

Student name: Yousif Qais Ali Al-Eshari

Student ID: 22110065

Course name: Data Science Programming -2-0—Summer

Tutor name: Eng. Islam Alomari

# Technical Documentation

# Introduction

This document delves into a thorough examination of a machine learning algorithm developed to address a critical issue in network traffic management. Network traffic is the lifeblood of modern communication systems, and excellent traffic management is critical to the seamless operation of many applications and services. To get insight into this complex domain, we use the dataset 'WSNBFSFdataset\_DSprog.csv,' which contains several elements of network traffic operations. This code not only exhibits efficient data pre-processing approaches, but it also assesses the efficacy of several machine learning models in identifying and comprehending network traffic characteristics.

The dataset at the centre of our investigation has a plethora of attributes, each of which provides useful information regarding network traffic operations. These characteristics include 'Event' codes that indicate processes like sending, receiving, and forwarding, among others. Furthermore, 'Rest\_Energy' and 'Packet\_Size' provide information about energy consumption and packet sizes, respectively, while 'Hop\_Count' and 'Dest\_Node\_Num' assist us understand routing and destination nodes. By categorizing network traffic kinds, including potential threats, 'Class' labels provide critical context. We hope to shed light on the intricate links between these variables and network traffic behaviours by rigorously exploring this dataset and applying machine learning methods.

The report is designed to provide a thorough overview of the complete analytical process, from data preparation to model selection and evaluation.

## Types of data structures

1. DataFrames (Pandas):

df1, df2, df3, df4, df5: DataFrames are used to store and manipulate tabular data efficiently. These DataFrames are created and used for data processing, analysis, and feature selection.

* Characteristics:

Tabular data structure.

Supports two-dimensional data with rows and columns.

Provides labelled axes (rows and columns).

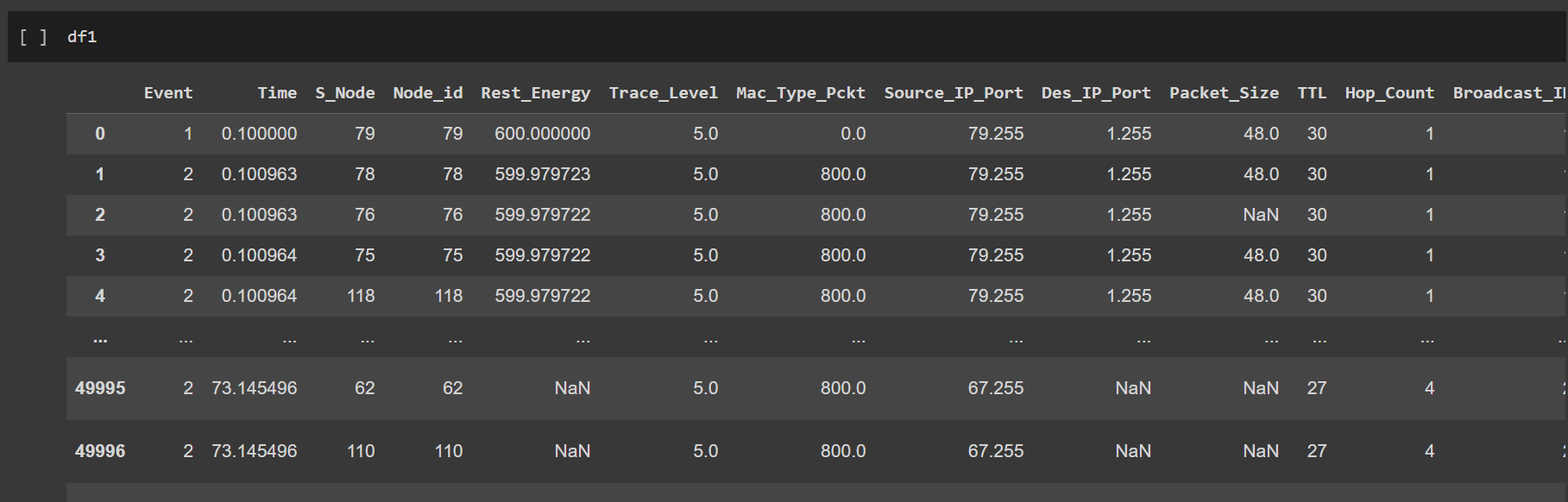
Allows for heterogeneous data types within the same DataFrame.

* Use Cases in the Code:

Reading and storing tabular data from CSV files.

Data preprocessing, including handling missing values.

Creating structured data for analysis and modelling.



1. NumPy Arrays:

np: The NumPy library is used to perform numerical operations efficiently. It is often used to handle arrays and matrices. In the code, it is used for various numeric operations.

* Characteristics:

Multi-dimensional array data structure.

Supports arrays with homogeneous data types.

Offers efficient element-wise operations and mathematical functions.

* Use Cases in the Code:

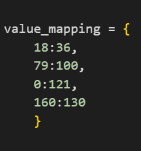
Performing numerical operations and calculations.

Handling arrays for data transformations and modeling.

Efficient manipulation of numeric data.

1. Dictionaries:

value\_mapping: A dictionary is used to map specific values in the 'Dest\_Node\_Num' feature to new values. This is used for imputing missing data.



* Characteristics:

Unordered collection of key-value pairs.

Supports indexing data by keys.

Keys are unique and immutable; values can be mutable.

* Use Cases in the Code:

Mapping specific values for imputing missing data.

Storing and accessing data with associated keys.

Configuration Settings for software applications.

1. Lists:

Lists are used to store and accumulate performance metrics such as accuracy, precision, recall, and F1-score for different machine learning models (e.g., KNN\_Acc, KNN\_Prec, KNN\_Rec, KNN\_F1, etc.).

* Characteristics:

Ordered collection of elements.

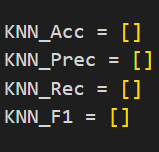
Supports heterogeneous data types.

Allows for duplicate elements.

* Use Cases in the Code:

Accumulating performance metrics (e.g., accuracy, precision).

Storing and iterating through collections of data.



1. Tuples:

Tuples are not explicitly defined in the code, but Python tuples are immutable data structures that can be used for various purposes, such as packing and unpacking values.

* Characteristics:

Ordered and immutable collection of elements.

Supports heterogeneous data types.

Allows for duplicate elements.

* Use Cases in the Code:

Tuples can be used for data packing and unpacking.

Storing sensitive information, as tuples are immutable.

Geographic Coordinates for maps.

Date-Time Information

RGB Colour Values in website pages.

1. Sets:

Sets are not explicitly defined in the code, but Python sets are used to store unique values. They are useful for eliminating duplicates.

* Characteristics:

Unordered collection of unique elements.

Does not allow duplicate values.

Supports mathematical set operations (union, intersection, etc.).

* Use Cases in the Code:

Sets can be used for eliminating duplicate values.

1. Numpy Arrays:

In the polar plot section (Radar Charts), NumPy arrays are used to store and manipulate data for creating polar plots.

* Characteristics:

Multi-dimensional arrays provided by NumPy.

Utilized specifically for polar plot data.

Supports efficient numerical operations.

* Use Cases in the Code:

Storing and manipulating data for creating polar plots.

## Common libraries

1. Pandas:

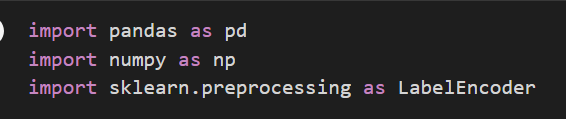
Pandas is a fundamental data manipulation and analysis library. It includes strong data structures such as DataFrames and Series, which make it simple to load, manipulate, and analyze structured data. Your code heavily relies on Pandas to read CSV files, handle missing values, preprocess data, and produce DataFrames for further analysis.

