76	50

4. Friedman Test

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

# Load the dataset from a CSV file
data = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/jm1.csv.xls')

# Perform the Friedman test
friedman_stat, p_value = friedmanchisquare(data['n'], data['v'], data['t'])

# Print the results
print('Friedman statistic:', friedman_stat)
print('p-value:', p_value)
```

```
Friedman statistic: 10372.614517524318
p-value: 0.0
```

5. Nemenyi Test

```
d=np.array([sample1, sample2, sample3])
sp.posthoc_nemenyi_friedman(d.T)
```

	0	1	2
0	1.000000	0.809299	0.746632
1	0.809299	1.000000	0.409304
2	0.746632	0.409304	1.000000

OBSERVATION

- From the outputs received, we reject the Null hypothesis
- From the output table we can clearly conclude that the 2 groups to have statistically significantly different means are Group 1 and Group 2.

LEARNING

Key learnings for implementing the Nemenyi test in a program include understanding its applications in posthoc analysis of multiple comparison tests, familiarity with requirements and assumptions such as ranked or continuous data and multiple group comparisons, implementation in a programming language or statistical software, interpretation of results including critical difference values and significance levels, and consideration of appropriate use and limitations of the Nemenyi test for posthoc analysis and valid interpretation of results in the context of statistical hypothesis testing.

EXPERIMENT – 12

Objective

Conduct ANNOVA test

Theory

ANOVA stands for ANalysis Of VAriance. It is an excellent statistical test to compare group means across different levels of one categorical variable. In case there are just two levels of a categorical variable then two sample t-test can be used, but in case number of levels in categorical variable increases from two then ANOVA is preferred approach. Alternatively, one can run two-sample t-test for all possible combinition of groups but that is a cumbersome task to execute.

Here we will explore one-way ANOVA only. One way ANOVA compares three or more levels for a categorical variable to establish if there is difference between sample means or not. Sample means will always be different but is the difference statstically significant or not? This is the main goal of ANOVA. It is called one-way ANOVA because groups are under one categorical variable. If there are two factors then it is called two-way ANOVA or ANCOVA if one variable is continious.

Assumptions for running ANOVA tests are:

- Normal distribution of populations
- Equal variances
- Samples are independent

Here null hypothesis for one-way ANOVA is that group means are same and there is no statistical differences between group means:

Code and Output

1.Dataframe overview

```
]:
df_ameshousing=pd.read_csv('../input/ames-housing-dataset/AmesHousing.csv')
```

```
]:
df_ameshousing.head()
```

]:

	Order	PID	MS SubClass	MS Zoning	Lot Frontage	Lot Area	Street	Alley	Lot Shape	Land Contour	...	Pool Area	Pool QC	Fence	Misc Feature	Misc Val	Mo Sold	Yr Sold	Sale Type	Sale Conditio
0	1	526301100	20	RL	141.0	31770	Pave	NaN	IR1	Lvl	...	0	NaN	NaN	NaN	0	5	2010	WD	Normal
1	2	526350040	20	RH	80.0	11622	Pave	NaN	Reg	Lvl	...	0	NaN	MnPrv	NaN	0	6	2010	WD	Normal
2	3	526351010	20	RL	81.0	14267	Pave	NaN	IR1	Lvl	...	0	NaN	NaN	Gar2	12500	6	2010	WD	Normal
3	4	526353030	20	RL	93.0	11160	Pave	NaN	Reg	Lvl	...	0	NaN	NaN	NaN	0	4	2010	WD	Normal
4	5	527105010	60	RL	74.0	13830	Pave	NaN	IR1	Lvl	...	0	NaN	MnPrv	NaN	0	3	2010	WD	Normal

```
df_ameshousing['Heating'].unique()
```

```
array(['GasA', 'GasW', 'Grav', 'Wall', 'Floor', 'OthW'], dtype=object)
```

Null hypothesis for this test is that variance for all groups is same.

```
Stat,p = f_oneway( df_ameshousing['SalePrice_log_t'][df_ameshousing['Heating'] == 'GasA'],
     df_ameshousing['SalePrice_log_t'][df_ameshousing['Heating'] == 'GasW'],
     df_ameshousing['SalePrice_log_t'][df_ameshousing['Heating'] == 'Grav'],
     df_ameshousing['SalePrice_log_t'][df_ameshousing['Heating'] == 'Wall'],
     df_ameshousing['SalePrice_log_t'][df_ameshousing['Heating'] == 'Floor'],
     df_ameshousing['SalePrice_log_t'][df_ameshousing['Heating'] == 'OthW']
                 )
```

```
print('P-value for one way anova test:',p)
```

```
P-value for one way anova test: 3.9927708306619703e-14
```

Findings and Learnings

As p-value for one-way ANOVA is less than 0.05 so we can conclude that atleast one group mean is different from others. One-way ANOVA only tell us that atleast one group mean is different, it doesn't tell which one is different

EXPERIMENT – 13

AIM

To analyze the performance of the model developed.

THEORY

The performance analysis of a model for software defect prediction encompasses several crucial steps to ensure its effectiveness. Beginning with meticulous data preparation, including dataset splitting for training and testing, the process progresses to selecting suitable machine learning algorithms and relevant features. Model training involves optimization through hyperparameter tuning while monitoring for overfitting. Performance evaluation entails assessing metrics such as accuracy, precision, recall, and F1-score, along with techniques like ROC analysis and confusion matrices. Error analysis aids in understanding misclassifications, while cross-validation ensures consistency and generalization. Comparison with baselines provides context, and interpretability enhances the model's utility. Documenting the entire process comprehensively facilitates clear communication of results and insights, empowering informed decision-making in software quality assurance endeavors**.**

CODE AND OUTPUTS(considering Naïve Bayes Model)

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, random_state = 0)

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

#Summary of the predictions made by the classifier
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
#Accuracy score
from sklearn.metrics import accuracy_score
print("ACC: ",accuracy_score(y_pred,y_test))
```

```
                precision    recall  f1-score   support

     Redesign       0.93      0.94      0.94       319
    Succesful       0.99      0.99      0.99      1858

    micro avg       0.98      0.98      0.98      2177
    macro avg       0.96      0.97      0.96      2177
 weighted avg       0.98      0.98      0.98      2177

[[ 301   18]
 [  22 1836]]
ACC:  0.9816260909508497
```

Learnings: The overall accuray of the Naïve Bayes model for cross-project defect prediction was 98 percent with recall and f1 scores as shown above.