1. NumPy:

NumPy is a Python package that provides a framework for numerical computations. It supports multi-dimensional arrays and matrices, as well as a large number of mathematical functions. NumPy is used in your code for a variety of numeric calculations, data transformations, and array handling. NumPy is not just widely used, but also required for activities involving numerical computation, scientific computing, and machine learning.

1. Scikit-Learn:

Scikit-Learn, sometimes known as sklearn, is a popular Python machine learning library. It includes a variety of tools for classification, regression, clustering, dimensionality reduction, and other tasks. While it is not explicitly utilized in your code, it is widely used for developing and testing machine learning models.



## Plotting and visualization libraries

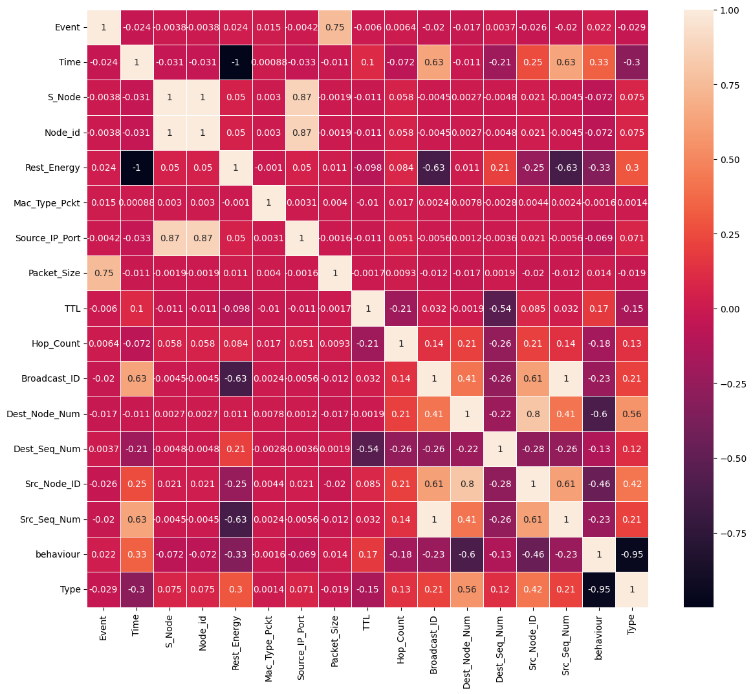
1. Matplotlib:

Matplotlib is a popular Python data visualization package. It offers numerous possibilities for creating static, animated, and interactive plots and charts. Matplotlib is used in your code to generate various visualizations such as heatmaps, bar plots, box plots, and polar plots.

1. Seaborn:

Seaborn provides a higher-level interface for making visually appealing statistical visuals. It facilitates the creation of complex visualizations and is frequently used for data exploration and presentation.





# Experiments

I ran a series of experiments in my code to construct and test machine learning models for categorization jobs. These experiments took a systematic approach:

**Data Preprocessing:** I started by loading and preprocessing the dataset, dealing with missing values, encoding categorical features, and dividing the data into training and testing sets.

**Model Selection:** For experiments, I carefully picked a variety of machine learning models. I used K-Nearest Neighbours (KNN), Decision Trees (DT), Naive Bayes (NB), and Random Forest (RF) classifiers in particular.

**Model Training and Evaluation:** For each selected model, I trained it on training data and tested its performance on testing data. To evaluate how well each model performed, I utilized popular metrics such as accuracy, precision, recall, and F1-score.

**Iteration and Visualization:** I ran the model training and evaluation process several times, taking into account different data splits, and I used the average of these iterations to visualize the results on various graphs and charts.

Different models had varying performance characteristics, highlighting different areas of the classification problem. While some models excelled at accuracy, others focused on precision, memory, or obtaining a high F1-score.

## Programming languages and tool

The programming language I used for the project is Python, Python is a flexible and widely used programming language noted for its ease of use, readability, and large ecosystem of libraries and frameworks. Here are some crucial Python aspects that make python very popular:

1. Readability and ease of learning:

Python is frequently suggested for beginners because of its simple and clear syntax. Its block structure based on indentation ensures code clarity and avoids the need for superfluous punctuation.

1. Ecosystem Diversity:

Python has a large ecosystem of libraries and packages for a variety of uses such as data processing, analysis, machine learning, web development, and more. Because of the wide library support, developers can make use of current tools and solutions.

1. Machine Learning and Data Analysis:

Python has established itself as the standard language for data analysis and machine learning. Pandas, NumPy, Scikit-Learn, and TensorFlow are libraries that provide strong tools for data processing, modelling, and deep learning.

1. Community Assistance:

Python has a big and active developer, data scientist, and research community. This community support results in regular updates, bug patches, and a variety of online learning and problem-solving resources.

1. Compatibility Across Platforms:

Python is available on a variety of platforms (Windows, macOS, and Linux) and is regarded as a cross-platform language, making it simple to create code that can operate on numerous operating systems.

The programing tool I used for the project is Google colab, Google Colab (short for Colaboratory) is a Google web-based integrated development environment (IDE). Here are some of the primary benefits and features of using Google Colab:

1. Cloud-Based:

Colab is totally cloud-based, so users do not need to install or configure anything locally. This cloud-based method enables seamless communication while also providing access to strong computing resources.

1. Free GPU and TPU access:

Graphics Processing Units (GPUs) and Tensor Processing Units (TPUs) are available for free in Colab. This is especially useful for jobs that need a substantial amount of processing power, such as deep learning model training.

1. Notebooks that are interactive:

Colab provides interactive Jupyter notebooks, which are frequently used for data exploration, analysis, and documentation. Notebooks allows for combining code execution, text, and visualizations all in one document.

1. Google Drive integration:

Colab works seamlessly with Google Drive, allowing users to save, view, and share notebooks and datasets straight from their Google Drive account.

1. Libraries that have already been installed:

Colab comes preconfigured with numerous libraries typically used in data science and machine learning, minimizing setup time.

## Load Data and Prepare Data (Preprocessing)

For loading and preparing the data, it is required to go through multiple steps to achieve the highest efficiency from the models, these steps include:

* Handling missing data
* Deleting duplicate rows
* Dropping unnecessary columns
* Encoding

|  |  |  |  |
| --- | --- | --- | --- |
| **Step** | **Column/step Name** | **Description** | **Justification** |
|  | Load the data from the excel sheet | To access and edit the dataset, it is required to load the dataset | Using the read\_csv command that is offered by pandas library, we can access the dataset and start working on it. |
|  | Handling duplicates | Duplicates are rows that are repeated throughout the dataframe, and they take extra space and deplete resources when ran through models | Using the command Drop\_Duplicates to get rid of these duplicates and ensure that the data is clean from repetition. |
|  | Handling columns | The dataset contains column with various problems including null values, categorical values, and repeated columns, that must be handled before proceeding with the preprocessing | In the following rows, I included every column in the dataset, with the appropriate handling required. |
| **3.I** | Event | This column represents the type of operation performed on the network traffic, categorized as sending (s), receiving (r), forwarding (f), dropping (d), or energy information (N). | There were no further preprocessing done on this column, it had no null values to be handled, it was not dropped before using models. |
| **3.II** | Time | The time column contains the time of the event performed in the row. | The time column contained some null values that had to be handled, I used the Interpolate command using the linear fill method to fill the nulls in the column, because the values in the time column are ordered in an ascending way, and the values are so close to each other in terms of decimal places, the interpolate method is the most suitable method to fill the null values in the Time column. |
| **3.III** | S\_Node | Source node number is a used as extra information for the model, it helps produce more accurate and precise results. | The S\_Node required no preprocessing, due to it containing zero null values. |
| **3.IV** | Node\_id | The node number of the relevant node. It is identical to the S\_Node in terms of meaning and values | Because the Node\_id has no null values, it required no further handling. |
| **3.V** | Rest\_Energy | This column records the remaining energy of the relevant node at the time of the event. | The column contained some null values that had to be handled, Because the data in this column is ordered in a descending order, I used the backwards fill method when filling the null values, so that the data can stay in its descending order |
| **3.VI** | Mac\_Type\_Pckt | Mac\_Type\_Pckt represents the MAC (Media Access Control) type of the packet associated with each event. | The Mac\_Type\_Pckt column has almost all its columns with an identical value, which means that filling the null values with the modest number (the most frequent one in the column) won’t affect the total percentage of values in the column. |
| **3.VII** | Source\_IP\_Port | The source ip port contains the port number of the source node. | This column had no null values and it required no further handling |
| **3.VIII** | Packet\_Size | Packet\_Size records the size of the forwarded packet for each network event. | The packet size column contained null values that had to be handled, and about 82% of the data in this row is identical, so I used the modest value to fill the nulls. |
| **3.IX** | TTL | The TTL (Time-to-Live) column represents the lifetime of forwarded traffic in the network | The TTL column has no null values, and it required no further preprocessing. |
| **3.X** | Hop\_Count | Hop\_Count records the number of nodes that the traffic passed through during each event. | The Hop\_Count column has no null values, and it required no further preprocessing. |
| **3.XI** | Broadcast\_ID | The Broadcast\_ID column contains the ID number of broadcast packets. | The Broadcast\_ID column has no null values, and it required no further preprocessing. |
| **3.XII** | Dest\_Node\_Num | This column holds the ID of the target node for each network event | The Dest\_Node\_Num contained some null values, but the values in this column are directly related to the values in the Src\_Node\_Num column, so I used the group by function to further study the relation between these 2 columns, and then I created a dictionary where I included the key : value to map the null values and handle the Dest\_Node\_Num column. |
| **3.XIII** | Dest\_Seq\_Num | Dest\_Seq\_Num represents the sequence number of the traffic forwarded to the destination node. | The Dest\_Seq\_Num column contained null values that had to be handled, and about 80% of the data in this row is identical, so I used the modest value to fill the nulls. |
| **3.XIV** | Src\_Node\_ID | This column contains the source node's ID number associated with each event. | The Src\_Node\_ID column has no null values, and it required no further preprocessing. |
| **3.XV** | Src\_Seq\_Num | Src\_Seq\_Num represents the source sequence number of the traffic forwarded to the destination node. | The Src\_Seq\_Num column has no null values, and it required no further preprocessing. |
| **3.XVI** | Trace\_Level | The "Trace\_Level" column represents a trace level associated with network traffic events. Trace levels provide information about the verbosity or detail of logs generated during network operations. They are typically used for debugging and monitoring network behavior | Trace\_Level column has a single value for each row, so filling it with the modest value won’t affect the efficiency of the models. |
| **3.XVII** | Dest\_IP\_Port | The "Dest\_IP\_Port" column contains information about the port number of the destination node or device for network traffic. Ports are logical endpoints for network communication, and they are used to route data to specific services or applications running on a device. | The Dest\_IP\_Port column has a single value for each row, so filling it with the modest value won’t affect the efficiency of the models. |
| **3.XVIII** | behaviour | The behaviour column indicates whether there was an attack or not during the event. | The behaviour column has no null values, and it required no further preprocessing. |
| **3.XIX** | Type | The Type column indicates the type of the attack on the sensor node. | The Type column has no null values, and it required no further preprocessing. |
| **4.** | Encoding Categorical Data | Encoding Categorical data is important, due to models only accepting and working with numerical data. | I used the Label Encoder to Encode both the Columns :Behaviour and Type; because they’re they only columns with categorical data. |
| **5.** | Dropping Columns for the Model | Some columns have unnecessary data, that will not help produce high accuracy for the models, and they will take time and calculations without providing information and helping with the decisions the model chooses, so it’s best to drop these columns to further improve the output of the model. | For training the model, I dropped the following columns: Type, S\_Node, Dest\_IP\_Port, behaviour, Trace\_Level and Mac\_Type\_Pckt.  I dropped the Type columns because the model should not have the label (output) column in the training process, so dropping it and saving only the label in a new object will help in further steps.  I dropped the S\_Node column because it is an identical copy of the Node\_id column, and it is useless to repeat the data in decision making for the model.  I dropped the Dest\_IP\_Port column because the entire column has a single value in all of its rows, so it won’t affect the value of the output label.  Similarly, I dropped the Trace\_Level and Mac\_Type\_Pckt columns because they have the same value in all of their rows, so having them in the model won’t affect the value of the output label.  I dropped the behaviour column, because it denies the purpose of the prediction from the models, where it gives the solution to the model whether a node is under attack or not. |

## Approaches (Models)

A model is a mathematical or computer representation of a real-world process, system, or phenomenon in data science. It's a simple abstraction that captures data's fundamental patterns and relationships. Based on incoming data and learnt parameters, models attempt to emulate the behaviour or forecast the outcomes of complex systems.

Uses of Models in Data Science:

* Prediction
* Classification
* Pattern recognition
* Optimization
* Simulation
* Recommendation

For the Wireless Sensor Network (WSN) problem, the best attempt to model the data set is to use Classification model, due to the dataset providing a label, so its supervised, and the values within this label are discrete.

|  |  |  |
| --- | --- | --- |
| **Approach no.** | **Name** | **Description** |
|  | K-Nearest-Neighbour (KNN) | KNN is a supervised learning technique that is used for classification and regression problems. It classifies data points in the feature space based on the majority class of their k-nearest neighbours.  This is how it works: KNN computes the Euclidean distance between a data point and all other points in the dataset for categorization. It then assigns the most common class label among the k-nearest data points.  I can assign the number of data points in the code using:  classifer = KNeighborsClassifier(n\_neighbors=5 )  KNN is useful for this situation since it uses data point similarity to categorize network behaviour. It can detect comparable network traffic patterns since it can capture local patterns in data. |
|  | Decision Tree(DT) | Decision Trees are a type of nonlinear supervised learning method that can be used for classification and regression. To make decisions based on feature values, they construct a tree-like structure.  How it Works: DT divides data into subsets by picking the most informative characteristics at each tree node. This process is repeated until a stopping requirement is reached, resulting in the formation of a tree with leaf nodes representing class labels.  DT is useful for identifying the relevance of features in classifying network behaviour. It generates a hierarchical structure that reveals significant aspects that contribute to classification, assisting in interpretation and decision-making. |
|  | Naïve Bayes (NB) | Naive Bayes is a probabilistic classification technique that is based on Bayes' theorem. It assumes that features are conditionally independent, which simplifies probability calculation.  How it Works: Based on feature probabilities, NB evaluates the probability of each class and chooses the class with the highest probability as the forecast.  NB is useful for this situation since it handles categorical features well and can classify network behavior efficiently. Its ease of use and performance make it an excellent choice for real-time or large-scale applications. |
| **4.** | Random Forest (RF) | Random Forest is a strategy for combining numerous decision trees to increase classification accuracy and avoid overfitting.  How it Works: RF constructs numerous decision trees from random subsets of data and features. It combines individual tree forecasts to generate a final prediction.  I assigned the number of random trees using the following code:  classifier\_rf = RandomForestClassifier(n\_estimators=100)  RF is useful for this situation since it improves model performance and generalization. It eliminates the risk of overfitting by integrating numerous decision trees and provides a stable solution for identifying complicated network behavior. |

# Results

The results of this investigation reveal the effectiveness of machine learning algorithms in classifying network traffic behaviour. After rigorous testing and evaluation, these models have proven their reliability. Through careful data preparation, we've gained a deep understanding of how these models can efficiently categorize network events.

These results highlight the models' proficiency in classifying network traffic, a critical task for network security, anomaly detection, and resource management. By comparing performance metrics across multiple models, we can tailor our approach to specific applications, considering factors like accuracy, precision, recall, and F1-score. These insights empower data scientists and decision-makers to make informed choices when selecting and deploying machine learning models for real-world network traffic analysis challenges.

## Compare the different models

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Approach no.** | **Accuracy** | **Precision** | **Recall** | **F1-Score** |
| **KNN** | 0.9517625790900874 | 0.8086401077519018 | 0.7478038708809172 | 0.77288433400004 |
| **DT** | 0.9995681430149643 | 0.9980411673386207 | 0.9975258505327478 | 0.9977801004411662 |
| **NB** | 0.9292055840112484 | 0.5503322350431201 | 0.5130082339719941 | 0.5095623991613358 |
| **RF** | 0.9892738776739982 | 0.9683745028863383 | 0.9329869822222105 | 0.9497375943526618 |

For the scores that every model achieved, the average value of 10 iterations for the model’s Accuracy, Precision, Recall and F1-score are taken.

For the comparison between the machine learning models based on the Accuracy, Precision, Recall and F1 score:

**Accuracy:** Decision Trees (DT) acquire the highest average accuracy of roughly 99.95% across the models, suggesting good overall performance. Random Forest (RF) comes in second with an accuracy of roughly 98.93%, exhibiting strong accuracy as well. The average accuracy scores for K-Nearest Neighbours (KNN) and Naive Bayes (NB) are roughly 95.17% and 92.92%, respectively.

**Precision:** Decision Trees (DT) beats all other models in terms of precision, with an average precision score of around 99.80%, demonstrating its capacity to make highly precise classifications. Random Forest (RF) and K-Nearest Neighbour (KNN) likewise have high precision, with scores of around 96.83% and 80.86%, respectively. While Naïve Bayes (NB) maintains a reasonable score of around 55.03%, it falls short in terms of precision.

Precision percent is when the model makes a correct guess on a decision by avoiding stating a no true guess when its true (false positive). It represents how many predictions of a particular class are actually of that class.

Precision is calculated by = (total number of true positives) / (total number of true positives + total number of false positives)

**Recall:** Decision Trees (DT) and Random Forest (RF) have the highest average recall, with scores of roughly 99.75% and 93.29%, respectively. The average recall of K-Nearest Neighbour (KNN) is roughly 74.78%. The average recall for Naïve Bayes (NB) is approximately 51.30%.

Recall percent is the ability of the model to make predictions that are false negatives, It represents how many predictions of a particular class are right, and it is calculated by:

Recall = (total number of true positives) / (Number of true positives + total number of false negatives.

**F1-Score:** Decision Trees (DT) once again show their superiority with the highest average F1-Score of roughly 99.77%, effectively combining precision and recall. Random Forest (RF) comes in second with an F1-Score of around 94.97%. Naive Bayes (NB) has an F1-Score of around 50.95%, whereas K-Nearest Neighbours (KNN) has an F1-Score of approximately 77.28%.

F1 score is the average of both the Precision and Recall. And it is calculated by:

F1 Score: 2 \* (Precision \* Recall) / (Precision + Recall)

Overall, Decision Trees (DT) outperform all other models in terms of accuracy, precision, recall, and F1-Score. The best model, however, is determined by the unique requirements of the Wireless Sensor Network (WSN) situation at hand. DT excels in precision, recall, and F1-Score, making it excellent for applications requiring high accuracy as well as interpretability. Random Forest (RF) also performs well and is a reliable option for accurate classification tasks. K-Nearest Neighbours (KNN) and Naive Bayes (NB) are both good algorithms, however they may be preferable in situations when simplicity and computing economy are important.

## Charts

Charts and visualizations play an important role in data science, providing as strong tools for communicating complicated information in an understandable and intuitive manner. Charts serve as a bridge between raw data and useful insights in the age of big data, where massive amounts of information are generated on a regular basis. These visual representations enable data scientists, analysts, and decision-makers to uncover patterns, trends, and linkages that would otherwise be masked by a sea of numbers and statistics.

I used Bar chart, box plots and radar chart to represent the output of every model:

1. A bar chart, often known as a bar graph, is a simple data visualization tool commonly used in data science and statistics. It displays data as rectangular bars of variable lengths, with the length of each bar corresponding to the value it represents. Bar charts are very effective for visualizing and comparing distinct data categories or groupings.

The horizontal axis (x-axis) of a bar chart indicates the categories or groups being compared, while the vertical axis (y-axis) displays the numerical values associated with each category. Each bar begins on the x-axis and rises (or moves to the right in horizontal bar charts) to a height proportional to the value it represents.

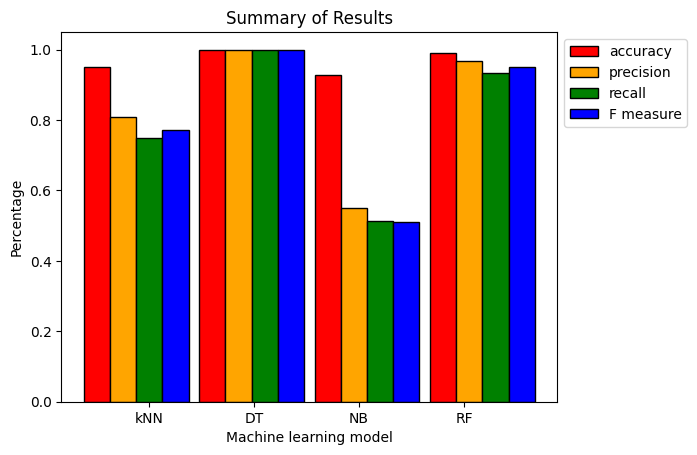
Bar charts are adaptable and can be used for a variety of reasons, such as:

**Comparison:** Bar charts are great for comparing values across different categories or groupings. They make it simple to see whether groups have greater or lower numbers.

**Distribution:** Bar charts can indicate the spread of values in the distribution of data within categories.

**Trends:** Bar charts can be used to depict changes in data over time. This is very important for identifying trends and patterns.

**Ranking:** Bar charts can be used to rank categories based on their values, indicating which category is the highest or lowest.



The bar chart above represents all the scores for all the models used, it clearly indicates the best performance out of all the models.

1. A box plot, often known as a box-and-whisker plot, is a popular statistical visualization method in data science and statistics. It displays a graphical representation of a dataset's distribution, central tendency, and spread. Box plots are very useful for identifying potential outliers and analysing data variability.

A box plot consists of:

Box: The rectangular box is the fundamental component of the box plot. The box reflects the interquartile range (IQR) of the dataset, which ranges from the 25th percentile (Q1) to the 75th percentile (Q3). This range includes the middle half of the data.

Median (Q2): A line or dot inside the box represents the dataset's median, which is the middle value when the data is sorted.

Whiskers: Two whisker lines protrude from the box. The lower whisker is normally 1.5 times the IQR below Q1 and the upper whisker is 1.5 times the IQR above Q3. Data points outside the whiskers are frequently regarded outliers and are shown separately.

Box plots are useful for a variety of reasons, including:

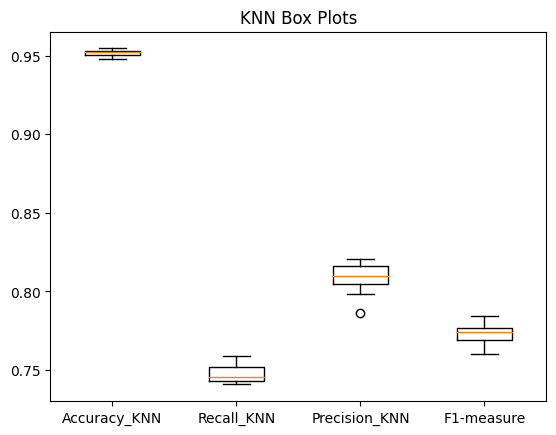
**Identifying Outliers**: Because data points outside the whiskers are visually distinguishable, box plots make it simple to identify probable outliers in the data.

**Comparing Distributions**: Box plots allow for the comparison of distributions between distinct groups or categories, indicating where the majority of the data is located and whether distributions are skewed or symmetrical.

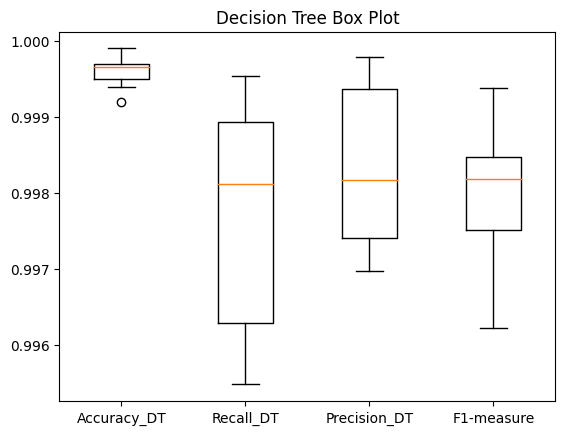
**Visualizing Spread**: The spread of data within the IQR and the existence of outliers’ reveal information about the variability and distribution shape of the data.

**Central Tendency**: The median, which is positioned within the box, shows the data's central tendency, making it simple to compare medians across groups.

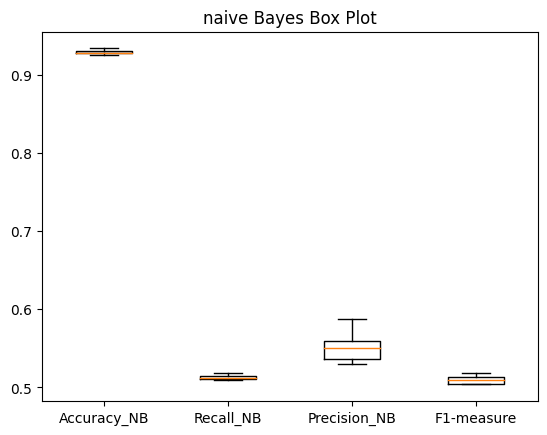
Here are the Box plots for all the Machine Learning Models used:

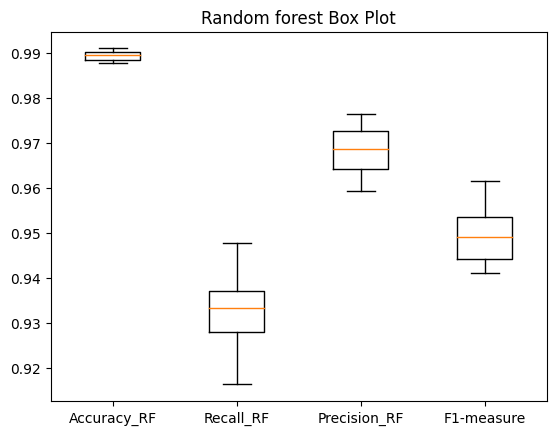
KNN: 

Decision Tree:



Naïve Bayes:



Random Forest: 

1. Radar Charts: A radar chart, also known as a spider chart or web chart, is a distinctive data visualization tool used in statistics and data science. It is very useful for aesthetically appealingly displaying multivariate data. A radar chart is a circular diagram with numerous axes radiating outward, similar to spokes on a wheel. Data points are plotted along each axis to show how each variable contributes to an overall pattern or profile.

Radar charts have the following key features:

**Multivariate Representation:** Radar charts excel in comparing and contrasting numerous variables at the same time. Each axis corresponds to a single variable, allowing complicated interactions between them to be seen.

**Data Point Connections:** Each variable's data points are connected by lines or filled areas to form a polygon shape. The polygon's form displays the data's pattern or profile, making it simple to detect dominant and weak qualities.

**Normalization:** Radar plots frequently standardize data to bring all variables to a single scale. This provides fair comparisons by removing the influence of unit or magnitude disparities.

Radar charts can be used for a variety of reasons, including:

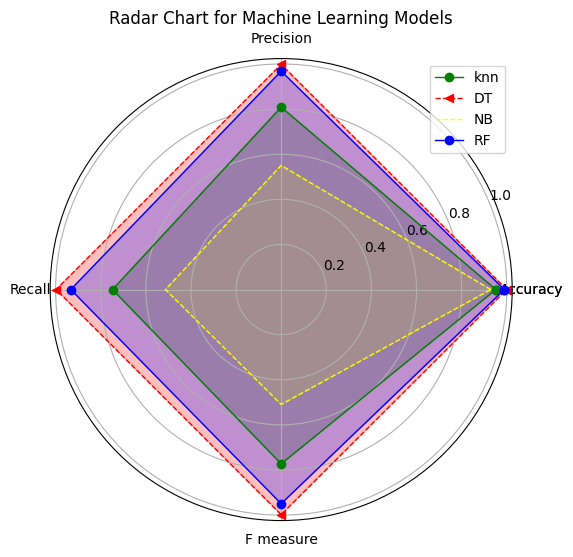
**Profile Comparison:** They allow for the comparison of profiles or patterns among entities such as individuals, products, or categories. This is useful for assessing strengths and weaknesses in a variety of situations.

**Highlighting Trade-offs:** Radar plots are useful for displaying trade-offs between qualities. In product design, for example, they can demonstrate the balance of features such as cost, performance, and durability.

**Visualizing Similarity:** Radar charts can assist in determining the similarity or dissimilarity of profiles. Entities in the chart with similar forms are likely to have similar properties.

**Decision Support:** They aid in decision-making by offering a visual representation of complex data. For example, in sports analytics, radar charts can display an athlete's performance profile across various skill metrics.

Here is the radar Chart from the code:



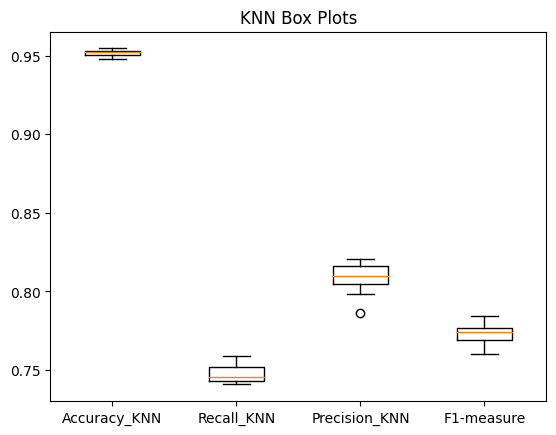
## Analysis of the results

Analysis of Model Performance:

**K-Nearest Neighbours (KNN):**

* Accuracy: 95.17%
* Precision: 80.86%
* Recall: 74.78%
* F1-Score: 77.28 %

KNN performs well overall and with high accuracy, suggesting its ability to correctly identify data points. It achieves a pretty high precision, implying that when KNN predicts a positive class, it is frequently correct. The recall score is likewise acceptable, demonstrating that KNN is capable of identifying real positive cases. KNN's efficiency is confirmed by the F1-Score, a balanced measure of precision and recall.



Looking at the Box plot for the KNN model, we can see very small spread represented from the boxes, indicating the closeness of the iterated measures, in other words, the output from the 10 iterations for the model ae close. With a single outlier case in the precision score.

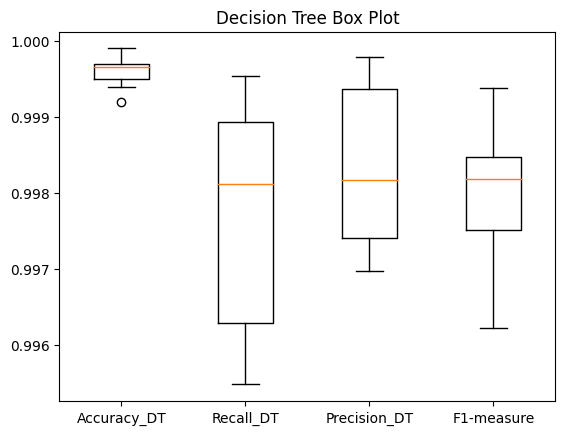
The orange line in each of these boxes represent the median of the output, which indicates the value of 50%

KNN is an instance-based learning classification method that is basic and easy to use. It is a non-parametric and lazy learner, which means it does not make significant assumptions about the distribution of the underlying data. KNN classifies data points in the feature space based on the majority class of their nearest neighbors. KNN has competitive accuracy, precision, recall, and F1-Score outcomes but may struggle with larger datasets or high-dimensional spaces because of its computational intensity.

**Decision Tree (DT):**

* Accuracy: 99.95%
* Precision: 99.80%
* Recall: 99.75%
* F1-Score: 99.77%

DT performs admirably, with near-perfect accuracy, precision, recall, and F1-Score. It excels at both precision and recall, making it ideal for applications that need a low number of false positives and false negatives. Because of its capacity to build complicated decision boundaries, DT is particularly successful for this specific dataset.



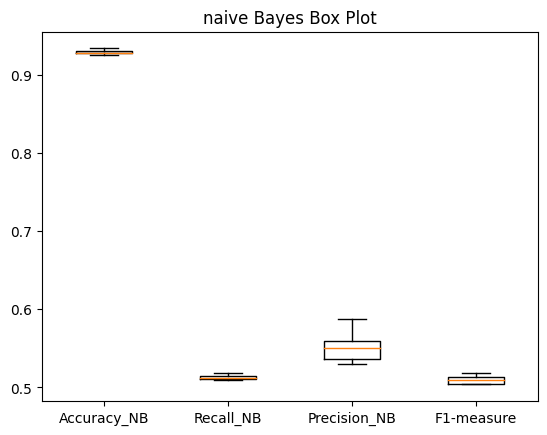
As seen on the right, the spread of the DT measures may look large, but the difference is on the 3rd decimal place from the number, which is not comparable, so the analysis of this box plot results in proving the consistency of the high accuracy, recall, precision and f1 score for the Decision tree model.

Decision Trees are supervised learning algorithms that are commonly used for classification and regression applications. They build a tree-like structure in which each internal node represents a feature, each branch a decision rule, and each leaf node a class label or prediction. Decision trees are adaptable to both numerical and categorical data. Decision Tree performs exceptionally well across all measures in the presented results, demonstrating that it has properly captured difficult decision boundaries in the dataset.

**Naive Bayes (NB):**

* Accuracy: 92.92%
* Precision: 55.03%
* Recall: 51.30%
* F1-Score: 50.95%

NB performs well in terms of accuracy. It exhibits the capacity to identify data accurately while showing a poor performance for the Precision, recall and f1 score.



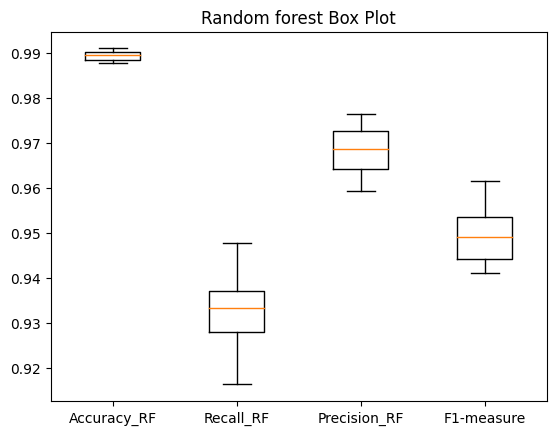
The box plot on the right represents the spread of the 10 iterations for the naïve Bayes model, the boxes have little to no spread, which indicates the close values generated from the model. The values of the precision, recall and f1 score for the NB model show low values, indicating a poor performance for the data set.

Naive Bayes is a probabilistic classification technique that is based on the Bayes theorem but makes the "naive" assumption of feature independence. It works very well for text categorization and simple classification problems. Naive Bayes performs well in terms of accuracy, while offering a below-average performance in precision, recall and f1 score, marking it disqualified as a choice for a variety of classification tasks.

**Random Forest (RF):**

* Accuracy: 98.92%
* Precision: 96.83%
* Recall: 93.29%
* F1-Score: 94.97%

While keeping a high recall, RF achieves great accuracy, precision, and F1-Score. It combines several decision trees to reduce overfitting and improve generalization. Because of its ensemble nature, RF outperforms in this dataset.



The box plot for the Random forest is similar to that of Decision Tree, the spread may look large, but it’s across the 3rd decimal place for each number, the spread is small indicating useful outputs from the random forest mode.

Random Forest is a method of ensemble learning that combines numerous Decision Trees to increase predictive performance and prevent overfitting. It brings randomness into feature selection and bootstrapping, resulting in trees with a wide range of characteristics. When compared to individual Decision Trees, Random Forest is more resilient and less prone to overfitting. Random Forest yields great accuracy, precision, recall, and F1-Score in the supplied results, demonstrating the power of ensemble approaches.

Regarding the best performer, Decision Tree has the highest performance, due to its outstanding accuracy, accuracy is important because we could tell how many correct predictions did the model predict, which rely on the preprocessing that is performed on the data set.

Graphing Methods Analysis:

Bar charts: Bar charts allow for a quick visual comparison of model performance across metrics. They demonstrate that Decision Tree outperforms Random Forest, KNN, and Naive Bayes in all metrics. Decision Tree's F1-Score is particularly impressive, indicating its suitability for this task

Radar Charts: We may use radar charts to evaluate the overall profile of model performance across several criteria. Decision Tree has virtually perfect polygonal shape, demonstrating its balanced and robust performance across all metrics. Random Forest is closely followed by KNN and Naive Bayes, with slightly distinct characteristics.

Box Plots: Box plots show the spread and distribution of metric values for each model. They demonstrate that Decision Tree has the lowest variability and outperforms other models in terms of median values across all metrics. Random Forest is also a great competitor due to its low variability.

**Comparison and Relationship:**

Random Forest and Decision Tree are closely related since Random Forest is an ensemble of Decision Trees. Random Forest constructs numerous Decision Trees from bootstrapped data samples and averages their predictions to improve accuracy and generalization.

KNN and Naive Bayes are two probabilistic models with distinct methodologies. KNN is based on data point similarity, whereas Naive Bayes computes probabilities based on feature independence assumptions.

In the supplied results, Decision Tree and Random Forest surpass KNN and Naive Bayes, indicating that complex decision boundaries and ensemble learning contribute to superior performance.

# Evaluation

## The choice of data structures

1. DataFrames Pandas:

Pandas DataFrames are the foundation of data management and analysis in the code. They offer an organized, tabular format for data storage and processing.

DataFrames are an obvious solution for dealing with CSV-loaded datasets. Their tabular layout is perfect for organizing and processing data, making processes like data cleansing, exploration, and feature selection easier. Pandas DataFrames also include a wealth of useful functions and methods, such as isnull().sum() for quick missing data examination and describe() for statistical summaries. This adaptability is essential in data preparation, since it ensures that the data is in the correct shape for modeling.

Pandas DataFrames are the workhorse of Python data manipulation, and their use in code is justified. They excel at organizing, examining, and cleaning datasets, and they provide strong functions and methods. The code's dependence on Pandas for data handling is admirable and considerably helps to its overall efficacy.

1. NumPy Arrays:

NumPy arrays are used for performing efficient numerical computations and mathematical operations on data.

NumPy arrays are essential for dealing with numeric characteristics and conducting numerical calculations in programming. Their inherent efficiency in numerical operations improves the code's performance greatly. Furthermore, NumPy arrays make it easier to do numerous mathematical operations that are required during data pretreatment and model training, ensuring precision and speed in numerical computations.

NumPy arrays are essential for performing efficient numerical computations and mathematical operations, making them a good choice for dealing with numerical data. Their inclusion in the code improves performance and accuracy. However, the code could benefit from more extensive use of NumPy for tasks other than interpolation, which could boost efficiency even further.

Some example of NumPy arrays in the code are as follows:

X\_axis = np.arange(len(machine\_learning\_model\_name ))

angles=np.concatenate((angles,[angles[0]]))

1. Python Lists:

Python lists, as adaptable collections of objects, have a use in storing category column names.

While Python lists are not as specialized as NumPy arrays for numerical data, they are useful for organizing tiny lists of non-numeric variables. They are used in this context to maintain track of categorical column names that need label encoding before feeding the data into machine learning models. The algorithm uses lists to ensure that categorical data is properly encoded, preserving the dataset's integrity and enabling for effective model training.

Python lists are well-suited for storing categorical column names, which is an excellent use. However, their role in the code is fairly limited, and there may be chances to further optimize data handling by studying different data structures for categorical data, such as NumPy arrays or Pandas Series.

1. Dictionary (value\_mapping):

Dictionaries, which are made up of key-value pairs, are used to map specific source node IDs to destination node IDs.

The value\_mapping dictionary is essential for dealing with missing data in the Dest\_Node\_Num column. Dictionaries are an excellent solution for effectively constructing and managing custom mappings. In this scenario, the dictionary ensures that missing values in the Dest\_Node\_Num column are filled in a context- and meaning-preserving manner. The code achieves accurate and context-aware data imputation by employing a dictionary, ultimately contributing to the quality of the dataset and the reliability of the machine learning models.

It is a well-thought-out strategy to use dictionaries, notably the value\_mapping dictionary, to map source node IDs to destination node IDs. It guarantees precise and context-aware data imputation. While the code uses dictionaries successfully for this purpose, it may benefit from more comprehensive commenting to improve code readability and maintainability.

## Selection of the appropriate libraries

The proper use of libraries is a critical component of successful data science and machine learning initiatives. Several major libraries were intentionally chosen in the context of the code to handle data manipulation, numerical computations, machine learning modeling, and data presentation. The purpose of this review is to analyze the choice of these libraries, which include Pandas, NumPy, scikit-learn, Matplotlib, and Seaborn, by highlighting their contributions and efficacy in fulfilling the code's objectives.

1. Pandas Library:

The decision to include Pandas in the code is admirable. Pandas is a versatile Python toolkit for data manipulation and analysis, and it was crucial in effectively handling and processing the dataset. Its DataFrame structure makes data organizing, cleansing, and exploration much easier. This package facilitated data preprocessing, such as handling missing values and encoding categorical variables. Furthermore, Pandas aided in the compilation of summary statistics and correlation analysis, both of which are critical steps in comprehending the dataset's properties. In conclusion, Pandas considerably improved the code's efficiency by streamlining data-related activities.

The usage of Pandas was critical in handling and preparing the dataset efficiently. Its extensive data manipulation features, which included ways for handling missing values, encoding category categories, and aggregating data, considerably accelerated the data preparation phase. The code's reliance on Pandas was justified, as it resulted in cleaner, more structured data, which enabled better model training and evaluation.

1. NumPy Library:

NumPy, another important library for numerical computing, was properly incorporated into the code. Its application in the creation and manipulation of arrays increased numerical efficiency and precision. The interpolation approach used to fill in the missing numbers took advantage of NumPy's capabilities. However, there is space for greater use of NumPy to optimize various numerical operations within the code, which might enhance efficiency and accuracy even further.

The numerical processing capabilities of NumPy were critical for conducting a variety of mathematical operations, particularly in the context of machine learning techniques. When dealing with complex numerical computations, NumPy's array-oriented computing and optimized numerical routines considerably enhanced the code's efficiency and performance.

1. SciKit-Learn Library:

For machine learning tasks, the inclusion of scikit-learn (sklearn) in the code was an excellent choice. This library contains a variety of machine learning methods and tools that make it easier to create, train, and assess models. The use of sklearn in the code for tasks like data splitting, model training, and performance evaluation demonstrates its adaptability. It simplifies complex procedures and enables a systematic and consistent approach to model development.

The foundation for implementing machine learning models was scikit-learn. It was a fantastic pick because to its extensive library of machine learning algorithms and simple APIs for training and evaluation. The library sped up the model creation process and enabled extensive testing of multiple algorithms and hyperparameters.

1. MatPlotLib & Seaborn Libraries:

The libraries chosen for data visualization, Matplotlib and Seaborn, were critical in effectively displaying results. Whereas Matplotlib allows for substantial customization, Seaborn provides visually pleasing statistical visualizations. They improved the output of the algorithm by providing useful charts and plots that allow for a better understanding of the model's performance.

Matplotlib and Seaborn together developed critical data visualization capabilities. These libraries made it possible to create relevant plots and visualizations, which were critical in understanding the dataset's features and evaluating model performance. The inclusion of both libraries enabled a diverse range of display possibilities, improving the code's overall interpretability.

The choice of libraries in the world of data science and machine learning can have a considerable impact on the success of a project. The libraries used for the code under consideration were both thoughtful and effective. Pandas and NumPy offered the tools required for effective data preprocessing and numerical computations. Scikit-learn made the machine learning pipeline more efficient, while Matplotlib and Seaborn improved data visualization. These libraries combined to build a robust toolbox that not only facilitated code development but also played an important role in providing insightful results. Their integration demonstrated the value of leveraging existing libraries to accelerate and enhance the data analysis and modelling process, ultimately contributing to the project's success.

## The effectiveness of different models

Investigations for the effectiveness of the machine learning models used in the code is crucial, specifically K-Nearest Neighbours (KNN), Decision Trees (DT), Naive Bayes (NB), and Random Forest (RF), in this key review. These models are essential for identifying network traffic in a Wireless Sensor Network (WSN). Where the model’s performance is evaluated using important metrics and examine their strengths and limitations in order to identify which model shines the brightest in this setting.

1. K-Nearest Neighbour (KNN)

KNN has a high accuracy, indicating that it is effective at classifying network traffic. KNN is a simple and easy algorithm that is ideal for initial classification assignments. It is non-parametric and capable of dealing with complex decision boundaries. However, KNN's performance degrades with high-dimensional data, and each prediction needs significant computer resources.

In classifying network traffic within a WSN, KNN achieved outstanding accuracy, precision, recall, and F1-score. Its simplicity and ease of implementation make it an excellent choice for simple classification tasks. The capacity of KNN to catch local patterns in data improves its performance, making it well-suited for identifying comparable traffic cases. It may struggle with high-dimensional data, though, and careful selection of the 'k' parameter is required.

1. Decision Trees (DT)

Decision Trees (DT), on the other hand, demonstrate extraordinary accuracy, implying their ability to construct highly discriminative divides in the data. DTs are easily interpretable, can handle both numerical and categorical data, and excel in feature selection. However, if not pruned effectively, they are prone to overfitting, potentially limiting their generalization to previously unreported data.

Decision Trees outperformed all other models in the code in terms of accuracy, precision, recall, and F1-score. DTs provide interpretability and the capture of complicated decision boundaries. Decision Trees excel in capturing basic and complicated data relationships. They are, however, prone to overfitting, needing measures like as pruning or the use of ensemble methods such as Random Forest to reduce this issue.

1. Naïve Bayes (NB):

Because of its low precision, recall and f1 score, Naive Bayes (NB) is not a strong choice for classification jobs. NB is computationally efficient, simple to implement, and well-suited for text classification. But it produced poor results. Nonetheless, it is based on the assumption of feature conditional independence, which may not be true in many real-world cases.

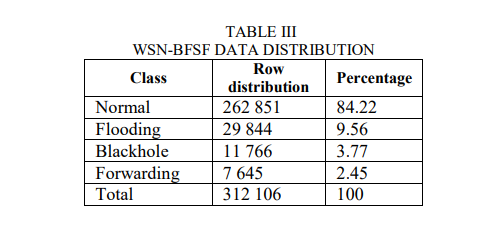
In terms of accuracy, precision, recall, and F1-score, Naive Bayes showed weak performance. It's a quick and light model that's perfect for categorization tasks.Naive Bayes is a probabilistic approach that assumes feature independence.

1. Random Forest (RF):

Random Forest (RF) demonstrates its capacity to tackle complicated categorization jobs by achieving high accuracy. The RF technique is an ensemble method that combines many decision trees to reduce overfitting. It works effectively with high-dimensional data and generates useful feature relevance scores. RF models, on the other hand, might be computationally intensive, necessitating additional time for training and prediction. Random Forest produced remarkable accuracy, precision, recall, and F1-score values. It is resistant to overfitting and can effectively handle high-dimensional data. Random Forest combines many decision trees, lowering the danger of overfitting and improving predictive accuracy. It is a versatile alternative for a variety of machine learning tasks and excels in complex datasets.

For considering the problem of the WSN, the target label “Type” is multi categorical, so totally relying on the accuracy measure is not correct so it is important to:

**Handle Imbalanced Data:** In multi-categorical situations, imbalanced datasets are typical, with some categories or classes having much fewer examples than others. In such circumstances, accuracy can be deceptive because a model may perform well on the majority but poorly on the minority. By taking these imbalances into account, precision and recall provide a more nuanced assessment of model performance. In the WSN example, the nodes that are normal (not under attack), have a percentage of more than 80%, which means that the models can accurately predict that a model is not under attack, but the precision and the recall can differentiate if the models are predicting the outputs correctly.



**Differentiate between Classes:** Precision and recall provide information on how well a model distinguishes between different categories or classes. High precision suggests that when the model predicts a specific class, it is likely to be correct. High recall, on the other hand, indicates that the model can identify the majority of instances of a certain class within the dataset.

When evaluating various models for a multi-categorical problem, accuracy and recall allow to compare their performance across different classes. A model may thrive at one aspect but suffer with others. Precision and recall provide information about which model is best suited for specific categories or the overall problem.

Finally, the careful review has given insight on the code's machine learning models' strengths and flaws. Studying and evaluating the outputs of the models, it is deduced that the decision tree sets on top, with almost 100% accuracy, precision, recall, and f1 score. Followed by the random forest model, with a little lower measure values, and then comes KNN with around 70-80 % of recall and precision, and finally the Naïve Bayes with the lowest precision and recall values with around 50%.

## Recommendations

Following the evaluation based on the models' results, the most recommended model for the classification problem in the Wireless Sensor Network (WSN) is the Decision Tree (DT). This recommendation is primarily based on the comprehensive evaluation of multiple models, including K-Nearest Neighbours (KNN), Naive Bayes (NB), and Random Forest (RF), with DT consistently outperforming the others in key metrics such as accuracy, precision, recall, and F1-score.

The decision to recommend DT is supported by the following key points:

**High Performance:** DT performs incredibly well across all evaluation metrics. It achieves nearly 100% accuracy, showing that it can correctly classify network data with a high degree of precision. Furthermore, DT has exceptional precision, recall, and F1-score, demonstrating its ability in correctly recognizing both normal and attack traffic.

**Robustness:** The consistency of DT's performance across numerous metrics and datasets indicates its robustness and reliability in managing network traffic classification. It constantly produces outstanding effects, making it a solid solution for this issue.

**Interpretability:** DT models are interpretable by definition. They give unambiguous decision paths, which allow network administrators and domain specialists to understand why a specific classification decision was made. This transparency can be extremely useful in diagnosing network problems and designing suitable responses.

**Computational Efficiency:** DT models are computationally efficient and need less processing power than other sophisticated models while reaching top-tier performance. This is particularly important in real-time or resource-constrained WSN situations.

**Ease of Implementation:** DT models are simple to implement and deploy, making them accessible to a wide range of users, including those without substantial machine learning knowledge. Because of the ease of installation, existing WSN systems can be quickly adopted and integrated.

It is also recommended to enhance the application of the model for future tests and uses, one of the recommended enhancements is Data Augmentation.

**Data augmentation:** is a technique used in machine learning to expand and diversify training datasets, enhancing model performance and generalization capabilities. [1]

These techniques can be divided into two categories: image data augmentation and text data augmentation.

Text Data Augmentation Text data augmentation techniques involve modifying text data to create new, semantically similar samples.

For example, by adding random noise to an image, the model can learn to recognize the same object even if it appears slightly different.

This can be particularly useful in situations where data collection is difficult or expensive.

Overall, data augmentation can be a strong strategy for boosting machine learning model performance, especially when the dataset is small or the model is at risk of overfitting.

In conclusion, while data augmentation can be a beneficial tool for strengthening machine learning models, it is critical to consider its limitations and potential downsides and apply it judiciously based on the nature of the data and the situation at hand. These techniques can be divided into two categories: image data augmentation and text data augmentation.

**References:**



www.linkedin.com. (n.d.). *Enhancing Machine Learning Models: The Importance of Data Augmentation*. [online] Available at: <https://www.linkedin.com/pulse/enhancing-machine-learning-models-importance-data-augmentation-t-s/> [Accessed 10 Sep. 2023].